

Contents

1 Angle between two Planes in 3D	15
Source	21
2 Angular Sweep (Maximum points that can be enclosed in a circle of given radius)	22
Source	28
3 Arc length from given Angle	29
Source	34
4 Area of a Circular Sector	35
Source	39
5 Area of a Circumscribed Circle of a Square	40
Source	44
6 Area of a Hexagon	45
Source	48
7 Area of a polygon with given n ordered vertices	49
Source	55
8 Area of square Circumscribed by Circle	56
Source	60
9 Bresenham's Algorithm for 3-D Line Drawing	61
Source	65
10 Calculate Volume and Surface area Of Sphere	66
Source	72
11 Calculate Volume of Dodecahedron	73
Source	76
12 Calculate volume and surface area of a cone	77
Source	82
13 Centered Dodecagonal Number	83
Source	85

14 Centered Octadecagonal Number	86
Source	90
15 Centered Octagonal Number	91
Source	95
16 Centered Pentadecagonal Number	96
Source	100
17 Centered cube number	101
Source	105
18 Centered decagonal number	106
Source	111
19 Centered dodecahedral number	112
Source	116
20 Centered nonadecagonal number	117
Source	121
21 Centered pentagonal number	122
Source	127
22 Centered tridecagonal number	128
Source	132
23 Chain Code for 2D Line	133
Source	137
24 Check if a given circle lies completely inside the ring formed by two concentric circles	138
Source	144
25 Check if a line at 45 degree can divide the plane into two equal weight parts	145
Source	147
26 Check if a line passes through the origin	148
Source	151
27 Check if a line touches or intersects a circle	152
Source	157
28 Check if a point lies on or inside a rectangle Set-2	158
Source	162
29 Check if four segments form a rectangle	163
Source	166
30 Check if given four integers (or sides) make rectangle	167

Source	173
31 Check if it is possible to create a polygon with a given angle	174
Source	177
32 Check if right triangle possible from given area and hypotenuse	178
Source	183
33 Check if three straight lines are concurrent or not	184
Source	189
34 Check if two given circles touch or intersect each other	190
Source	195
35 Check whether a given point lies inside a rectangle or not	196
Source	202
36 Check whether a given point lies inside a triangle or not	203
Source	209
37 Check whether a given point lies on or inside the rectangle Set 3	210
Source	214
38 Check whether a point exists in circle sector or not.	215
Source	220
39 Check whether a point lies inside a sphere or not	221
Source	227
40 Check whether four points make a parallelogram	228
Source	231
41 Check whether given circle resides in boundary maintained by two other circles	232
Source	238
42 Check whether triangle is valid or not if sides are given	239
Source	243
43 Circle and Lattice Points	244
Source	250
44 Classify a triangle	251
Source	254
45 Closest Pair of Points using Divide and Conquer algorithm	255
Source	261
46 Closest Pair of Points O(nlogn) Implementation	262
Source	266

47 Convex Hull using Divide and Conquer Algorithm	267
Source	274
48 Convex Hull Set 1 (Jarvis's Algorithm or Wrapping)	275
Source	281
49 Convex Hull Set 2 (Graham Scan)	282
Source	288
50 Coordinates of rectangle with given points lie inside	289
Source	294
51 Count Integral points inside a Triangle	295
Source	298
52 Count maximum points on same line	299
Source	302
53 Count of acute, obtuse and right triangles with given sides	303
Source	306
54 Count of different straight lines with total n points with m collinear	307
Source	313
55 Count of obtuse angles in a circle with 'k' equidistant points between 2 given points	314
Source	319
56 Count of parallelograms in a plane	320
Source	322
57 Count ways to divide circle using N non-intersecting chords	323
Source	327
58 Deleting points from Convex Hull	328
Source	335
59 Direction of a Point from a Line Segment	336
Source	338
60 Distance between a point and a Plane in 3 D	339
Source	345
61 Distance between two parallel Planes in 3-D	346
Source	353
62 Divide cuboid into cubes such that sum of volumes is maximum	354
Source	359
63 Draw geometric shapes on images using OpenCV	360
Source	367

64 Dynamic Convex hull Adding Points to an Existing Convex Hull	368
Source	372
65 Enneadecagonal number	373
Source	377
66 Equable Shapes	378
Source	385
67 Find Corners of Rectangle using mid points	386
Source	390
68 Find Simple Closed Path for a given set of points	391
Source	395
69 Find all angles of a given triangle	396
Source	400
70 Find all angles of a triangle in 3D	401
Source	404
71 Find all possible coordinates of parallelogram	405
Source	409
72 Find all sides of a right angled triangle from given hypotenuse and area	
Set 1	410
Source	413
73 Find an Integer point on a line segment with given two ends	414
Source	416
74 Find area of parallelogram if vectors of two adjacent sides are given	417
Source	419
75 Find if it's possible to rotate the page by an angle or not.	420
Source	425
76 Find if two rectangles overlap	426
Source	427
77 Find intersection point of lines inside a section	428
Source	430
78 Find maximum and minimum distance between magnets	431
Source	437
79 Find minimum radius such that atleast k point lie inside the circle	438
Source	443
80 Find mirror image of a point in 2-D plane	444
Source	447

81 Find number of diagonals in n sided convex polygon	448
Source	451
82 Find perimeter of shapes formed with 1s in binary matrix	452
Source	456
83 Find points at a given distance on a line of given slope	457
Source	460
84 Find the Missing Point of Parallelogram	461
Source	468
85 Find the Surface area of a 3D figure	469
Source	477
86 Find the center of the circle using endpoints of diameter	478
Source	482
87 Find the dimensions of Right angled triangle	483
Source	488
88 Find the other end point of a line with given one end and mid	489
Source	493
89 Find the perimeter of a cylinder	494
Source	497
90 Finding Quadrant of a Coordinate with respect to a Circle	498
Source	506
91 Finding the vertex, focus and directrix of a parabola	507
Source	512
92 Given equation of a circle as string, find area	513
Source	519
93 Given n line segments, find if any two segments intersect	520
Source	523
94 Haversine formula to find distance between two points on a sphere	524
Source	528
95 Hendecagonal number	529
Source	534
96 Heptadecagonal number	535
Source	539
97 Heptagonal number	540
Source	544

98 Hexadecagonal number	545
Source	549
99 How to check if a given point lies inside or outside a polygon?	550
Source	554
100 How to check if given four points form a square	555
Source	557
101 How to check if two given line segments intersect?	558
Source	562
102 Icosagonal number	563
Source	567
103 Icosahedral Number	568
Source	571
104 Icosidigonal number	572
Source	576
105 Intersecting rectangle when bottom-left and top-right corners of two rectangles are given	577
Source	582
106 Klee's Algorithm (Length Of Union Of Segments of a line)	583
Source	586
107 LS3/NS3 sphere generation algorithm and its implementation	587
Source	596
108 Lexicographically Kth smallest way to reach given coordinate from origin	597
Source	602
109 Line Clipping Set 1 (Cohen–Sutherland Algorithm)	603
Source	612
110 Maximize a value for a semicircle of given radius	613
Source	617
111 Maximize volume of cuboid with given sum of sides	618
Source	627
112 Maximum area of quadrilateral	628
Source	633
113 Maximum distinct lines passing through a single point	634
Source	636
114 Maximum height when coins are arranged in a triangle	637
Source	643

115 Maximum integral co-ordinates with non-integer distances	644
Source	645
116 Maximum number of 2×2 squares that can be fit inside a right isosceles triangle	646
Source	650
117 Maximum number of pieces in N cuts	651
Source	653
118 Maximum number of squares that can fit in a right angle isosceles triangle	654
Source	658
119 Maximum points of intersection n circles	659
Source	662
120 Maximum possible intersection by moving centers of line segments	663
Source	667
121 Minimum Cost Polygon Triangulation	668
Source	673
122 Minimum Perimeter of n blocks	674
Source	679
123 Minimum area of a Polygon with three points given	680
Source	682
124 Minimum block jumps to reach destination	683
Source	685
125 Minimum distance to travel to cover all intervals	686
Source	689
126 Minimum height of a triangle with given base and area	690
Source	693
127 Minimum length of the shortest path of a triangle	694
Source	696
128 Minimum lines to cover all points	697
Source	699
129 Minimum number of points to be removed to get remaining points on one side of axis	700
Source	702
130 Minimum number of square tiles required to fill the rectangular floor	703
Source	710

131 Minimum revolutions to move center of a circle to a target	711
Source	715
132 Minimum squares to cover a rectangle	716
Source	717
133 Mirror of a point through a 3 D plane	718
Source	723
134 Neighbors of a point on a circle using Bresenham's algorithm	724
Source	729
135 Non-crossing lines to connect points in a circle	730
Source	737
136 Number of Integral Points between Two Points	738
Source	740
137 Number of Pentagons and Hexagons on a Football	741
Source	744
138 Number of Triangles that can be formed given a set of lines in Euclidean Plane	745
Source	749
139 Number of horizontal or vertical line segments to connect 3 points	750
Source	757
140 Number of jump required of given length to reach a point of form (d, 0) from origin in 2D plane	758
Source	764
141 Number of ordered points pair satisfying line equation	765
Source	765
142 Number of parallelograms when n horizontal parallel lines intersect m vertical parallel lines	776
Source	780
143 Number of possible pairs of Hypotenuse and Area to form right angled triangle	781
Source	787
144 Number of quadrilaterals possible from the given points	788
Source	796
145 Number of rectangles in N*M grid	797
Source	800
146 Number of rectangles in a circle of radius R	801

Source	811
147 Number of squares of maximum area in a rectangle	812
Source	817
148 Number of triangles in a plane if no more than two points are collinear	818
Source	823
149 Number of unique rectangles formed using N unit squares	824
Source	828
150 Optimum location of point to minimize total distance	829
Source	832
151 Orientation of 3 ordered points	833
Source	836
152 Paper Cut into Minimum Number of Squares	837
Source	841
153 Path in a Rectangle with Circles	842
Source	846
154 Pentagonal Pyramidal Number	847
Source	854
155 Pentatope number	855
Source	859
156 Perpendicular distance between a point and a Line in 2 D	860
Source	864
157 Pizza cut problem (Or Circle Division by Lines)	865
Source	867
158 Polygon Clipping Sutherland–Hodgman Algorithm	868
Source	874
159 Probability that the pieces of a broken stick form a n sided polygon	875
Source	876
160 Program for Area And Perimeter Of Rectangle	877
Source	881
161 Program for Area Of Square	882
Source	885
162 Program for Circumference of a Parallelogram	886
Source	890
163 Program for Point of Intersection of Two Lines	891

Source	896
164 Program for Surface Area of Octahedron	897
Source	900
165 Program for Volume and Surface Area of Cube	901
Source	905
166 Program for Volume and Surface Area of Cuboid	906
Source	910
167 Program for Volume and Surface area of Frustum of Cone	911
Source	919
168 Program for distance between two points on earth	920
Source	927
169 Program for volume of Pyramid	928
Source	935
170 Program to calculate Area Of Octagon	936
Source	939
171 Program to calculate Volume and Surface area of Hemisphere	940
Source	944
172 Program to calculate area and perimeter of Trapezium	945
Source	950
173 Program to calculate area and perimeter of equilateral triangle	951
Source	956
174 Program to calculate area and volume of a Tetrahedron	957
Source	963
175 Program to calculate area of Circumcircle of an Equilateral Triangle	964
Source	968
176 Program to calculate area of Enneagon	969
Source	972
177 Program to calculate the Surface Area of a Triangular Prism	973
Source	977
178 Program to calculate volume of Ellipsoid	978
Source	981
179 Program to calculate volume of Octahedron	982
Source	986
180 Program to check congruency of two triangles	987

Source	993
181 Program to check if tank will overflow, underflow or filled in given time	994
Source	999
182 Program to check if the points are parallel to X axis or Y axis	1000
Source1004
183 Program to check if three points are collinear	1005
Source1011
184 Program to check if water tank overflows when n solid balls are dipped in the water tank	1012
Source1019
185 Program to check similarity of given two triangles	1020
Source1033
186 Program to check whether 4 points in a 3-D plane are Coplanar	1034
Source1037
187 Program to determine the octant of the axial plane	1038
Source1042
188 Program to find third side of triangle using law of cosines	1043
Source1049
189 Program to find Circumcenter of a Triangle	1050
Source1054
190 Program to find Circumference of a Circle	1055
Source1059
191 Program to find Perimeter / Circumference of Square and Rectangle	1060
Source1066
192 Program to find area of a Circular Segment	1067
Source1075
193 Program to find area of a Trapezoid	1076
Source1079
194 Program to find area of a circle	1080
Source1083
195 Program to find area of a triangle	1084
Source1093
196 Program to find equation of a plane passing through 3 points	1094
Source1101

197 Program to find line passing through 2 Points	1102
Source1104
198 Program to find slope of a line	1105
Source1108
199 Program to find smallest difference of angles of two parts of a given circle	1109
Source1114
200 Program to find the Volume of a Triangular Prism	1115
Source1119
201 Program to find the mid-point of a line	1120
Source1123
202 Pythagorean Quadruple	1124
Source1128
203 Queries on count of points lie inside a circle	1129
Source1134
204 Quickhull Algorithm for Convex Hull	1135
Source1139
205 Reflection of a point about a line in C++	1140
Source1143
206 Reflection of a point at 180 degree rotation of another point	1144
Source1147
207 Regular polygon using only 1s in a binary numbered circle	1148
Source1155
208 Represent a given set of points by the best possible straight line	1156
Source1163
209 Rotation of a point about another point in C++	1164
Source1166
210 Section formula (Point that divides a line in given ratio)	1167
Source1172
211 Shortest distance between a Line and a Point in a 3-D plane	1173
Source1178
212 Slope of perpendicular to line	1179
Source1183
213 Smallest square formed with given rectangles	1184
Source1186

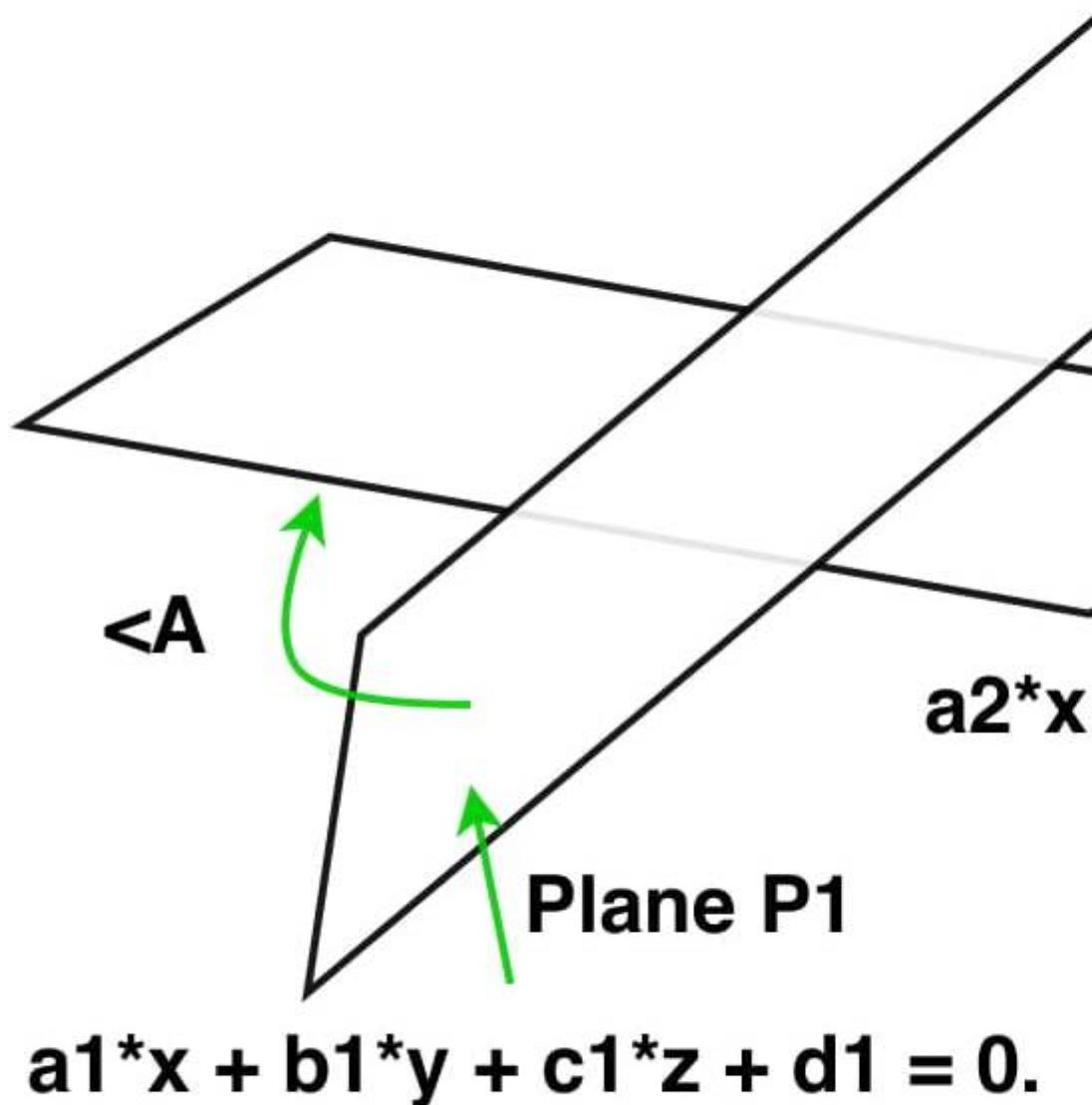
214 Sum of Areas of Rectangles possible for an array	1187
Source1198
215 Sum of Manhattan distances between all pairs of points	1199
Source1208
216 Tangents between two Convex Polygons	1209
Source1214
217 Tetradezagonal number	1215
Source1219
218 Total area of two overlapping rectangles	1220
Source1222
219 Triangle with no point inside	1223
Source1228
220 Triangular Matchstick Number	1229
Source1232
221 Ways to choose three points with distance between the most distant points $\leq L$	1233
Source1239
222 n'th Pentagonal Number	1240
Source1243

Chapter 1

Angle between two Planes in 3D

Angle between two Planes in 3D - GeeksforGeeks

Given two planes **P1:** $a_1 * x + b_1 * y + c_1 * z + d_1 = 0$ and **P2:** $a_2 * x + b_2 * y + c_2 * z + d_2 = 0$. The task is to find the angle between these two planes in 3D.



$$a_1*x + b_1*y + c_1*z + d_1 = 0.$$

Examples:

Input: a1 = 1, b1 = 1, c1 = 2, d1 = 1, a2 = 2, b2 = -1, c2 = 1, d2 = -4

Output: Angle is 60.0 degree

Input: a1 = 2, b1 = 2, c1 = -3, d1 = -5, a2 = 3, b2 = -3, c2 = 5, d2 = -6

Output: Angle is 123.696598882 degree

Approach: Consider the below equations of given two planes:

P1 : a1 * x + b1 * y + c1 * z + d1 = 0 and,

P2 : a2 * x + b2 * y + c2 * z + d2 = 0,

where a1, b1, c1, and a2, b2, c2 are direction ratios of normal to the plane P1 and P2.

The angle between two planes is equal to the angle determined by the normal vectors of the planes.

Angle between these planes is given by using the following formula:-

$$\cos A = \frac{a_1 a_2 + b_1 b_2 + c_1 c_2}{\sqrt{a_1^2 + b_1^2 + c_1^2} \sqrt{a_2^2 + b_2^2 + c_2^2}}$$

Using inverse property, we get:

$$A = \arccos \left(\frac{a_1 a_2 + b_1 b_2 + c_1 c_2}{\sqrt{a_1^2 + b_1^2 + c_1^2} \sqrt{a_2^2 + b_2^2 + c_2^2}} \right)$$

Below is the implementation of the above formulae:

C

```
// C program to find
// the Angle between
// two Planes in 3 D.
#include<stdio.h>
#include<math.h>

// Function to find Angle
void distance(float a1, float b1,
              float c1, float a2,
              float b2, float c2)
{
    float d = (a1 * a2 + b1 *
               b2 + c1 * c2);
    float e1 = sqrt(a1 * a1 + b1 *
                   b1 + c1 * c1);
    float e2 = sqrt(a2 * a2 + b2 *
                   b2 + c2 * c2);
```

```
d = d / (e1 * e2);
float pi = 3.14159;
float A = (180 / pi) * (acos(d));
printf("Angle is %.2f degree", A);
}

// Driver Code
int main()
{
    float a1 = 1;
    float b1 = 1;
    float c1 = 2;
    float d1 = 1;
    float a2 = 2;
    float b2 = -1;
    float c2 = 1;
    float d2 = -4;
    distance(a1, b1, c1,
              a2, b2, c2);
    return 0;
}

// This code is contributed
// by Amber_Saxena.
```

Java

```
// Java program to find
// the Angle between
// two Planes in 3 D.
import java.io.*;
import java.lang.Math;

class GFG
{

    // Function to find Angle
    static void distance(float a1, float b1,
                          float c1, float a2,
                          float b2, float c2)
    {

        float d = (a1 * a2 + b1 *
                   b2 + c1 * c2);
        float e1 = (float)Math.sqrt(a1 * a1 + b1 *
                                   b1 + c1 * c1);
        float e2 = (float)Math.sqrt(a2 * a2 + b2 *
                                   b2 + c2 * c2);
```

```
d = d / (e1 * e2);
float pi = (float)3.14159;
float A = (180 / pi) * (float)(Math.acos(d));
System.out.println("Angle is "+ A +" degree");
}

// Driver code
public static void main(String[] args)
{
    float a1 = 1;
    float b1 = 1;
    float c1 = 2;
    float d1 = 1;
    float a2 = 2;
    float b2 = -1;
    float c2 = 1;
    float d2 = -4;
    distance(a1, b1, c1,
              a2, b2, c2);
}
}

// This code is contributed
// by Amber_Saxena.
```

Python

```
# Python program to find the Angle between
# two Planes in 3 D.

import math

# Function to find Angle
def distance(a1, b1, c1, a2, b2, c2):

    d = ( a1 * a2 + b1 * b2 + c1 * c2 )
    e1 = math.sqrt( a1 * a1 + b1 * b1 + c1 * c1)
    e2 = math.sqrt( a2 * a2 + b2 * b2 + c2 * c2)
    d = d / (e1 * e2)
    A = math.degrees(math.acos(d))
    print("Angle is"), A, ("degree")

# Driver Code
a1 = 1
b1 = 1
c1 = 2
d1 = 1
a2 = 2
```

```
b2 = -1
c2 = 1
d2 = -4
distance(a1, b1, c1, a2, b2, c2)
```

C#

```
// C# program to find
// the Angle between
// two Planes in 3 D.
using System;

class GFG
{
    // Function to find Angle
    static void distance(float a1, float b1,
    float c1, float a2,
    float b2, float c2)
    {
        float d = (a1 * a2 + b1 *
        b2 + c1 * c2);
        float e1 = (float)Math.Sqrt(a1 * a1 + b1 *
        b1 + c1 * c1);
        float e2 = (float)Math.Sqrt(a2 * a2 + b2 *
        b2 + c2 * c2);
        d = d / (e1 * e2);
        float pi = (float)3.14159;
        float A = (180 / pi) * (float)(Math.Acos(d));
        Console.WriteLine("Angle is "+ A +" degree");
    }

    // Driver code
    public static void Main()
    {
        float a1 = 1;
        float b1 = 1;
        float c1 = 2;
        float a2 = 2;
        float b2 = -1;
        float c2 = 1;

        distance(a1, b1, c1,
        a2, b2, c2);
    }
}

// This code is contributed
// by ChitraNayal
```

PHP

```
<?php
// PHP program to find the Angle
// between two Planes in 3 D.

// Function to find Angle
function distance($a1, $b1,
                  $c1, $a2,
                  $b2, $c2)
{
    $d = ($a1 * $a2 + $b1 *
          $b2 + $c1 * $c2);
    $e1 = sqrt($a1 * $a1 + $b1 *
               $b1 + $c1 * $c1);
    $e2 = sqrt($a2 * $a2 + $b2 *
               $b2 + $c2 * $c2);
    $d = $d / ($e1 * $e2);
    $pi = 3.14159;
    $A = (180 / $pi) * (acos($d));
    echo sprintf("Angle is %.2f degree", $A);
}

// Driver Code
$a1 = 1;
$b1 = 1;
$c1 = 2;
$d1 = 1;
$a2 = 2;
$b2 = -1;
$c2 = 1;
$d2 = -4;
distance($a1, $b1, $c1,
          $a2, $b2, $c2);

// This code is contributed
// by Amber_Saxena.
?>
```

Output:

Angle is 60.0 degree

Improved By : [Amber_Saxena](#), [ChitraNayal](#)

Source

<https://www.geeksforgeeks.org/angle-between-two-planes-in-3d/>

Chapter 2

Angular Sweep (Maximum points that can be enclosed in a circle of given radius)

Angular Sweep (Maximum points that can be enclosed in a circle of given radius) - Geeks-forGeeks

Given ‘n’ points on 2-D plane, find the maximum number of points that can be enclosed by a fixed-radius circle of radius ‘R’.

Note: The point is considered to be inside the circle even when it lies on the circumference.

Examples:

```
Input : R = 1
points[] = {(6.47634, 7.69628), (5.16828 4.79915),
             (6.69533 6.20378)}
```

Output : 2

The maximum number of points are 2

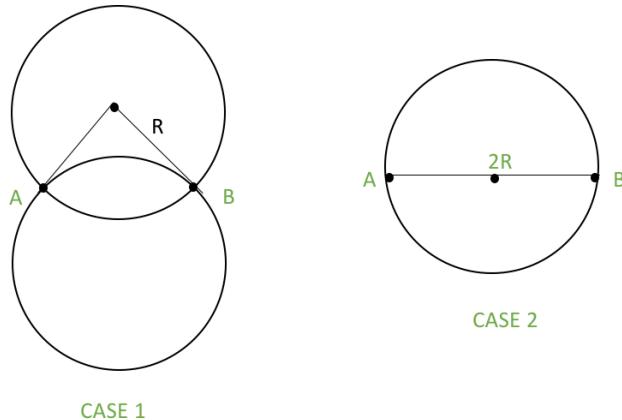
```
Input : R = 1
points[] = {(6.65128, 5.47490), (6.42743, 6.26189)
             (6.35864, 4.61611), (6.59020 4.54228), (4.43967 5.70059)
             (4.38226, 5.70536), (5.50755 6.18163), (7.41971 6.13668)
             (6.71936, 3.04496), (5.61832, 4.23857), (5.99424, 4.29328)
             (5.60961, 4.32998), (6.82242, 5.79683), (5.44693, 3.82724)
             (6.70906, 3.65736), (7.89087, 5.68000), (6.23300, 4.59530)
             (5.92401, 4.92329), (6.24168, 3.81389), (6.22671, 3.62210)}
```

Output : 11

The maximum number of points are 11

Naive Algorithm

1. For an arbitrary pair of points in the given set (say A and B), construct the circles with radius ‘R’ that touches both the points. There are maximum 2 such possible circles. As we can see here maximum possible circles is for CASE 1 i.e. 2.



2. For each of the constructed circle, check for each point in the set if it lies inside the circle or not.
3. The circle with maximum number of points enclosed is returned.

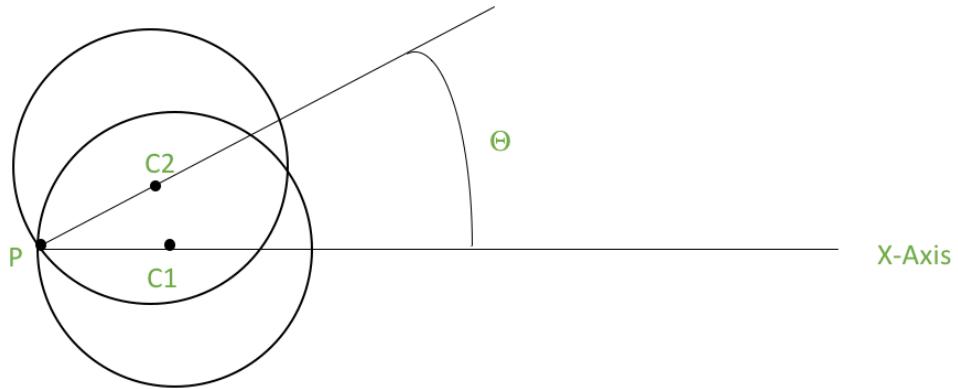
Time Complexity: There are ${}^n C_2$ pair of points corresponding to which we can have $2^n C_2$ circles at maximum. For each circle, $(n-2)$ points have to be checked. This makes the naive algorithm $O(n^3)$.

Angular Sweep Algorithm

By using Angular Sweep, we can solve this problem in $O(n^2 \log n)$. The basic logical idea of this algorithm is described below.

We pick an arbitrary point P from the given set. We then rotate a circle with fixed-radius ‘R’ about the point P. During the entire rotation P lies on the circumference of the circle and we maintain a count of the number of points in the circle at a given value of Θ where the parameter Θ determines the angle of rotation. The state of a circle can thus be determined by a single parameter Θ because the radius is fixed.

We can also see that the value of the count maintained will change only when a point from the set enters or exits the circle.



In the given diagram, C1 is the circle with $\Theta = 0$ and C2 is the circle constructed when we rotate the circle at a general value of Θ .

After this, the problem reduces to, how to maintain the value of count.

For any given point except P (say Q), we can easily calculate the value of Θ for which it enters the circle (Let it be α) and the value of Θ for which it exits the circle (Let it be β). We have angles A and B defined as under,

- A is the angle between PQ and the X-Axis.
- B is the angle between PC and PQ where C is the center of the circle.

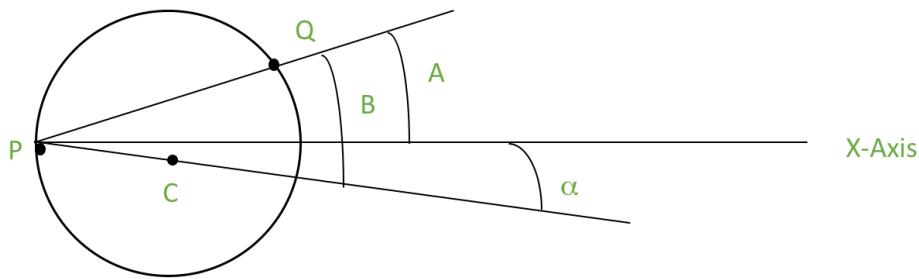
where, x and y represent the coordinates of a point and 'd' is the distance between P and Q.

Now, from the diagrams we can see that,

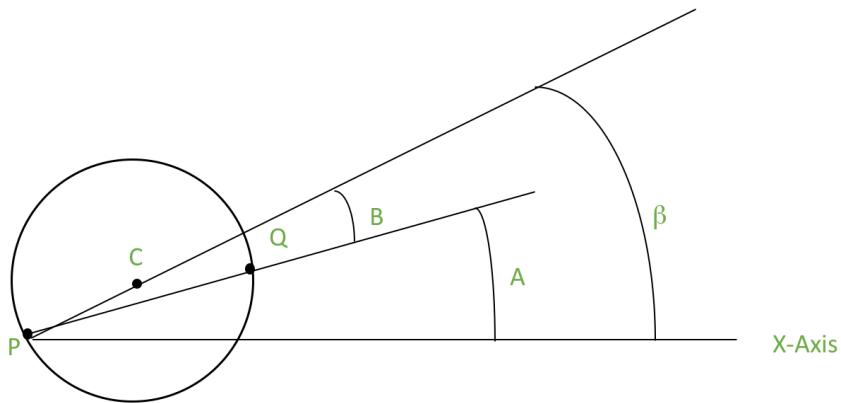
$$\begin{aligned} &= A - B \\ &= A + B \end{aligned}$$

(Note: All angles are w.r.t. to X-Axis. Thus, it becomes 'A-B' and not 'B-A').

When Q enters the circle



When Q exits the circle



We can calculate angles A and B for all points excluding P. Once these angles are found, we sort them and then traverse them in increasing order. Now we maintain a counter which tells us how many points are inside the circle at a particular moment.

Count will change only when a point enters the circle or exits it. In case we find an entry angle we increase the counter by 1 and in case we find an exit angle we decrease the counter by 1. The check that the angle is entry or exit can be easily realised using a flag.

Proceeding like this, the counter always gives us a valid value for number of points inside the circle in a particular state.

Important Note: The points which have ' $d > 2R$ ' do not have to be considered because they will never enter or exit the circle.

The angular sweep algorithm can be described as:

1. Calculate the distance between every pair of nC_2 points and store them.
2. For an arbitrary point (say P), get the maximum number of points that can lie inside the circle rotated about P using the `getPointsInside()` function.

3. The maximum of all values returned will be the final answer.

This algorithm has been described in the following C++ implementation.

```
// C++ program to find the maximum number of
// points that can be enclosed by a fixed-radius
// circle
#include <bits/stdc++.h>
using namespace std;

const int MAX_POINTS = 500;

// complex class which is available in STL has
// been used to implement points. This helps to
// ensure greater functionality easily
typedef complex<double> Point;

Point arr[MAX_POINTS];
double dis[MAX_POINTS][MAX_POINTS];

// This function returns the maximum points that
// can lie inside the circle of radius 'r' being
// rotated about point 'i'
int getPointsInside(int i, double r, int n)
{
    // This vector stores alpha and beta and flag
    // is marked true for alpha and false for beta
    vector<pair<double, bool>> angles;

    for (int j=0; j<n; j++)
    {
        if (i != j && dis[i][j] <= 2*r)
        {
            // acos returns the arc cosine of the complex
            // used for cosine inverse
            double B = acos(dis[i][j]/(2*r));

            // arg returns the phase angle of the complex
            double A = arg(arr[j]-arr[i]);
            double alpha = A-B;
            double beta = A+B;
            angles.push_back(make_pair(alpha, true));
            angles.push_back(make_pair(beta, false));
        }
    }

    // angles vector is sorted and traversed
    sort(angles.begin(), angles.end());
}
```

```
// count maintains the number of points inside
// the circle at certain value of theta
// res maintains the maximum of all count
int count = 1, res = 1;
vector<pair<double, bool>>::iterator it;
for (it=angles.begin(); it!=angles.end(); ++it)
{
    // entry angle
    if ((*it).second)
        count++;

    // exit angle
    else
        count--;

    if (count > res)
        res = count;
}

return res;
}

// Returns count of maximum points that can lie
// in a circle of radius r.
int maxPoints(Point arr[], int n, int r)
{
    // dis array stores the distance between every
    // pair of points
    for (int i=0; i<n-1; i++)
        for (int j=i+1; j<n; j++)

            // abs gives the magnitude of the complex
            // number and hence the distance between
            // i and j
            dis[i][j] = dis[j][i] = abs(arr[i]-arr[j]);

    // This loop picks a point p
    int ans = 0;
    for (int i=0; i<n; i++)

        // maximum number of points for point arr[i]
        ans = max(ans, getPointsInside(i, r, n));

    return ans;
}

// Driver code
```

```
int main()
{
    Point arr[] = {Point(6.47634, 7.69628),
                   Point(5.16828, 4.79915),
                   Point(6.69533, 6.20378)};
    int r = 1;

    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "The maximum number of points are: "
         << maxPoints(arr, n, r);

    return 0;
}
```

Output :

The maximum number of points are: 2

Time Complexity: There are n points for which we call the function `getPointsInside()`. This function works on ' $n-1$ ' points for which we get $2*(n-1)$ size of the vector 'angles' (one entry angle and one exit angle). Now this 'angles' vector is sorted and traversed which gives complexity of the `getPointsInside()` function equal to $O(n\log n)$. This makes the Angular Sweep Algorithm $O(n^2\log n)$.

Related Resources: Using the complex class available in stl for implementing solutions to geometry problems.

<http://codeforces.com/blog/entry/22175>

Source

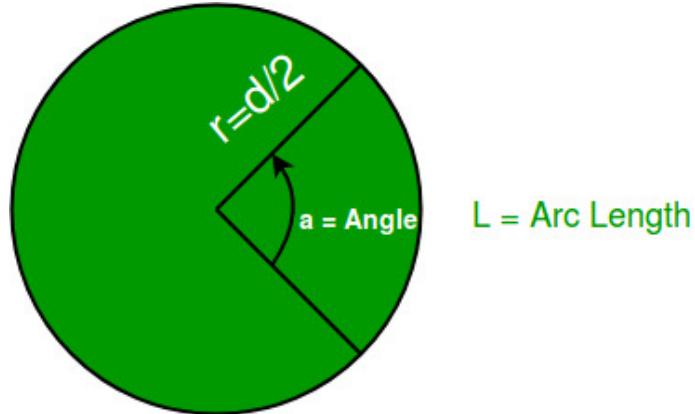
<https://www.geeksforgeeks.org/angular-sweep-maximum-points-can-enclosed-circle-given-radius/>

Chapter 3

Arc length from given Angle

Arc length from given Angle - GeeksforGeeks

An angle is a geometrical figure when two rays meet at a common point on a plane. These rays form the sides of the angle and the meeting point is referred as the vertex of the angle. There is something that we need to keep in mind that the plane that forms an angle doesn't need to be a Euclidean plane. Now, in a circle, the length of an arc is a portion of the circumference. The figure explains the various parts we have discussed:



Given an angle and the diameter of a circle, we can calculate the length of the arc using the formula:

```
ArcLength = ( 2 * pi * radius ) * ( angle / 360 )
Where pi = 22/7,
diameter = 2 * radius,
angle is in degree.
```

Examples :

```
Input :  
Diameter = 25  
Angle = 45  
Explanation : ((22/7) * 25) * (45/360)  
Output : 9.821 (rounded)
```

```
Input :  
Diameter = 80  
Angle = 60  
Explanation : ((22/7) * 80) * (60/360)  
Output : 41.905 (rounded)
```

Note: If angle is greater than or equal to 360 degree, then the arc length cannot be calculated, since no angle is possible.

C++

```
// C++ program to calculate  
// length of an arc  
#include <iostream>  
using namespace std;  
  
// function to calculate  
// arc length  
double arcLength(double diameter,  
                  double angle)  
{  
    double pi = 22.0 / 7.0;  
    double arc;  
  
    if (angle >= 360)  
    {  
        cout<< "Angle cannot",  
            " be formed";  
        return 0;  
    }  
    else  
    {  
        arc = (pi * diameter) *  
              (angle / 360.0);  
        return arc;  
    }  
}  
  
// Driver Code  
int main()  
{
```

```
double diameter = 25.0;
double angle = 45.0;

double arc_len = arcLength(diameter,
                           angle);
cout << (arc_len);

return 0;
}
```

Java

```
// Java program to calculate
// length of an arc
public class Arc {

    // function to calculate arc length
    static double arcLength(double diameter,
                           double angle)
    {
        double pi = 22.0 / 7.0;
        double arc;

        if (angle >= 360) {
            System.out.println("Angle cannot"
                               + " be formed");
            return 0;
        }
        else {
            arc = (pi * diameter) * (angle / 360.0);
            return arc;
        }
    }

    // Driver Code
    public static void main(String args[])
    {
        double diameter = 25.0;
        double angle = 45.0;
        double arc_len = arcLength(diameter, angle);
        System.out.println(arc_len);
    }
}
```

Python3

```
# Python3 code to calculate
```

```
# length of an arc
import math

# function to calculate arc length
def arcLength(diameter, angle):
    if angle >= 360:
        print("Angle cannot be formed")
        return 0
    else:
        arc = (3.142857142857143 * diameter) * (angle / 360.0)
        return arc

# Driver Code
diameter = 25.0
angle = 45.0
arc_len = arcLength(diameter, angle)
print(arc_len)

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# program to calculate length of an arc
using System;

public class GFG {

    // function to calculate arc length
    static double arcLength(double diameter,
                           double angle)
    {
        double pi = 22.0 / 7.0;
        double arc;

        if (angle >= 360) {
            Console.WriteLine("Angle cannot"
                            + " be formed");
            return 0;
        }
        else {
            arc = (pi * diameter) * (angle / 360.0);
            return arc;
        }
    }

    // Driver Code
    public static void Main()
    {
```

```
    double diameter = 25.0;
    double angle = 45.0;

    double arc_len = arcLength(diameter, angle);

    Console.WriteLine(arc_len);
}
}

// This code is contributed by Anant Agarwal.
```

PHP

```
<?php
// PHP program to calculate
// length of an arc

// function to calculate
// arc length
function arcLength($diameter,
                   $angle)
{
    $pi = 22.0 / 7.0;
    $arc;

    if ($angle >= 360)
    {
        echo "Angle cannot",
             " be formed";
        return 0;
    }
    else
    {
        $arc = ($pi * $diameter) *
              ($angle / 360.0);
        return $arc;
    }
}

// Driver Code
$diameter = 25.0;
$angle = 45.0;
$arc_len = arcLength($diameter, $angle);
echo ($arc_len);

// This code is contributed by ajit
?>
```

Output:

9.821428571428571

Improved By : [jit_t](#)

Source

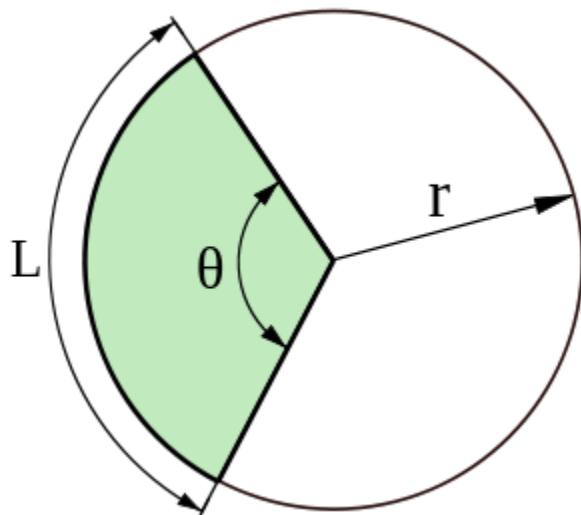
<https://www.geeksforgeeks.org/arc-length-angle/>

Chapter 4

Area of a Circular Sector

Area of a Circular Sector - GeeksforGeeks

A circular sector or circle sector is the portion of a disk enclosed by two radii and an arc, where the smaller area is known as the minor sector and the larger being the major sector. Let's look at this figure and try to figure out the sector:



source: Wikipedia (<https://goo.gl/mWijn2>)

In this figure the green shaded part is a sector, “r” is the Radius and “theta” is the angle as shown. Here, we can say that the shaded portion is the minor sector and the other portion is the major sector. “L” is the Arc of the Sector. For more, visit [Sector](#).

Now let's see the formula using which the sector of a circle can be calculated.

Sector = (pi * r²) * (angle / 360)

Where,

Sector = Area of the Sector,

pi = 22/7,

r = radius,

angle is in degree.

The area of the sector is similar to the calculation of the [area of the circle](#), just multiply the area of the circle with the angle of the sector.

Examples:

Input:

radius = 9

angle = 60

Explanation:

Sector = (pi * 9*9) * (60 / 360)

Output: 42.42857142857142

Input:

radius = 20

angle = 145

Explanation:

Sector = (pi * 20*20) * (145 / 360)

Output: 506.3492063492063

C++

```
// C++ program to find Area of a Sector
#include <iostream>
using namespace std;

void SectorArea(double radius,double angle)
{
    if(angle >= 360)
        cout<<"Angle not possible";

    // Calculating area of the sector
    else
    {
        double sector = ((22 * radius * radius) / 7)
                    * (angle / 360);
```

```
        cout<<sector;
    }
}

// Driver code
int main()
{
    double radius = 9;
    double angle = 60;
    SectorArea(radius, angle);
    return 0;
}

// This code is contributed by Anant Agarwal.
```

Java

```
// Java program to find Area of a Sector

class GFG
{
    static void SectorArea(double radius,double angle)
    {
        if(angle >= 360)
            System.out.println("Angle not possible");

        // Calculating area of the sector
        else
        {
            double sector =((22 * radius * radius) / 7)
                          * (angle / 360);
            System.out.println(sector);
        }
    }

    // Driver code
    public static void main (String[] args)
    {
        double radius = 9;
        double angle = 60;
        SectorArea(radius, angle);
    }
}
// This code is contributed by Anant Agarwal.
```

Python3

```
# Python program to find Area of a Sector
```

```
def SectorArea(radius, angle):
    pi = 22 / 7

    # Constraint or Limit
    if angle >= 360:
        print("Angle not possible")
        return

    # Calculating area of the sector
    else:
        sector = (pi * radius ** 2) * (angle / 360)
        print(sector)
        return

# Driver code
radius = 9
angle = 60
SectorArea(radius, angle)
```

C#

```
// C# program to find Area of a Sector
using System;

class GFG {

    static void SectorArea(double radius, double angle)
    {

        if (angle >= 360)
            Console.WriteLine("Angle not possible");

        // Calculating area of the sector
        else {
            double sector = ((22 * radius * radius) / 7)
                         * (angle / 360);

            Console.WriteLine(sector);
        }
    }

    // Driver code
    public static void Main()
    {
        double radius = 9;
        double angle = 60;
```

```
        SectorArea(radius, angle);
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find Area of a Sector

function SectorArea( $radius, $angle)
{
    if($angle >= 360)
        echo("Angle not possible");

    // Calculating area of the sector
    else
    {
        $sector = ((22 * $radius * $radius)
                   / 7) * ($angle / 360);
        echo($sector);
    }
}

// Driver code

$radius = 9;
$angle = 60;
SectorArea($radius, $angle);

// This code is contributed by vt_m.
?>
```

Output:

42.42857142857142

Reference: Wikipedia ([Circular Sector](#))

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/area-of-a-sector/>

Chapter 5

Area of a Circumscribed Circle of a Square

Area of a Circumscribed Circle of a Square - GeeksforGeeks

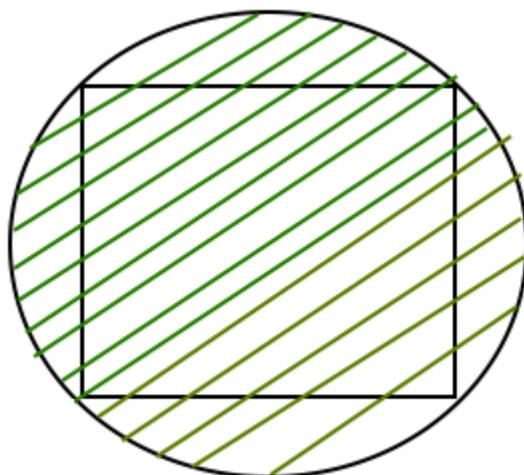
Given the side of a square then find the area of a Circumscribed circle around it.

Examples:

```
Input : a = 6
Output : Area of a circumscribed circle is : 56.55
```

```
Input : a = 4
Output : Area of a circumscribed circle is : 25.13
```

All four sides of a square are of equal length and all four angles are 90 degree. The circle is circumscribed on a given square shown by a shaded region in the below diagram.



Properties of Circumscribed circle are as follows:

- The center of the circumcircle is the point where the two diagonals of a square meet.
- Circumscribed circle of a square is made through the four vertices of a square.
- The radius of a circumcircle of a square is equal to the radius of a square.

Formula used to calculate the area of inscribed circle is:

$$(\text{PI} * a * a)/2$$

where, a is the side of a square in which a circle is circumscribed.

How does this formula work?

We know **area of circle** = $\text{PI} * r * r$.

We also know radius of circle = (square diagonal)/2

Length of diagonal = $\sqrt{2} * a$

Radius = $\sqrt{2} * a / 2 = \sqrt{a^2 / 2}$

Area = $\text{PI} * r * r = (\text{PI} * a * a) / 2$

C++

```
// C++ Program to find the
// area of a circumscribed circle
#include <stdio.h>
#define PI 3.14159265

float areacircumscribed(float a)
{
    return (a * a * (PI / 2));
}

// Driver code
int main()
{
    float a = 6;
    printf(" Area of an circumscribed circle is : %.2f ",
           areacircumscribed(a));
    return 0;
}
```

Java

```
// Java program to calculate
// area of a circumscribed circle-square
import java.io.*;
class Gfg {
    // Utility Function
    static float areacircumscribed(float a)
    {
```

```
        float PI = 3.14159265f;
        return (a * a * (PI / 2));
    }

    // Driver Function
    public static void main(String arg[])
    {
        float a = 6;
        System.out.print("Area of an circumscribed"
                        + "circle is :");
        System.out.println(areacircumscribed(a));
    }
}

// The code is contributed by Anant Agarwal.
```

Python3

```
# Python3 Program to find the
# area of a circumscribed circle
PI = 3.14159265

def areacircumscribed(a):

    return (a * a * (PI / 2))

# Driver code
a = 6
print(" Area of an circumscribed circle is :",
      round(areacircumscribed(a), 2))

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
// C# Program to find the
// area of a circumscribed circle
using System;

class GFG {

    public static double PI= 3.14159265 ;

    static float areacircumscribed(float a)
    {
        return (a * a * (float)(PI / 2));
    }
}
```

```
// Driver code
public static void Main()
{
    float a = 6;

    Console.WriteLine(" Area of an circumscribed"
                      + " circle is : {0}",
                      Math.Round(areacircumscribed(a), 2));
}

// This code is contributed by
// Smitha Dinesh Semwal
```

PHP

```
<?php
// PHP Program to find the
// area of a circumscribed
// circle

$PI = 3.14159265;

// function returns the area
function areacircumscribed($a)
{
    global $PI;
    return ($a * $a * ($PI / 2));
}

// Driver code
$a = 6;
echo " Area of an circumscribed circle is : ",
     areacircumscribed($a);

// The code is contributed by anuj_67.
?>
```

Output :

Area of an circumscribed circle is : 56.55

Improved By : [Smitha Dinesh Semwal, vt_m](#)

Source

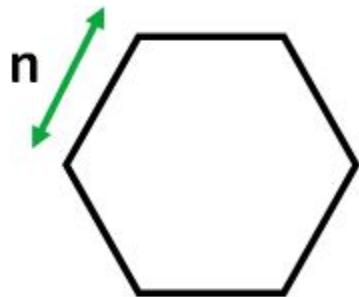
<https://www.geeksforgeeks.org/area-circumscribed-circle-square/>

Chapter 6

Area of a Hexagon

Area of a Hexagon - GeeksforGeeks

A hexagon is a 6-sided, 2-dimensional geometric figure. The total of the internal angles of any hexagon is 720° . A regular hexagon has 6 rotational symmetries and 6 reflection symmetries. All internal angles are 120 degrees.



Examples :

Input: 4

Output: 41.5692

Input: 6

Output: 93.5307

Number of vertices: 6

Number of edges: 6

Internal angle: 120°

$$\text{Area} = (3 \sqrt{3}(\text{side})^2) / 2$$

C++

```
// CPP program to find
// area of a Hexagon
#include <iostream>
#include <math.h>
using namespace std;

// function for calculating
// area of the hexagon.
double hexagonArea(double s)
{
    return ((3 * sqrt(3) *
             (s * s)) / 2);
}

// Driver Code
int main()
{
    // Length of a side
    double s = 4;
    cout << "Area : "
        << hexagonArea(s);
    return 0;
}
```

Java

```
class GFG
{
    // Create a function for calculating
    // the area of the hexagon.
    public static double hexagonArea(double s)
    {
        return ((3 * Math.sqrt(3) *
                 (s * s)) / 2);
    }

    // Driver Code
    public static void main(String[] args)
    {
        // Length of a side
        double s = 4;
        System.out.print("Area: " +
                        hexagonArea(s));
    }
}
```

C#

```
// C# program to find
// area of a Hexagon
using System;

class GFG
{

    // Create a function for calculating
    // the area of the hexagon.
    public static double hexagonArea(double s)
    {
        return ((3 * Math.Sqrt(3) *
                 (s * s)) / 2);
    }

    // Driver Code
    public static void Main()
    {
        // Length of a side
        double s = 4;

        Console.WriteLine("Area: " +
                          hexagonArea(s));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find
// area of a Hexagon

// function for calculating
// area of the hexagon.
function hexagonArea( $s )
{
    return ((3 * sqrt(3) *
             ($s * $s)) / 2);
}

// Driver Code

// Length of a side
```

```
$s = 4;  
echo("Area : ");  
echo(hexagonArea($s));  
  
// This code is contributed by vt_m.  
?>
```

Output :

Area: 41.5692

Improved By : [vt_m, jit_t](#)

Source

<https://www.geeksforgeeks.org/area-of-a-hexagon/>

Chapter 7

Area of a polygon with given n ordered vertices

Area of a polygon with given n ordered vertices - GeeksforGeeks

Given ordered coordinates of a polygon with n vertices. Find area of the polygon. Here ordered mean that the coordinates are given either in clockwise manner or anticlockwise from first vertex to last.

Examples :

Input : X[] = {0, 4, 4, 0}, Y[] = {0, 0, 4, 4};
Output : 16

Input : X[] = {0, 4, 2}, Y[] = {0, 0, 4}
Output : 8

We can compute area of a polygon using [Shoelace formula](#).

Area

$$= | \frac{1}{2} [(x_1y_2 + x_2y_3 + \dots + x_{n-1}y_n + x_ny_1) - (x_2y_1 + x_3y_2 + \dots + x_ny_{n-1} + x_1y_n)] |$$

Below is implementation of above formula.

CPP

```
// C++ program to evaluate area of a polygon using
// shoelace formula
```

```
#include <bits/stdc++.h>
using namespace std;

// (X[i], Y[i]) are coordinates of i'th point.
double polygonArea(double X[], double Y[], int n)
{
    // Initialize area
    double area = 0.0;

    // Calculate value of shoelace formula
    int j = n - 1;
    for (int i = 0; i < n; i++)
    {
        area += (X[j] + X[i]) * (Y[j] - Y[i]);
        j = i; // j is previous vertex to i
    }

    // Return absolute value
    return abs(area / 2.0);
}

// Driver program to test above function
int main()
{
    double X[] = {0, 2, 4};
    double Y[] = {1, 3, 7};

    int n = sizeof(X)/sizeof(X[0]);

    cout << polygonArea(X, Y, n);
}
```

Java

```
// Java program to evaluate area
// of a polygon using shoelace formula
import java.io.*;

class GFG
{
    // (X[i], Y[i]) are coordinates of i'th point.
    public static double polygonArea(double X[], double Y[],
                                    int n)
    {
        // Initialize area
        double area = 0.0;

        // Calculate value of shoelace formula
```

```
int j = n - 1;
for (int i = 0; i < n; i++)
{
    area += (X[j] + X[i]) * (Y[j] - Y[i]);

    // j is previous vertex to i
    j = i;
}

// Return absolute value
return Math.abs(area / 2.0);
}

// Driver program
public static void main (String[] args)
{
    double X[] = {0, 2, 4};
    double Y[] = {1, 3, 7};

    int n = 3;
    System.out.println(polygonArea(X, Y, n));
}

}
// This code is contributed by Sunnysingh
```

Python3

```
# python3 program to evaluate
# area of a polygon using
# shoelace formula

# (X[i], Y[i]) are coordinates of i'th point.
def polygonArea(X, Y, n):

    # Initialize area
    area = 0.0

    # Calculate value of shoelace formula
    j = n - 1
    for i in range(0,n):
        area += (X[j] + X[i]) * (Y[j] - Y[i])
        j = i    # j is previous vertex to i

    # Return absolute value
    return int(abs(area / 2.0))
```

```
# Driver program to test above function
X = [0, 2, 4]
Y = [1, 3, 7]
n = len(X)
print(polygonArea(X, Y, n))

# This code is contributed by
# Smitha Dinesh Semwal

C#

// C# program to evaluate area
// of a polygon using shoelace formula
using System;

class GFG {

    // (X[i], Y[i]) are coordinates of i'th point.
    public static double polygonArea(double[] X,
                                    double[] Y, int n)
    {

        // Initialize area
        double area = 0.0;

        // Calculate value of shoelace formula
        int j = n - 1;

        for (int i = 0; i < n; i++) {
            area += (X[j] + X[i]) * (Y[j] - Y[i]);

            // j is previous vertex to i
            j = i;
        }

        // Return absolute value
        return Math.Abs(area / 2.0);
    }

    // Driver program
    public static void Main()
    {
        double[] X = { 0, 2, 4 };
        double[] Y = { 1, 3, 7 };

        int n = 3;
        Console.WriteLine(polygonArea(X, Y, n));
    }
}
```

```
}
```

```
// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to evaluate area of
// a polygon using shoelace formula

// (X[i] , Y[i]) are
// coordinates of i'th point.
function polygonArea($X, $Y, $n)
{
    // Initialize area
    $area = 0.0;

    // Calculate value of
    // shoelace formula
    $j = $n - 1;
    for ($i = 0; $i < $n; $i++)
    {
        $area += ($X[$j] + $X[$i]) *
            ($Y[$j] - $Y[$i]);

        // j is previous vertex to i
        $j = $i;
    }

    // Return absolute value
    return abs($area / 2.0);
}

// Driver Code
$X = array(0, 2, 4);
$Y = array(1, 3, 7);

$n = sizeof($X);

echo polygonArea($X, $Y, $n);

// This code is contributed by ajit
?>
```

Output :

2

Why is it called Shoelace Formula?

The formula is called so because of the way we evaluate it.

Example :

Let the input vertices be
(0, 1), (2, 3), and (4, 7).

Evaluation procedure matches with process of tying
shoelaces.

We write vertices as below

0	1
2	3
4	7
0	1 [written twice]

we evaluate positive terms as below

0	\	1
2	\	3
4	\	7
0		1

i.e., $0*3 + 2*7 + 4*1 = 18$

we evaluate negative terms as below

0	/	1
2	/	3
4	/	7
0	/	1

i.e., $0*7 + 4*3 + 2*1 = 14$

Area = $1/2 (18 - 14) = 2$

See this for a clearer image.

How does this work?

We can always divide a polygon into triangles. The area formula is derived by taking each edge AB, and calculating the (signed) area of triangle ABO with a vertex at the origin O, by taking the cross-product (which gives the area of a parallelogram) and dividing by 2. As one wraps around the polygon, these triangles with positive and negative area will overlap, and the areas between the origin and the polygon will be cancelled out and sum to 0, while only the area inside the reference triangle remains. [Source : [Wiki](#)]

Related articles :

[Minimum Cost Polygon Triangulation](#)

[Find Simple Closed Path for a given set of points](#)

This article is contributed by Utkarsh Trivedi. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/area-of-a-polygon-with-given-n-ordered-vertices/>

Chapter 8

Area of square Circumscribed by Circle

Area of square Circumscribed by Circle - GeeksforGeeks

Given the radius(r) of circle then find the area of square which is Circumscribed by circle.

Examples:

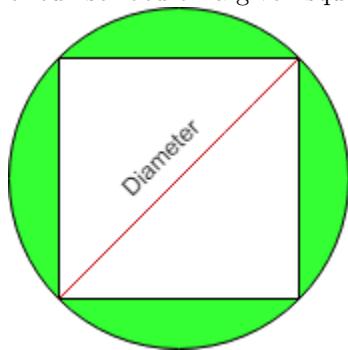
Input : $r = 3$

Output :Area of square = 18

Input : $r = 6$

Output :Area of square = 72

All four sides of a square are of equal length and all four angles are 90 degree. The circle is circumscribed on a given square shown by a shaded region in the below diagram.



Properties of Circumscribed circle are as follows:

- The center of the circumcircle is the point where the two diagonals of a square meet.

- Circumscribed circle of a square is made through the four vertices of a square.
- The radius of a circumcircle of a square is equal to the radius of a square.

Formula used to calculate the area of circumscribed square is:

$$2 * r^2$$

where, r is the radius of the circle in which a square is circumscribed by circle.

How does this formula work?

Assume diagonal of square is d and length of side is a.

We know from the Pythagoras Theorem, the diagonal of a square is $\sqrt{2}$ times the length of a side.

$$\text{i.e } d^2 = a^2 + a^2$$

$$d = 2 * a^2$$

$$d = \sqrt{2} * a$$

Now,

$$a = d / \sqrt{2}$$

and We know diagonal of square that are Circumscribed by Circle is equal to Diameter of circle.

so Area of square = $a * a$

$$= d / \sqrt{2} * d / \sqrt{2}$$

$$= d^2 / 2$$

$$= (2 * r)^2 / 2 \text{ (We know } d = 2 * r \text{)}$$

$$= 2 * r^2$$

CPP

```
// C++ program to find Area of
// square Circumscribed by Circle
#include <iostream>
using namespace std;

// Function to find area of square
int find_Area(int r)
{
    return (2 * r * r);
}

// Driver code
int main()
{
    // Radius of a circle
    int r = 3;

    // Call Function to find
    // an area of square
    cout << " Area of square = "
        << find_Area(r);
```

```
    return 0;
}
```

Java

```
// Java program to find Area of
// square Circumscribed by Circle
class GFG {

    // Function to find area of square
    static int find_Area(int r)
    {
        return (2 * r * r);
    }

    // Driver code
    public static void main(String[] args)
    {
        // Radius of a circle
        int r = 3;

        // Call Function to find
        // an area of square
        System.out.print(" Area of square = "
                        + find_Area(r));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python program to
# find Area of
# square Circumscribed
# by Circle

# Function to find
# area of square
def find_Area(r):

    return (2 * r * r)

# driver code
# Radius of a circle
r = 3
```

```
# Call Function to find
# an area of square
print(" Area of square = ", find_Area(r))

# This code is contributed
# by Anant Agarwal.
```

C#

```
// C# program to find Area of
// square Circumscribed by Circle
using System;

class GFG {

    // Function to find area of square
    static int find_Area(int r)
    {
        return (2 * r * r);
    }

    // Driver code
    public static void Main()
    {
        // Radius of a circle
        int r = 3;

        // Call Function to find
        // an area of square
        Console.WriteLine(" Area of square = "
                        + find_Area(r));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find Area of
// square Circumscribed by Circle

// Function to find area of square
function find_Area( $r )
{
    return (2 * $r * $r);
```

```
}

// Driver code
// Radius of a circle
$r = 3;

// Call Function to find
// an area of square
echo ("Area of square = ");
echo(find_Area($r));

// This code is contributed by vt_m.
?>
```

Output:

Area of square = 18

Time Complexity: O(1)

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/area-square-circumscribed-circle/>

Chapter 9

Bresenham's Algorithm for 3-D Line Drawing

Bresenham's Algorithm for 3-D Line Drawing - GeeksforGeeks

Given two 3-D co-ordinates we need to find the points on the line joining them. All points have integer co-ordinates.

Examples:

```
Input  : (-1, 1, 1), (5, 3, -1)
Output : (-1, 1, 1), (0, 1, 1), (1, 2, 0),
         (2, 2, 0), (3, 2, 0), (4, 3, -1),
         (5, 3, -1)
```

```
Input  : (-7, 0, -3), (2, -5, -1)
Output : (-7, 0, -3), (-6, -1, -3), (-5, -1, -3),
         (-4, -2, -2), (-3, -2, -2), (-2, -3, -2),
         (-1, -3, -2), (0, -4, -1), (1, -4, -1),
         (2, -5, -1)
```

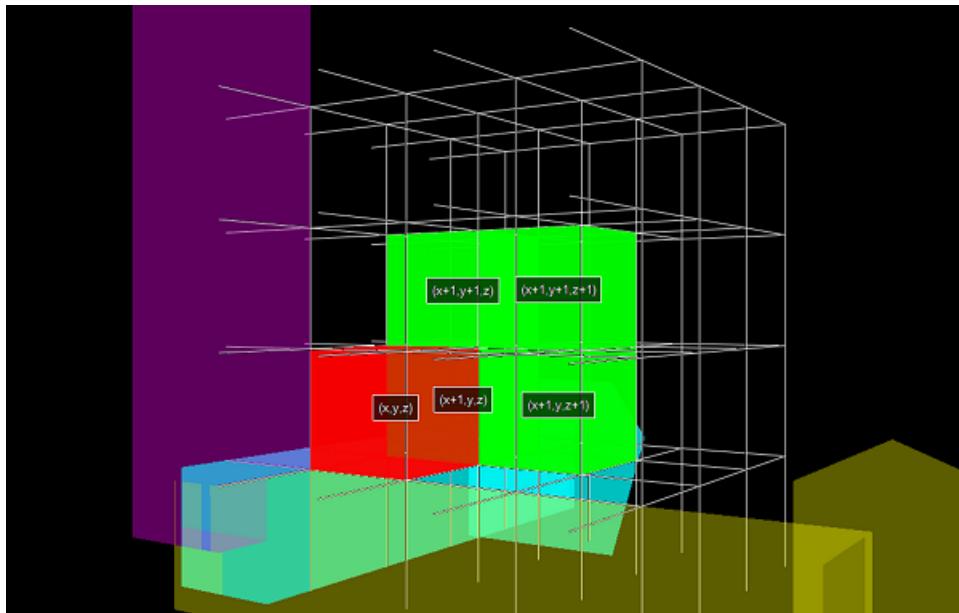
Bresenham's Algorithm is efficient as it avoids floating point arithmetic operations. As in the case of [2-D Line Drawing](#), we use a variable to store the slope-error i.e. the error in slope of the line being plotted from the actual geometric line. As soon as this slope-error exceeds the permissible value we modify the digital to negate the error.

The driving axis of the line to be plotted is the one along which the line travels the farthest i.e. the difference in axes co-ordinates is greatest. Thus the co-ordinate values increase

linearly by 1 along the driving axis and the slope-error variable is used to determine the change in the co-ordinate values of the other axis.

In case of a 2-D line we use one slope-error variable but in case of a 3-D line we need two (Δx , Δy) of them for each of the non-driving axes. If current point is $P(x, y, z)$ and the driving axis is the positive X-axis, then the next point P' could be

- $(x+1, y, z)$
- $(x+1, y+1, z)$
- $(x+1, y, z+1)$
- $(x+1, y+1, z+1)$



The value of slope-error variables are determined according to the following equations:-

$$\text{Error}_x = \frac{\Delta x}{\Delta z} \cdot \text{Error}_z - \frac{\Delta z}{\Delta x} \cdot \text{Error}_x = \frac{\Delta x}{\Delta z} \cdot \text{Error}_z - \frac{\Delta z}{\Delta x} \cdot \text{Error}_x$$

$$\text{Error}_y = \frac{\Delta y}{\Delta z} \cdot \text{Error}_z - \frac{\Delta z}{\Delta y} \cdot \text{Error}_y = \frac{\Delta y}{\Delta z} \cdot \text{Error}_z - \frac{\Delta z}{\Delta y} \cdot \text{Error}_y$$

The initial value of slope-error variables are given by the following equations:-

$$\text{Error}_x = \frac{\Delta x}{\Delta z} \cdot \text{Error}_z - \frac{\Delta z}{\Delta x} \cdot \text{Error}_x$$

$$\text{Error}_y = \frac{\Delta y}{\Delta z} \cdot \text{Error}_z - \frac{\Delta z}{\Delta y} \cdot \text{Error}_y$$

Here Δx , Δy , Δz denote the difference in co-ordinates of the two end points along the X, Y, Z axes.

Algorithm:-

1. Input the two endpoints and store the initial point as (x_0, y_0, z_0)
2. Plot (x_0, y_0, z_0)
3. Calculate constants $\Delta_x, \Delta_y, \Delta_z$ and determine the driving axis by comparing the absolute values of $\Delta_x, \Delta_y, \Delta_z$
 If Δ_x is maximum, then X-axis is the driving axis
 If Δ_y is maximum, then Y-axis is the driving axis
 If Δ_z is maximum, then Z-axis is the driving axis
4. Let's suppose that X-axis is the driving axis, then

$$\frac{\Delta_x}{\Delta_x} = \frac{\Delta_y}{\Delta_x} = \frac{\Delta_z}{\Delta_x}$$

$$x_k = x_0 + k \cdot \Delta_x$$
5. At each x_k along the line, starting at $k = 0$, check the following conditions and determine the next point:
 - If $p_{yk} < 0$ AND $p_{zk} < 0$, then
 plot $(x_k + 1, y_k, z_k)$ and
 set $p_{yk+1} = p_{yk} + 2\Delta_y - 2\Delta_z$ and $p_{zk+1} = p_{zk} + 2\Delta_z - 2\Delta_y$
 - Else If $p_{yk} > 0$ AND $p_{zk} < 0$, then
 plot $(x_k + 1, y_k + 1, z_k)$ and
 set $p_{yk+1} = p_{yk} + 2\Delta_y - 2\Delta_z$ and $p_{zk+1} = p_{zk} + 2\Delta_z - 2\Delta_y$
 - Else If $p_{yk} = 0$, then
 plot $(x_k + 1, y_k, z_k + 1)$ and
 set $p_{yk+1} = p_{yk} + 2\Delta_y + 2\Delta_z$ and $p_{zk+1} = p_{zk} + 2\Delta_z + 2\Delta_y$
 - Else then
 plot $(x_k + 1, y_k + 1, z_k + 1)$ and
 set $p_{yk+1} = p_{yk} + 2\Delta_y + 2\Delta_z$ and $p_{zk+1} = p_{zk} + 2\Delta_z + 2\Delta_y$
6. Repeat step 5 $k = k + 1$ times

Python3

```
# Python3 code for generating points on a 3-D line
# using Bresenham's Algorithm
```

```

def Bresenham3D(x1, y1, z1, x2, y2, z2):
    ListOfPoints = []
    ListOfPoints.append((x1, y1, z1))
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)
    dz = abs(z2 - z1)
    if (x2 > x1):
        xs = 1
    else:
        xs = -1
    if (y2 > y1):
        ys = 1
    else:
        ys = -1
    if (z2 > z1):
        zs = 1
    else:
        zs = -1

    # Driving axis is X-axis"
    if (dx >= dy and dx >= dz):
        p1 = 2 * dy - dx
        p2 = 2 * dz - dx
        while (x1 != x2):
            x1 += xs
            if (p1 >= 0):
                y1 += ys
                p1 -= 2 * dx
            if (p2 >= 0):
                z1 += zs
                p2 -= 2 * dx
            p1 += 2 * dy
            p2 += 2 * dz
            ListOfPoints.append((x1, y1, z1))

    # Driving axis is Y-axis"
    elif (dy >= dx and dy >= dz):
        p1 = 2 * dx - dy
        p2 = 2 * dz - dy
        while (y1 != y2):
            y1 += ys
            if (p1 >= 0):
                x1 += xs
                p1 -= 2 * dy
            if (p2 >= 0):
                z1 += zs
                p2 -= 2 * dy
            p1 += 2 * dx

```

```
p2 += 2 * dz
ListOfPoints.append((x1, y1, z1))

# Driving axis is Z-axis"
else:
    p1 = 2 * dy - dz
    p2 = 2 * dx - dz
    while (z1 != z2):
        z1 += zs
        if (p1 >= 0):
            y1 += ys
            p1 -= 2 * dz
        if (p2 >= 0):
            x1 += xs
            p2 -= 2 * dz
        p1 += 2 * dy
        p2 += 2 * dx
        ListOfPoints.append((x1, y1, z1))
    return ListOfPoints

def main():
    (x1, y1, z1) = (-1, 1, 1)
    (x2, y2, z2) = (5, 3, -1)
    ListOfPoints = Bresenham3D(x1, y1, z1, x2, y2, z2)
    print(ListOfPoints)

main()
```

Output:

```
[(-1, 1, 1), (0, 1, 1), (1, 2, 0), (2, 2, 0), (3, 2, 0), (4, 3, -1), (5, 3, -1)]
```

Source

<https://www.geeksforgeeks.org/bresenham-s-algorithm-for-3-d-line-drawing/>

Chapter 10

Calculate Volume and Surface area Of Sphere

Calculate Volume and Surface area Of Sphere - GeeksforGeeks

Given radius of sphere, calculate the volume and surface area of sphere.

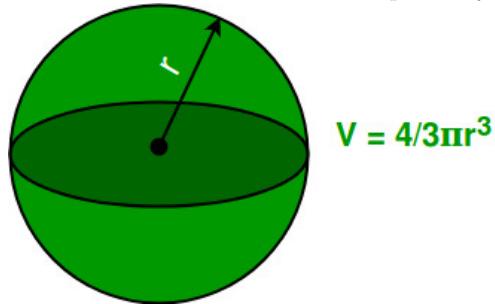
Sphere:

Just like a circle, which geometrically is a two-dimensional object, a sphere is defined mathematically as the set of points that are all at the same distance r from a given point, but in three-dimensional space. This distance r is the radius of the sphere, and the given point is the center of the sphere.

For a given surface area, the sphere is the one solid that has the greatest volume. This why it appears in nature so much, such as water drops, bubbles and planets etc.

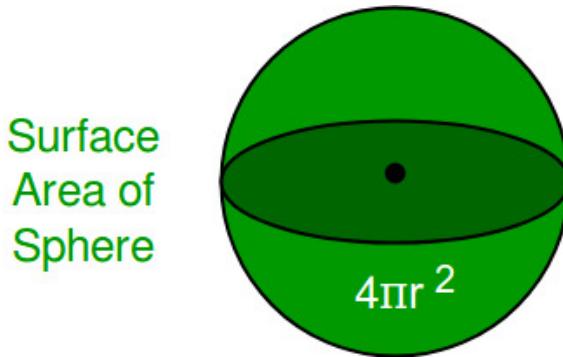
Volume Of Sphere:

The number of cubic units that will exactly fill a sphere or the storage capacity of sphere. We can calculate the volume of sphere by using formula:



Surface Area Of Sphere:

The surface area of a sphere object is a measure of the total area that the surface of the sphere occupies. We can calculate the volume of sphere by using formula:



Examples :

```
Input : Radius Of Sphere = 5
Output : Volume Of Sphere : 523.5987755982989
          Surface Area Of Sphere : 314.1592653589793
```

Explanation:

$$\text{Volume} = \left(\frac{4}{3}\right) * 3.14159 * 5 * 5 * 5 = 523.598$$
$$\text{Surface Area} = 4 * 3.14159 * 5 * 5 = 314.159$$

```
Input : Radius Of Sphere = 12
Output : Volume Of Sphere : 7238.229473870883
          Surface Area Of Sphere : 1809.5573684677208
```

C++

```
// CPP program to calculate Volume and
// Surface area of Sphere
#include<bits/stdc++.h>
using namespace std;

// Initializing Value Of PI
float pi = 3.14159;

// Function To Calculate Volume Of Sphere
float volume(float r)
{
    float vol;
    vol = (float(4) / float(3)) * pi * r * r * r;
    return vol;
}

// Function To Calculate Surface Area of Sphere
float surface_area(float r)
```

```
{  
    float sur_ar;  
    sur_ar = 4 * pi * r * r;  
    return sur_ar;  
}  
  
// Driver Function  
int main()  
{  
    float radius = 12;  
    float vol, sur_area;  
  
    // Function Call  
    vol = volume(radius);  
    sur_area = surface_area(radius);  
  
    // Printing Value Of Volume And Surface Area  
    cout << "Volume Of Sphere :" << vol << endl;  
    cout << "Surface Area Of Sphere :" << sur_area << endl;  
    return 0;  
}
```

Java

```
// Java program to calculate Volume and  
// Surface area of Sphere  
class GFG {  
  
    // Initializing Value Of PI  
    static float pi = 3.14159f;  
  
    // Function To Calculate Volume Of Sphere  
    static float volume(float r)  
{  
        float vol;  
        vol = ((float)4 / (float)3) * (pi * r * r * r);  
        return vol;  
    }  
  
    // Function To Calculate Surface Area of Sphere  
    static float surface_area(float r) {  
        float sur_ar;  
        sur_ar = 4 * pi * r * r;  
        return sur_ar;  
    }  
  
    // Driver Function  
    public static void main(String[] args)
```

```
{  
    float radius = 12;  
    float vol, sur_area;  
  
    // Function Call  
    vol = volume(radius);  
    sur_area = surface_area(radius);  
  
    // Printing Value Of Volume And Surface Area  
    System.out.println("Volume Of Sphere :" + vol);  
    System.out.println("Surface Area Of Sphere :" + sur_area);  
}  
}  
  
// This code is contributed by Anant Agarwal.
```

Python3

```
''' Python3 program to calculate Volume and  
Surface area of Sphere'''  
# Importing Math library for value Of PI  
import math  
pi = math.pi  
  
# Function to calculate Volume of Sphere  
def volume(r):  
    vol = (4 / 3) * pi * r * r * r  
    return vol  
  
# Function To Calculate Surface Area of Sphere  
def surfacearea(r):  
    sur_ar = 4 * pi * r * r  
    return sur_ar  
  
# Driver Code  
radius = float(12)  
print( "Volume Of Sphere : ", volume(radius) )  
print( "Surface Area Of Sphere : ", surfacearea(radius) )
```

C#

```
// C# program to calculate Volume and  
// Surface area of Sphere  
using System;  
  
class GFG {
```

```
// Initializing Value Of PI
static float pi = 3.14159f;

// Function To Calculate Volume
// Of Sphere
static float volume(float r)
{
    float vol;
    vol = ((float)4 / (float)3) *
          (pi * r * r * r);
    return vol;
}

// Function To Calculate Surface Area
// of Sphere
static float surface_area(float r) {
    float sur_ar;
    sur_ar = 4 * pi * r * r;
    return sur_ar;
}

// Driver Function
public static void Main()
{
    float radius = 12;
    float vol, sur_area;

    // Function Call
    vol = volume(radius);
    sur_area = surface_area(radius);

    // Printing Value Of Volume And
    // Surface Area
    Console.WriteLine("Volume Of Sphere :"
                      + vol);
    Console.WriteLine("Surface Area Of "
                      + "Sphere :" + sur_area);
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// CPP program to calculate Volume
// and Surface area of Sphere
```

```
// Function To Calculate
// Volume Of Sphere
function volume( $r)
{
    $pi = 3.14159;
    $vol = (4 / 3) * $pi * $r * $r * $r;
    return $vol;
}

// Function To Calculate
// Surface Area of Sphere
function surface_area( $r)
{
    $pi = 3.14159;

    $sur_ar = 4 * $pi * $r * $r;
    return $sur_ar;
}

// Driver Code
$radius = 12;
$vol; $sur_area;

// Function Call
$vol = volume($radius);
$sur_area = surface_area($radius);

// Printing Value Of
// Volume And Surface Area
echo ("Volume Of Sphere : " );
echo($vol);
echo( " \nSurface Area Of Sphere :");
echo( $sur_area);

// This code is contributed by vt_m.
?>
```

Output :

```
Volume Of Sphere :7238.22
Surface Area Of Sphere :1809.56
```

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/calculate-volume-surface-area-sphere/>

Chapter 11

Calculate Volume of Dodecahedron

Calculate Volume of Dodecahedron - GeeksforGeeks

Given the edge of the dodecahedron calculate its Volume. Volume is the amount of the space which the shapes takes up.

A **dodecahedron** is a 3-dimensional figure made up of 12 faces, or flat sides. All of the faces are pentagons of the same size. The word ‘dodecahedron’ comes from the Greek words dodeca (‘twelve’) and hedron (‘faces’).

Formula:

$$\text{Volume} = (15 + 7\sqrt{5}) \cdot e^3 / 4$$

Where e is length of an edge.

Examples :

Input : side = 4
Output : 490.44

Input : side = 9
Output : 5586.41

C++

```
// CPP program to calculate
// Volume of dodecahedron
#include <bits/stdc++.h>
using namespace std;

// utility Function
double vol_of_dodecahedron(int side)
```

```
{  
    return (((15 + (7 * (sqrt(5)))) / 4)  
           * (pow(side, 3))) ;  
}  
// Driver Function  
int main()  
{  
    int side = 4;  
  
    cout << "Volume of dodecahedron = "  
         << vol_of_dodecahedron(side);  
}
```

Java

```
// Java program to calculate  
// Volume of dodecahedron  
  
import java.io.*;  
  
class GFG  
{  
    // driver function  
    public static void main (String[] args)  
    {  
        int side = 4;  
        System.out.print("Volume of dodecahedron = ");  
        System.out.println(vol_of_dodecahedron(side));  
    }  
  
    static double vol_of_dodecahedron(int side)  
    {  
        return (((15 + (7 * (Math.sqrt(5)))) / 4)  
               * (Math.pow(side, 3)));  
    }  
}  
  
// This code is contributed  
// by Azkia Anam.
```

Python3

```
# Python3 program to calculate  
# Volume of dodecahedron  
import math  
  
# utility Function
```

```
def vol_of_dodecahedron(side) :  
  
    return (((15 + (7 * (math.sqrt(5)))) / 4)  
           * (math.pow(side, 3)))  
  
# Driver Function  
side = 4  
print("Volume of dodecahedron =",  
      round(vol_of_dodecahedron(side), 2))  
  
# This code is contributed by Smitha Dinesh Semwal
```

C#

```
// C# program to calculate  
// Volume of dodecahedron  
using System;  
  
public class GFG  
{  
  
    // utility Function  
    static float vol_of_dodecahedron(int side)  
    {  
        return (float)((15 + (7 * (Math.Sqrt(5)))) / 4)  
               * (Math.Pow(side, 3));  
    }  
  
    // Driver Function  
    static public void Main ()  
    {  
        int side = 4;  
  
        Console.WriteLine("Volume of dodecahedron = "  
                         + vol_of_dodecahedron(side));  
    }  
}  
  
/* This code is contributed by vt_m.*/
```

PHP

```
<?php  
// PHP program to calculate  
// Volume of dodecahedron
```

```
// utility Function
function vol_of_dodecahedron($side)
{
    return (((15 + (7 * (sqrt(5)))) / 4)
            * (pow($side, 3))) ;
}

// Driver Function
$side = 4;
echo ("Volume of dodecahedron = ");
echo(vol_of_dodecahedron($side));

// This code is contributed by vt_m.
?>
```

Output :

Volume of dodecahedron = 490.44

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/calculate-volume-dodecahedron/>

Chapter 12

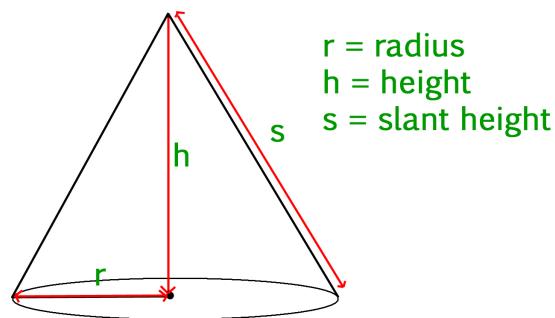
Calculate volume and surface area of a cone

Calculate volume and surface area of a cone - GeeksforGeeks

Given slant height, height and radius of a cone, we have to calculate the volume and surface area of the cone.

Cone :

Cone is a three dimensional geometric shape. It consists of a base having the shape of a circle and a curved side (the lateral surface) ending up in a tip called the apex or vertex.



Volume of a cone :

The volume of a cone is given by the formula –

$$\text{volume} = 1/3(\pi * r * r * h)$$

where r is the radius of the circular base, and h is the height (the perpendicular distance from the base to the vertex).

Surface area of a cone :

The surface area of a cone is given by the formula –

```
area = pi * r * s + pi * r^2
```

Where r is the radius of the circular base, and s is the slant height of the cone.

Examples :

Input :

```
radius = 5
slant_height = 13
height = 12
Output :
Volume Of Cone = 314.159
Surface Area Of Cone = 282.743
```

Input :

```
radius = 6
slant_height = 10
height = 8
Output :
Volume Of Cone = 301.593
Surface Area Of Cone = 301.593
```

C++

```
// CPP program to calculate Volume
// and Surface area of Cone
#include<iostream>
using namespace std;

float pi = 3.14159;

// Function to calculate
// Volume of cone
float volume(float r, float h)
{
    return (float(1) / float(3)) * pi *
           r * r * h;
}

// Function to calculate
// Surface area of cone
float surface_area(float r, float s)
{
    return pi * r * s + pi * r * r;
}

// Driver Code
```

```
int main()
{
    float radius = 5;
    float slant_height = 13;
    float height = 12;
    float vol, sur_area;

    // Printing value of volume
    // and surface area
    cout << "Volume Of Cone : "
        << volume(radius, height) << endl;
    cout << "Surface Area Of Cone : "
        << surface_area(radius, slant_height);
    return 0;
}
```

Java

```
// Java program to calculate
// Volume and Surface area of cone
class GFG
{
    static float pi = 3.14159f;

    // Function to calculate
    // Volume of cone
    public static float volume(float r,
                               float h)
    {
        return (float)1 / 3 * pi * h *
               r * r;
    }

    // Function to calculate
    // Surface area of cone
    public static float surface_area(float r,
                                     float s)
    {
        return pi * r * s + pi * r * r;
    }

    // Driver Code
    public static void main(String args[])
    {
        float radius = 5;
        float slant_height = 13;
        float height = 12;
        float vol, sur_area;
```

```
// Printing value of volume
// and surface area
System.out.print("Volume Of Cone : ");
System.out.println(volume(radius, height));

System.out.print("Surface Area Of Cone : ");
System.out.println(surface_area(radius,
                                slant_height));

}

}

// This code is contributed by "akanshgupta"
```

Python

```
''' Python3 program to calculate Volume and
Surface area of Cone'''

# Importing Math library for value Of PI
import math
pi = math.pi

# Function to calculate Volume of Cone
def volume(r, h):
    return (1 / 3) * pi * r * r * h

# Function To Calculate Surface Area of Cone
def surfacearea(r, s):
    return pi * r * s + pi * r * r

# Driver Code
radius = float(5)
height = float(12)
slat_height = float(13)
print( "Volume Of Cone : ", volume(radius, height) )
print( "Surface Area Of Cone : ", surfacearea(radius, slat_height) )
```

C#

```
// C# program to calculate
// Volume and Surface area of cone
using System;

class GFG
{
```

```
static float pi = 3.14159f;

// Function to calculate
// Volume of cone
public static float volume(float r,
                           float h)
{
    return (float)1 / 3 * pi * h *
           r * r;
}

// Function to calculate
// Surface area of cone
public static float surface_area(float r,
                                  float s)
{
    return pi * r * s + pi * r * r;
}

// Driver Code
public static void Main()
{
    float radius = 5;
    float slant_height = 13;
    float height = 12;
    //float vol, sur_area;

    // Printing value of volume
    // and surface area
    Console.Write("Volume Of Cone : ");
    Console.WriteLine(volume(radius,
                            height));

    Console.Write("Surface Area Of Cone : ");
    Console.WriteLine(surface_area(radius,
                                 slant_height));
}

}

// This code is contributed by "vt_m"
```

PHP

```
<?php
// PHP program to calculate Volume
// and Surface area of Cone
```

```
// Function to calculate Volume of cone
function volume( $r, $h)
{
    $pi = 3.14159;
    return (1 / 3) * $pi * $r *
           $r * $h;
}

// Function to calculate
// Surface area of cone
function surface_area($r, $s)
{
    $pi = 3.14159;
    return $pi * $r * $s + $pi *
           $r * $r;
}

// Driver Code

$radius = 5;
$slant_height = 13;
$height = 12;
//vol, sur_area;

// Printing value of volume
// and surface area
echo("Volume Of Cone : ");
echo( volume($radius, $height));
echo("\n");
echo("Surface Area Of Cone : ");
echo( surface_area($radius,
                  $slant_height));

// This code is contributed by vt_m.

?>
```

Output :

```
Volume Of Cone : 314.159
Surface Area Of Cone : 282.743
```

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/calculate-volume-surface-area-cone/>

Chapter 13

Centered Dodecagonal Number

Centered Dodecagonal Number - GeeksforGeeks

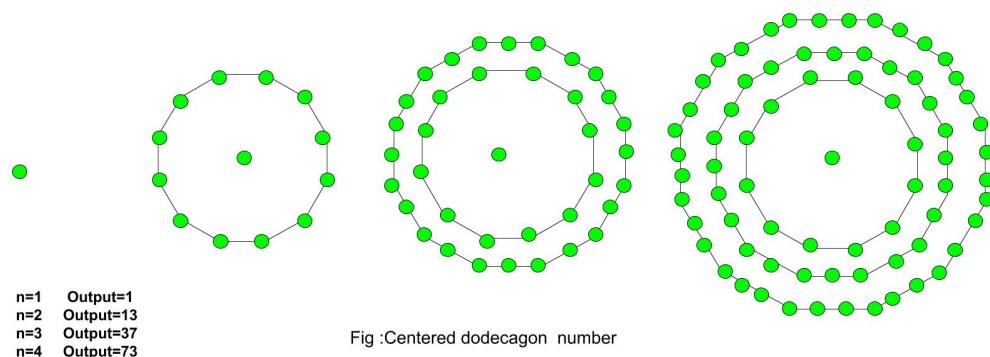
Given a number n, find the **nth** Centered Dodecagonal Number.

The **Centered Dodecagonal Number** represents a dot in the center and other dots surrounding it in successive dodecagonal(12 sided polygon) layers.

Examples :

Input : 3
Output : 37

Input : 7
Output : 253



The first few centered dodecagonal numbers are:

1, 13, 37, 73, 121, 181, 253, 337, 433, 541, 661.....

The formula for the nth Centered dodecagonal number:

C++

```
// C++ Program to find
// nth centered
// Dodecagonal number
#include <bits/stdc++.h>
using namespace std;

// Function to calculate Centered
// Dodecagonal number
int centeredDodecagonal(long int n)
{
    // Formula to calculate nth
    // centered Dodecagonal number
    return 6 * n * (n - 1) + 1;
}

// Drivers Code
int main()
{
    long int n = 2;
    cout << centeredDodecagonal(n);
    cout << endl;
    n = 9;
    cout << centeredDodecagonal(n);

    return 0;
}
```

Java

```
// Java Program to find
// nth centered
// Dodecagonal number
import java.io.*;

class GFG
{
// Function to calculate
// Centered Dodecagonal number
static long centeredDodecagonal(long n)
{
    // Formula to calculate nth
    // centered Dodecagonal number
```

```
        return 6 * n * (n - 1) + 1;
    }

// Driver Code
public static void main (String[] args)
{
    long n = 2;
    System.out.println(centeredDodecagonal(n));

    n = 9;
    System.out.println(centeredDodecagonal(n));
}
}

// This code is contributed by anuj_67.
```

Output :

```
13
433
```

References

<http://oeis.org/A003154>

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/centered-dodecagonal-number/>

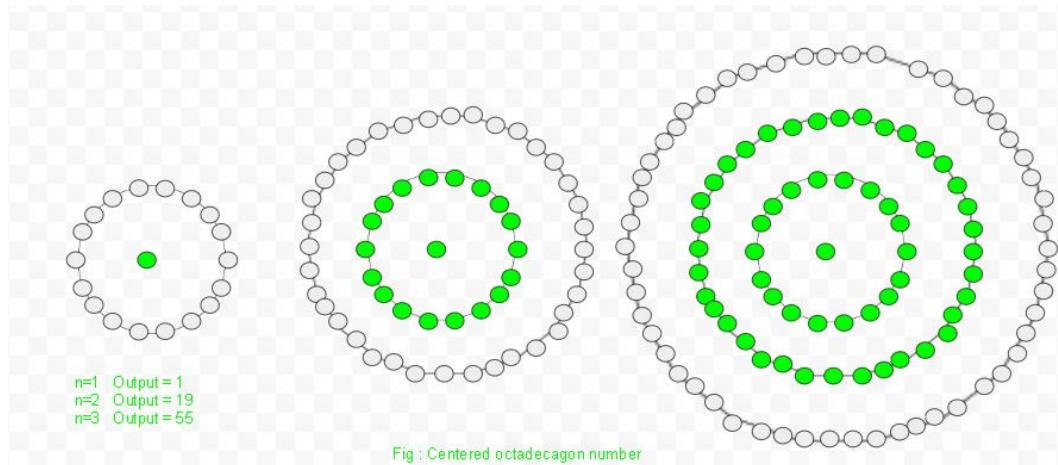
Chapter 14

Centered Octadecagonal Number

Centered Octadecagonal Number - GeeksforGeeks

Given a number n, find the **nth** Centered Octadecagonal number.

The **Centered Octadecagonal Number** represents a dot in the centre and others dot are arranged around it in successive layers of octadecagon(18 sided polygon).



Examples :

Input : 2

Output : 19

Input : 6

Output : 271

In mathematics, Centered Octadecagonal number for **n-th** term is given by:

Below is the basic implementation of the above idea:

C++

```
// C++ Program to find the
// nth centered octadecagonal
// number
#include <bits/stdc++.h>
using namespace std;

// centered octadecagon function
int center_octadecagon_num(long int n)
{
    // Formula to calculate nth
    // centered octadecagonal number
    return 9 * n * n - 9 * n + 1;
}

// Driver Code
int main()
{
    long int n = 3;
    cout << n << "th centered octadecagonal number : "
        << center_octadecagon_num(n);
    cout << endl;
    n = 13;
    cout << n << "th centered octadecagonal number : "
        << center_octadecagon_num(n);

    return 0;
}
```

Java

```
// Java Program to find the
// nth centered octadecagonal
// number
import java.io.*;

class GFG
{
```

```
// centered octadecagon function
static int center_octadecagon_num(int n)
{
    // Formula to calculate nth
    // centered octadecagonal number
    return 9 * n * n - 9 * n + 1;
}

// Driver Code
public static void main (String[] args)
{
    int n = 3;
    System.out.print(n + "th centered " +
                      "octadecagonal number : ");
    System.out.println(center_octadecagon_num(n));

    n = 13;
    System.out.print(n + "th centered " +
                      "octadecagonal number : ");
    System.out.println(center_octadecagon_num(n));
}

// This code is contributed by ajit
```

Python3

```
# Program to find nth
# centered octadecagonal number

# Centered octadecagonal
# number function
def center_octadecagon_num(n) :

    # Formula to calculate
    # nth centered octadecagonal
    # number & return it
    # into main function.
    return(9 * n * n -
          9 * n + 1)

# Driver Code
if __name__ == '__main__' :

    n = 3
    print(n,"rd centered octadecagonal " +
                      "number : ",
```

```
center_octadecagon_num(n)

n = 13
print(n,"th centered octadecagonal " +
      "number : ",
      center_octadecagon_num(n))

# This code is contributed
# by akt_mit
```

C#

```
// C# Program to find the
// nth centered octadecagonal
// number
using System;

class GFG
{

    // centered octadecagon function
    static int center_octadecagon_num(int n)
    {

        // Formula to calculate nth
        // centered octadecagonal number
        return 9 * n * n - 9 * n + 1;
    }

    // Driver Code
    static public void Main ()
    {

        int n = 3;
        Console.Write( n + "th centered " +
                      "octadecagonal number : ");
        Console.WriteLine( center_octadecagon_num(n));

        n = 13;
        Console.Write( n + "th centered " +
                      "octadecagonal number : ");
        Console.WriteLine(center_octadecagon_num(n));
    }
}

// This code is contributed by aj_36.
```

PHP

```
<?php
// PHP Program to find the
// nth centered octadecagonal
// number

// centered octadecagon function
function center_octadecagon_num($n)
{
    // Formula to calculate nth
    // centered octadecagonal number
    return (9 * $n * $n -
            9 * $n + 1);
}

// Driver Code
$n = 3;
echo $n , "th centered octadecagonal " .
           "number : ",
       center_octadecagon_num($n);
echo "\n";

$n = 13;
echo $n , "th centered octadecagonal " .
           "number : ",
       center_octadecagon_num($n);

// This code is contributed by m_kit
?>
```

Output :

```
3th centered octadecagonal number : 55
13th centered octadecagonal number : 1405
```

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/centered-octadecagonal-number/>

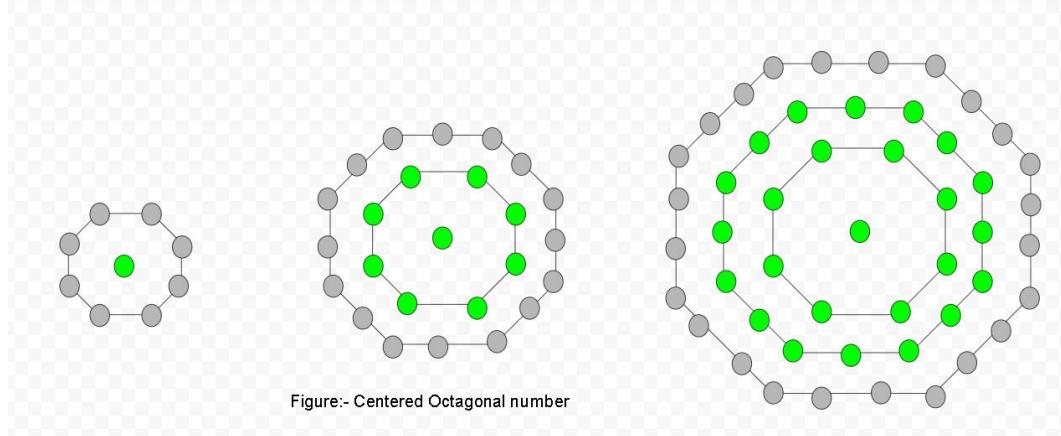
Chapter 15

Centered Octagonal Number

Centered Octagonal Number - GeeksforGeeks

Given a number n , find the n th centered octagonal number.

A **centered octagonal number** represents an octagon with a dot in the centre and others dots surrounding the centre dot in the successive octagonal layer.



Examples :

Input : 2
Output : 9

Input : 5
Output : 81

Centered Octagonal **n-th** Number is given by :

Basic Implementation of the above approach:

C++

```
// Program to find nth
// centered octagonal
// number
#include <bits/stdc++.h>
using namespace std;

// Centered octagonal number function
int cen_octagonalnum(long int n)
{
    // Formula to calculate nth
    // centered octagonal number &
    // return it into main function.
    return (4 * n * n - 4 * n + 1);
}

// Driver Code
int main()
{
    long int n = 6;
    cout << n << "th centered"
        << " octagonal number : ";
    cout << cen_octagonalnum(n);
    cout << endl;
    n = 11;
    cout << n << "th centered"
        << " octagonal number : ";
    cout << cen_octagonalnum(n);

    return 0;
}
```

Java

```
// Java Program to find nth
// centered octagonal number
import java.io.*;

class GFG
{
```

```
// Function to find centered
// octagonal number
static int centeredoctagonalNumber(int n)
{
    // Formula to calculate nth
    // centered octagonal number
    // and return it into main function
    return 4 * n * (n - 1) + 1;
}

// Driver Code
public static void main(String args[])
{
    int n = 6;
    System.out.print(n + "th centered " +
                      "octagonal number: ");
    System.out.println(
        centeredoctagonalNumber(n));

    n = 11;
    System.out.print(n + "th centered " +
                      "octagonal number: ");
    System.out.println(
        centeredoctagonalNumber(n));
}

}

// This code has been contributed by Prasad_Kshirsagar.
```

Python3

```
# Program to find nth
# centered octagonal number

def cen_octagonalnum(n) :

    # Formula to calculate nth
    # centered octagonal number
    return (4 * n * n -
           4 * n + 1)

# Driver Code
if __name__ == '__main__' :

    n = 6
    print(n,"th Centered",
          "octagonal number: ",
```

```
        cen_octagonalnum(n))
n = 11
print(n,"th Centered" ,
      "octagonal number: " ,
      cen_octagonalnum(n))

# This code is contributed
# by akt_mit

C#

// Program to find nth centered octagonal
// number
using System;

public class GFG{

    // Centered octagonal number function
    static long cen_octagonalnum(long n)
    {

        // Formula to calculate nth
        // centered octagonal number &
        // return it into main function.
        return (4 * n * n - 4 * n + 1);
    }

    // Driver code
    static public void Main ()
    {
        long n = 6;
        Console.WriteLine(n + "th centered"
                          + " octagonal number : "
                          + cen_octagonalnum(n));

        n = 11;
        Console.WriteLine(n + "th centered"
                          + " octagonal number : "
                          + cen_octagonalnum(n));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
```

```
// Program to find nth
// centered octagonal
// number

// Centered octagonal
// number function
function cen_octagonalnum($n)
{
    // Formula to calculate nth
    // centered octagonal number &
    // return it into main function.
    return (4 * $n * $n -
        4 * $n + 1);
}

// Driver Code
$n = 6;
echo $n , "th centered",
      " octagonal number : ";
echo cen_octagonalnum($n);
echo "\n";

$n = 11;
echo $n , "th centered",
      " octagonal number : ";
echo cen_octagonalnum($n);

// This code is contributed by ajit
?>
```

Output

```
6th centered octagonal number : 121
11th centered octagonal number : 441
```

Improved By : [vt_m, jit_t](#)

Source

<https://www.geeksforgeeks.org/centered-octagonal-number/>

Chapter 16

Centered Pentadecagonal Number

Centered Pentadecagonal Number - GeeksforGeeks

Given a number n, find the **nth** Centered Pentadecagonal Number .

A **Centered Pentadecagonal Number** represents a dot in the centre and other dots surrounding it in successive pentadecagonal(15-sided polygon) layers.

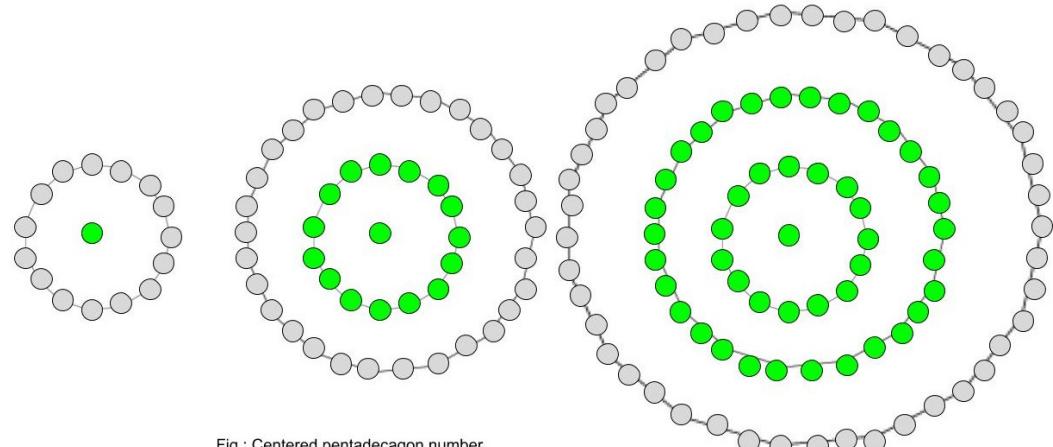


Fig : Centered pentadecagon number

Examples :

Input : 2

Output : 16

Input : 8

Output : 421

n-th term of Centered pentadecagonal number:-

Below is the basic implementation of the above idea.

C++

```
// C++ Program to find
// nth centered
// pentadecagonal number
#include <bits/stdc++.h>
using namespace std;

// centered pentadecagonal function
int center_pentadecagonal_num(long int n)
{
    // Formula to calculate nth
    // centered pentadecagonal number
    return (15 * n * n - 15 * n + 2) / 2;
}

// Driver Code
int main()
{
    long int n = 3;
    cout << n << "th number : "
        << center_pentadecagonal_num(n);
    cout << endl;
    n = 10;
    cout << n << "th number : "
        << center_pentadecagonal_num(n);

    return 0;
}
```

Java

```
// Java Program to find nth centered
// pentadecagonal number

import java.io.*;

class GFG {

    // centered pentadecagonal function
```

```
static long center_pentadecagonal_num(long n)
{
    // Formula to calculate nth
    // centered pentadecagonal number
    return (15 * n * n - 15 * n + 2) / 2;
}

// Driver Code
public static void main (String[] args)
{
    long n = 3;
    System.out.print(n + "th number : ");
    System.out.println(
        center_pentadecagonal_num(n));

    n = 10;
    System.out.print( n + "th number : ");
    System.out.println(
        center_pentadecagonal_num(n));
}
}

// This code is contributed by ajit.
```

Python3

```
# Program to find nth
#centered pentadecagonal number

def center_pentadecagonal_num(n) :

    # Formula to calculate nth
    # centered pentadecagonal number
    return (15 * n * n - 15 * n + 2) // 2

# Driver Code
if __name__ == '__main__' :

    n = 3
    print(n,"rd number : ",
          center_pentadecagonal_num(n))
    n = 10
    print(n,"th number : ",
          center_pentadecagonal_num(n))
```

```
# This code is contributed by m_kit

C#

// C# Program to find
// nth centered
// pentadecagonal number
using System;

class GFG
{

    // centered
    // pentadecagonal function
    static long center_pentadecagonal_num(long n)
    {

        // Formula to calculate
        // nth centered
        // pentadecagonal number
        return (15 * n * n -
               15 * n + 2) / 2;
    }

    // Driver Code
    static public void Main ()
    {
        long n = 3;
        Console.Write(n + "th number : ");
        Console.WriteLine(
            center_pentadecagonal_num(n));

        n = 10;
        Console.Write( n + "th number : ");
        Console.WriteLine(
            center_pentadecagonal_num(n));
    }
}

// This code is contributed by ajit.
```

PHP

```
<?php
// PHP Program to find
// nth centered
// pentadecagonal number
```

```
// centered pentadecagonal function
function center_pentadecagonal_num($n)
{
    // Formula to calculate nth
    // centered pentadecagonal number
    return (15 * $n * $n -
        15 * $n + 2) / 2;
}

// Driver Code
$n = 3;
echo $n , "th number : ",
      center_pentadecagonal_num($n);
echo "\n";
$n = 10;
echo $n , "th number : ",
      center_pentadecagonal_num($n);

// This code is contributed by m_kit
?>
```

Output :

```
3th number : 46
10th number : 676
```

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/centered-pentadecagonal-number/>

Chapter 17

Centered cube number

Centered cube number - GeeksforGeeks

Given a number n, find the **n-th** centered cube number.

The **Centered cube number** counts the number of points which are formed by a point that is surrounded by concentric cubical layers in 3D with i^2 points on the square faces of the i-th layer. Source[\[WIKI\]](#). Please see [this](#) image for more clarity.

The first few Centered cube numbers are:

1, 9, 35, 91, 189, 341, 559, 855, 1241, 172.....

Examples :

```
Input : n = 1
Output : 9
```

```
Input : n = 7
Output : 855
```

Mathematical formula for **nth** centered cube number is given by:

$$n\text{-th Centered Cube Number} = (2n + 1)(n^2 + n + 1)$$

Below is the basic implementation of the above formula:

C++

```
// Program to find nth Centered cube
// number
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to find
// Centered cube number
int centered_cube(int n)
{
    // Formula to calculate nth
    // Centered cube number &
    // return it into main function.
    return (2 * n + 1) * (n * n + n + 1);
}

// Driver Code
int main()
{
    int n = 3;
    cout << n << "th Centered cube number: ";
    cout << centered_cube(n);
    cout << endl;

    n = 10;
    cout << n << "th Centered cube number: ";
    cout << centered_cube(n);
    return 0;
}
```

Java

```
// Java Program to find nth Centered
// cube number
import java.io.*;

class GFG {

    // Function to find
    // Centered cube number
    static int centered_cube(int n)
    {
        // Formula to calculate nth
        // Centered cube number &
        // return it into main function.
        return (2 * n + 1) * (n * n + n + 1);
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 3;
        System.out.print (n + "th Centered"
```

```
        + " cube number: ");
System.out.println (centered_cube(n));

n = 10;
System.out.print ( n + "th Centered"
                  + " cube number: ");
System.out.println (centered_cube(n));
}
}

// This code is contributed by m_kit.
```

Python3

```
# Python 3 Program to find
# nth Centered cube number

# Centered cube
# number function
def centered_cube(n) :

    # Formula to calculate
    # nth Centered cube
    # number return it
    # into main function.
    return (2 * n + 1) * (
        n * n + n + 1)

# Driver Code
if __name__ == '__main__' :

    n = 3
    print(n,"th Centered cube " +
          "number : " ,
          centered_cube(n))

    n = 10
    print(n,"th Centered cube " +
          "number : " ,
          centered_cube(n))

# This code is contributed by ajit
```

C#

```
// C# Program to find nth
// Centered cube number
```

```
using System;

class GFG
{

    // Function to find
    // Centered cube number
    static int centered_cube(int n)
    {
        // Formula to calculate
        // nth Centered cube
        // number & return it
        // into main function.
        return (2 * n + 1) *
               (n * n + n + 1);
    }

    // Driver code
    static public void Main ()
    {
        int n = 3;
        Console.Write(n + "th Centered" +
                      " cube number: ");
        Console.WriteLine (centered_cube(n));

        n = 10;
        Console.Write( n + "th Centered" +
                      " cube number: ");
        Console.WriteLine(centered_cube(n));
    }
}

// This code is contributed by aj_36
```

PHP

```
<?php
// Program to find nth
// Centered cube number

// Function to find
// Centered cube number
function centered_cube($n)
{
    // Formula to calculate nth
    // Centered cube number &
    // return it into main function.
    return (2 * $n + 1) *
```

```
        ($n * $n + $n + 1);  
}  
  
// Driver Code  
$n = 3;  
echo $n , "th Centered cube number: ";  
echo centered_cube($n);  
echo "\n";  
  
$n = 10;  
echo $n , "th Centered cube number: ";  
echo centered_cube($n);  
  
// This code is contributed by m_kit  
?>
```

Output :

```
3th Centered cube number: 91  
10th Centered cube number: 2331
```

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/centered-cube-number/>

Chapter 18

Centered decagonal number

Centered decagonal number - GeeksforGeeks

Given a number n, find the **nth** Centered decagonal number .

A **Centered Decagonal Number** is centered figurative number that represents a decagon with dot in center and all other dot surrounding it in successive decagonal form.
Source[\[Wiki\]](#).

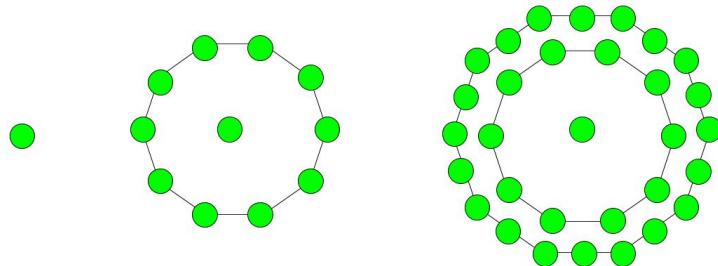


Fig :Centered decagonal number

The first few Centered Decagonal Numbers are :
1, 11, 31, 61, 101, 151, 211, 281, 361, 451, 551, 661.....

Examples :

Input : 3
Output : 31

```
Input : 6
Output : 151
```

In mathematics centered decagonal number for **n-th** term is given by :

Below is the basic implementation of the above idea.

C++

```
// Program to find nth
// centered decagonal
// number
#include <bits/stdc++.h>
using namespace std;

// Centered decagonal
// number function

int centereddecagonalnum(int n)
{
    // Formula to calculate nth
    // centered decagonal number &
    // return it into main function.
    return (5 * n * n + 5 * n + 1);
}

// Driver Code
int main()
{
    int n = 5;
    cout << n << "th centered decagonal"
        << "number: ";
    cout << centereddecagonalnum(n);
    cout << endl;
    n = 9;
    cout << n << "th centered decagonal"
        << "number: ";
    cout << centereddecagonalnum(n);

    return 0;
}
```

Java

```
// Java Program to find nth
```

```
// centered decagonal number
import java.io.*;

class GFG
{

    // Centered decagonal
    // number function
    static int centereddecagonalnum(int n)
    {

        // Formula to calculate nth
        // centered decagonal number &
        // return it into main function.
        return (5 * n * n + 5 * n + 1);
    }

    // Driver Code
    public static void main (String[] args)
    {
        int n = 5;
        System.out.print(n + "th centered " +
                          "decagonal number: ");
        System.out.println(centereddecagonalnum(n));

        n = 9;
        System.out.print(n + "th centered " +
                          "decagonal number: ");
        System.out.println(centereddecagonalnum(n));
    }
}

// This code is contributed by m_kit
```

Python 3

```
# Program to find nth
# centered decagonal number

# Centered decagonal
# number function
def centereddecagonalnum(n) :

    # Formula to calculate
    # nth centered decagonal
    # number & return it
    # into main function.
    return (5 * n * n +
```

```
5 * n + 1)

# Driver Code
if __name__ == '__main__':
    n = 5
    print(n,"th centered decagonal " +
          "number : ",
          centereddecagonalnum(n))

    n = 9
    print(n,"th centered decagonal " +
          "number : ",
          centereddecagonalnum(n))

# This code is contributed by m_kit

C#
// Program to find nth
// centered decagonal
// number
using System;

class GFG
{
// Centered decagonal
// number function
static int centereddecagonalnum(int n)
{
    // Formula to calculate nth
    // centered decagonal number &
    // return it into main function.
    return (5 * n * n + 5 * n + 1);
}

// Driver Code
static public void Main ()
{
int n = 5;
Console.Write(n + "th centered decagonal"+
              "number: ");
Console.WriteLine(centereddecagonalnum(n));

n = 9;
Console.Write(n + "th centered decagonal"+
              "number: ");
Console.WriteLine(centereddecagonalnum(n));
```

```
}
```

```
}
```

```
// This code is contributed by aj_36
```

PHP

```
<?php
// Program to find nth
// centered decagonal number

// Centered decagonal
// number function
function centereddecagonalnum($n)
{
    // Formula to calculate
    // nth centered decagonal
    // number & return it
    // into main function.
    return (5 * $n * $n +
           5 * $n + 1);
}

// Driver Code
$n = 5;
echo $n , "th centered decagonal",
      "number: ";
echo centereddecagonalnum($n);
echo "\n";

$n = 9;
echo $n , "th centered decagonal",
      "number: ";
echo centereddecagonalnum($n);

// This code is contributed by ajit
?>
```

Output

```
5th centered decagonalnumber: 151
9th centered decagonalnumber: 451
```

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/centered-decagonal-number/>

Chapter 19

Centered dodecahedral number

Centered dodecahedral number - GeeksforGeeks

Given a number n, find the **nth** Centered Dodecahedral Number.

A **Centered Dodecahedral number** is class of figurative number. It is formed by a central dot, surrounded by successive dodecahedral(polyhedral with 12 flat surfaces) layers.

The first few Centered dodecahedral numbers (**where n = 0, 1, 2, 3.....**) are :
1, 33, 155, 427, 909, 1661

Examples:

```
Input : 5  
Output : 1661
```

```
Input :1  
Output :33
```

Mathematical formula for the **nth** Centered dodecahedral number is given by:

Below is the basic implementation of the above idea:

C++

```
// Program to find nth centered  
// dodecahedral number  
#include <bits/stdc++.h>  
using namespace std;
```

```
// Function to find
// centered dodecahedral number
int CenteredDodecahedral_num(long int n)
{
    // Formula to calculate nth
    // centered dodecahedral number
    // and return it into main function.
    return (2 * n + 1) * (5 * n * n + 5 * n + 1);
}
// Driver Code
int main()
{
    long int n = 3;
    // print result
    cout << n << "th Centered Dodecahedral number : ";
    cout << CenteredDodecahedral_num(n) << endl;

    n = 10;
    // print result
    cout << n << "th Centered Dodecahedral number : ";
    cout << CenteredDodecahedral_num(n);

    return 0;
}
```

Java

```
// Java Program to find nth
// centered dodecahedral number
import java.io.*;

class GFG {

    // Function to find centered
    // dodecahedral number
    static int CenteredDodecahedral_num(int n)
    {

        // Formula to calculate nth
        // centered dodecahedral number
        // and return it into main function.

        return (2 * n + 1) *
               (5 * n * n + 5 * n + 1);
    }

    // Driver Code
    public static void main (String[] args)
```

```
{  
  
    int n = 3;  
  
    // print result  
    System.out.print( n + "th Centered "  
                      + "Dodecahedral number : ");  
    System.out.println (  
                      CenteredDodecahedral_num(n));  
  
    n = 10;  
  
    // print result  
    System.out.print( n + "th Centered "  
                      + "Dodecahedral number : ");  
    System.out.println(  
                      CenteredDodecahedral_num(n));  
}  
}  
  
// This code is contributed by m_kit.
```

Python3

```
# Program to find nth centered  
# dodecahedral number  
  
# Function to find centered  
# dodecahedral number  
def CenteredDodecahedral_num(n) :  
  
    # Formula to calculate nth  
    # centered dodecahedral number  
    return (2 * n + 1) * (5 * n * n + 5 * n + 1)  
  
# Driver Code  
if __name__ == '__main__' :  
  
    n = 3  
    print(n,"rd centered dodecahedral number: ",  
          CenteredDodecahedral_num(n))  
    n = 10  
    print(n,"th centered dodecahedral number : ",  
          CenteredDodecahedral_num(n))  
  
# This code is contributed by aj_36
```

C#

```
// C# Program to find
// nth centered
// dodecahedral number
using System;

class GFG
{

    // Function to find
    // nth centered
    // dodecahedral number
    static int CenteredDodecahedral_num(int n)
    {

        // Formula to calculate
        // nth centered dodecahedral
        // number and return it
        // into main function.
        return (2 * n + 1) *
            (5 * n * n +
            5 * n + 1);
    }

    // Driver Code
    static public void Main ()
    {
        int n = 3;

        // print result
        Console.Write( n + "th Centered " +
                      "Dodecahedral number : ");
        Console.WriteLine(
                      CenteredDodecahedral_num(n));

        n = 10;

        // print result
        Console.Write( n + "th Centered " +
                      "Dodecahedral number : ");
        Console.WriteLine(
                      CenteredDodecahedral_num(n));
    }
}

// This code is contributed by ajit
```

PHP

```
<?php
// Program to find nth centered
// dodecahedral number

// Function to find
// centered dodecahedral number
function CenteredDodecahedral_num($n)
{
    // Formula to calculate nth
    // centered dodecahedral number
    // and return it into main function.
    return (2 * $n + 1) *
        (5 * $n * $n +
         5 * $n + 1);
}

// Driver Code
$n = 3;
// print result
echo $n, "th Centered Dodecahedral " .
          "number : ";
echo CenteredDodecahedral_num($n), "\n";

$n = 10;
// print result
echo $n, "th Centered Dodecahedral " .
          "number : ";
echo CenteredDodecahedral_num($n);

// This code is contributed by akt_mit
?>
```

Output :

```
3th Centered Dodecahedral number : 427
10th Centered Dodecahedral number : 11571
```

References:

https://en.wikipedia.org/wiki/Centered_dodecahedral_number

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/centered-dodecahedral-number/>

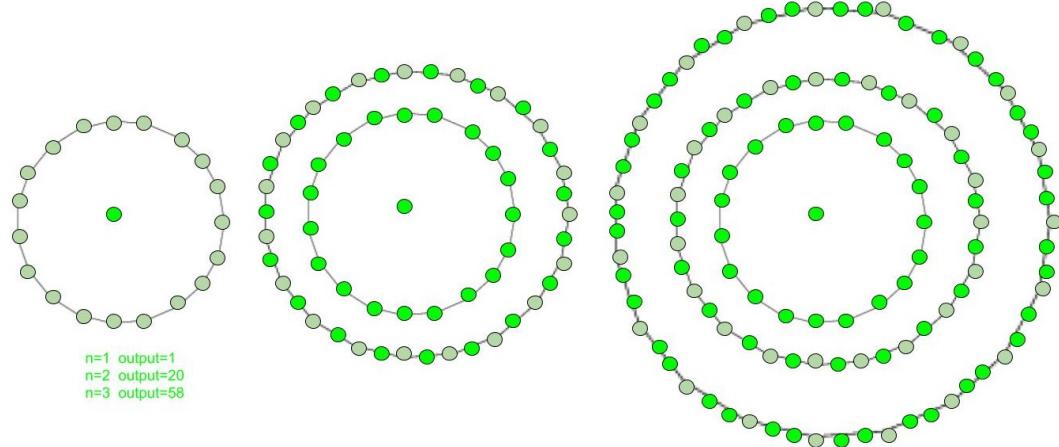
Chapter 20

Centered nonadecagonal number

Centered nonadecagonal number - GeeksforGeeks

Given a number n, find the **nth** Centered Nonadecagonal number.

A **Centered Nonadecagonal Number** represents a dot in the centre and other dots surrounding it in successive nonadecagon(19 sided polygon) layers.



The first few Centered Nonadecagonal numbers are:

1, 20, 58, 115, 191, 286, 400, 533, 685, 856, 1046, 1255.....

Examples :

Input : 3

Output : 58

```
Input : 13
Output :1483
```

In mathematics, Centered nonadecagonal number for **n-th** term is given by :

Below is the basic implementation of the above idea:

C++

```
// C++ Program to find
// nth centered
// nonadecagonal number
#include <bits/stdc++.h>
using namespace std;

// centered nonadecagonal
// function
int center_nonadecagon_num( long int n )
{
    // Formula to calculate nth
    // centered nonadecagonal number
    return (19 * n * n - 19 * n + 2) / 2;
}

// Driver Code
int main()
{
    long int n = 2;
    cout << n << "th centered nonadecagonal number : "
        << center_nonadecagon_num(n);
    cout << endl;
    n = 7;
    cout << n << "th centered nonadecagonal number : "
        << center_nonadecagon_num(n);

    return 0;
}
```

Java

```
// Java Program to find nth centered
// nonadecagonal number
import java.io.*;
```

```
class GFG {

    // centered nonadecagonal
    // function
    static int center_nonadecagon_num(int n)
    {
        // Formula to calculate nth
        // centered nonadecagonal number
        return (19 * n * n - 19 * n + 2) / 2;
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 2;
        System.out.print ( n + "th centered "
            + "nonadecagonal number : ");
        System.out.println (
            center_nonadecagon_num(n));

        n = 7;
        System.out.print ( n + "th centered "
            + "nonadecagonal number : ");
        System.out.println(
            center_nonadecagon_num(n));
    }
}

// This code is contributed by m_kit
```

Python3

```
# Program to find nth
# centered nonadecagonal number
def center_nonadecagon_num(n) :

    # Formula to calculate
    # nth centered nonadecagonal
    # number & return it into
    # main function.

    return (19 * n * n -
           19 * n + 2) // 2

# Driver Code
if __name__ == '__main__' :

    n = 2
```

```
print(n,"nd centered nonadecagonal " +
      "number : ",
      center_nonadecagon_num(n))

n = 7
print(n,"nd centered nonadecagonal " +
      "number : ",
      center_nonadecagon_num(n))

# This code is contributed by ajit
```

C#

```
// C# Program to find
// nth centered
// nonadecagonal number
using System;

class GFG
{

    // centered nonadecagonal
    // function
    static int center_nonadecagon_num(int n)
    {
        // Formula to calculate nth
        // centered nonadecagonal number
        return (19 * n * n -
               19 * n + 2) / 2;
    }

    // Driver code
    static public void Main ()
    {

        int n = 2;
        Console.Write ( n + "th centered " +
                       "nonadecagonal number : ");
        Console.WriteLine(
            center_nonadecagon_num(n));

        n = 7;
        Console.Write( n + "th centered " +
                      "nonadecagonal number : ");
        Console.WriteLine(
            center_nonadecagon_num(n));
    }
}
```

```
// This code is contributed by ajit
```

PHP

```
<?php
// PHP Program to find
// nth centered
// nonadecagonal number

// centered nonadecagonal
// function
function center_nonadecagon_num( $n )
{
    // Formula to calculate nth
    // centered nonadecagonal number
    return (19 * $n * $n -
            19 * $n + 2) / 2;
}

// Driver Code
$n = 2;
echo $n , "th centered " +
        "nonadecagonal number : ",
        center_nonadecagon_num($n);
echo "\n";
$n = 7;
echo $n , "th centered " +
        "nonadecagonal number : ",
        center_nonadecagon_num($n);

// This code is contributed by ajit
?>
```

Output :

```
2th centered nonadecagonal number : 20
7th centered nonadecagonal number : 400
```

References:

<http://oeis.org/A069132>

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/centered-nonadecagonal-number/>

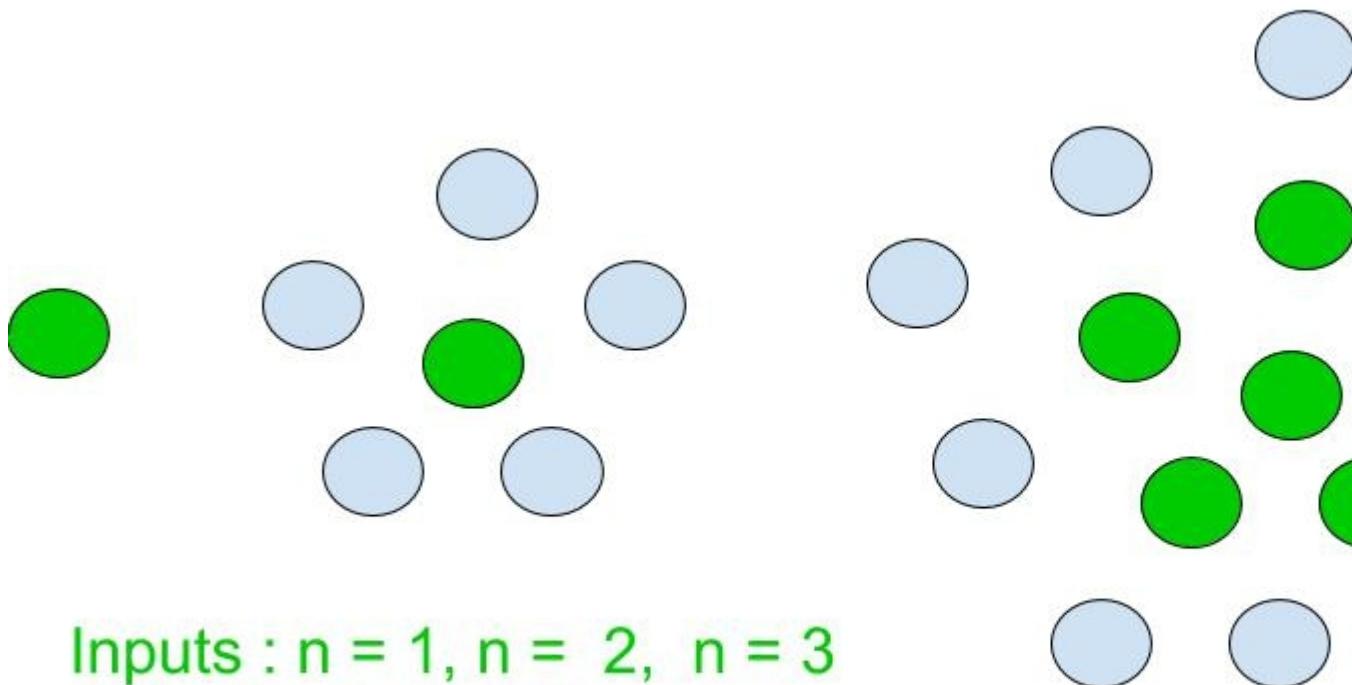
Chapter 21

Centered pentagonal number

Centered pentagonal number - GeeksforGeeks

Given an integer n, find the nth Centered pentagonal number.

A **Centered Pentagonal Number** is a centered figurate number that represents a pentagon with a dot in the centre and other dots surrounding it in pentagonal layers successively
[Source: [Wiki](#)]



Inputs : $n = 1, n = 2, n = 3$
Outputs : 1, 6, 16

Few Centred pentagonal Number are :
1, 6, 16, 31, 51, 76, 106, 141, 181, 226, 276, 331, 391.....

Examples :

Input : 3
Output : 16

Input : 9
Output : 181

Approach:
Centered pentagonal for **n-th** term is given by :

Basic implementation of the above approach :

C++

```
// Program to find nth
// Centered pentagonal number.
#include <bits/stdc++.h>
using namespace std;

// centered pentagonal number function

int centered_pentagonal_Num(int n)
{
    // Formula to calculate nth
    // Centered pentagonal number
    // and return it into main function.

    return (5 * n * n - 5 * n + 2) / 2;
}

// Driver Code
int main()
{
    int n = 7;
    cout << n << "th Centered pentagonal number: ";
    cout << centered_pentagonal_Num(n);
    return 0;
}
```

Java

```
// Program to find nth
// Centered pentagonal number
import java.io.*;

class GFG
{

    // centered pentagonal
    // number function
    static int centered_pentagonal_Num(int n)
    {
        // Formula to calculate
        // nth Centered pentagonal
        // number and return it
        // into main function.
```

```
        return (5 * n * n - 5 * n + 2) / 2;
    }

// Driver Code
public static void main (String[] args)
{
    int n = 7;
    System.out.print(n + "th Centered " +
                      "pentagonal number: ");
    System.out.println(centred_pentagonal_Num(n));
}
}

// This code is contributed by anuj_67.
```

Python3

```
# Python program to find Nth
# Centered pentagonal number.

# Function to calculate
# Centered pentagonal number.

def centred_pentagonal_Num(n):

    # Formula to calculate nth
    # Centered pentagonal number.

    return (5 * n * n - 5 * n + 2) // 2

# Driver Code
n = 7
print("%sth Centered pentagonal number : " %n,
      centred_pentagonal_Num(n))

# This code is contributed by ajit
```

C#

```
// C# Program to find nth
// Centered pentagonal number
using System;

class GFG
{

    // centered pentagonal
```

```
// number function
static int centered_pentagonal_Num(int n)
{
    // Formula to calculate
    // nth Centered pentagonal
    // number and return it
    // into main function.

    return (5 * n * n - 5 * n + 2) / 2;
}

// Driver Code
public static void Main ()
{
    int n = 7;
    Console.Write(n + "th Centered " +
                  "pentagonal number: ");
    Console.WriteLine(centered_pentagonal_Num(n));
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP Program to find nth
// Centered pentagonal number.

// Centered pentagonal number function

function centered_pentagonal_Num($n)
{
    // Formula to calculate nth
    // Centered pentagonal number
    // and return it into main function.

    return (5 * $n * $n - 5 * $n + 2) / 2;
}

// Driver Code
$n = 7;
echo $n , "th Centered pentagonal number: ";
echo centered_pentagonal_Num($n);

// This code is contributed by aj_36
?>
```

Output :

7th Centered pentagonal number: 106

Improved By : [jit_t](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/centered-pentagonal-number/>

Chapter 22

Centered tridecagonal number

Centered tridecagonal number - GeeksforGeeks

Given a number n, the task is to find the nth Centered Tridecagonal Number.

A **Centered tridecagonal number** represents a dot at the center and other dots surrounding the center dot in the successive tridecagonal(13 sided polygon) layer.

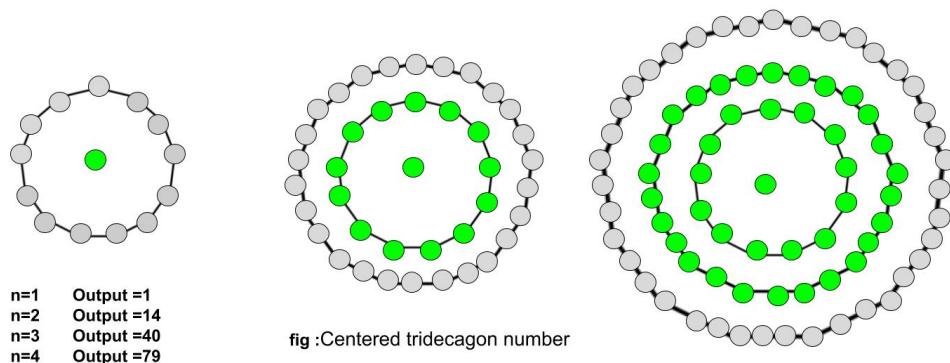
Examples :

Input : 2

Output : 14

Input : 9

Output : 469



Formula for nth Centered tridecagonal number:

C++

```
// C++ Program to find nth
// centered tridecagonal number
#include <bits/stdc++.h>
using namespace std;

// Function to find nth centered
// tridecagonal number
int centeredTridecagonalNum(long int n)
{
    // Formula to calculate nth
    // centered tridecagonal number
    return (13 * n * (n - 1) + 2) / 2;
}

// Drivers code
int main()
{
    long int n = 3;
    cout << centeredTridecagonalNum(n);
    cout << endl;
    n = 10;
    cout << centeredTridecagonalNum(n);

    return 0;
}
```

Java

```
// Java Program to find nth
// centered tridecagonal number
import java.io.*;

class GFG
{

// Function to find nth centered
// tridecagonal number
static long centeredTridecagonalNum(long n)
{
    // Formula to calculate nth
    // centered tridecagonal number
    return (13 * n * (n - 1) + 2) / 2;
}
```

```
}

// Driver Code
public static void main (String[] args)
{
    long n = 3;
    System.out.println(centeredTridecagonalNum(n));
    n = 10;
    System.out.println(centeredTridecagonalNum(n));
}
}

// This code is contributed by anuj_67.
```

Python3

```
# Program to find nth centered
# tridecagonal number

# Function to find centered
# tridecagonal number
def centeredTridecagonalNum(n) :

    # Formula to calculate nth
    # centered tridecagonal number
    return (13 * n *
           (n - 1) + 2) // 2

# Driver Code
if __name__ == '__main__' :

    n = 3
    print(centeredTridecagonalNum(n))
    n = 10
    print(centeredTridecagonalNum(n))

# This code is contributed
# by akt_mit
```

C#

```
// C# Program to find nth
// centered tridecagonal number
using System;

class GFG
{
```

```
// Function to find nth centered
// tridecagonal number
static long centeredTridecagonalNum(long n)
{
    // Formula to calculate nth
    // centered tridecagonal number
    return (13 * n * (n - 1) + 2) / 2;
}

// Driver Code
public static void Main ()
{
    long n = 3;
    Console.WriteLine(centeredTridecagonalNum(n));
    n = 10;
    Console.WriteLine(centeredTridecagonalNum(n));
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP Program to find nth
// centered tridecagonal number

// Function to find nth centered
// tridecagonal number
function centeredTridecagonalNum( $n )
{
    // Formula to calculate nth
    // centered tridecagonal number
    return (13 * $n *
           ($n - 1) + 2) / 2;
}

// Driver Code
$n = 3;
echo centeredTridecagonalNum($n);
echo"\n";

$n = 10;
echo centeredTridecagonalNum($n);

// This code is contributed by anuj_67.
?>
```

Output :

40
586

Reference: http://oeis.org/wiki/Figurate_numbers

Improved By : [vt_m, jit_t](#)

Source

<https://www.geeksforgeeks.org/centered-tridecagonal-number/>

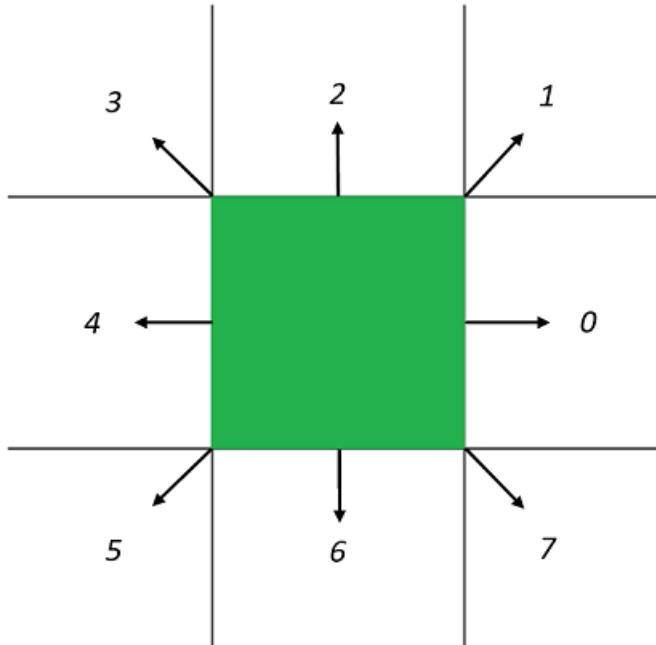
Chapter 23

Chain Code for 2D Line

Chain Code for 2D Line - GeeksforGeeks

[Chain code](#)is a lossless compression technique used for representing an object in images. The co-ordinates of any continuous boundary of an object can be represented as a string of numbers where each number represents a particular direction in which the next point on the connected line is present. One point is taken as the reference/starting point and on plotting the points generated from the chain, the original figure can be re-drawn.

This article describes how to generate a 8-neighbourhood chain code of a 2-D straight line. In a rectangular grid, a point can have at most 8 surrounding points as shown below. The next point on the line has to be one of these 8 surrounding points. Each direction is assigned a code. Using this code we can find out which of the surrounding point should be plotted next.



GeeksforGeeks
A computer science portal for geeks

The chain codes could be generated by using conditional statements for each direction but it becomes very tedious to describe for systems having large number of directions(3-D grids can have up to 26 directions). Instead we use a hash function. The difference in X(Δx) and Y(Δy) co-ordinates of two successive points are calculated and hashed to generate the key for the chain code between the two points.

Chain code list: [5, 6, 7, 4, -1, 0, 3, 2, 1]

Hash function: $\mathcal{H}(\Delta x, \Delta y) = \Delta y + \Delta x + 4$

Hash table:-

Δx	Δy	$\mathcal{H}(\Delta x, \Delta y)$	Chain Code []
1	0	5	0
1	1	8	1
0	1	7	2
-1	1	6	3
-1	0	3	4
-1	-1	0	5

dx	dy	$O(dx, dy)$	ChainCode[O]
0	-1	1	6
1	-1	2	7

The function does not generate the value 4 so a dummy value is stored there.

Examples:

Input : (2, -3), (-4, 2)

Output : Chain code for the straight line from (2, -3) to (-4, 2) is 333433

Input : (-7, -4), (9, 3)

Output : Chain code for the straight line from (-7, -4) to (9, 3) is 0101010100101010

Python3

```
# Python3 code for generating 8-neighbourhood chain
# code for a 2-D line

codeList = [5, 6, 7, 4, -1, 0, 3, 2, 1]

# This function generates the chaincode
# for transition between two neighbour points
def getChainCode(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    hashKey = 3 * dy + dx + 4
    return codeList[hashKey]

'''This function generates the list of
chaincodes for given list of points'''
def generateChainCode(ListOfPoints):
    chainCode = []
    for i in range(len(ListOfPoints) - 1):
        a = ListOfPoints[i]
        b = ListOfPoints[i + 1]
        chainCode.append(getChainCode(a[0], a[1], b[0], b[1]))
    return chainCode

'''This function generates the list of points for
a staright line using Bresenham's Algorithm'''
```

```

def Bresenham2D(x1, y1, x2, y2):
    ListOfPoints = []
    ListOfPoints.append([x1, y1])
    xdif = x2 - x1
    ydif = y2 - y1
    dx = abs(xdif)
    dy = abs(ydif)
    if(xdif > 0):
        xs = 1
    else:
        xs = -1
    if (ydif > 0):
        ys = 1
    else:
        ys = -1
    if (dx > dy):

        # Driving axis is the X-axis
        p = 2 * dy - dx
        while (x1 != x2):
            x1 += xs
            if (p >= 0):
                y1 += ys
                p -= 2 * dx
            p += 2 * dy
            ListOfPoints.append([x1, y1])
    else:

        # Driving axis is the Y-axis
        p = 2 * dx-dy
        while(y1 != y2):
            y1 += ys
            if (p >= 0):
                x1 += xs
                p -= 2 * dy
            p += 2 * dx
            ListOfPoints.append([x1, y1])
    return ListOfPoints

def DriverFunction():
    (x1, y1) = (-9, -3)
    (x2, y2) = (10, 1)
    ListOfPoints = Bresenham2D(x1, y1, x2, y2)
    chainCode = generateChainCode(ListOfPoints)
    chainCodeString = "" .join(str(e) for e in chainCode)
    print ('Chain code for the straight line from', (x1, y1),
          'to', (x2, y2), 'is', chainCodeString)

```

DriverFunction()

Output:

Chain code for the straight line from (-9, -3) to (10, 1) is 0010000100010000100

Source

<https://www.geeksforgeeks.org/chain-code-for-2d-line/>

Chapter 24

Check if a given circle lies completely inside the ring formed by two concentric circles

Check if a given circle lies completely inside the ring formed by two concentric circles - GeeksforGeeks

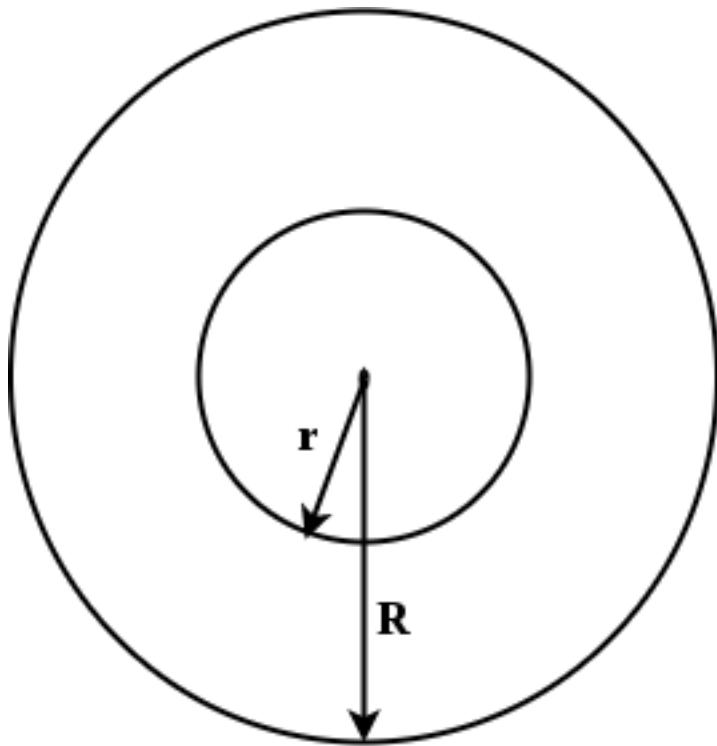
Given two circles of radius r and R , both have their centre at the origin. Now, given another circle of radius r_1 and centre at (x_1, y_1) . Check, if the third circle(circle of radius r_1) lies completely inside the ring formed by two circles of radius r and R .

Examples :

```
Input : r = 8 R = 4
        r1 = 2 x1 = 6 y1 = 0
Output : yes
```

```
Input : r = 8 R = 4
        r1 = 2 x1 = 5 y1 = 0
Output : no
```

Important : Concentric circles are those circles which have same centre. The region lying between two concentric circles is called annulus or the circular ring.

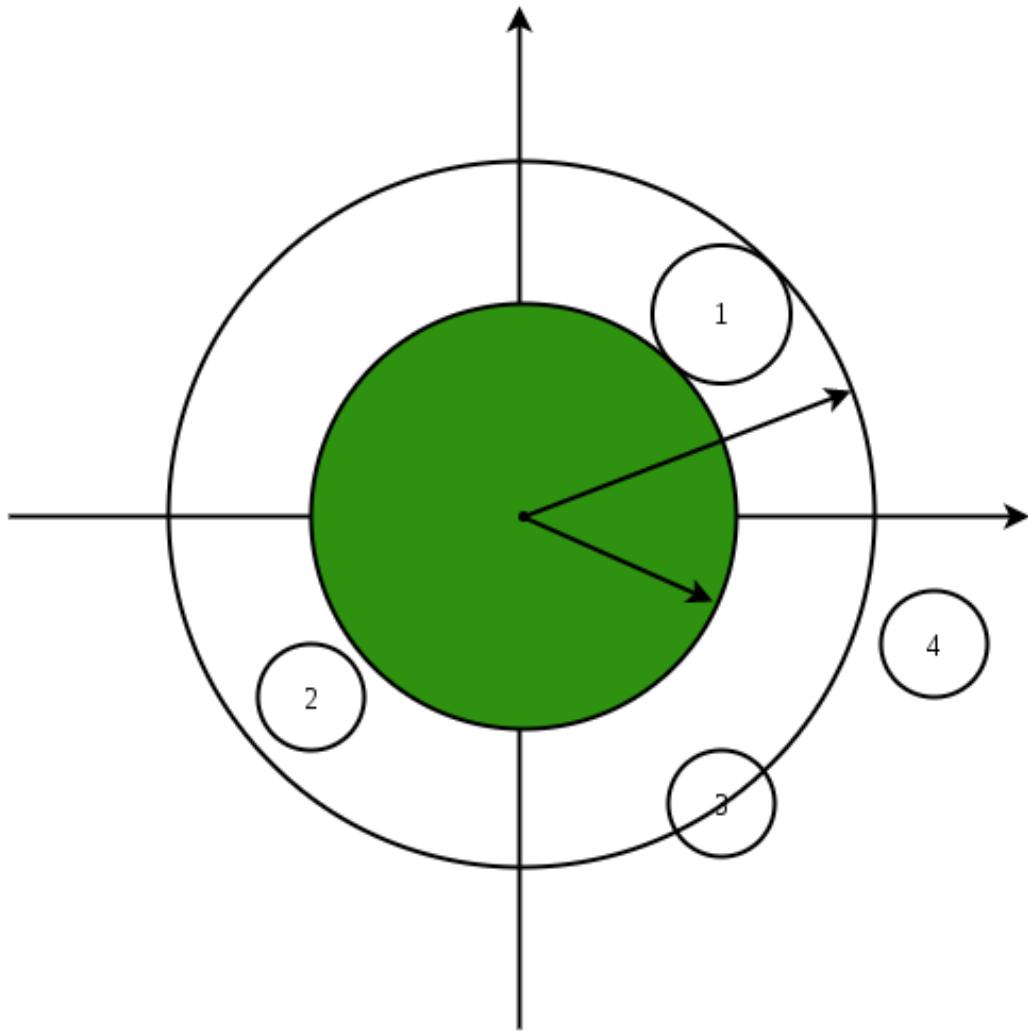


Example :

There are two concentric circles with their centre at origin(0, 0) and radius as $r = 8$ and $R = 4$.

- 1.) Circle 1 and 2 lies inside the ring.
- 2.) Circle 3 and 4 are outside the ring.

The complete figure can be visualised as given below :



Approach :

This problem can be solved using **Pythagoras Theorem**. Compute the distance between the centre of the circle and origin using Pythagoras theorem, suppose it is denoted by ‘dis’. After computing the distance just check that the value of $(\text{dis} - r1) >= 0$ and $(\text{dis} + r1) <= R$. If both these conditions hold then the circle lies completely inside the ring.

C++

```
// CPP code to check if a circle
// lies in the ring
#include <bits/stdc++.h>
using namespace std;

// Function to check if circle
// lies in the ring
```

```
bool checkcircle(int r, int R, int r1,
                 int x1, int y1)
{
    // distance between center of circle
    // center of concentric circles(origin)
    // using Pythagoras theorem
    int dis = sqrt(x1*x1+y1*y1);

    // Condition to check if circle is
    // strictly inside the ring
    return (dis-r1 >= R && dis+r1 <= r);
}

// Driver Code
int main()
{
    // Both circle with radius 'r'
    // and 'R' have center (0,0)
    int r = 8, R = 4, r1 = 2, x1 = 6, y1 = 0;
    if (checkcircle(r, R, r1, x1, y1))
        cout << "yes" << endl;
    else
        cout << "no" << endl;

    return 0;
}
```

Java

```
// Java code to check if a
// circle lies in the ring
import java.io.*;

class ring
{
    // Function to check if circle
    // lies in the ring
    public static boolean checkcircle(int r, int R,
                                      int r1, int x1, int y1)
    {
        // distance between center of circle
        // center of concentric circles(origin)
        // using Pythagoras theorem
        int dis = (int)Math.sqrt(x1 * x1 +
                               y1 * y1);

        // Condition to check if circle
        // is strictly inside the ring
```

```
        return (dis - r1 >= R && dis + r1 <= r);
    }

// Driver Code
public static void main(String args[])
{
    // Both circle with radius 'r'
    // and 'R' have center (0,0)
    int r = 8, R = 4, r1 = 2, x1 = 6, y1 = 0;

    if (checkcircle(r, R, r1, x1, y1))
        System.out.println("yes");
    else
        System.out.println("no");
}
```

Python3

```
# Python3 code to check if
# a circle lies in the ring
import math

# Function to check if circle
# lies in the ring
def checkcircle(r, R, r1, x1, y1):

    # distance between center of circle
    # center of concentric circles(origin)
    # using Pythagoras theorem
    dis = int(math.sqrt(x1 * x1 + y1 * y1))

    # Condition to check if circle is
    # strictly inside the ring
    return (dis-r1 >= R and dis+r1 <= r)

# Driver Code

# Both circle with radius 'r'
# and 'R' have center (0,0)
r = 8; R = 4; r1 = 2; x1 = 6; y1 = 0
if (checkcircle(r, R, r1, x1, y1)):
    print("yes")
else:
    print("no")

# This code is contributed by Smitha Dinesh Semwal.
```

C#

```
// C# code to check if a
// circle lies in the ring
using System;

class ring {

    // Function to check if circle
    // lies in the ring
    public static bool checkcircle(int r, int R,
                                    int r1, int x1, int y1)
    {
        // distance between center of circle
        // center of concentric circles(origin)
        // using Pythagoras theorem
        int dis = (int)Math.Sqrt(x1 * x1 + y1 * y1);

        // Condition to check if circle
        // is strictly inside the ring
        return (dis - r1 >= R && dis + r1 <= r);
    }

    // Driver Code
    public static void Main()
    {
        // Both circle with radius 'r'
        // and 'R' have center (0, 0)
        int r = 8, R = 4, r1 = 2, x1 = 6, y1 = 0;

        if (checkcircle(r, R, r1, x1, y1))
            Console.WriteLine("yes");
        else
            Console.WriteLine("no");
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP code to check if a circle
// lies in the ring

// Function to check if circle
// lies in the ring
```

```
function checkcircle($r, $R, $r1,
                     $x1, $y1)
{
    // distance between center of circle
    // center of concentric circles(origin)
    // using Pythagoras theorem
    $dis = sqrt($x1 * $x1 + $y1 * $y1);

    // Condition to check if circle is
    // strictly inside the ring
    return ($dis-$r1 >= $R && $dis + $r1 <= $r);
}

// Driver Code
// Both circle with radius 'r'
// and 'R' have center (0,0)
$r = 8; $R = 4;
$r1 = 2; $x1 = 6;
$y1 = 0;
if (checkcircle($r, $R, $r1, $x1, $y1))

    echo "yes" ,"\n";
else
    echo "no" ,"\n";

// This code is contributed by ajit.
?>
```

Output:

yes

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/check-given-circle-lies-completely-inside-ring-formed-two-concentric-circles/>

Chapter 25

Check if a line at 45 degree can divide the plane into two equal weight parts

Check if a line at 45 degree can divide the plane into two equal weight parts - GeeksforGeeks

Given a set of n points (x_i, y_i) in 2D coordinate. Each point has some weight w_i . The task is to check whether a line at 45 degree can be drawn so that sum of weights of points on each side are equal.

Examples:

```
Input : x1 = -1, y1 = 1, w1 = 3  
x2 = -2, y2 = 1, w2 = 1  
x3 = 1, y3 = -1, w3 = 4
```

```
Output : Yes
```

```
Input : x1 = 1, y1 = 1, w1 = 2  
x2 = -1, y2 = 1, w2 = 1  
x3 = 1, y3 = -1, w3 = 2
```

```
Output : No
```

First, let's try to solve above problem for a vertical line i.e if a line $x = i$ can divide the plane into two part such that the sum of weight at each side is equal.

Observe, multiple points with the same x-coordinate can be treated as one point with weight equal to the sum of weights of all points with the same x-coordinate.

Now, traverse through all x-coordinates from the minimum x-coordinate to maximum x-coordinate. So, make an array **prefix_sum[]**, which will store the sum of weights till the

point $x = i$.

So, there can be two options for which the answer can be ‘Yes’:

- Either $\text{prefix_sum}[1, 2, \dots, i-1] = \text{prefix_sum}[i+1, \dots, n]$
- or there exist a point i such that a line passes somewhere in between $x = i$ and $x = i+1$ and $\text{prefix_sum}[1, \dots, i] = \text{prefix_sum}[i+1, \dots, n]$, where $\text{prefix_sum}[i, \dots, j]$ is the sum of weight of points from i to j .

```
int is_possible = false;
for (int i = 1; i < prefix_sum.size(); i++)
    if (prefix_sum[i] == total_sum - prefix_sum[i])
        is_possible = true

if (prefix_sum[i-1] == total_sum - prefix_sum[i])
    is_possible = true
```

Now, to solve for a line at 45 degrees, we will rotate each point by 45 degrees.

Refer: [2D Transformation or Rotation of objects](#)

So, point at (x, y) , after 45 degree rotation will become $((x - y)/\sqrt{2}, (x + y)/\sqrt{2})$. We can ignore the $\sqrt{2}$ since it is the scaling factor. Also, we don’t need to care about y -coordinate after rotation because a vertical line cannot distinguish between the point having the same x -coordinate. (x, y_1) and (x, y_2) will lie to the right, left or on any line of the form $x = k$.

```
#include <bits/stdc++.h>
using namespace std;

// Checking if a plane can be divided by a line
// at 45 degrees such that weight sum is equal
void is_partition_possible(int n, int x[],
                           int y[], int w[])
{
    map<int, int> weight_at_x;
    int max_x = -2e3, min_x = 2e3;

    // Rotating each point by 45 degrees and
    // calculating prefix sum.
    // Also, finding maximum and minimum x
    // coordinates
    for (int i = 0; i < n; i++) {
        int new_x = x[i] - y[i];
        max_x = max(max_x, new_x);
        min_x = min(min_x, new_x);

        // storing weight sum upto x - y point
        weight_at_x[new_x] += w[i];
    }
}
```

```
}

vector<int> sum_till;
sum_till.push_back(0);

// Finding prefix sum
for (int x = min_x; x <= max_x; x++) {
    sum_till.push_back(sum_till.back() +
                        weight_at_x[x]);
}

int total_sum = sum_till.back();

int partition_possible = false;
for (int i = 1; i < sum_till.size(); i++) {
    if (sum_till[i] == total_sum - sum_till[i])
        partition_possible = true;

    // Line passes through i, so it neither
    // falls left nor right.
    if (sum_till[i - 1] == total_sum - sum_till[i])
        partition_possible = true;
}

printf(partition_possible ? "YES\n" : "NO\n");
}

// Driven Program
int main()
{
    int n = 3;
    int x[] = { -1, -2, 1 };
    int y[] = { 1, 1, -1 };
    int w[] = { 3, 1, 4 };
    is_partition_possible(n, x, y, w);

    return 0;
}
```

Output

Yes

Source

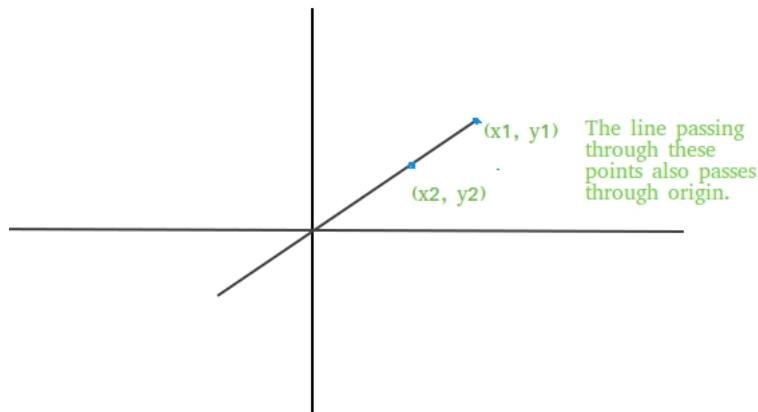
<https://www.geeksforgeeks.org/check-if-a-line-at-45-degree-can-divide-the-plane-into-two-equal-weight-parts/>

Chapter 26

Check if a line passes through the origin

Check if a line passes through the origin - GeeksforGeeks

Given two coordinates of a line as (x_1, y_1) and (x_2, y_2) , find if the line passing through these points also passes through origin or not.



Examples:

Input : $(x_1, y_1) = (10, 0)$
 $(x_2, y_2) = (20, 0)$

Output : Yes

The line passing through these points clearly passes through the origin as the line is x axis.

Input : $(x_1, y_1) = (1, 28)$

```
(x2, y2) = (2, 56)
Output : Yes
```

Approach: Equation of a line passing through two points (x_1, y_1) and (x_2, y_2) is given by $y - y_1 = ((y_2 - y_1) / (x_2 - x_1))(x - x_1) + c$

If line is also passing through origin, then $c=0$, so equation of line becomes

$y - y_1 = ((y_2 - y_1) / (x_2 - x_1))(x - x_1)$

Keeping $x=0, y=0$ in the above equation we get,

$x_1(y_2 - y_1) = y_1(x_2 - x_1)$

So above equation must be satisfied if any line passing through two coordinates (x_1, y_1) and (x_2, y_2) also passes through origin $(0, 0)$.

C++

```
/* C++ program to find if line passing through
   two coordinates also passes through origin
   or not */
#include <bits/stdc++.h>
using namespace std;

bool checkOrigin(int x1, int y1, int x2, int y2)
{
    return (x1 * (y2 - y1) == y1 * (x2 - x1));
}

// Driver code
int main()
{
    if (checkOrigin(1, 28, 2, 56) == true)
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
// Java program to find if line passing through
// two coordinates also passes through origin
// or not
import java.io.*;

class GFG {

    static boolean checkOrigin(int x1, int y1,
                               int x2, int y2)
    {
        return (x1 * (y2 - y1) == y1 * (x2 - x1));
    }
}
```

```
}

// Driver code
public static void main (String[] args)
{
    if (checkOrigin(1, 28, 2, 56) == true)
        System.out.println("Yes");
    else
        System.out.println("No");
}
}

// This code is contributed by Ajit.
```

Python3

```
# Python program to find if line
# passing through two coordinates
# also passes through origin or not

def checkOrigin(x1, y1, x2, y2):
    return (x1 * (y2 - y1) == y1 * (x2 - x1))

# Driver code
if (checkOrigin(1, 28, 2, 56) == True):
    print("Yes")
else:
    print("No")

# This code is contributed
# by Anant Agarwal.
```

C#

```
// C# program to find if line passing through
// two coordinates also passes through origin
// or not
using System;

class GFG {

    static bool checkOrigin(int x1, int y1,
                          int x2, int y2)
    {
        return (x1 * (y2 - y1) == y1 * (x2 - x1));
    }
}
```

```
// Driver code
public static void Main()
{
    if (checkOrigin(1, 28, 2, 56) == true)
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find if
// line passing through
// two coordinates also
// passes through origin
// or not

function checkOrigin($x1, $y1,
                     $x2, $y2)
{
    return ($x1 * ($y2 - $y1) ==
            $y1 * ($x2 - $x1));
}

// Driver code
if (checkOrigin(1, 28, 2, 56) == true)
    echo("Yes");
else
    echo("No");

// This code is contributed by Ajit.
?>
```

Output:

Yes

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/check-line-passes-origin/>

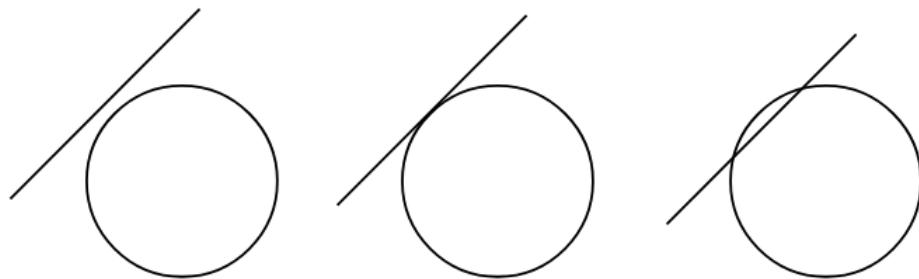
Chapter 27

Check if a line touches or intersects a circle

Check if a line touches or intersects a circle - GeeksforGeeks

Given coordinate of the center and radius > 1 of a circle and the equation of a line. The task is to check if the given line collide with the circle or not. There are three possibilities :

1. Line intersect the circle.
2. Line touches the circle.
3. Line is outside the circle.



Line outside the circle

Line touches the circle

Line through the circle

Note: General equation of a line is $a*x + b*y + c = 0$, so only constant a , b , c are given in the input.

Examples :

```
Input : radius = 5, center = (0, 0),
       a = 1, b = -1, c = 0.
```

Output : Intersect

```
Input : radius = 5, center = (0, 0),
       a = 5, b = 0, c = 0.
```

Output : Touch

```
Input : radius = 5, center = (0, 0),
       a = 1, b = 1, c = -16.
```

Output : Outside

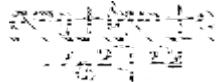
The idea is to compare the perpendicular distance between center of circle and line with the radius of the circle.

Algorithm:

1. Find the perpendicular (say p) between center of circle and given line.
2. Compare this distance p with radius r.
 -a) If $p > r$, then line lie outside the circle.
 -b) If $p = r$, then line touches the circle.
 -c) If $p < r$, then line intersect the circle.

How to find the perpendicular distance ?

Distance of a line from a point can be computed using below formula:



Refer [Wiki](#) for details of above formula.

C++

```
// CPP program to check if a line touches or
// intersects or outside a circle.
#include <bits/stdc++.h>
using namespace std;

void checkCollision(int a, int b, int c,
                    int x, int y, int radius)
{
    // Finding the distance of line from center.
    int dist = (abs(a * x + b * y + c)) /
               sqrt(a * a + b * b);

    // Checking if the distance is less than,
    // greater than or equal to radius.
    if (radius == dist)
        cout << "Touch" << endl;
```

```
    else if (radius > dist)
        cout << "Intersect" << endl;
    else
        cout << "Outside" << endl;
}

// Driven Program
int main()
{
    int radius = 5;
    int x = 0, y = 0;
    int a = 3, b = 4, c = 25;
    checkCollision(a, b, c, x, y, radius);
    return 0;
}
```

Java

```
// Java program to check if a line touches or
// intersects or outside a circle.

import java.io.*;

class GFG {

    static void checkCollision(int a, int b, int c,
                               int x, int y, int radius)
    {
        // Finding the distance of line from center.
        double dist = (Math.abs(a * x + b * y + c)) /
                      Math.sqrt(a * a + b * b);

        // Checking if the distance is less than,
        // greater than or equal to radius.
        if (radius == dist)
            System.out.println( "Touch" );
        else if (radius > dist)
            System.out.println( "Intersect" );
        else
            System.out.println( "Outside" );
    }

    // Driven Program
    public static void main (String[] args)
    {
        int radius = 5;
        int x = 0, y = 0;
        int a = 3, b = 4, c = 25;
```

```
    checkCollision(a, b, c, x, y, radius);

}

}

// This article is contributed by vt_m.
```

Python3

```
# python program to check if a line
# touches or intersects or outside
# a circle.

import math

def checkCollision(a, b, c, x, y, radius):

    # Finding the distance of line
    # from center.
    dist = ((abs(a * x + b * y + c)) /
             math.sqrt(a * a + b * b))

    # Checking if the distance is less
    # than, greater than or equal to radius.
    if (radius == dist):
        print("Touch")
    elif (radius > dist):
        print("Intersect")
    else:
        print("Outside")

# Driven Program
radius = 5
x = 0
y = 0
a = 3
b = 4
c = 25
checkCollision(a, b, c, x, y, radius)

# This code is contributed by Sam007
```

C#

```
// C# program to check if a line touches or
// intersects or outside a circle.
using System;
```

```
class GFG {

    static void checkCollision(int a, int b, int c,
                               int x, int y, int radius)
    {
        // Finding the distance of line from center.
        double dist = (Math.Abs(a * x + b * y + c)) /
                      Math.Sqrt(a * a + b * b);

        // Checking if the distance is less than,
        // greater than or equal to radius.
        if (radius == dist)
            Console.WriteLine ("Touch");
        else if (radius > dist)
            Console.WriteLine("Intersect");
        else
            Console.WriteLine("Outside");
    }

    // Driven Program
    public static void Main ()
    {
        int radius = 5;
        int x = 0, y = 0;
        int a = 3, b = 4, c = 25;

        checkCollision(a, b, c, x, y, radius);
    }
}

// This article is contributed by vt_m.
```

PHP

```
<?php
// PHP program to check if a line
// touches or intersects or outside
// a circle.

function checkCollision($a, $b, $c,
                       $x, $y, $radius)
{
    // Finding the distance
    // of line from center.
    $dist = (abs($a * $x + $b * $y + $c)) /
            sqrt($a * $a + $b * $b);
```

```
// Checking if the distance is less than,
// greater than or equal to radius.
if ($radius == $dist)
    echo "Touch";
else if ($radius > $dist)
    echo "Intersect";
else
    echo "Outside" ;
}

// Driver Code
$radius = 5;
$x = 0;
$y = 0;
$a = 3;
$b = 4;
$c = 25;
checkCollision($a, $b, $c, $x, $y, $radius);

// This code is contributed by Sam007
?>
```

Output :

Touch

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/check-line-touches-intersects-circle/>

Chapter 28

Check if a point lies on or inside a rectangle | Set-2

Check if a point lies on or inside a rectangle | Set-2 - GeeksforGeeks

Given coordinates of bottom-left and top-right corners of a rectangle. Check if a point (x, y) lies inside this rectangle or not.

Examples:

Input: bottom-left: (0, 0), top-right: (10, 8), point: (1, 5)

Output: Yes

Input: bottom-left: (-1, 4), top-right: (2, 3), point: (0, 4)

Output: No

This problem is already discussed in a [previous post](#). In this post we have discussed a new approach.

Approach: The above problem can be solved by observation. A point lies inside or not the rectangle if and only if it's x-coordinate lies between the x-coordinate of the given bottom-right and top-left coordinates of the rectangle and y-coordinate lies between the y-coordinate of the given bottom-right and top-left coordinates.

Below is the implementation of the above approach:

C++

```
// CPP program to Check if a
// point lies on or inside a rectangle | Set-2
#include <bits/stdc++.h>
using namespace std;

// function to find if given point
```

```
// lies inside a given rectangle or not.
bool FindPoint(int x1, int y1, int x2,
               int y2, int x, int y)
{
    if (x > x1 and x < x2 and y > y1 and y < y2)
        return true;

    return false;
}

// Driver code
int main()
{
    // bottom-left and top-right
    // corners of rectangle
    int x1 = 0, y1 = 0, x2 = 10, y2 = 8;

    // given point
    int x = 1, y = 5;

    // function call
    if (FindPoint(x1, y1, x2, y2, x, y))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
// Java program to Check if
// a point lies on or inside
// a rectangle | Set-2
class GFG
{
    // function to find if given point
    // lies inside a given rectangle or not.
    static boolean FindPoint(int x1, int y1, int x2,
                           int y2, int x, int y)
    {
        if (x > x1 && x < x2 && y > y1 && y < y2) return true; return false; } // Driver code
    public static void main(String[] args) { // bottom-left and top-right // corners of rectangle
        int x1 = 0, y1 = 0, x2 = 10, y2 = 8; // given point
        int x = 1, y = 5; // function call
        if (FindPoint(x1, y1, x2, y2, x, y)) System.out.println("Yes"); else System.out.println("No");
    } } // This code is contributed // by ChitraNayal [tabby title="Python3"]
```

```
# Python3 program to Check
```

```
# if a point lies on or
# inside a rectangle | Set-2

# function to find if
# given point lies inside
# a given rectangle or not.
def FindPoint(x1, y1, x2,
              y2, x, y) :
    if (x > x1 and x < x2 and
        y > y1 and y < y2) :
        return True
    else :
        return False

# Driver code
if __name__ == "__main__":
    # bottom-left and top-right
    # corners of rectangle.
    # use multiple assignment
    x1, y1, x2, y2 = 0, 0, 10, 8

    # given point
    x, y = 1, 5

    # function call
    if FindPoint(x1, y1, x2,
                 y2, x, y) :
        print("Yes")
    else :
        print("No")

# This code is contributed
# by Ankit Rai
```

C#

```
// C# program to Check if a
// point lies on or inside
// a rectangle | Set-2
using System;

class GFG
{
    // function to find if given
    // point lies inside a given
    // rectangle or not.
    static bool FindPoint(int x1, int y1, int x2,
```

```
int y2, int x, int y)
{
if (x > x1 && x < x2 && y > y1 && y < y2) return true; return false; } // Driver code
public static void Main() { // bottom-left and top-right // corners of rectangle int x1 = 0,
y1 = 0, x2 = 10, y2 = 8; // given point int x = 1, y = 5; // function call if (FindPoint(x1,
y1, x2, y2, x, y)) Console.WriteLine("Yes"); else Console.WriteLine("No"); } } // This code is
contributed // by ChitraNayal [tabby title="PHP"]

<?php
// PHP program to Check if
// a point lies on or inside
// a rectangle | Set-2

// function to find if given
// point lies inside a given
// rectangle or not.
function FindPoint($x1, $y1, $x2,
                  $y2, $x, $y)
{
    if ($x > $x1 and $x < $x2 and
        $y > $y1 and $y < $y2)
        return true;

    return false;
}

// Driver code

// bottom-left and top-right
// corners of rectangle
$x1 = 0; $y1 = 0;
$x2 = 10; $y2 = 8;

// given point
$x = 1; $y = 5;

// function call
if (FindPoint($x1, $y1, $x2,
              $y2, $x, $y))
    echo "Yes";
else
    echo "No";

// This code is contributed
// by Akanksha Rai(Addy_akku)
?>
```

Output:

Yes

Improved By : ANKITRAI1, Abby_akku, ChitraNayal

Source

<https://www.geeksforgeeks.org/check-if-a-point-lies-on-or-inside-a-rectangle-set-2/>

Chapter 29

Check if four segments form a rectangle

Check if four segments form a rectangle - GeeksforGeeks

We are given four segments as a pair of coordinates of their end points. We need to tell whether those four line segments make a rectangle or not.

Examples:

```
Input : segments[] = [(4, 2), (7, 5),
                      (2, 4), (4, 2),
                      (2, 4), (5, 7),
                      (5, 7), (7, 5)]
Output : Yes
Given these segment make a rectangle of length 3X2.
```

```
Input : segment[] = [(7, 0), (10, 0),
                     (7, 0), (7, 3),
                     (7, 3), (10, 2),
                     (10, 2), (10, 0)]
Output : Not
These segments do not make a rectangle.
```

Above examples are shown in below diagram.

This problem is mainly an extension of [How to check if given four points form a square](#)

We can solve this problem by using properties of a rectangle. First, we check total unique end points of segments, if count of these points is not equal to 4 then the line segment can't make a rectangle. Then we check distances between all pair of points, there should

be at most 3 different distances, one for diagonal and two for sides and at the end we will check the relation among these three distances, for line segments to make a rectangle these distance should satisfy Pythagorean relation because sides and diagonal of rectangle makes a right angle triangle. If they satisfy mentioned conditions then we will flag polygon made by line segment as rectangle otherwise not.

```
// C++ program to check whether it is possible
// to make a rectangle from 4 segments
#include <bits/stdc++.h>
using namespace std;
#define N 4

// structure to represent a segment
struct Segment
{
    int ax, ay;
    int bx, by;
};

// Utility method to return square of distance
// between two points
int getDis(pair<int, int> a, pair<int, int> b)
{
    return (a.first - b.first)*(a.first - b.first) +
           (a.second - b.second)*(a.second - b.second);
}

// method returns true if line Segments make
// a rectangle
bool isPossibleRectangle(Segment segments[])
{
    set< pair<int, int> > st;

    // putting all end points in a set to
    // count total unique points
    for (int i = 0; i < N; i++)
    {
        st.insert(make_pair(segments[i].ax, segments[i].ay));
        st.insert(make_pair(segments[i].bx, segments[i].by));
    }

    // If total unique points are not 4, then
    // they can't make a rectangle
    if (st.size() != 4)
        return false;

    // dist will store unique 'square of distances'
    set<int> dist;
```

```
// calculating distance between all pair of
// end points of line segments
for (auto it1=st.begin(); it1!=st.end(); it1++)
    for (auto it2=st.begin(); it2!=st.end(); it2++)
        if (*it1 != *it2)
            dist.insert(getDis(*it1, *it2));

// if total unique distance are more than 3,
// then line segment can't make a rectangle
if (dist.size() > 3)
    return false;

// copying distance into array. Note that set maintains
// sorted order.
int distance[3];
int i = 0;
for (auto it = dist.begin(); it != dist.end(); it++)
    distance[i++] = *it;

// If line segments form a square
if (dist.size() == 2)
    return (2*distance[0] == distance[1]);

// distance of sides should satisfy pythagorean
// theorem
return (distance[0] + distance[1] == distance[2]);
}

// Driver code to test above methods
int main()
{
    Segment segments[] =
    {
        {4, 2, 7, 5},
        {2, 4, 4, 2},
        {2, 4, 5, 7},
        {5, 7, 7, 5}
    };

    (isPossibleRectangle(segments))?cout << "Yes\n":cout << "No\n";
}
```

Output:

Yes

Improved By : [harrypotter0](#)

Source

<https://www.geeksforgeeks.org/check-four-segments-form-rectangle/>

Chapter 30

Check if given four integers (or sides) make rectangle

Check if given four integers (or sides) make rectangle - GeeksforGeeks

Given four positive integers, determine if there's a rectangle such that the lengths of its sides are a, b, c and d (in any order).

Examples :

Input : 1 1 2 2
Output : Yes

Input : 1 2 3 4
Output : No

Approach 1 :- We will check, if any of the two integers are equal and make sure rest of two are also equal using few if else conditions.

C++

```
// A simple program to find if given 4
// values can represent 4 sides of rectangle
#include <iostream>
using namespace std;

// Function to check if the given
// integers value make a rectangle
bool isRectangle(int a, int b, int c, int d)
{
    // Square is also a rectangle
```

```
if (a == b == c == d)
    return true;

else if (a == b && c == d)
    return true;
else if (a == d && c == b)
    return true;
else if (a == c && d == b)
    return true;
else
    return false;
}

// Driver code
int main()
{
    int a, b, c, d;
    a = 1, b = 2, c = 3, d = 4;
    if (isRectangle(a, b, c, d))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
// A simple program to find if
// given 4 values can represent
// 4 sides of rectangle
class GFG {

    // Function to check if the given
    // integers value make a rectangle
    static boolean isRectangle(int a, int b,
                               int c, int d)
    {
        // Square is also a rectangle
        if (a == b && a == c &&
            a == d && c == d &&
            b == c && b == d)
            return true;

        else if (a == b && c == d)
            return true;
        else if (a == d && c == b)
            return true;
        else if (a == c && d == b)
```

```
        return true;
    else
        return false;
}

// Driver code
public static void main(String[] args)
{
    int a = 1, b = 2, c = 3, d = 4;
    if (isRectangle(a, b, c, d))
        System.out.println("Yes");
    else
        System.out.println("No");
}
}

// This code is contributed by prerna saini.
```

Python3

```
# A simple program to find if given 4
# values can represent 4 sides of rectangle

# Function to check if the given
# integers value make a rectangle
def isRectangle(a, b, c, d):

    # Square is also a rectangle
    if a == b == c == d:
        return True

    elif a == b and c == d:
        return True

    elif a == d and c == b:
        return True

    elif a == c and d == b:
        return True

    return False

# Driver code
a, b, c, d = 1, 2, 3, 4
print("Yes" if isRectangle(a, b, c, d) else "No")
```

```
# This code is contributed by Ansu Kumari.
```

C#

```
// A simple program to find if
// given 4 values can represent
// 4 sides of rectangle
using System;

class GFG {

    // Function to check if the given
    // integers value make a rectangle
    static bool isRectangle(int a, int b,
                           int c, int d)
    {
        // Square is also a rectangle
        if (a == b && a == c && a == d &&
            c == d && b == c && b == d)
            return true;

        else if (a == b && c == d)
            return true;

        else if (a == d && c == b)
            return true;

        else if (a == c && d == b)
            return true;

        else
            return false;
    }

    // Driver code
    public static void Main()
    {

        int a = 1, b = 2, c = 3, d = 4;
        if (isRectangle(a, b, c, d))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
}

// This code is contributed by vt_m.
```

Output :

No

Approach 2 :- XORing the given lengths, will give value 0.

C++

```
// An efficient program to find if given 4
// values can represent 4 sides of rectangle
#include <iostream>
using namespace std;

// Function to check if the given
// integers value make a rectangle
bool isRectangle(int a, int b, int c, int d)
{
    if (a ^ b ^ c ^ d)
        return false;
    else
        return true;
}

// Driver code
int main()
{
    int a, b, c, d;
    a = 3, b = 2, c = 3, d = 2;
    if (isRectangle(a, b, c, d))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
// An efficient Java program to find if given 4
// values can represent 4 sides of rectangle

class GFG
{
    // Function to check if the given
    // integers value make a rectangle
    static boolean isRectangle(int a, int b,
```

```
        int c, int d)
{
    if ((a ^ b ^ c ^ d) != 0)
        return false;
    else
        return true;
}

// Driver code
public static void main(String[] args)
{
    int a, b, c, d;
    a = 3; b = 2; c = 3; d = 2;
    if (isRectangle(a, b, c, d))
        System.out.println("Yes");
    else
        System.out.println("No");

}
}

// This code is contributed by
// Smitha Dinesh Semwal
```

Python3

```
# An efficient program to find if given 4
# values can represent 4 sides of rectangle

# Function to check if the given
# integers value make a rectangle
def isRectangle(a, b, c, d):

    if a ^ b ^ c ^ d:
        return False

    return True

# Driver code
a, b, c, d = 3, 2, 3, 2
print("Yes"    if isRectangle(a, b, c, d) else "No")

# This code is contributed by Ansu Kumari.
```

C#

```
// An efficient C# program to find
// if given 4 values can represent
// 4 sides of rectangle
using System;

class GFG {

    // Function to check if the given
    // integers value make a rectangle
    static bool isRectangle(int a, int b,
                           int c, int d)
    {
        if ((a ^ b ^ c ^ d) != 0)
            return false;
        else
            return true;
    }

    // Driver code
    public static void Main()
    {
        int a, b, c, d;
        a = 3;
        b = 2;
        c = 3;
        d = 2;

        if (isRectangle(a, b, c, d))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
}

// This code is contributed by vt_m
```

Output :

Yes

Improved By : Aditya raj 9

Source

<https://www.geeksforgeeks.org/check-given-four-integers-sides-make-rectangle/>

Chapter 31

Check if it is possible to create a polygon with a given angle

Check if it is possible to create a polygon with a given angle - GeeksforGeeks

Given an angle a where, $1 \leq a \leq 180$. The task is to check whether it is possible to make a regular polygon with all of its interior angle equal to a . If possible then print "YES", otherwise print "NO" (without quotes).

Examples:

Input: angle = 90

Output: YES

Polygons with sides 4 is possible with angle 90 degrees.

Input: angle = 30

Output: NO

Approach: The Interior angle is defined as the angle between any two adjacent sides of a regular polygon.

It is given by $\text{Interior angle} = \frac{\text{angle} = \frac{(n-2) \times 180}{n}}$ where, n is the number of sides in the polygon.

This can be written as $\frac{180(n-2)}{n}$.

On rearranging terms we get,

Thus, if n is an Integer the answer is "YES" otherwise, answer is "NO".

Below is the implementation of the above approach:

C++

```
// C++ implementation of above approach
#include <bits/stdc++.h>
using namespace std;

// Function to check whether it is possible
// to make a regular polygon with a given angle.
void makePolygon(float a)
{
    // N denotes the number of sides
    // of polygons possible
    float n = 360 / (180 - a);
    if (n == (int)n)
        cout << "YES";
    else
        cout << "NO";
}

// Driver code
int main()
{
    float a = 90;

    // function to print the required answer
    makePolygon(a);

    return 0;
}
```

Java

```
class GFG
{
// Function to check whether
// it is possible to make a
// regular polygon with a given angle.
static void makePolygon(double a)
{
    // N denotes the number of
    // sides of polygons possible
    double n = 360 / (180 - a);
    if (n == (int)n)
        System.out.println("YES");
    else
        System.out.println("NO");
}
```

```
// Driver code
public static void main (String[] args)
{
    double a = 90;

    // function to print
    // the required answer
    makePolygon(a);
}
}

// This code is contributed by Bilal
```

Python3

```
# Python 3 implementation
# of above approach

# Function to check whether
# it is possible to make a
# regular polygon with a
# given angle.
def makePolygon(a) :

    # N denotes the number of sides
    # of polygons possible
    n = 360 / (180 - a)

    if n == int(n) :
        print("YES")

    else :
        print("NO")

# Driver Code
if __name__ == "__main__":
    a = 90

    # function calling
    makePolygon(a)

# This code is contributed
# by ANKITRAI1
```

C#

```
// C# implementation of
```

```
// above approach
using System;

class GFG
{
// Function to check whether
// it is possible to make a
// regular polygon with a
// given angle.
static void makePolygon(double a)
{
    // N denotes the number of
    // sides of polygons possible
    double n = 360 / (180 - a);
    if (n == (int)n)
        Console.WriteLine("YES");
    else
        Console.WriteLine("NO");
}

// Driver code
static void Main()
{
    double a = 90;

    // function to print
    // the required answer
    makePolygon(a);
}
}

// This code is contributed by mits
```

PHP

Output:

YES

Improved By : [bilal-hungund](#), [Mithun Kumar](#), [ANKITRAI1](#), [ChitraNayal](#)

Source

<https://www.geeksforgeeks.org/check-if-it-is-possible-to-create-a-polygon-with-a-given-angle/>

Chapter 32

Check if right triangle possible from given area and hypotenuse

Check if right triangle possible from given area and hypotenuse - GeeksforGeeks

Given area and hypotenuse, the aim is to print all sides if right triangle can exist, else print -1. We need to print all sides in ascending order.

Examples:

```
Input : 6 5
Output : 3 4 5
```

```
Input : 10 6
Output : -1
```

We have discussed a solution of this problem in below post.

[Find all sides of a right angled triangle from given hypotenuse and area | Set 1](#)

In this post, a new solution with below logic is discussed.

Let the two unknown sides be a and b

Area : $A = 0.5 * a * b$

Hypotenuse Square : $H^2 = a^2 + b^2$

Substituting b, we get $H^2 = a^2 + (4 * A^2)/a^2$

On re-arranging, we get the equation $a^4 - (H^2)(a^2) + 4*(A^2)$

The discriminant D of this equation would be $D = H^4 - 16*(A^2)$

If $D = 0$, then roots are given by the linear equation formula, $\text{roots} = (-b \pm \sqrt{D})/2*a$
these roots would be equal to the square of the sides, finding the square roots would give us the sides.

C++

```
// C++ program to check existence of
// right triangle.
#include <bits/stdc++.h>
using namespace std;

// Prints three sides of a right triangle
// from given area and hypotenuse if triangle
// is possible, else prints -1.
void findRightAngle(int A, int H)
{
    // Descriminant of the equation
    long D = pow(H, 4) - 16 * A * A;

    if (D >= 0)
    {
        // applying the linear equation
        // formula to find both the roots
        long root1 = (H * H + sqrt(D)) / 2;
        long root2 = (H * H - sqrt(D)) / 2;

        long a = sqrt(root1);
        long b = sqrt(root2);

        if (b >= a)
            cout << a << " " << b << " " << H;
        else
            cout << b << " " << a << " " << H;
    }
    else
        cout << "-1";
}

// Driver code
int main()
{
    findRightAngle(6, 5);
}

// This code is contributed By Anant Agarwal.
```

Java

```
// Java program to check existence of
// right triangle.

class GFG {
```

```
// Prints three sides of a right trianlge
// from given area and hypotenuse if triangle
// is possible, else prints -1.
static void findRightAngle(double A, double H)
{
    // Descriminant of the equation
    double D = Math.pow(H, 4) - 16 * A * A;

    if (D >= 0)
    {
        // applying the linear equation
        // formula to find both the roots
        double root1 = (H * H + Math.sqrt(D)) / 2;
        double root2 = (H * H - Math.sqrt(D)) / 2;

        double a = Math.sqrt(root1);
        double b = Math.sqrt(root2);
        if (b >= a)
            System.out.print(a + " " + b + " " + H);
        else
            System.out.print(b + " " + a + " " + H);
    }
    else
        System.out.print("-1");
}

// Driver code
public static void main(String arg[])
{
    findRightAngle(6, 5);
}
}

// This code is contributed by Anant Agarwal.
```

Python

```
# Python program to check existence of
# right triangle.
from math import sqrt

# Prints three sides of a right trianlge
# from given area and hypotenuse if triangle
# is possible, else prints -1.
def findRightAngle(A, H):

    # Descriminant of the equation
    D = pow(H,4) - 16 * A * A
```

```
if D >= 0:

    # applying the linear equation
    # formula to find both the roots
    root1 = (H * H + sqrt(D))/2
    root2 = (H * H - sqrt(D))/2

    a = sqrt(root1)
    b = sqrt(root2)
    if b >= a:
        print a, b, H
    else:
        print b, a, H
    else:
        print "-1"

# Driver code
# Area is 6 and hypotenuse is 5.
findRightAngle(6, 5)
```

C#

```
// C# program to check existence of
// right triangle.
using System;

class GFG {

    // Prints three sides of a right trianlge
    // from given area and hypotenuse if triangle
    // is possible, else prints -1.
    static void findRightAngle(double A, double H)
    {

        // Descriminant of the equation
        double D = Math.Pow(H, 4) - 16 * A * A;

        if (D >= 0) {

            // applying the linear equation
            // formula to find both the roots
            double root1 = (H * H + Math.Sqrt(D)) / 2;
            double root2 = (H * H - Math.Sqrt(D)) / 2;

            double a = Math.Sqrt(root1);
            double b = Math.Sqrt(root2);

            if (b >= a)
```

```
        Console.WriteLine(a + " " + b + " " + H);
    else
        Console.WriteLine(b + " " + a + " " + H);
    }
else
    Console.WriteLine("-1");
}

// Driver code
public static void Main()
{
    findRightAngle(6, 5);
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to check existence of
// right triangle.

// Prints three sides of a right triangle
// from given area and hypotenuse if
// triangle is possible, else prints -1.
function findRightAngle($A, $H)
{

    // Descriminant of the equation
    $D = pow($H, 4) - 16 * $A * $A;

    if ($D >= 0)
    {

        // applying the linear equation
        // formula to find both the roots
        $root1 = ($H * $H + sqrt($D)) / 2;
        $root2 = ($H * $H - sqrt($D)) / 2;

        $a = sqrt($root1);
        $b = sqrt($root2);

        if ($b >= $a)
            echo $a , " ", $b , " " , $H;
        else
            echo $b , " " , $a , " " , $H;
    }
}
```

```
    else
        echo "-1";
}

// Driver code
findRightAngle(6, 5);

// This code is contributed By Anuj_67
?>
```

Output:

3 4 5

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/check-right-angles-possible-given-area-hypotenuse/>

Chapter 33

Check if three straight lines are concurrent or not

Check if three straight lines are concurrent or not - GeeksforGeeks

Given three lines equation,

$$a_1x + b_1y + c_1 = 0$$

$$a_2x + b_2y + c_2 = 0$$

$$a_3x + b_3y + c_3 = 0$$

The task is to check whether the given three lines are concurrent or not. Three straight lines are said to be concurrent if they pass through a point i.e., they meet at a point.

Examples:

```
Input : a1 = 2, b1 = -3, c1 = 5  
        a2 = 3, b2 = 4, c2 = -7  
        a3 = 9, b3 = -5, c3 = 8
```

Output : Yes

```
Input : a1 = 2, b1 = -3, c1 = 5  
        a2 = 3, b2 = 4, c2 = -7  
        a3 = 9, b3 = -5, c3 = 4
```

Output : No

Let

$$a_1x + b_1y + c_1 = 0 \dots\dots\dots (1)$$

$$a_2x + b_2y + c_2 = 0 \dots\dots\dots (2)$$

$$a_3x + b_3y + c_3 = 0 \dots\dots\dots (3)$$

Suppose the eqn (i) and (ii) intersects at (x_1, y_1) . Then (x_1, y_1) will satisfy both the equations. Therefore, solving (i) and (ii) using [method of cross-multiplication](#), we get,
 $(x_1/b_1c_2 - b_2c_1) = (y_1/c_1a_2 - c_2a_1) = (1/a_1b_2 - a_2b_1)$

Therefore,

$$x_1 = (b_1c_2 - b_2c_1/a_1b_2 - a_2b_1) \text{ and}$$

$$y_1 = (c_1a_2 - c_2a_1/a_1b_2 - a_2b_1), a_1b_2 - a_2b_1 \neq 0$$

Therefore, the required coordinates of the point of intersection of the lines (i) and (ii) are $(b_1c_2 - b_2c_1/a_1b_2 - a_2b_1, c_1a_2 - c_2a_1/a_1b_2 - a_2b_1)$

For, three of line to be concurrent, (x_1, y_1) must satisfy the equation (iii) as well.

So,

$$a_3x + b_3y + c_3 = 0$$

$$\Rightarrow a_3(b_1c_2 - b_2c_1/a_1b_2 - a_2b_1) + b_3(c_1a_2 - c_2a_1/a_1b_2 - a_2b_1) + c_3 = 0$$

$$\Rightarrow a_3(b_1c_2 - b_2c_1) + b_3(c_1a_2 - c_2a_1) + c_3(a_1b_2 - a_2b_1) = 0$$

So, we only need to check if above condition satisfy or not.

Below is the implemenatation of this approach:

C++

```
// CPP Program to check if three straight
// line are concurrent or not
#include <bits/stdc++.h>
using namespace std;

// Return true if three line are concurrent,
// else false.
bool checkConcurrent(int a1, int b1, int c1,
                      int a2, int b2, int c2,
                      int a3, int b3, int c3)
{
    return (a3 * (b1 * c2 - b2 * c1) +
            b3 * (c1 * a2 - c2 * a1) +
            c3 * (a1 * b2 - a2 * b1) == 0);
}

// Driven Program
int main()
{
    int a1 = 2, b1 = -3, c1 = 5;
    int a2 = 3, b2 = 4, c2 = -7;
    int a3 = 9, b3 = -5, c3 = 8;

    (checkConcurrent(a1, b1, c1, a2, b2, c2,
                     a3, b3, c3) ? (cout << "Yes") : (cout << "No"));
    return 0;
}
```

Java

```
// Java Program to check if three straight
// line are concurrent or no
import java.io.*;

class GFG {

    // Return true if three line are concurrent,
    // else false.
    static boolean checkConcurrent(int a1, int b1,
                                    int c1, int a2, int b2, int c2,
                                    int a3, int b3, int c3)
    {
        return (a3 * (b1 * c2 - b2 * c1) +
               b3 * (c1 * a2 - c2 * a1) +
               c3 * (a1 * b2 - a2 * b1) == 0);
    }

    // Driven Program
    public static void main (String[] args)
    {
        int a1 = 2, b1 = -3, c1 = 5;
        int a2 = 3, b2 = 4, c2 = -7;
        int a3 = 9, b3 = -5, c3 = 8;

        if(checkConcurrent(a1, b1, c1, a2, b2,
                           c2, a3, b3, c3))
            System.out.println( "Yes");
        else
            System.out.println( "No");
    }
}

// This code is contributed by anuj_67.
```

Python 3

```
# Python3 Program to check if three straight
# line are concurrent or not

# Return true if three line are concurrent,
# else false.
def checkConcurrent(a1, b1, c1, a2, b2, c2,
                     a3, b3, c3):

    return (a3 * (b1 * c2 - b2 * c1) +
           b3 * (c1 * a2 - c2 * a1) +
           c3 * (a1 * b2 - a2 * b1) == 0)
```

```
# Driven Program
a1 = 2
b1 = -3
c1 = 5
a2 = 3
b2 = 4
c2 = -7
a3 = 9
b3 = -5
c3 = 8

if(checkConcurrent(a1, b1, c1, a2, b2, c2,
                    a3, b3, c3)):
    print("Yes")
else:
    print("No")

# This code is contributed by Smitha

C#
// C# Program to check if three straight
// line are concurrent or no
using System;

class GFG {

    // Return true if three line are concurrent,
    // else false.
    static bool checkConcurrent(int a1, int b1,
                                int c1, int a2, int b2, int c2,
                                int a3, int b3, int c3)
    {
        return (a3 * (b1 * c2 - b2 * c1) +
                b3 * (c1 * a2 - c2 * a1) +
                c3 * (a1 * b2 - a2 * b1) == 0);
    }

    // Driven Program
    public static void Main ()
    {
        int a1 = 2, b1 = -3, c1 = 5;
        int a2 = 3, b2 = 4, c2 = -7;
        int a3 = 9, b3 = -5, c3 = 8;

        if(checkConcurrent(a1, b1, c1, a2, b2,
                           c2, a3, b3, c3))
```

```
        Console.WriteLine( "Yes");
    else
        Console.WriteLine( "No");
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP Program to check if three straight
// line are concurrent or not

// Return true if three line are
// concurrent, else false.
function checkConcurrent($a1, $b1, $c1,
                         $a2, $b2, $c2,
                         $a3, $b3, $c3)
{
    return ($a3 * ($b1 * $c2 - $b2 * $c1) +
            $b3 * ($c1 * $a2 - $c2 * $a1) +
            $c3 * ($a1 * $b2 - $a2 * $b1) == 0);
}

// Driver Code
$a1 = 2; $b1 = -3; $c1 = 5;
$a2 = 3; $b2 = 4; $c2 = -7;
$a3 = 9; $b3 = -5; $c3 = 8;

if(checkConcurrent($a1, $b1, $c1, $a2, $b2,
                   $c2, $a3, $b3, $c3))
    echo "Yes";
else
    echo "No";

// This code is contributed by anuj_67.
?>
```

Output:

Yes

Improved By : [vt_m](#), Smitha Dinesh Semwal

Source

<https://www.geeksforgeeks.org/check-three-straight-lines-concurrent-not/>

Chapter 34

Check if two given circles touch or intersect each other

Check if two given circles touch or intersect each other - GeeksforGeeks

There are two circle A and B with their centers **C1(x₁, y₁)** and **C2(x₂, y₂)** and radius **R₁** and **R₂**. Task is to check both circles A and B touch each other or not.

Examples :

```
Input : C1 = (3, 4)
        C2 = (14, 18)
        R1 = 5, R2 = 8
Output : Circles do not touch each other.
```

```
Input : C1 = (2, 3)
        C2 = (15, 28)
        R1 = 12, R2 = 10
Output : Circles intersect with each other.
```

```
Input : C1 = (-10, 8)
        C2 = (14, -24)
        R1 = 30, R2 = 10
Input : -10 8
        14 -24
        30 10
Output : Circle touch each other.
```

Distance between centers C₁ and C₂ is calculated as
$$C_1C_2 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$
.
There are three condition arises.

1. If $C_1C_2 == R_1 + R_2$
Circle A and B are touch to each other.
2. If $C_1C_2 > R_1 + R_2$
Circle A and B are not touch to each other.
3. If $C_1C_2 < R_1 + R_2$
Circle intersects each other.

C++

```
// C++ program to check if two
// circles touch each other or not.
#include <bits/stdc++.h>
using namespace std;

int circle(int x1, int y1, int x2,
           int y2, int r1, int r2)
{
    int distSq = (x1 - x2) * (x1 - x2) +
                 (y1 - y2) * (y1 - y2);
    int radSumSq = (r1 + r2) * (r1 + r2);
    if (distSq == radSumSq)
        return 1;
    else if (distSq > radSumSq)
        return -1;
    else
        return 0;
}

// Driver code
int main()
{
    int x1 = -10, y1 = 8;
    int x2 = 14, y2 = -24;
    int r1 = 30, r2 = 10;
    int t = circle(x1, y1, x2,
                   y2, r1, r2);
    if (t == 1)
        cout << "Circle touch to"
             << " each other.";
    else if (t < 0)
        cout << "Circle not touch"
             << " to each other.";
    else
        cout << "Circle intersect"
             << " to each other.";
    return 0;
}
```

Java

```
// Java program to check if two
// circles touch each other or not.
import java.io.*;

class GFG
{
    static int circle(int x1, int y1, int x2,
                      int y2, int r1, int r2)
    {
        int distSq = (x1 - x2) * (x1 - x2) +
                    (y1 - y2) * (y1 - y2);
        int radSumSq = (r1 + r2) * (r1 + r2);
        if (distSq == radSumSq)
            return 1;
        else if (distSq > radSumSq)
            return -1;
        else
            return 0;
    }

    // Driver code
    public static void main (String[] args)
    {
        int x1 = -10, y1 = 8;
        int x2 = 14, y2 = -24;
        int r1 = 30, r2 = 10;
        int t = circle(x1, y1, x2,
                       y2, r1, r2);
        if (t == 1)
            System.out.println ("Circle touch to"
                               " each other.");
        else if (t < 0)
            System.out.println ("Circle not touch"
                               " to each other.");
        else
            System.out.println ("Circle intersect"
                               " to each other.");
    }
}

// This article is contributed by vt_m.
```

Python3

```
# Python3 program to
```

```
# check if two circles touch
# each other or not.

def circle(x1, y1, x2, y2, r1, r2):

    distSq = (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2);
    radSumSq = (r1 + r2) * (r1 + r2);
    if (distSq == radSumSq):
        return 1
    elif (distSq > radSumSq):
        return -1
    else:
        return 0

# Driver code
x1 = -10
y1 = 8
x2 = 14
y2 = -24
r1 = 30
r2 = 10

t = circle(x1, y1, x2, y2, r1, r2)
if (t == 1):
    print("Circle touch to each other.")
elif (t < 0):
    print("Circle not touch to each other.")
else:
    print("Circle intersect to each other.")

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
// C# program to check if two
// circles touch each other or not.
using System;

class GFG
{
    static int circle(int x1, int y1, int x2,
                      int y2, int r1, int r2)
    {
        int distSq = (x1 - x2) * (x1 - x2) +
                    (y1 - y2) * (y1 - y2);
        int radSumSq = (r1 + r2) * (r1 + r2);
```

```
        if (distSq == radSumSq)
            return 1;
        else if (distSq > radSumSq)
            return -1;
        else
            return 0;
    }

    // Driver code
    public static void Main ()
    {
        int x1 = -10, y1 = 8;
        int x2 = 14, y2 = -24;
        int r1 = 30, r2 = 10;
        int t = circle(x1, y1, x2,
                        y2, r1, r2);
        if (t == 1)
            Console.WriteLine ( "Circle touch" +
                                " to each other.");
        else if (t < 0)
            Console.WriteLine( "Circle not touch" +
                                " to each other.");
        else
            Console.WriteLine ( "Circle intersect" +
                                " to each other.");
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to check if two
// circles touch each other or not.

function circle($x1, $y1, $x2,
                $y2, $r1, $r2)
{
    $distSq = ($x1 - $x2) * ($x1 - $x2) +
              ($y1 - $y2) * ($y1 - $y2);
    $radSumSq = ($r1 + $r2) * ($r1 + $r2);
    if ($distSq == $radSumSq)
        return 1;
    else if ($distSq > $radSumSq)
        return -1;
    else
```

```
        return 0;
}

// Driver code
$x1 = -10; $y1 = 8;
$x2 = 14; $y2 = -24;
$r1 = 30; $r2 = 10;
$t = circle($x1, $y1, $x2,
            $y2, $r1, $r2);
if ($t == 1)
    echo "Circle touch to each other.";
else if ($t < 0)
    echo "Circle not touch to each other.";
else
    echo "Circle intersect to each other.";

// This code is contributed by vt_m.
?>
```

Output :

Circle touch to each other.

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/check-two-given-circles-touch-intersect/>

Chapter 35

Check whether a given point lies inside a rectangle or not

Check whether a given point lies inside a rectangle or not - GeeksforGeeks

Given four points of a rectangle, and one more point P. Write a function to check whether P lies within the given rectangle or not.

Examples:

```
Input : R = [(10, 10), (10, -10),
             (-10, -10), (-10, 10)]
        P = (0, 0)
Output : yes
Illustration :
```

```
Input : R = [(10, 10), (10, -10),
             (-10, -10), (-10, 10)],
        P = (20, 20)
Output : no
Illustration :
```

Prerequisite: [Check whether a given point lies inside a triangle or not](#)

Approach : Let the coordinates of four corners be A(x₁, y₁), B(x₂, y₂), C(x₃, y₃) and D(x₄, y₄). And coordinates of the given point P be (x, y)

1) Calculate area of the given rectangle, i.e., area of the rectangle ABCD as area of triangle ABC + area of triangle ACD.

Area A = [x₁(y₂ - y₃) + x₂(y₃ - y₁) + x₃(y₁-y₂)]/2 + [x₁(y₄ - y₃) + x₄(y₃ - y₁) + x₃(y₁-y₄)]/2

- 2) Calculate area of the triangle PAB as A1.
- 3) Calculate area of the triangle PBC as A2.
- 4) Calculate area of the triangle PCD as A3.
- 5) Calculate area of the triangle PAD as A4.
- 6) If P lies inside the triangle, then A1 + A2 + A3 + A4 must be equal to A.

C++

```
#include <bits/stdc++.h>
using namespace std;

/* A utility function to calculate area of
triangle formed by (x1, y1), (x2, y2) and
(x3, y3) */
float area(int x1, int y1, int x2, int y2,
           int x3, int y3)
{
    return abs((x1 * (y2 - y3) + x2 * (y3 - y1) +
               x3 * (y1 - y2)) / 2.0);
}

/* A function to check whether point P(x, y)
lies inside the rectangle formed by A(x1, y1),
B(x2, y2), C(x3, y3) and D(x4, y4) */
bool check(int x1, int y1, int x2, int y2, int x3,
           int y3, int x4, int y4, int x, int y)
{
    /* Calculate area of rectangle ABCD */
    float A = area(x1, y1, x2, y2, x3, y3) +
              area(x1, y1, x4, y4, x3, y3);

    /* Calculate area of triangle PAB */
    float A1 = area(x, y, x1, y1, x2, y2);

    /* Calculate area of triangle PBC */
    float A2 = area(x, y, x2, y2, x3, y3);

    /* Calculate area of triangle PCD */
    float A3 = area(x, y, x3, y3, x4, y4);

    /* Calculate area of triangle PAD */
    float A4 = area(x, y, x1, y1, x4, y4);

    /* Check if sum of A1, A2, A3 and A4
       is same as A */
    return (A == A1 + A2 + A3 + A4);
}

/* Driver program to test above function */
```

```
int main()
{
    /* Let us check whether the point P(10, 15)
       lies inside the rectangle formed by A(0, 10),
       B(10, 0) C(0, -10) D(-10, 0) */
    if (check(0, 10, 10, 0, 0, -10, -10, 0, 10, 15))
        cout << "yes";
    else
        cout << "no";
    return 0;
}
```

Java

```
class GFG
{
    /* A utility function to calculate area of
       triangle formed by (x1, y1), (x2, y2) and
       (x3, y3) */
    static float area(int x1, int y1, int x2,
                      int y2, int x3, int y3)
    {
        return (float)Math.abs((x1 * (y2 - y3) +
                               x2 * (y3 - y1) + x3 * (y1 - y2)) / 2.0);
    }

    /* A function to check whether point P(x, y)
       lies inside the rectangle formed by A(x1, y1),
       B(x2, y2), C(x3, y3) and D(x4, y4) */
    static boolean check(int x1, int y1, int x2, int y2,
                        int x3, int y3, int x4, int y4, int x, int y)
    {

        /* Calculate area of rectangle ABCD */
        float A = area(x1, y1, x2, y2, x3, y3) +
                  area(x1, y1, x4, y4, x3, y3);

        /* Calculate area of triangle PAB */
        float A1 = area(x, y, x1, y1, x2, y2);

        /* Calculate area of triangle PBC */
        float A2 = area(x, y, x2, y2, x3, y3);

        /* Calculate area of triangle PCD */
        float A3 = area(x, y, x3, y3, x4, y4);

        /* Calculate area of triangle PAD */
        float A4 = area(x, y, x1, y1, x4, y4);
    }
}
```

```
/* Check if sum of A1, A2, A3 and A4
is same as A */
return (A == A1 + A2 + A3 + A4);
}

// Driver code
public static void main (String[] args)
{

    /* Let us check whether the point P(10, 15)
    lies inside the rectangle formed by A(0, 10),
    B(10, 0) C(0, -10) D(-10, 0) */
    if (check(0, 10, 10, 0, 0, -10, -10, 0, 10, 15))
        System.out.print("yes");
    else
        System.out.print("no");
}
}

// This code is contributed by Anant Agarwal.
```

C#

```
// C# program to Check whether a given
// point lies inside a rectangle or not
using System;

class GFG {

    // A utility function to calculate area
    // of triangle formed by (x1, y1),
    // (x2, y2) and (x3, y3)
    static float area(int x1, int y1, int x2,
                      int y2, int x3, int y3)
    {
        return (float)Math.Abs((x1 * (y2 - y3) +
                               x2 * (y3 - y1) +
                               x3 * (y1 - y2)) / 2.0);
    }

    // A function to check whether point P(x, y)
    // lies inside the rectangle formed by A(x1, y1),
    // B(x2, y2), C(x3, y3) and D(x4, y4)
    static bool check(int x1, int y1, int x2,
                      int y2, int x3, int y3,
                      int x4, int y4, int x, int y)
{
```

```
// Calculate area of rectangle ABCD
float A = area(x1, y1, x2, y2, x3, y3) +
          area(x1, y1, x4, y4, x3, y3);

// Calculate area of triangle PAB
float A1 = area(x, y, x1, y1, x2, y2);

// Calculate area of triangle PBC
float A2 = area(x, y, x2, y2, x3, y3);

// Calculate area of triangle PCD
float A3 = area(x, y, x3, y3, x4, y4);

// Calculate area of triangle PAD
float A4 = area(x, y, x1, y1, x4, y4);

// Check if sum of A1, A2, A3
// and A4 is same as A
return (A == A1 + A2 + A3 + A4);
}

// Driver code
public static void Main ()
{

    // Let us check whether the point
    // P(10, 15) lies inside the rectangle
    // formed by A(0, 10), B(10, 0),
    // C(0, -10), D(-10, 0)
    if (check(0, 10, 10, 0, 0, -10, -10, 0, 10, 15))
        Console.WriteLine("yes");
    else
        Console.WriteLine("no");
}

// This code is contributed by Nitin Mittal.
```

PHP

```
<?php
// PHP program to check whether a
// given point lies inside a
// rectangle or not

// A utility function to
// calculate area of
```

```
// triangle formed by
// (x1, y1), (x2, y2)
// and (x3, y3)
function area($x1, $y1, $x2,
             $y2, $x3, $y3)
{
    return abs(($x1 * ($y2 - $y3) +
                $x2 * ($y3 - $y1) +
                $x3 * ($y1 - $y2)) / 2.0);
}

/* A function to check
whether point P(x, y)
lies inside the rectangle
formed by A(x1, y1),
B(x2, y2), C(x3, y3)
and D(x4, y4) */
function check($x1, $y1, $x2, $y2, $x3,
               $y3, $x4, $y4, $x, $y)
{
    // Calculate area of rectangle ABCD
    $A = area($x1, $y1, $x2, $y2, $x3, $y3) +
        area($x1, $y1, $x4, $y4, $x3, $y3);

    // Calculate area of triangle PAB
    $A1 = area($x, $y, $x1, $y1, $x2, $y2);

    // Calculate area of triangle PBC
    $A2 = area($x, $y, $x2, $y2, $x3, $y3);

    // Calculate area of triangle PCD
    $A3 = area($x, $y, $x3, $y3, $x4, $y4);

    // Calculate area of triangle PAD
    $A4 = area($x, $y, $x1, $y1, $x4, $y4);

    // Check if sum of A1, A2,
    // A3 and A4 is same as A
    return ($A == $A1 + $A2 + $A3 + $A4);
}

// Driver Code

// Let us check whether
// the point P(10, 15)
// lies inside the rectangle
// formed by A(0, 10),
// B(10, 0) C(0, -10)
```

```
// D(-10, 0)
if (check(0, 10, 10, 0, 0, -10,
           -10, 0, 10, 15))
    echo "yes";
else
    echo "no";

// This code is contributed by vt_m.
?>
```

Output:

no

Improved By : [nitin mittal](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/check-whether-given-point-lies-inside-rectangle-not/>

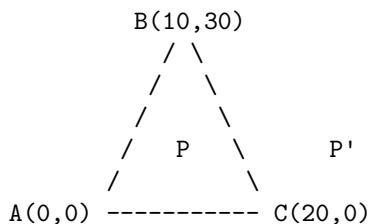
Chapter 36

Check whether a given point lies inside a triangle or not

Check whether a given point lies inside a triangle or not - GeeksforGeeks

Given three corner points of a triangle, and one more point P. Write a function to check whether P lies within the triangle or not.

For example, consider the following program, the function should return true for P(10, 15) and false for P'(30, 15)



Solution:

Let the coordinates of three corners be (x_1, y_1) , (x_2, y_2) and (x_3, y_3) . And coordinates of the given point P be (x, y)

- 1) Calculate area of the given triangle, i.e., area of the triangle ABC in the above diagram.
Area $A = [x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)]/2$
- 2) Calculate area of the triangle PAB. We can use the same formula for this. Let this area be A_1 .
- 3) Calculate area of the triangle PBC. Let this area be A_2 .
- 4) Calculate area of the triangle PAC. Let this area be A_3 .
- 5) If P lies inside the triangle, then $A_1 + A_2 + A_3$ must be equal to A.

C++

```
#include <bits/stdc++.h>
using namespace std;

/* A utility function to calculate area of triangle formed by (x1, y1),
   (x2, y2) and (x3, y3) */
float area(int x1, int y1, int x2, int y2, int x3, int y3)
{
    return abs((x1*(y2-y3) + x2*(y3-y1)+ x3*(y1-y2))/2.0);
}

/* A function to check whether point P(x, y) lies inside the triangle formed
   by A(x1, y1), B(x2, y2) and C(x3, y3) */
bool isInside(int x1, int y1, int x2, int y2, int x3, int y3, int x, int y)
{
    /* Calculate area of triangle ABC */
    float A = area (x1, y1, x2, x3, y3);

    /* Calculate area of triangle PBC */
    float A1 = area (x, y, x2, y3);

    /* Calculate area of triangle PAC */
    float A2 = area (x1, y1, x, y3, y3);

    /* Calculate area of triangle PAB */
    float A3 = area (x1, y1, x2, y2, x, y);

    /* Check if sum of A1, A2 and A3 is same as A */
    return (A == A1 + A2 + A3);
}

/* Driver program to test above function */
int main()
{
    /* Let us check whether the point P(10, 15) lies inside the triangle
       formed by A(0, 0), B(20, 0) and C(10, 30) */
    if (isInside(0, 0, 20, 0, 10, 30, 10, 15))
        printf ("Inside");
    else
        printf ("Not Inside");

    return 0;
}
```

Java

```
// JAVA Code for Check whether a given point
// lies inside a triangle or not
import java.util.*;
```

```
class GFG {

    /* A utility function to calculate area of triangle
       formed by (x1, y1) (x2, y2) and (x3, y3) */
    static double area(int x1, int y1, int x2, int y2,
                       int x3, int y3)
    {
        return Math.abs((x1*(y2-y3) + x2*(y3-y1)+
                        x3*(y1-y2))/2.0);
    }

    /* A function to check whether point P(x, y) lies
       inside the triangle formed by A(x1, y1),
       B(x2, y2) and C(x3, y3) */
    static boolean isInside(int x1, int y1, int x2,
                           int y2, int x3, int y3, int x, int y)
    {
        /* Calculate area of triangle ABC */
        double A = area (x1, y1, x2, y2, x3, y3);

        /* Calculate area of triangle PBC */
        double A1 = area (x, y, x2, y2, x3, y3);

        /* Calculate area of triangle PAC */
        double A2 = area (x1, y1, x, y, x3, y3);

        /* Calculate area of triangle PAB */
        double A3 = area (x1, y1, x2, y2, x, y);

        /* Check if sum of A1, A2 and A3 is same as A */
        return (A == A1 + A2 + A3);
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        /* Let us check whether the point P(10, 15)
           lies inside the triangle formed by
           A(0, 0), B(20, 0) and C(10, 30) */
        if (isInside(0, 0, 20, 0, 10, 30, 10, 15))
            System.out.println("Inside");
        else
            System.out.println("Not Inside");

    }
}
```

```
// This code is contributed by Arnav Kr. Mandal.
```

Python

```
# A utility function to calculate area
# of triangle formed by (x1, y1),
# (x2, y2) and (x3, y3)

def area(x1, y1, x2, y2, x3, y3):

    return abs((x1 * (y2 - y3) + x2 * (y3 - y1)
               + x3 * (y1 - y2)) / 2.0)

# A function to check whether point P(x, y)
# lies inside the triangle formed by
# A(x1, y1), B(x2, y2) and C(x3, y3)
def isInside(x1, y1, x2, y2, x3, y3, x, y):

    # Calculate area of triangle ABC
    A = area (x1, y1, x2, y2, x3, y3)

    # Calculate area of triangle PBC
    A1 = area (x, y, x2, y2, x3, y3)

    # Calculate area of triangle PAC
    A2 = area (x1, y1, x, y, x3, y3)

    # Calculate area of triangle PAB
    A3 = area (x1, y1, x2, y2, x, y)

    # Check if sum of A1, A2 and A3
    # is same as A
    if(A == A1 + A2 + A3):
        return True
    else:
        return False

# Driver program to test above function
# Let us check whether the point P(10, 15)
# lies inside the triangle formed by
# A(0, 0), B(20, 0) and C(10, 30)
if (isInside(0, 0, 20, 0, 10, 30, 10, 15)):
    print('Inside')
else:
    print('Not Inside')

# This code is contributed by Danish Raza
```

C#

```
// C# Code to Check whether a given point
// lies inside a triangle or not
using System;

class GFG {

    /* A utility function to calculate area of triangle
    formed by (x1, y1) (x2, y2) and (x3, y3) */
    static double area(int x1, int y1, int x2,
                       int y2, int x3, int y3)
    {
        return Math.Abs((x1 * (y2 - y3) +
                         x2 * (y3 - y1) +
                         x3 * (y1 - y2)) / 2.0);
    }

    /* A function to check whether point P(x, y) lies
    inside the triangle formed by A(x1, y1),
    B(x2, y2) and C(x3, y3) */
    static bool isInside(int x1, int y1, int x2,
                        int y2, int x3, int y3,
                        int x, int y)
    {
        /* Calculate area of triangle ABC */
        double A = area(x1, y1, x2, y2, x3, y3);

        /* Calculate area of triangle PBC */
        double A1 = area(x, y, x2, y2, x3, y3);

        /* Calculate area of triangle PAC */
        double A2 = area(x1, y1, x, y, x3, y3);

        /* Calculate area of triangle PAB */
        double A3 = area(x1, y1, x2, y2, x, y);

        /* Check if sum of A1, A2 and A3 is same as A */
        return (A == A1 + A2 + A3);
    }

    /* Driver program to test above function */
    public static void Main()
    {
        /* Let us check whether the point P(10, 15)
        lies inside the triangle formed by
        A(0, 0), B(20, 0) and C(10, 30) */
        if (isInside(0, 0, 20, 0, 10, 30, 10, 15))
```

```
        Console.WriteLine("Inside");
    else
        Console.WriteLine("Not Inside");
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php

/* A utility function to calculate
area of triangle formed by (x1, y1),
(x2, y2) and (x3, y3) */
function area($x1, $y1, $x2,
              $y2, $x3, $y3)
{
    return abs(($x1 * ($y2 - $y3) +
                $x2 * ($y3 - $y1) +
                $x3 * ($y1 - $y2)) / 2.0);
}

/* A function to check whether
P(x, y) lies inside the
triangle formed by A(x1, y1),
B(x2, y2) and C(x3, y3) */
function isInside($x1, $y1, $x2, $y2,
                  $x3, $y3, $x, $y)
{

    /* Calculate area of triangle ABC */
    $A = area ($x1, $y1, $x2, $y2, $x3, $y3);

    /* Calculate area of triangle PBC */
    $A1 = area ($x, $y, $x2, $y2, $x3, $y3);

    /* Calculate area of triangle PAC */
    $A2 = area ($x1, $y1, $x, $y, $x3, $y3);

    /* Calculate area of triangle PAB */
    $A3 = area ($x1, $y1, $x2, $y2, $x, $y);

    /* Check if sum of A1, A2
and A3 is same as A */
    return ($A == $A1 + $A2 + $A3);
}
```

```
// Driver Code
/* Let us check whether the
P(10, 15) lies inside the
triangle formed by A(0, 0),
B(20, 0) and C(10, 30) */
if (isInside(0, 0, 20, 0, 10, 30, 10, 15))
    echo "Inside";
else
    echo "Not Inside";

// This code is contributed by anuj_67.
?>
```

Ouptut:

Inside

Exercise: Given coordinates of four corners of a rectangle, and a point P. Write a function to check whether P lies inside the given rectangle or not.

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/check-whether-a-given-point-lies-inside-a-triangle-or-not/>

Chapter 37

Check whether a given point lies on or inside the rectangle | Set 3

Check whether a given point lies on or inside the rectangle | Set 3 - GeeksforGeeks

Given two numbers a and b where $b < a$ form a rectangle with points $(0, b)$, $(b, 0)$, $(a-b, b)$, $(b, a-b)$. Given a point (x, y) , the task is to check whether this point lies inside or on the rectangle or not.

Examples:

Input: $a = 7$, $b = 2$, $x = 5$, $y = 2$;

Output: Given point does not lie on the rectangle

Input: $a = 7$, $b = 2$, $x = 4$, $y = 5$;

Output: Given point lies inside the rectangle

Approach: An efficient way is to form 4 line's equation of a rectangle. Then, if a given point lies in or on the rectangle if and only if, substituting the given point on:

1. The right sideline ($x-y-b = 0$) must give the value less than or equals to zero.
2. The left sideline ($x-y+a = 0$) must give the value greater than or equals to one.
3. The upper sideline ($x+y-2*a+b = 0$) must give the value less than or equals to zero.
4. The lower sideline ($x+y-b = 0$) must give the value greater than or equals to one.

C++

```
// C++ program to Check whether a given point
// lies inside or on the rectangle or not
#include <bits/stdc++.h>
```

```
using namespace std;

// function to Check whether a given point
// lies inside or on the rectangle or not
bool LiesInsideRectangle(int a, int b, int x, int y)
{
    if (x - y - b <= 0 && x - y + b >= 0
        && x + y - 2 * a + b <= 0 && x + y - b >= 0)
        return true;

    return false;
}

// Driver code
int main()
{
    int a = 7, b = 2, x = 4, y = 5;

    if (LiesInsideRectangle(a, b, x, y))
        cout << "Given point lies inside the rectangle";
    else
        cout << "Given point does not lie on the rectangle";

    return 0;
}
```

Java

```
// Java program to Check whether
// a given point lies inside or
// on the rectangle or not
class GFG
{

    // function to Check whether
    // a given point lies inside
    // or on the rectangle or not
    static boolean LiesInsideRectangle(int a, int b,
    int x, int y)
    {
        if (x - y - b <= 0 && x - y + b >= 0 &&
            x + y - 2 * a + b <= 0 && x + y - b >= 0)
            return true;

        return false;
    }

    // Driver code
    public static void main(String[] args)
```

```
{  
int a = 7, b = 2, x = 4, y = 5;  
if (LiesInsideRectangle(a, b, x, y))  
    System.out.println("Given point lies " +  
        "inside the rectangle");  
else  
    System.out.println("Given point does not " +  
        "lie on the rectangle");  
}  
}  
  
// This code is contributed  
// by ChitraNayal
```

Python3

```
# Python 3 program to Check whether  
# a given point lies inside or on  
# the rectangle or not  
  
# function to Check whether a given  
# point lies inside or on the  
# rectangle or not  
def LiesInsideRectangle(a, b, x, y) :  
  
    if(x - y - b <= 0 and  
        x - y + b >= 0 and  
        x + y - 2 * a + b <= 0 and  
        x + y - b >= 0) :  
        return True  
  
    return False  
  
# Driver code  
if __name__ == "__main__" :  
  
    # multiple assignments  
    a, b, x, y = 7, 2, 4, 5  
  
    if LiesInsideRectangle(a, b, x, y) :  
        print("Given point lies inside"  
              " the rectangle")  
    else :  
        print("Given point does not lie"  
              " on the rectangle")  
  
# This code is contributed by ANKITRAI1
```

C#

```
// C# program to Check whether
// a given point lies inside
// or on the rectangle or not
using System;
class GFG
{
    // function to Check whether
    // a given point lies inside
    // or on the rectangle or not
    static bool LiesInsideRectangle(int a, int b,
        int x, int y)
    {
        if (x - y - b <= 0 && x - y + b >= 0 &&
            x + y - 2 * a + b <= 0 && x + y - b >= 0)
            return true;
        return false;
    }
    // Driver code
    public static void Main()
    {
        int a = 7, b = 2, x = 4, y = 5;
        if (LiesInsideRectangle(a, b, x, y))
            Console.Write("Given point lies " +
                "inside the rectangle");
        else
            Console.Write("Given point does not " +
                "lie on the rectangle");
    }
}

// This code is contributed
// by ChitraNayal
```

PHP

```
<?php
// PHP program to Check whether
// a given point lies inside
// or on the rectangle or not

// function to Check whether
// a given point lies inside
// or on the rectangle or not
function LiesInsideRectangle($a, $b,
    $x, $y)
{
    if ($x - $y - $b <= 0 &&
```

```
$x - $y + $b >= 0 &&
$x + $y - 2 * $a + $b <= 0 &&
$x + $y - $b >= 0)
return true;

return false;
}

// Driver code
$a = 7;
$b = 2;
$x = 4;
$y = 5;

if (LiesInsideRectangle($a, $b,
                      $x, $y))
    echo "Given point lies ".
        "inside the rectangle";
else
    echo "Given point does not".
        " lie on the rectangle";

// This code is contributed by mits
?>
```

Output:

Given point lies inside the rectangle

Improved By : [ANKITRAI1](#), Mithun Kumar, ChitraNayal

Source

<https://www.geeksforgeeks.org/check-whether-a-given-point-lies-on-or-inside-the-rectangle-set-3/>

Chapter 38

Check whether a point exists in circle sector or not.

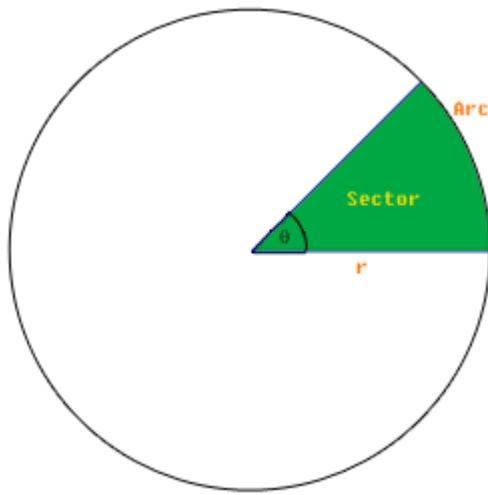
Check whether a point exists in circle sector or not. - GeeksforGeeks

We have a circle centered at origin (0, 0). As input we are given with starting angle of the circle sector and the size of the circle sector in percentage.

Examples:

```
Input : Radius = 8
        StartAngle = 0
        Percentage = 12
        x = 3 y = 4
Output : Point (3, 4) exists in the circle
          sector

Input : Radius = 12
        Startangle = 45
        Percentage = 25
        x = 3 y = 4
Output : Point (3, 4) does not exist in
          the circle sector
```



Source:wikibooks.org

In this image starting angle is 0 degree, radius r and suppose that percentage of colored area is 12% then we calculate Ending Angle as $360/\text{percentage} + \text{starting angle}$.

To find whether a point (x, y) exists in a circle sector (centered at origin) or not we find polar coordinates of that point and then go through the following steps:

1. Convert x, y to polar coordinates using this
 $\text{Angle} = \text{atan}(y/x); \text{Radius} = \sqrt{x^2 + y^2};$
2. Then Angle must be between StartingAngle and EndingAngle, and Radius between 0 and your Radius.

C++

```
// C++ program to check if a point lies inside a circle
// sector.
#include<bits/stdc++.h>
using namespace std;

void checkPoint(int radius, int x, int y, float percent,
                float startAngle)
{
    // calculate endAngle
    float endAngle = 360/percent + startAngle;

    // Calculate polar co-ordinates
    float polarradius = sqrt(x*x+y*y);
    float Angle = atan(y/x);

    // Check whether polarradius is less than radius of circle
    // or not and Angle is between startAngle and endAngle
```

```
// or not
if (Angle>=startAngle && Angle<=endAngle && polarradius<radius)
    printf("Point (%d, %d) exist in the circle sector\n", x, y);
else
    printf("Point (%d, %d) does not exist in the circle sector\n",
           x, y);
}

// Driver code
int main()
{
    int radius = 8, x = 3, y = 4;
    float percent = 12, startAngle = 0;
    checkPoint(radius, x, y, percent, startAngle);
    return 0;
}
```

Java

```
// Java program to check if
// a point lies inside a circle
// sector.

class GFG
{
    static void checkPoint(int radius, int x, int y, float percent,
                           float startAngle)
    {

        // calculate endAngle
        float endAngle = 360/percent + startAngle;

        // Calculate polar co-ordinates
        double polarradius = Math.sqrt(x*x+y*y);
        double Angle = Math.atan(y/x);

        // Check whether polarradius is
        // less than radius of circle
        // or not and Angle is between
        // startAngle and endAngle
        // or not
        if (Angle>=startAngle && Angle<=endAngle && polarradius<radius)
            System.out.print("Point"+("+"x+", "+y)+" "+
                            " exist in the circle sector\n");
        else
            System.out.print("Point"+("+"x+", "+y)+" "+
                            " exist in the circle sector\n");
    }
}
```

```
// Driver Program to test above function
public static void main(String arg[])
{
    int radius = 8, x = 3, y = 4;
    float percent = 12, startAngle = 0;
    checkPoint(radius, x, y, percent, startAngle);
}
}

// This code is contributed
// by Anant Agarwal.
```

Python3

```
# Python3 program to check if a point
# lies inside a circle sector.

import math

def checkPoint(radius, x, y, percent, startAngle):

    # calculate endAngle
    endAngle = 360 / percent + startAngle

    # Calculate polar co-ordinates
    polarradius = math.sqrt(x * x + y * y)
    Angle = math.atan(y / x)

    # Check whether polarradius is less
    # then radius of circle or not and
    # Angle is between startAngle and
    # endAngle or not
    if (Angle >= startAngle and Angle <= endAngle
        and polarradius < radius):
        print("Point (", x, ", ", y, ") "
              "exist in the circle sector")
    else:
        print("Point (", x, ", ", y, ") "
              "does not exist in the circle sector")

# Driver code
radius, x, y = 8, 3, 4
percent, startAngle = 12, 0

checkPoint(radius, x, y, percent, startAngle)

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
// C# program to check if a point lies
// inside a circle sector.
using System.IO;
using System;

class GFG {

    static void checkPoint(int radius, int x, int y,
                          float percent, float startAngle)
    {

        // calculate endAngle
        float endAngle = 360 / percent + startAngle;

        // Calculate polar co-ordinates
        float polarradius =
            (float)Math.Sqrt(x * x + y * y);

        float Angle = (float)Math.Atan(y / x);

        // Check whether polarradius is less than
        // radius of circle or not and Angle is
        // between startAngle and endAngle or not
        if (Angle >= startAngle && Angle <= endAngle
            && polarradius < radius)
            Console.WriteLine("Point ({0}, {1}) exist in "
                + "the circle sector", x, y);
        else
            Console.WriteLine("Point ({0}, {1}) does not "
                + "exist in the circle sector", x, y);
    }

    // Driver code
    public static void Main()
    {
        int radius = 8, x = 3, y = 4;
        float percent = 12, startAngle = 0;
        checkPoint(radius, x, y, percent, startAngle);
    }
}

// This code is contributed by Smitha Dinesh Semwal
```

Output :

```
Point(3, 4) exists in the circle sector
```

Time complexity = O(1)

Improved By : [Smitha Dinesh Semwal](#)

Source

<https://www.geeksforgeeks.org/check-whether-point-exists-circle-sector-not/>

Chapter 39

Check whether a point lies inside a sphere or not

Check whether a point lies inside a sphere or not - GeeksforGeeks

Given co-ordinates of the center of a sphere (**cx**, **cy**, **cz**) and its radius **r**. Our task is to check whether a point (**x**, **y**, **z**) lies inside, outside or on this sphere.

Examples:

```
Input : Centre(0, 0, 0) Radius 3
        Point(1, 1, 1)
Output :Point is inside the sphere
```

```
Input :Centre(0, 0, 0) Radius 3
        Point(2, 1, 2)
Output :Point lies on the sphere
```

```
Input :Centre(0, 0, 0) Radius 3
        Point(10, 10, 10)
Output :Point is outside the sphere
```

Approach:

Whether a point lies inside a sphere or not, depends upon its distance from the centre.
A point (**x**, **y**, **z**) is inside the sphere with center (**cx**, **cy**, **cz**) and radius **r** if

$$(x-cx)^2 + (y-cy)^2 + (z-cz)^2 < r^2$$

A point (**x**, **y**, **z**) lies on the sphere with center (**cx**, **cy**, **cz**) and radius **r** if

$$(x-cx)^2 + (y-cy)^2 + (z-cz)^2 = r^2$$

A point (x, y, z) is outside the sphere with center (cx, cy, cz) and radius r if

$$(x - cx)^2 + (y - cy)^2 + (z - cz)^2 > r^2$$

C++

```
// CPP code to illustrate above approach
#include <bits/stdc++.h>
using namespace std;
// function to calculate the distance between centre and the point
int check(int cx, int cy, int cz, int x, int y, int z)
{
    int x1 = pow((x - cx), 2);
    int y1 = pow((y - cy), 2);
    int z1 = pow((z - cz), 2);

    // distance between the centre
    // and given point
    return (x1 + y1 + z1);
}

// Driver program to test above function
int main()
{
    // coordinates of centre
    int cx = 1, cy = 2, cz = 3;

    int r = 5; // radius of the sphere
    int x = 4, y = 5, z = 2; // coordinates of point

    int ans = check(cx, cy, cz, x, y, z);

    // distance btw centre and point is less
    // than radius
    if (ans < (r * r))
        cout << "Point is inside the sphere";

    // distance btw centre and point is
    // equal to radius
    else if (ans == (r * r))
        cout << "Point lies on the sphere";

    // distance btw center and point is
    // greater than radius
    else
        cout << "Point is outside the sphere";
    return 0;
}
```

Java

```
// Java code to illustrate above approach
import java.io.*;
import java.util.*;
import java.lang.*;

class GfG {

    // function to calculate the distance
    // between centre and the point
    public static int check(int cx, int cy,
                           int cz, int x, int y, int z)
    {
        int x1 = (int)Math.pow((x - cx), 2);
        int y1 = (int)Math.pow((y - cy), 2);
        int z1 = (int)Math.pow((z - cz), 2);

        // distance between the centre
        // and given point
        return (x1 + y1 + z1);
    }

    // Driver program to test above function
    public static void main(String[] args)
    {
        // coordinates of centre
        int cx = 1, cy = 2, cz = 3;

        int r = 5; // radius of the sphere

        // coordinates of point
        int x = 4, y = 5, z = 2;

        int ans = check(cx, cy, cz, x, y, z);

        // distance btw centre and point is less
        // than radius
        if (ans < (r * r))
            System.out.println("Point is inside"
                               + " the sphere");

        // distance btw centre and point is
        // equal to radius
        else if (ans == (r * r))
            System.out.println("Point lies on"
                               + " the sphere");
    }
}
```

```
// distance btw center and point is
// greater than radius
else
    System.out.println("Point is outside"
                        + " the sphere");
}
}

// This code is contributed by Sagar Shukla.
```

Python

```
# Python3 code to illustrate above approach

import math
# function to calculate distance btw center and given point
def check(cx, cy, cz, x, y, z, ):

    x1 = math.pow((x-cx), 2)
    y1 = math.pow((y-cy), 2)
    z1 = math.pow((z-cz), 2)
    return (x1 + y1 + z1) # distance between the centre and given point

# driver code
cx = 1
cy = 2 # coordinates of centre
cz = 3

r = 5 # radius of sphere

x = 4
y = 5 # coordinates of the given point
z = 2
# function call to calculate distance btw centre and given point
ans = check(cx, cy, cz, x, y, z);

# distance btw centre and point is less than radius
if ans<(r**2):
    print("Point is inside the sphere")

# distance btw centre and point is equal to radius
elif ans == (r**2):
    print("Point lies on the sphere")

# distance btw centre and point is greater than radius
else:
    print("Point is outside the sphere")
```

C#

```
// C# code to illustrate
// above approach
using System;

class GFG
{

    // function to calculate
    // the distance between
    // centre and the point
    public static int check(int cx, int cy,
                           int cz, int x,
                           int y, int z)
    {
        int x1 = (int)Math.Pow((x - cx), 2);
        int y1 = (int)Math.Pow((y - cy), 2);
        int z1 = (int)Math.Pow((z - cz), 2);

        // distance between the
        // centre and given point
        return (x1 + y1 + z1);
    }

    // Driver Code
    public static void Main()
    {
        // coordinates of centre
        int cx = 1, cy = 2, cz = 3;

        int r = 5; // radius of the sphere

        // coordinates of point
        int x = 4, y = 5, z = 2;

        int ans = check(cx, cy, cz,
                        x, y, z);

        // distance btw centre
        // and point is less
        // than radius
        if (ans < (r * r))
            Console.WriteLine("Point is inside" +
                              " the sphere");

        // distance btw centre
        // and point is
```

```
// equal to radius
else if (ans == (r * r))
    Console.WriteLine("Point lies on" +
                      " the sphere");

// distance btw center
// and point is greater
// than radius
else
    Console.WriteLine("Point is outside" +
                      " the sphere");
}

}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// function to calculate
// the distance between
// centre and the point
function check($cx, $cy, $cz,
              $x, $y, $z)
{
    $x1 = pow(($x - $cx), 2);
    $y1 = pow(($y - $cy), 2);
    $z1 = pow(($z - $cz), 2);

    // distance between the
    // centre and given point
    return ($x1 + $y1 + $z1);
}

// Driver Code

// coordinates of centre
$cx = 1;
$cy = 2;
$cz = 3;

$r = 5; // radius of the sphere
$x = 4;
$y = 5;
$z = 2; // coordinates of point

$ans = check($cx, $cy, $cz,
             $x, $y, $z);
```

```
// distance btw centre and
// point is less than radius
if ($ans < ($r * $r))
    echo "Point is inside " .
        "the sphere";

// distance btw centre and
// point is equal to radius
else if ($ans == ($r * $r))
    echo "Point lies on the sphere";

// distance btw center and point
// is greater than radius
else
    echo "Point is outside " .
        "the sphere";

// This code is contributed
// by shiv_bhakt
?>
```

Output:

Point is inside the sphere

Time Complexity:O(1)

Improved By : [vt_m](#), [shiv_bhakt](#)

Source

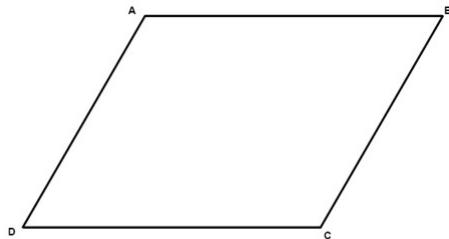
<https://www.geeksforgeeks.org/check-whether-a-point-lies-inside-a-sphere-or-not/>

Chapter 40

Check whether four points make a parallelogram

Check whether four points make a parallelogram - GeeksforGeeks

Given four points in a 2-dimensional space we need to find out whether they make a parallelogram or not.



A parallelogram has four sides. Two opposite sides are parallel and are of same lengths.

Examples:

```
Points = [(0, 0), (4, 0), (1, 3), (5, 3)]  
Above points make a parallelogram.
```

```
Points = [(0, 0), (2, 0), (4, 0), (2, 2)]  
Above points does not make a parallelogram  
as first three points itself are linear.
```

Problems for checking square and rectangle can be read from [Square checking](#) and [Rectangle checking](#) but in this problem, we need to check for the parallelogram. The main properties of the parallelogram are that opposite sides of parallelogram are parallel and of equal length

and diagonals of parallelogram bisect each other. We use the second property to solve this problem. As there are four points, we can get total 6 midpoints by considering each pair. Now for four points to make a parallelogram, 2 of the midpoints should be equal and rest of them should be different.

In below code, we have created a map, which stores pairs corresponding to each midpoint. After calculating all midpoints, we have iterated over the map and check the occurrence of each midpoint, If exactly one midpoint occurred twice and other have occurred once, then given four points make a parallelogram otherwise not.

```
// C++ code to test whether four points make a
// parallelogram or not
#include <bits/stdc++.h>
using namespace std;

// structure to represent a point
struct point {
    double x, y;
    point() { }
    point(double x, double y)
        : x(x), y(y) { }

    // defining operator < to compare two points
    bool operator<(const point& other) const
    {
        if (x < other.x) {
            return true;
        } else if (x == other.x) {
            if (y < other.y) {
                return true;
            }
        }
        return false;
    }
};

// Utility method to return mid point of two points
point getMidPoint(point points[], int i, int j)
{
    return point((points[i].x + points[j].x) / 2.0,
                 (points[i].y + points[j].y) / 2.0);
}

// method returns true if point of points array form
// a parallelogram
bool isParallelogram(point points[])
{
    map<point, vector<point>> midPointMap;
```

```
// looping over all pairs of point to store their
// mid points
int P = 4;
for (int i = 0; i < P; i++) {
    for (int j = i + 1; j < P; j++) {
        point temp = getMidPoint(points, i, j);

        // storing point pair, corresponding to
        // the mid point
        midPointMap[temp].push_back(point(i, j));
    }
}

int two = 0, one = 0;

// looping over (midpoint, (corresponding pairs))
// map to check the occurrence of each midpoint
for (auto x : midPointMap) {

    // updating midpoint count which occurs twice
    if (x.second.size() == 2)
        two++;

    // updating midpoing count which occurs once
    else if (x.second.size() == 1)
        one++;

    // if midpoint count is more than 2, then
    // parallelogram is not possible
    else
        return false;
}

// for parallelogram, one mid point should come
// twice and other mid points should come once
if (two == 1 && one == 4)
    return true;

return false;
}

// Driver code to test above methods
int main()
{
    point points[4];

    points[0] = point(0, 0);
    points[1] = point(4, 0);
```

```
points[2] = point(1, 3);
points[3] = point(5, 3);

if (isParallelogram(points))
    cout << "Given points form a parallelogram";
else
    cout << "Given points does not form a "
        "parallelogram";
return 0;
}
```

Output:

```
Given points form a parallelogram
```

Source

<https://www.geeksforgeeks.org/check-whether-four-points-make-parallelogram/>

Chapter 41

Check whether given circle resides in boundary maintained by two other circles

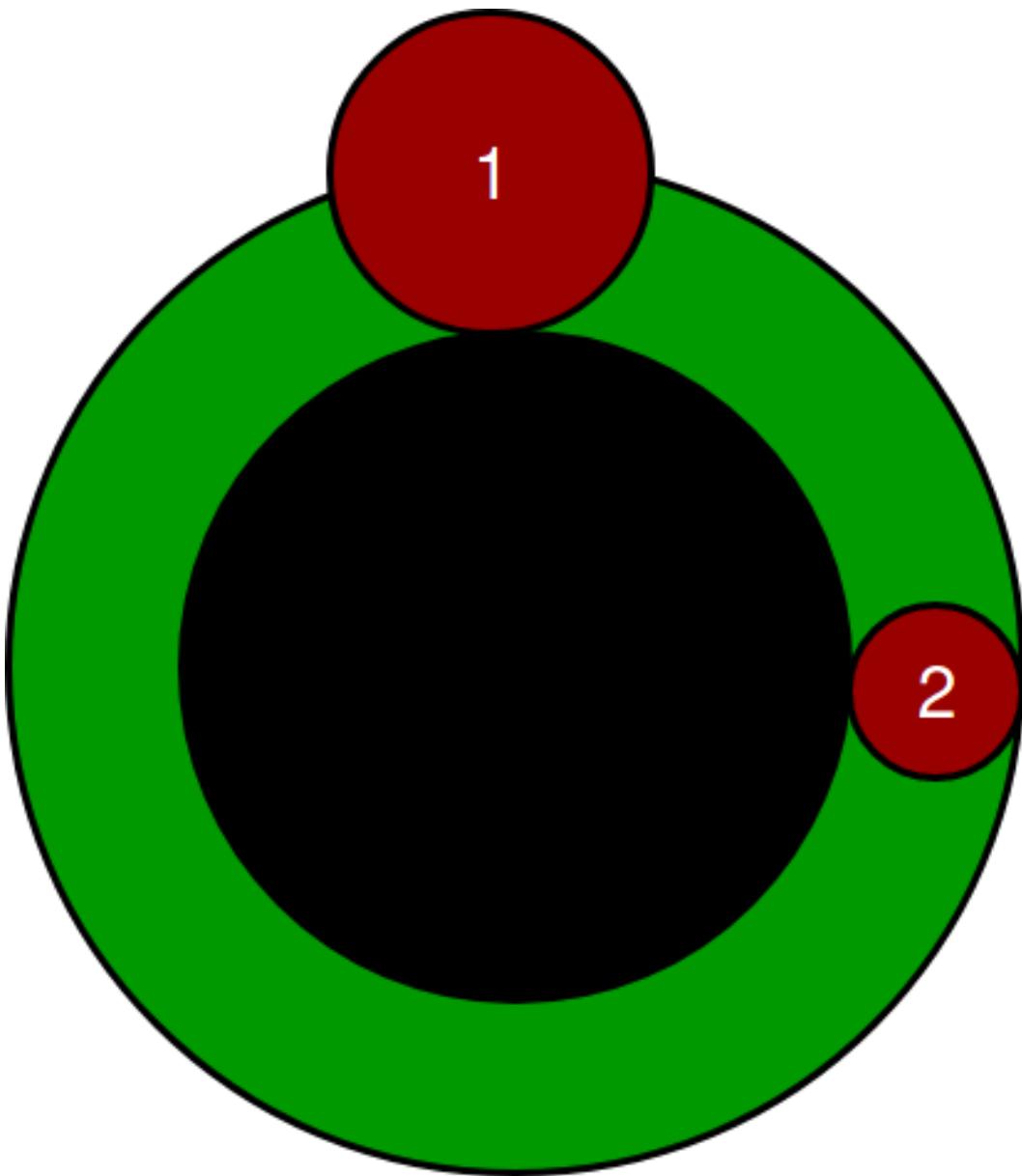
Check whether given circle resides in boundary maintained by two other circles - Geeks-forGeeks

Given outer circle radius R and inner circle radius r, making circles from same center and forming boundary between them. Now, given X,Y co-ordinates which denotes center of new circle to be formed with radius rad, your task is to check whether the circle with co-ordinate X,Y as center can fit in the boundary of circles formed or not.

Examples:

```
Input : R = 8, r = 4
x = 5, y = 3, rad = 1
Output : Fits
```

```
Input : R = 8, r = 4
x = 5, y = 3, rad = 3.
Output : Doesn't Fit
```



1 – Doesn't fits

2 – Fits

The idea is to calculate the distance between the center $(0, 0)$ and the co-ordinates of the circle to be checked. If distance + radius (of the circle to be checked) is less than or equal to Outer Radius and distance - radius (of the circle to be checked) is greater than or equal to Radius of Outer circle - Radius Inner circle

It fits.

Here is the implementation :

C++

```
// CPP program to check whether circle with given
// co-ordinates reside within the boundary
// of outer circle and inner circle
#include <bits/stdc++.h>
using namespace std;

// function to check if given circle fit in
// boundary or not
void fitOrNotFit(int R, int r, int x, int y,
                  int rad) {

    // Distance from the center
    double val = sqrt(pow(x, 2) + pow(y, 2));

    // Checking the corners of circle
    if (val + rad <= R && val - rad >= R - r)
        cout << "Fits\n";
    else
        cout << "Doesn't Fit\n";
}

// driver program
int main()
{
    // Radius of outer circle and inner circle
    // respectively
    int R = 8, r = 4;

    // Co-ordinates and radius of the circle
    // to be checked
    int x = 5, y = 3, rad = 3;
    fitOrNotFit(R, r, x, y, rad);
    return 0;
}
```

Java

```
// Java program to check whether circle with given
// co-ordinates reside within the boundary
// of outer circle and inner circle
import java.util.*;

class GFG
{
    // function to check if given circle fit in
```

```
// boundary or not
static void fitOrNotFit(int R, int r, int x, int y,
                        int rad)
{
    // Distance from the center
    double val = Math.sqrt(Math.pow(x, 2) +
                           Math.pow(y, 2));

    // Checking the corners of circle
    if (val + rad <= R && val - rad >= R - r)
        System.out.println("Fits");
    else
        System.out.println("Doesn't Fit");
}

// driver program
public static void main (String[] args)
{
    // Radius of outer circle and inner circle
    // respectively
    int R = 8, r = 4;

    // Co-ordinates and radius of the circle
    // to be checked
    int x = 5, y = 3, rad = 3;
    fitOrNotFit(R, r, x, y, rad);
}
}
/*
 * This Code is contributed by Kriti Shukla */

```

Python3

```
# Python3 program to check
# whether circle with given
# co-ordinates reside
# within the boundary
# of outer circle
# and inner circle

import math

# function to check if
# given circle fit in
# boundary or not
def fitOrNotFit(R, r, x, y, rad) :

    # Distance from the center
    val = math.sqrt(math.pow(x, 2) + math.pow(y, 2))
```

```
# Checking the corners of circle
if (val + rad <= R and val - rad >= R - r) :
    print("Fits\n")
else:
    print("Doesn't Fit")

# driver program

# Radius of outer circle and inner circle
# respectively
R = 8
r = 4

# Co-ordinates and radius of the circle
# to be checked
x = 5
y = 3
rad = 3

fitOrNotFit(R, r, x, y, rad)

# This code is contributed by
# Smitha Dinesh Semwal

C#
// C# program to check whether circle with given
// co-ordinates reside within the boundary
// of outer circle and inner circle
using System;

class GFG
{
    // function to check if given circle fit in
    // boundary or not
    static void fitOrNotFit(int R, int r, int x, int y,
                           int rad)
    {
        // Distance from the center
        double val = Math.Sqrt(Math.Pow(x, 2) +
                               Math.Pow(y, 2));

        // Checking the corners of circle
        if (val + rad <= R && val - rad >= R - r)
            Console.WriteLine("Fits");
        else
```

```
        Console.WriteLine("Doesn't Fit");
    }

    // Driver program
    public static void Main ()
    {
        // Radius of outer circle and inner circle
        // respectively
        int R = 8, r = 4;

        // Co-ordinates and radius of the circle
        // to be checked
        int x = 5, y = 3, rad = 3;
        fitOrNotFit(R, r, x, y, rad);
    }
}

// This Code is contributed by Anant Agarwal.
```

PHP

```
<?php
// PHP program to check whether
// circle with given co-ordinates
// reside within the boundary
// of outer circle and inner circle

// function to check if given
// circle fit in boundary or not
function fitOrNotFit($R, $r, $x, $y,
                     $rad)
{
    // Distance from the center
    $val = sqrt(pow($x, 2) + pow($y, 2));

    // Checking the corners of circle
    if ($val + $rad <= $R && $val -
        $rad >= $R - $r)
        echo "Fits\n";
    else
        echo "Doesn't Fit\n";
}

// Driver Code

// Radius of outer circle and
// inner circle respectively
```

```
$R = 8; $r = 4;

// Co-ordinates and radius of
// the circle to be checked
$x = 5; $y = 3; $rad = 3;
fitOrNotFit($R, $r, $x, $y, $rad);

// This Code is contributed by vt_m.
?>
```

Output:

Doesn't Fit

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/check-whether-given-circle-reside-boundary-maintained-outer-circle-inner-circle/>

Chapter 42

Check whether triangle is valid or not if sides are given

Check whether triangle is valid or not if sides are given - GeeksforGeeks

Given three sides, check whether triangle is valid or not.

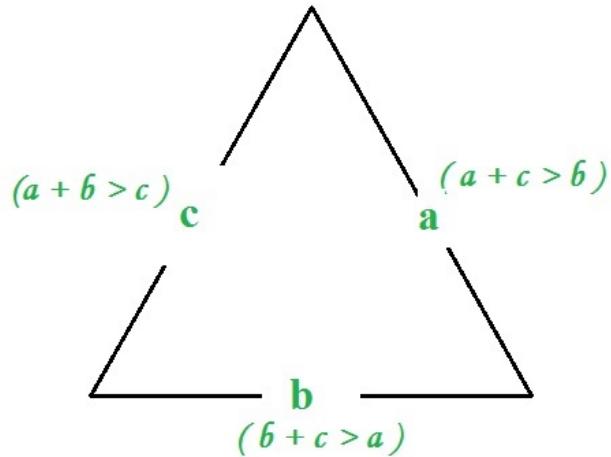
Examples:

Input : a = 7, b = 10, c = 5
Output : Valid

Input : a = 1 b = 10 c = 12
Output : Invalid

Approach: A triangle is valid if sum of its two sides is greater than the third side. If three sides are a, b and c, then three conditions should be met.

- 1.a + b > c
- 2.a + c > b
- 3.b + c > a



C++

```
// C++ program to check if three
// sides form a triangle or not
#include<bits/stdc++.h>
using namespace std;

// function to check if three sider
// form a triangle or not
bool checkValidity(int a, int b, int c)
{
    // check condition
    if (a + b <= c || a + c <= b || b + c <= a)
        return false;
    else
        return true;
}

// Driver function
int main()
{
    int a = 7, b = 10, c = 5;

    if (checkValidity(a, b, c))
        cout << "Valid";
    else
        cout << "Invalid";
}
```

Java

```
// Java program to check
// validity of any triangle

public class GFG {

    // Function to calculate for validity
    public static int checkValidity(int a,
                                    int b, int c)
    {
        // check condition
        if (a + b <= c || a + c <= b || b + c <= a)
            return 0;
        else
            return 1;
    }

    // Driver function
    public static void main(String args[])
    {

        int a = 7, b = 10, c = 5;

        // function calling and print output
        if ((checkValidity(a, b, c)) == 1)
            System.out.print("Valid");
        else
            System.out.print("Invalid");

    }
}

// This article is contributed by 'Akansh Gupta'
```

Python3

```
# Python3 program to check if three
# sides form a triangle or not

# function to check if three sides
# form a triangle or not
def checkValidity(a, b, c):

    # check condition
    if (a + b <= c) or (a + c <= b) or (b + c <= a) :
```

```
        return False
else:
    return True

# driver code
a = 7
b = 10
c = 5
if checkValidity(a, b, c):
    print("Valid")
else:
    print("Invalid")
```

C#

```
// C# program to check
// validity of any triangle
using System;

class GFG {

    // Function to calculate for validity
    public static int checkValidity(int a, int b,
                                    int c)
    {

        // check condition
        if (a + b <= c || a + c <= b ||
            b + c <= a)
            return 0;
        else
            return 1;
    }

    // Driver code
    public static void Main()
    {
        int a = 7, b = 10, c = 5;

        // function calling and print output
        if ((checkValidity(a, b, c)) == 1)
            Console.WriteLine("Valid");
        else
            Console.WriteLine("Invalid");

    }
}
```

// This code is contributed by Nitin Mittal.

PHP

```
<?php
// PHP program to check if three
// sides form a triangle or not

// function to check if three sider
// form a triangle or not
function checkValidity($a, $b, $c)
{

    // check condition
    if ($a + $b <= $c || 
        $a + $c <= $b || 
        $b + $c <= $a)
        return false;
    else
        return true;
}

// Driver Code
$a = 7;
$b = 10;
$c = 5;

if (checkValidity($a, $b, $c))
    echo "Valid";
else
    echo "Invalid";

// This code is contributed by nitin mittal.
?>
```

Output :

Valid

Improved By : [Koushik Mondal, nitin mittal](#)

Source

<https://www.geeksforgeeks.org/check-whether-triangle-valid-not-sides-given/>

Chapter 43

Circle and Lattice Points

Circle and Lattice Points - GeeksforGeeks

Given a circle of radius r in 2-D with origin or $(0, 0)$ as center. The task is to find the total lattice points on circumference. Lattice Points are points with coordinates as integers in 2-D space.

Example:

```
Input : r = 5.
Output : 12
Below are lattice points on a circle with
radius 5 and origin as (0, 0).
(0,5), (0,-5), (5,0), (-5,0),
(3,4), (-3,4), (-3,-4), (3,-4),
(4,3), (-4,3), (-4,-3), (4,-3).
are 12 lattice point.
```

To find lattice points, we basically need to find values of (x, y) which satisfy the equation $x^2 + y^2 = r^2$.

For any value of (x, y) that satisfies the above equation we actually have total 4 different combination which that satisfy the equation. For example if $r = 5$ and $(3, 4)$ is a pair which satisfies the equation, there are actually 4 combinations $(3, 4)$, $(-3, 4)$, $(-3, -4)$, $(3, -4)$. There is an exception though, in case of $(0, r)$ or $(r, 0)$ there are actually 2 points as there is no negative 0.

```
// Initialize result as 4 for (r, 0), (-r, 0),
// (0, r) and (0, -r)
result = 4
```

Loop for $x = 1$ to $r-1$ and do following for every x .

If $r \cdot r - x \cdot x$ is a perfect square, then add 4 to result.

Below is the implementation of above idea.

CPP

```
// C++ program to find countLattice points on a circle
#include<bits/stdc++.h>
using namespace std;

// Function to count Lattice points on a circle
int countLattice(int r)
{
    if (r <= 0)
        return 0;

    // Initialize result as 4 for (r, 0), (-r, 0),
    // (0, r) and (0, -r)
    int result = 4;

    // Check every value that can be potential x
    for (int x=1; x<r; x++)
    {
        // Find a potential y
        int ySquare = r*r - x*x;
        int y = sqrt(ySquare);

        // checking whether square root is an integer
        // or not. Count increments by 4 for four
        // different quadrant values
        if (y*y == ySquare)
            result += 4;
    }

    return result;
}

// Driver program
int main()
{
    int r = 5;
    cout << countLattice(r);
    return 0;
}
```

Java

```
// Java program to find
// countLattice points on a circle

class GFG
{

    // Function to count
    // Lattice points on a circle
    static int countLattice(int r)
    {
        if (r <= 0)
            return 0;

        // Initialize result as 4 for (r, 0), (-r, 0),
        // (0, r) and (0, -r)
        int result = 4;

        // Check every value that can be potential x
        for (int x=1; x<r; x++)
        {
            // Find a potential y
            int ySquare = r*r - x*x;
            int y = (int)Math.sqrt(ySquare);

            // checking whether square root is an integer
            // or not. Count increments by 4 for four
            // different quadrant values
            if (y*y == ySquare)
                result += 4;
        }

        return result;
    }

    // Driver code
    public static void main(String arg[])
    {
        int r = 5;
        System.out.println(countLattice(r));
    }
}
// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 program to find
# countLattice podesf on a circle
```

```
import math

# Function to count Lattice
# points on a circle
def countLattice(r):

    if (r <= 0):
        return 0

    # Initialize result as 4 for (r, 0), (-r, 0),
    # (0, r) and (0, -r)
    result = 4

    # Check every value that can be potential x
    for x in range(1, r):

        # Find a potential y
        ySquare = r*r - x*x
        y = int(math.sqrt(ySquare))

        # checking whether square root is an integer
        # or not. Count increments by 4 for four
        # different quadrant values
        if (y*y == ySquare):
            result += 4

    return result
```

```
# Driver program
r = 5
print(countLattice(r))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
// C# program to find countLattice
// points on a circle
using System;

class GFG {

    // Function to count Lattice
    // points on a circle
    static int countLattice(int r)
```

```
{  
    if (r <= 0)  
        return 0;  
  
    // Initialize result as 4  
    // for (r, 0), (-r, 0),  
    // (0, r) and (0, -r)  
    int result = 4;  
  
    // Check every value that  
    // can be potential x  
    for (int x = 1; x < r; x++)  
    {  
  
        // Find a potential y  
        int ySquare = r*r - x*x;  
        int y = (int)Math.Sqrt(ySquare);  
  
        // checking whether square root  
        // is an integer or not. Count  
        // increments by 4 for four  
        // different quadrant values  
        if (y*y == ySquare)  
            result += 4;  
    }  
  
    return result;  
}  
  
// Driver code  
public static void Main()  
{  
    int r = 5;  
  
    Console.WriteLine(countLattice(r));  
}  
}  
  
// This code is contributed by nitin mittal.
```

PHP

```
<?php  
// PHP program to find countLattice  
// points on a circle  
  
// Function to count Lattice  
// points on a circle
```

```
function countLattice($r)
{
    if ($r <= 0)
        return 0;

    // Initialize result as 4
    // for (r, 0), (-r, 0),
    // (0, r) and (0, -r)
    $result = 4;

    // Check every value that
    // can be potential x
    for ($x = 1; $x < $r; $x++)
    {

        // Find a potential y
        $ySquare = $r * $r - $x * $x;
        $y = ceil(sqrt($ySquare));

        // checking whether square
        // root is an integer
        // or not. Count increments
        // by 4 for four different
        // quadrant values
        if ($y * $y == $ySquare)
            $result += 4;
    }

    return $result;
}

// Driver Code
$r = 5;
echo countLattice($r);

// This code is contributed by nitin mittal
?>
```

Output:

12

Reference:

<http://mathworld.wolfram.com/CircleLatticePoints.html>

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/circle-lattice-points/>

Chapter 44

Classify a triangle

Classify a triangle - GeeksforGeeks

We are given co-ordinates of a triangle. The task is to classify this triangle on basis of sides and angle.

Examples:

Input : p1 = (3, 0), p2 = (0, 4), p3 = (4, 7)
Output : Right Angle triangle and Isosceles

Input : p1 = (0, 0), p2 = (1, 1), p3 = (1, 2);
Output : Triangle is obtuse and Scalene

We can solve this problem by first calculating the side length and then classifying on comparing of side lengths. Classification by sides is simple, if all sides are equal, triangle will be **equilateral**, if any two sides are equal triangle will be **Isosceles** otherwise it will be **Scalene**.

Now angle can be classified by Pythagoras theorem, if sum of square of two sides is equal to square of third side, triangle will be **right angle**, if less triangle will be **acute angle** else it will be **obtuse angle** triangle.

Below is written simple code for classification of triangle :

```
// C/C++ program to classify a given triangle
#include <bits/stdc++.h>
using namespace std;

struct point
{
    int x, y;
    point()    {}
    point(int x, int y) : x(x), y(y)    {}
}
```

```
};

// Utility method to return square of x
int square(int x)
{
    return x * x;
}

// Utility method to sort a, b, c; after this
// method a <= b <= c
void order(int &a, int &b, int &c)
{
    int copy[3];
    copy[0] = a;
    copy[1] = b;
    copy[2] = c;
    sort(copy, copy + 3);
    a = copy[0];
    b = copy[1];
    c = copy[2];
}

// Utility method to return Square of distance
// between two points
int euclidDistSquare(point p1, point p2)
{
    return square(p1.x - p2.x) + square(p1.y - p2.y);
}

// Method to classify side
string getSideClassification(int a, int b, int c)
{
    // if all sides are equal
    if (a == b && b == c)
        return "Equilateral";

    // if any two sides are equal
    else if (a == b || b == c)
        return "Isosceles";

    else
        return "Scalene";
}

// Method to classify angle
string getAngleClassification(int a, int b, int c)
{
    // If addition of sum of square of two side
```

```
// is less, then acute
if (a + b > c)
    return "acute";

// by pythagoras theorem
else if (a + b == c)
    return "right";

else
    return "obtuse";
}

// Method to classify triangle by sides and angles
void classifyTriangle(point p1, point p2, point p3)
{
    // Find squares of distances between points
    int a = euclidDistSquare(p1, p2);
    int b = euclidDistSquare(p1, p3);
    int c = euclidDistSquare(p2, p3);

    // Sort all squares of distances in increasing order
    order(a, b, c);

    cout << "Triangle is " +
        getAngleClassification(a, b, c) + " and " +
        getSideClassification(a, b, c) << endl;
}

// Driver code
int main()
{
    point p1, p2, p3;
    p1 = point(3, 0);
    p2 = point(0, 4);
    p3 = point(4, 7);
    classifyTriangle(p1, p2, p3);

    p1 = point(0, 0);
    p2 = point(1, 1);
    p3 = point(1, 2);
    classifyTriangle(p1, p2, p3);
    return 0;
}
```

Output:

```
Triangle is right and Isosceles
```

Triangle is obtuse and Scalene

Source

<https://www.geeksforgeeks.org/classify-a-triangle/>

Chapter 45

Closest Pair of Points using Divide and Conquer algorithm

Closest Pair of Points using Divide and Conquer algorithm - GeeksforGeeks

We are given an array of n points in the plane, and the problem is to find out the closest pair of points in the array. This problem arises in a number of applications. For example, in air-traffic control, you may want to monitor planes that come too close together, since this may indicate a possible collision. Recall the following formula for distance between two points p and q.

$$\|Pq\| = \sqrt{(Px - Qx)^2 + (Py - Qy)^2}$$

The Brute force solution is $O(n^2)$, compute the distance between each pair and return the smallest. We can calculate the smallest distance in $O(n \log n)$ time using Divide and Conquer strategy. In this post, a $O(n \times (\log n)^2)$ approach is discussed. We will be discussing a $O(n \log n)$ approach in a separate post.

Algorithm

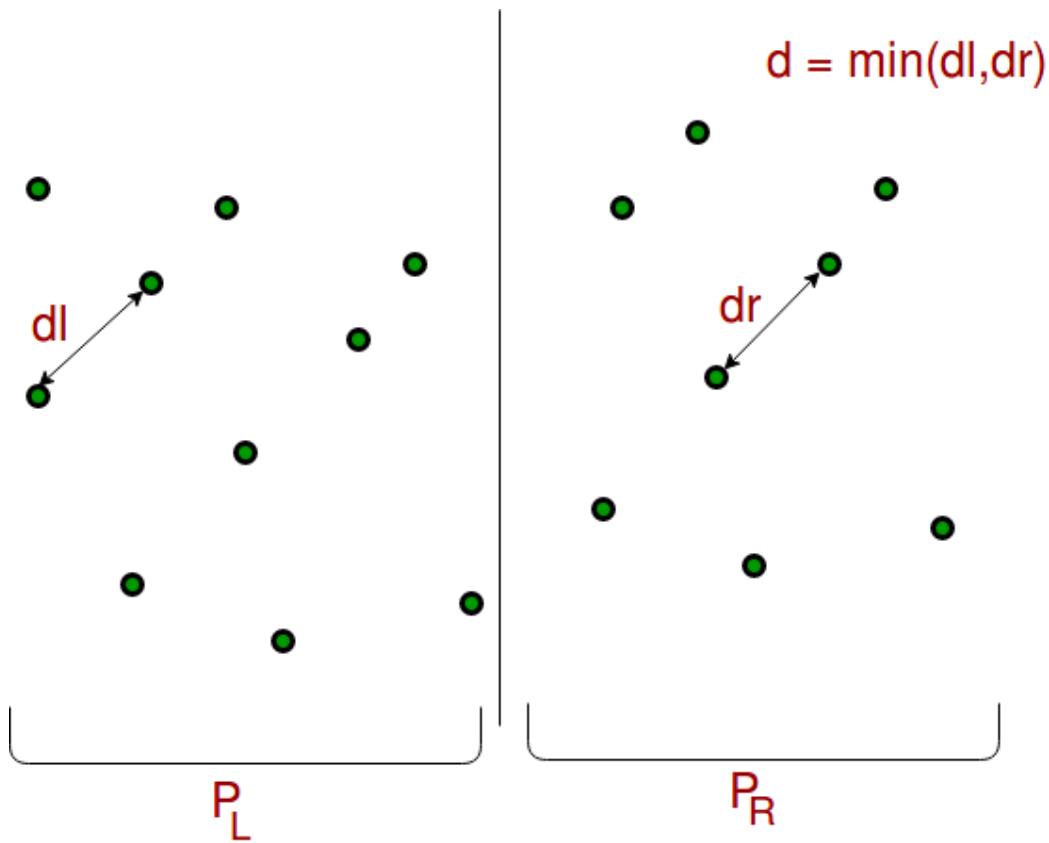
Following are the detailed steps of a $O(n (\log n)^2)$ algorithm.

Input: An array of n points $P[]$

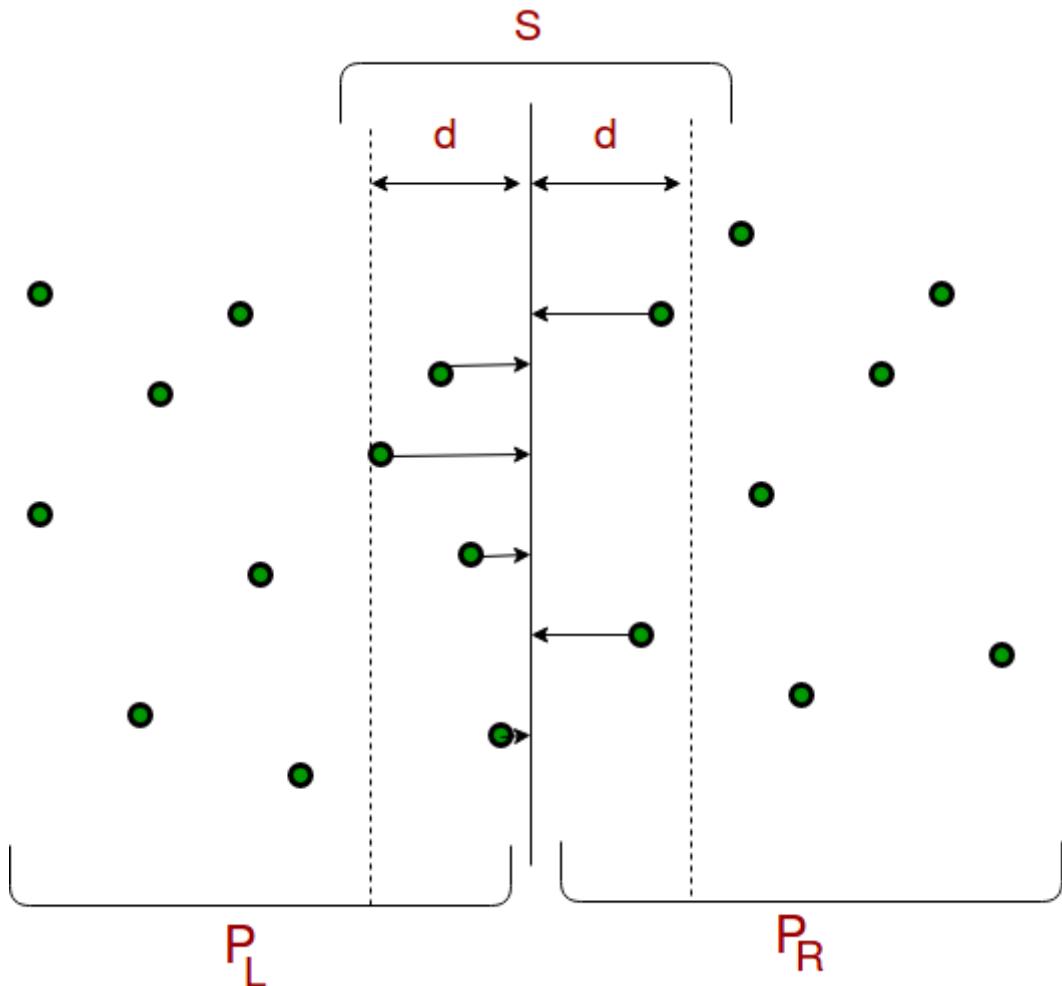
Output: The smallest distance between two points in the given array.

As a pre-processing step, input array is sorted according to x coordinates.

- 1) Find the middle point in the sorted array, we can take $P[n/2]$ as middle point.
- 2) Divide the given array in two halves. The first subarray contains points from $P[0]$ to $P[n/2]$. The second subarray contains points from $P[n/2+1]$ to $P[n-1]$.
- 3) Recursively find the smallest distances in both subarrays. Let the distances be dl and dr . Find the minimum of dl and dr . Let the minimum be d .



4) From above 3 steps, we have an upper bound d of minimum distance. Now we need to consider the pairs such that one point in pair is from left half and other is from right half. Consider the vertical line passing through $P[n/2]$ and find all points whose x coordinate is closer than d to the middle vertical line. Build an array $\text{strip}[]$ of all such points.



5) Sort the array $\text{strip}[]$ according to y coordinates. This step is $O(n \log n)$. It can be optimized to $O(n)$ by recursively sorting and merging.

6) Find the smallest distance in $\text{strip}[]$. This is tricky. From first look, it seems to be a $O(n^2)$ step, but it is actually $O(n)$. It can be proved geometrically that for every point in strip, we only need to check at most 7 points after it (note that strip is sorted according to Y coordinate). See [this](#) for more analysis.

7) Finally return the minimum of d and distance calculated in above step (step 6)

Implementation

Following is C/C++ implementation of the above algorithm.

```
// A divide and conquer program in C/C++ to find the smallest distance from a
// given set of points.
```

```
#include <stdio.h>
```

```
#include <float.h>
#include <stdlib.h>
#include <math.h>

// A structure to represent a Point in 2D plane
struct Point
{
    int x, y;
};

/* Following two functions are needed for library function qsort().
   Refer: http://www.cplusplus.com/reference/cstdlib/qsort/ */

// Needed to sort array of points according to X coordinate
int compareX(const void* a, const void* b)
{
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->x - p2->x);
}

// Needed to sort array of points according to Y coordinate
int compareY(const void* a, const void* b)
{
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->y - p2->y);
}

// A utility function to find the distance between two points
float dist(Point p1, Point p2)
{
    return sqrt( (p1.x - p2.x)*(p1.x - p2.x) +
                 (p1.y - p2.y)*(p1.y - p2.y)
               );
}

// A Brute Force method to return the smallest distance between two points
// in P[] of size n
float bruteForce(Point P[], int n)
{
    float min = FLT_MAX;
    for (int i = 0; i < n; ++i)
        for (int j = i+1; j < n; ++j)
            if (dist(P[i], P[j]) < min)
                min = dist(P[i], P[j]);
    return min;
}

// A utility function to find minimum of two float values
float min(float x, float y)
```

```

{
    return (x < y)? x : y;
}

// A utility function to find the distance between the closest points of
// strip of given size. All points in strip[] are sorted according to
// y coordinate. They all have an upper bound on minimum distance as d.
// Note that this method seems to be a O(n^2) method, but it's a O(n)
// method as the inner loop runs at most 6 times
float stripClosest(Point strip[], int size, float d)
{
    float min = d; // Initialize the minimum distance as d

    qsort(strip, size, sizeof(Point), compareY);

    // Pick all points one by one and try the next points till the difference
    // between y coordinates is smaller than d.
    // This is a proven fact that this loop runs at most 6 times
    for (int i = 0; i < size; ++i)
        for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
            if (dist(strip[i],strip[j]) < min)
                min = dist(strip[i], strip[j]);

    return min;
}

// A recursive function to find the smallest distance. The array P contains
// all points sorted according to x coordinate
float closestUtil(Point P[], int n)
{
    // If there are 2 or 3 points, then use brute force
    if (n <= 3)
        return bruteForce(P, n);

    // Find the middle point
    int mid = n/2;
    Point midPoint = P[mid];

    // Consider the vertical line passing through the middle point
    // calculate the smallest distance dl on left of middle point and
    // dr on right side
    float dl = closestUtil(P, mid);
    float dr = closestUtil(P + mid, n-mid);

    // Find the smaller of two distances
    float d = min(dl, dr);
}

```

```

// Build an array strip[] that contains points close (closer than d)
// to the line passing through the middle point
Point strip[n];
int j = 0;
for (int i = 0; i < n; i++)
    if (abs(P[i].x - midPoint.x) < d)
        strip[j] = P[i], j++;

// Find the closest points in strip. Return the minimum of d and closest
// distance is strip[]
return min(d, stripClosest(strip, j, d));
}

// The main function that finds the smallest distance
// This method mainly uses closestUtil()
float closest(Point P[], int n)
{
    qsort(P, n, sizeof(Point), compareX);

    // Use recursive function closestUtil() to find the smallest distance
    return closestUtil(P, n);
}

// Driver program to test above functions
int main()
{
    Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};
    int n = sizeof(P) / sizeof(P[0]);
    printf("The smallest distance is %f ", closest(P, n));
    return 0;
}

```

Output:

The smallest distance is 1.414214

Time Complexity Let Time complexity of above algorithm be $T(n)$. Let us assume that we use a $O(n\log n)$ sorting algorithm. The above algorithm divides all points in two sets and recursively calls for two sets. After dividing, it finds the strip in $O(n)$ time, sorts the strip in $O(n\log n)$ time and finally finds the closest points in strip in $O(n)$ time. So $T(n)$ can be expressed as follows

$$\begin{aligned}
 T(n) &= 2T(n/2) + O(n) + O(n\log n) + O(n) \\
 T(n) &= 2T(n/2) + O(n\log n) \\
 T(n) &= T(n \times \log n \times \log n)
 \end{aligned}$$

Notes

- 1) Time complexity can be improved to $O(n\log n)$ by optimizing step 5 of the above algorithm. We will soon be discussing the optimized solution in a separate post.

2) The code finds smallest distance. It can be easily modified to find the points with smallest distance.

3) The code uses quick sort which can be $O(n^2)$ in worst case. To have the upper bound as $O(n (\log n)^2)$, a $O(n \log n)$ sorting algorithm like merge sort or heap sort can be used

References:

<http://www.cs.umd.edu/class/fall2013/cmsc451/Lects/lect10.pdf>

<http://www.youtube.com/watch?v=vS4Zn1a9KUc>

<http://www.youtube.com/watch?v=T3T7T8Ym20M>

http://en.wikipedia.org/wiki/Closest_pair_of_points_problem

Source

<https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-and-conquer-algorithm/>

Chapter 46

Closest Pair of Points | O(nlogn) Implementation

Closest Pair of Points | O(nlogn) Implementation - GeeksforGeeks

We are given an array of n points in the plane, and the problem is to find out the closest pair of points in the array. This problem arises in a number of applications. For example, in air-traffic control, you may want to monitor planes that come too close together, since this may indicate a possible collision. Recall the following formula for distance between two points p and q.

$$d(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$$

We have discussed a [divide and conquer solution](#) for this problem. The time complexity of the implementation provided in the previous post is $O(n (\text{Log}n)^2)$. In this post, we discuss an implementation with time complexity as $O(n\text{Log}n)$.

Following is a recap of the algorithm discussed in the previous post.

- 1) We sort all points according to x coordinates.
- 2) Divide all points in two halves.
- 3) Recursively find the smallest distances in both subarrays.
- 4) Take the minimum of two smallest distances. Let the minimum be d.
- 5) Create an array strip[] that stores all points which are at most d distance away from the middle line dividing the two sets.
- 6) Find the smallest distance in strip[].
- 7) Return the minimum of d and the smallest distance calculated in above step 6.

The great thing about the above approach is, if the array strip[] is sorted according to y coordinate, then we can find the smallest distance in strip[] in $O(n)$ time. In the implementation discussed in previous post, strip[] was explicitly sorted in every recursive call that made the time complexity $O(n (\text{Log}n)^2)$, assuming that the sorting step takes $O(n\text{Log}n)$.

time.

In this post, we discuss an implementation where the time complexity is $O(n\log n)$. The idea is to presort all points according to y coordinates. Let the sorted array be $Py[]$. When we make recursive calls, we need to divide points of $Py[]$ also according to the vertical line. We can do that by simply processing every point and comparing its x coordinate with x coordinate of middle line.

Following is C++ implementation of $O(n\log n)$ approach.

```
// A divide and conquer program in C++ to find the smallest distance from a
// given set of points.

#include <iostream>
#include <float.h>
#include <stdlib.h>
#include <math.h>
using namespace std;

// A structure to represent a Point in 2D plane
struct Point
{
    int x, y;
};

/* Following two functions are needed for library function qsort().
   Refer: http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */

// Needed to sort array of points according to X coordinate
int compareX(const void* a, const void* b)
{
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->x - p2->x);
}

// Needed to sort array of points according to Y coordinate
int compareY(const void* a, const void* b)
{
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->y - p2->y);
}

// A utility function to find the distance between two points
float dist(Point p1, Point p2)
{
    return sqrt( (p1.x - p2.x)*(p1.x - p2.x) +
                 (p1.y - p2.y)*(p1.y - p2.y)
               );
}
```

```

// A Brute Force method to return the smallest distance between two points
// in P[] of size n
float bruteForce(Point P[], int n)
{
    float min = FLT_MAX;
    for (int i = 0; i < n; ++i)
        for (int j = i+1; j < n; ++j)
            if (dist(P[i], P[j]) < min)
                min = dist(P[i], P[j]);
    return min;
}

// A utility function to find minimum of two float values
float min(float x, float y)
{
    return (x < y)? x : y;
}

// A utility function to find the distance between the closest points of
// strip of given size. All points in strip[] are sorted according to
// y coordinate. They all have an upper bound on minimum distance as d.
// Note that this method seems to be a  $O(n^2)$  method, but it's a  $O(n)$ 
// method as the inner loop runs at most 6 times
float stripClosest(Point strip[], int size, float d)
{
    float min = d; // Initialize the minimum distance as d

    // Pick all points one by one and try the next points till the difference
    // between y coordinates is smaller than d.
    // This is a proven fact that this loop runs at most 6 times
    for (int i = 0; i < size; ++i)
        for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
            if (dist(strip[i],strip[j]) < min)
                min = dist(strip[i], strip[j]);

    return min;
}

// A recursive function to find the smallest distance. The array Px contains
// all points sorted according to x coordinates and Py contains all points
// sorted according to y coordinates
float closestUtil(Point Px[], Point Py[], int n)
{
    // If there are 2 or 3 points, then use brute force
    if (n <= 3)
        return bruteForce(Px, n);
}

```

```

// Find the middle point
int mid = n/2;
Point midPoint = Px[mid];

// Divide points in y sorted array around the vertical line.
// Assumption: All x coordinates are distinct.
Point Pyl[mid+1]; // y sorted points on left of vertical line
Point Pyr[n-mid-1]; // y sorted points on right of vertical line
int li = 0, ri = 0; // indexes of left and right subarrays
for (int i = 0; i < n; i++)
{
    if (Py[i].x <= midPoint.x)
        Pyl[li++] = Py[i];
    else
        Pyr[ri++] = Py[i];
}

// Consider the vertical line passing through the middle point
// calculate the smallest distance dl on left of middle point and
// dr on right side
float dl = closestUtil(Px, Pyl, mid);
float dr = closestUtil(Px + mid, Pyr, n-mid);

// Find the smaller of two distances
float d = min(dl, dr);

// Build an array strip[] that contains points close (closer than d)
// to the line passing through the middle point
Point strip[n];
int j = 0;
for (int i = 0; i < n; i++)
    if (abs(Py[i].x - midPoint.x) < d)
        strip[j] = Py[i], j++;

// Find the closest points in strip. Return the minimum of d and closest
// distance is strip[]
return min(d, stripClosest(strip, j, d));
}

// The main function that finds the smallest distance
// This method mainly uses closestUtil()
float closest(Point P[], int n)
{
    Point Px[n];
    Point Py[n];
    for (int i = 0; i < n; i++)
    {

```

```
Px[i] = P[i];
Py[i] = P[i];
}

qsort(Px, n, sizeof(Point), compareX);
qsort(Py, n, sizeof(Point), compareY);

// Use recursive function closestUtil() to find the smallest distance
return closestUtil(Px, Py, n);
}

// Driver program to test above functions
int main()
{
    Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};
    int n = sizeof(P) / sizeof(P[0]);
    cout << "The smallest distance is " << closest(P, n);
    return 0;
}
```

Output:

The smallest distance is 1.41421

Time Complexity: Let Time complexity of above algorithm be $T(n)$. Let us assume that we use a $O(n\log n)$ sorting algorithm. The above algorithm divides all points in two sets and recursively calls for two sets. After dividing, it finds the strip in $O(n)$ time. Also, it takes $O(n)$ time to divide the Py array around the mid vertical line. Finally finds the closest points in strip in $O(n)$ time. So $T(n)$ can be expressed as follows

$$T(n) = 2T(n/2) + O(n) + O(n) + O(n)$$

$$T(n) = 2T(n/2) + O(n)$$

$$T(n) = T(n\log n)$$

References:

<http://www.cs.umd.edu/class/fall2013/cmsc451/Lects/lect10.pdf>

<http://www.youtube.com/watch?v=vS4Zn1a9KUc>

<http://www.youtube.com/watch?v=T3T7T8Ym20M>

http://en.wikipedia.org/wiki/Closest_pair_of_points_problem

Source

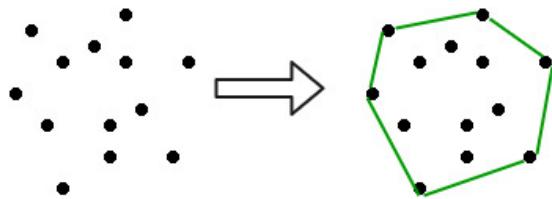
<https://www.geeksforgeeks.org/closest-pair-of-points-onlogn-implementation/>

Chapter 47

Convex Hull using Divide and Conquer Algorithm

Convex Hull using Divide and Conquer Algorithm - GeeksforGeeks

A convex hull is the smallest convex polygon containing all the given points.



Input is an array of points specified by their x and y coordinates. The output is the convex hull of this set of points.

Examples:

```
Input : points[] = {(0, 0), (0, 4), (-4, 0), (5, 0),
                    (0, -6), (1, 0)};
Output : (-4, 0), (5, 0), (0, -6), (0, 4)
```

Pre-requisite:

[Tangents between two convex polygons](#)

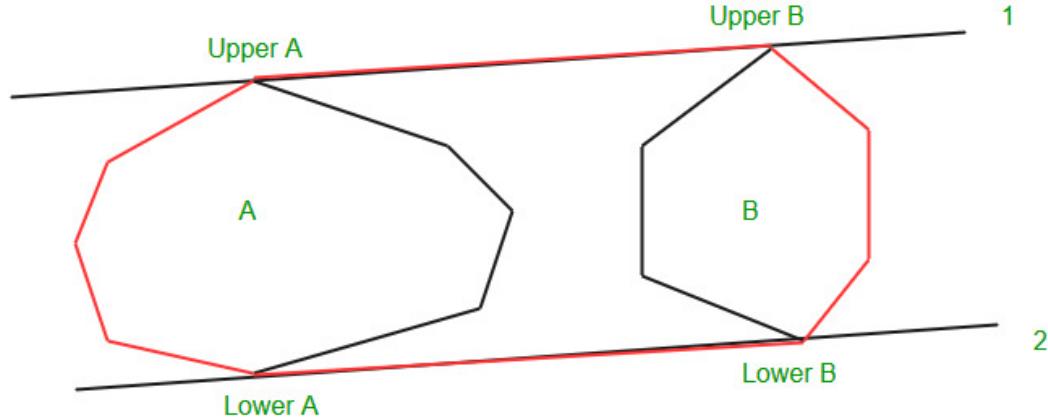
Algorithm:

Given the set of points for which we have to find the convex hull. Suppose we know the convex hull of the left half points and the right half points, then the problem now is to merge these two convex hulls and determine the convex hull for the complete set.

This can be done by finding the upper and lower tangent to the right and left convex hulls. This is illustrated here [Tangents between two convex polygons](#)

Let the left convex hull be a and the right convex hull be b. Then the lower and upper tangents are named as 1 and 2 respectively, as shown in the figure.

Then the red outline shows the final convex hull.



Now the problem remains, how to find the convex hull for the left and right half. Now recursion comes into the picture, we divide the set of points until the number of points in the set is very small, say 5, and we can find the convex hull for these points by the brute algorithm. The merging of these halves would result in the convex hull for the complete set of points.

Note:

We have used the brute algorithm to find the convex hull for a small number of points and

it has a time complexity of $O(n^2)$. But some people suggest the following, the convex hull for 3 or fewer points is the complete set of points. This is correct but the problem comes when we try to merge a left convex hull of 2 points and right convex hull of 3 points, then the program gets trapped in an infinite loop in some special cases. So, to get rid of

this problem I directly found the convex hull for 5 or fewer points by $O(n^2)$ algorithm, which is somewhat greater but does not affect the overall complexity of the algorithm.

```
// A divide and conquer program to find convex
// hull of a given set of points.
#include<bits/stdc++.h>
using namespace std;

// stores the centre of polygon (It is made
// global because it is used in compare function)
pair<int, int> mid;

// determines the quadrant of a point
// (used in compare())
int quad(pair<int, int> p)
{
    if (p.first >= 0 && p.second >= 0)
        return 1;
    else if (p.first <= 0 && p.second >= 0)
        return 2;
    else if (p.first <= 0 && p.second <= 0)
        return 3;
    else
        return 4;
}
```

```

if (p.first <= 0 && p.second >= 0)
    return 2;
if (p.first <= 0 && p.second <= 0)
    return 3;
return 4;
}

// Checks whether the line is crossing the polygon
int orientation(pair<int, int> a, pair<int, int> b,
                pair<int, int> c)
{
    int res = (b.second-a.second)*(c.first-b.first) -
              (c.second-b.second)*(b.first-a.first);

    if (res == 0)
        return 0;
    if (res > 0)
        return 1;
    return -1;
}

// compare function for sorting
bool compare(pair<int, int> p1, pair<int, int> q1)
{
    pair<int, int> p = make_pair(p1.first - mid.first,
                                  p1.second - mid.second);
    pair<int, int> q = make_pair(q1.first - mid.first,
                                  q1.second - mid.second);

    int one = quad(p);
    int two = quad(q);

    if (one != two)
        return (one < two);
    return (p.second*q.first < q.second*p.first);
}

// Finds upper tangent of two polygons 'a' and 'b'
// represented as two vectors.
vector<pair<int, int>> merger(vector<pair<int, int>> a,
                                 vector<pair<int, int>> b)
{
    // n1 -> number of points in polygon a
    // n2 -> number of points in polygon b
    int n1 = a.size(), n2 = b.size();

    int ia = 0, ib = 0;
    for (int i=1; i<n1; i++)

```

```

        if (a[i].first > a[ia].first)
            ia = i;

        // ib -> leftmost point of b
        for (int i=1; i<n2; i++)
            if (b[i].first < b[ib].first)
                ib=i;

        // finding the upper tangent
        int inda = ia, indb = ib;
        bool done = 0;
        while (!done)
        {
            done = 1;
            while (orientation(b[indb], a[inda], a[(inda+1)%n1]) >=0)
                inda = (inda + 1) % n1;

            while (orientation(a[inda], b[indb], b[(n2+indb-1)%n2]) <=0)
            {
                indb = (n2+indb-1)%n2;
                done = 0;
            }
        }

        int uppera = inda, upperb = indb;
        inda = ia, indb=ib;
        done = 0;
        int g = 0;
        while (!done)//finding the lower tangent
        {
            done = 1;
            while (orientation(a[inda], b[indb], b[(indb+1)%n2])>=0)
                indb=(indb+1)%n2;

            while (orientation(b[indb], a[inda], a[(n1+inda-1)%n1])<=0)
            {
                inda=(n1+inda-1)%n1;
                done=0;
            }
        }

        int lowera = inda, lowerb = indb;
        vector<pair<int, int>> ret;

        //ret contains the convex hull after merging the two convex hulls
        //with the points sorted in anti-clockwise order
        int ind = uppera;
        ret.push_back(a[uppera]);
    }
}

```

```

while (ind != lowera)
{
    ind = (ind+1)%n1;
    ret.push_back(a[ind]);
}

ind = lowerb;
ret.push_back(b[lowerb]);
while (ind != upperb)
{
    ind = (ind+1)%n2;
    ret.push_back(b[ind]);
}
return ret;
}

// Brute force algorithm to find convex hull for a set
// of less than 6 points
vector<pair<int, int>> bruteHull(vector<pair<int, int>> a)
{
    // Take any pair of points from the set and check
    // whether it is the edge of the convex hull or not.
    // if all the remaining points are on the same side
    // of the line then the line is the edge of convex
    // hull otherwise not
    set<pair<int, int> >s;

    for (int i=0; i<a.size(); i++)
    {
        for (int j=i+1; j<a.size(); j++)
        {
            int x1 = a[i].first, x2 = a[j].first;
            int y1 = a[i].second, y2 = a[j].second;

            int a1 = y1-y2;
            int b1 = x2-x1;
            int c1 = x1*y2-y1*x2;
            int pos = 0, neg = 0;
            for (int k=0; k<a.size(); k++)
            {
                if (a1*a[k].first+b1*a[k].second+c1 <= 0)
                    neg++;
                if (a1*a[k].first+b1*a[k].second+c1 >= 0)
                    pos++;
            }
            if (pos == a.size() || neg == a.size())
            {

```

```

        s.insert(a[i]);
        s.insert(a[j]);
    }
}

vector<pair<int, int>>ret;
for (auto e:s)
    ret.push_back(e);

// Sorting the points in the anti-clockwise order
mid = {0, 0};
int n = ret.size();
for (int i=0; i<n; i++)
{
    mid.first += ret[i].first;
    mid.second += ret[i].second;
    ret[i].first *= n;
    ret[i].second *= n;
}
sort(ret.begin(), ret.end(), compare);
for (int i=0; i<n; i++)
    ret[i] = make_pair(ret[i].first/n, ret[i].second/n);

return ret;
}

// Returns the convex hull for the given set of points
vector<pair<int, int>> divide(vector<pair<int, int>> a)
{
    // If the number of points is less than 6 then the
    // function uses the brute algorithm to find the
    // convex hull
    if (a.size() <= 5)
        return bruteHull(a);

    // left contains the left half points
    // right contains the right half points
    vector<pair<int, int>>left, right;
    for (int i=0; i<a.size()/2; i++)
        left.push_back(a[i]);
    for (int i=a.size()/2; i<a.size(); i++)
        right.push_back(a[i]);

    // convex hull for the left and right sets
    vector<pair<int, int>>left_hull = divide(left);
    vector<pair<int, int>>right_hull = divide(right);
}

```

```
// merging the convex hulls
return merger(left_hull, right_hull);
}

// Driver code
int main()
{
    vector<pair<int, int> > a;
    a.push_back(make_pair(0, 0));
    a.push_back(make_pair(1, -4));
    a.push_back(make_pair(-1, -5));
    a.push_back(make_pair(-5, -3));
    a.push_back(make_pair(-3, -1));
    a.push_back(make_pair(-1, -3));
    a.push_back(make_pair(-2, -2));
    a.push_back(make_pair(-1, -1));
    a.push_back(make_pair(-2, -1));
    a.push_back(make_pair(-1, 1));

    int n = a.size();

    // sorting the set of points according
    // to the x-coordinate
    sort(a.begin(), a.end());
    vector<pair<int, int> >ans = divide(a);

    cout << "convex hull:\n";
    for (auto e:ans)
        cout << e.first << " "
            << e.second << endl;

    return 0;
}
```

Output:

```
Convex Hull:
-5 -3
-1 -5
1 -4
0 0
-1 1
```

Time Complexity: The merging of the left and the right convex hulls take $O(n)$ time and as we are dividing the points into two equal parts, so the time complexity of the above algorithm is $O(n * \log n)$.

Related Articles :

- Convex Hull | Set 1 (Jarvis's Algorithm or Wrapping)
- Convex Hull | Set 2 (Graham Scan)
- Quickhull Algorithm for Convex Hull

Source

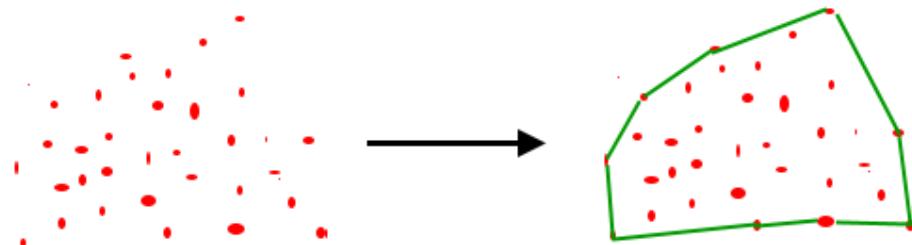
<https://www.geeksforgeeks.org/convex-hull-using-divide-and-conquer-algorithm/>

Chapter 48

Convex Hull | Set 1 (Jarvis's Algorithm or Wrapping)

[Convex Hull | Set 1 \(Jarvis's Algorithm or Wrapping\) - GeeksforGeeks](#)

Given a set of points in the plane. the convex hull of the set is the smallest convex polygon that contains all the points of it.



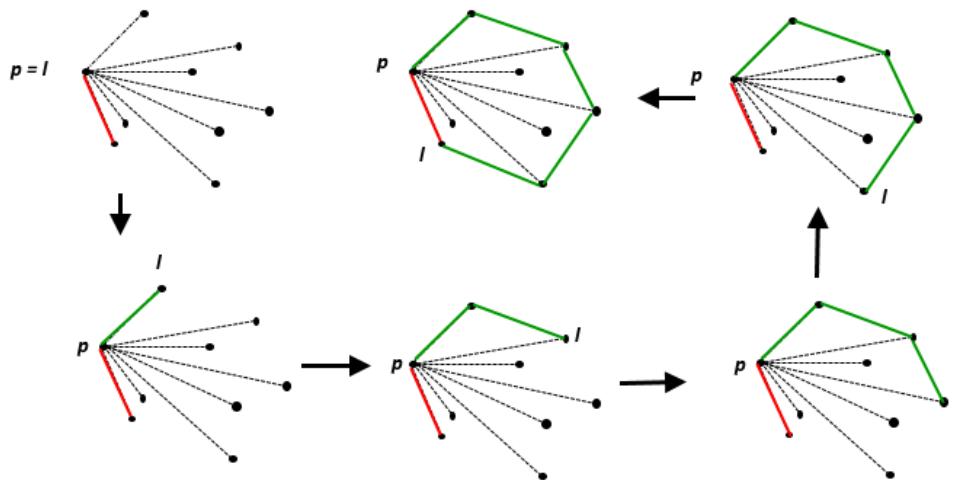
We strongly recommend to see the following post first.

[How to check if two given line segments intersect?](#)

The idea of Jarvis's Algorithm is simple, we start from the leftmost point (or point with minimum x coordinate value) and we keep wrapping points in counterclockwise direction. The big question is, given a point p as current point, how to find the next point in output? The idea is to use [orientation\(\)](#) here. Next point is selected as the point that beats all other points at counterclockwise orientation, i.e., next point is q if for any other point r, we have "orientation(p, r, q) = counterclockwise". Following is the detailed algorithm.

- 1) Initialize p as leftmost point.
- 2) Do following while we don't come back to the first (or leftmost) point.

-a) The next point q is the point such that the triplet (p, q, r) is counterclockwise for any other point r.
-b) next[p] = q (Store q as next of p in the output convex hull).
-c) p = q (Set p as q for next iteration).



The execution of jarvis's March

Below is the implementation of above algorithm.

C++

```
// A C++ program to find convex hull of a set of points. Refer
// https://www.geeksforgeeks.org/orientation-3-ordered-points/
// for explanation of orientation()
#include <bits/stdc++.h>
using namespace std;

struct Point
{
    int x, y;
};

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);
    if (val == 0)
        return 0;
    else if (val < 0)
        return 1;
    else
        return 2;
}
```

```

        if (val == 0) return 0; // colinear
        return (val > 0)? 1: 2; // clock or counterclock wise
    }

// Prints convex hull of a set of n points.
void convexHull(Point points[], int n)
{
    // There must be at least 3 points
    if (n < 3) return;

    // Initialize Result
    vector<Point> hull;

    // Find the leftmost point
    int l = 0;
    for (int i = 1; i < n; i++)
        if (points[i].x < points[l].x)
            l = i;

    // Start from leftmost point, keep moving counterclockwise
    // until reach the start point again. This loop runs O(h)
    // times where h is number of points in result or output.
    int p = l, q;
    do
    {
        // Add current point to result
        hull.push_back(points[p]);

        // Search for a point 'q' such that orientation(p, x,
        // q) is counterclockwise for all points 'x'. The idea
        // is to keep track of last visited most counterclock-
        // wise point in q. If any point 'i' is more counterclock-
        // wise than q, then update q.
        q = (p+1)%n;
        for (int i = 0; i < n; i++)
        {
            // If i is more counterclockwise than current q, then
            // update q
            if (orientation(points[p], points[i], points[q]) == 2)
                q = i;
        }

        // Now q is the most counterclockwise with respect to p
        // Set p as q for next iteration, so that q is added to
        // result 'hull'
        p = q;
    }
}
```

```
} while (p != 1); // While we don't come to first point

// Print Result
for (int i = 0; i < hull.size(); i++)
    cout << "(" << hull[i].x << ", "
          << hull[i].y << ")\n";
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 3}, {2, 2}, {1, 1}, {2, 1},
                      {3, 0}, {0, 0}, {3, 3}};
    int n = sizeof(points)/sizeof(points[0]);
    convexHull(points, n);
    return 0;
}
```

Java

```
// Java program to find convex hull of a set of points. Refer
// https://www.geeksforgeeks.org/orientation-3-ordered-points/
// for explanation of orientation()
import java.util.*;

class Point
{
    int x, y;
    Point(int x, int y){
        this.x=x;
        this.y=y;
    }
}

class GFG {

    // To find orientation of ordered triplet (p, q, r).
    // The function returns following values
    // 0 --> p, q and r are colinear
    // 1 --> Clockwise
    // 2 --> Counterclockwise
    public static int orientation(Point p, Point q, Point r)
    {
        int val = (q.y - p.y) * (r.x - q.x) -
                  (q.x - p.x) * (r.y - q.y);

        if (val == 0) return 0; // collinear
        return (val > 0)? 1: 2; // clock or counterclock wise
    }
}
```

```
}

// Prints convex hull of a set of n points.
public static void convexHull(Point points[], int n)
{
    // There must be at least 3 points
    if (n < 3) return;

    // Initialize Result
    Vector<Point> hull = new Vector<Point>();

    // Find the leftmost point
    int l = 0;
    for (int i = 1; i < n; i++)
        if (points[i].x < points[l].x)
            l = i;

    // Start from leftmost point, keep moving
    // counterclockwise until reach the start point
    // again. This loop runs O(h) times where h is
    // number of points in result or output.
    int p = l, q;
    do
    {
        // Add current point to result
        hull.add(points[p]);

        // Search for a point 'q' such that
        // orientation(p, x, q) is counterclockwise
        // for all points 'x'. The idea is to keep
        // track of last visited most counterclock-
        // wise point in q. If any point 'i' is more
        // counterclock-wise than q, then update q.
        q = (p + 1) % n;

        for (int i = 0; i < n; i++)
        {
            // If i is more counterclockwise than
            // current q, then update q
            if (orientation(points[p], points[i], points[q])
                == 2)
                q = i;
        }

        // Now q is the most counterclockwise with
        // respect to p. Set p as q for next iteration,
        // so that q is added to result 'hull'
        p = q;
    }
}
```

```
    } while (p != 1); // While we don't come to first
                  // point

    // Print Result
    for (Point temp : hull)
        System.out.println("(" + temp.x + ", " +
                           temp.y + ")");
}

/* Driver program to test above function */
public static void main(String[] args)
{
    Point points[] = new Point[7];
    points[0]=new Point(0, 3);
    points[1]=new Point(2, 3);
    points[2]=new Point(1, 1);
    points[3]=new Point(2, 1);
    points[4]=new Point(3, 0);
    points[5]=new Point(0, 0);
    points[6]=new Point(3, 3);

    int n = points.length;
    convexHull(points, n);

}
}

// This code is contributed by Arnav Kr. Mandal.
```

Output: The output is points of the convex hull.

```
(0, 3)
(0, 0)
(3, 0)
(3, 3)
```

Time Complexity: For every point on the hull we examine all the other points to determine the next point. Time complexity is $?(m * n)$ where n is number of input points and m is number of output or hull points ($m \leq n$). In worst case, time complexity is $O(n^2)$. The worst case occurs when all the points are on the hull ($m = n$)

Set 2- Convex Hull (Graham Scan)

Sources:

<http://www.cs.uiuc.edu/~jeffe/teaching/373/notes/x05-convexhull.pdf>
<http://www.dcs.gla.ac.uk/~pat/52233/slides/Hull1x1.pdf>

Source

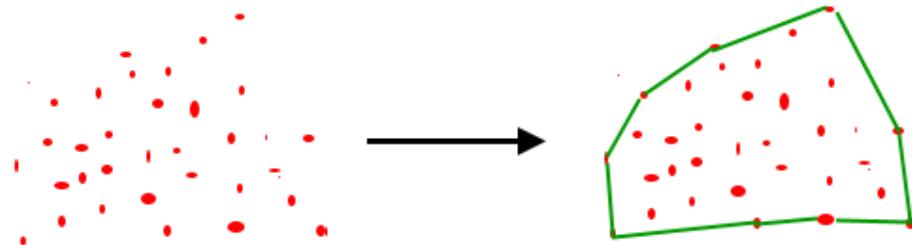
<https://www.geeksforgeeks.org/convex-hull-set-1-jarviss-algorithm-or-wrapping/>

Chapter 49

Convex Hull | Set 2 (Graham Scan)

[Convex Hull | Set 2 \(Graham Scan\) - GeeksforGeeks](#)

Given a set of points in the plane. the convex hull of the set is the smallest convex polygon that contains all the points of it.



We strongly recommend to see the following post first.

[How to check if two given line segments intersect?](#)

We have discussed [Jarvis's Algorithm](#) for Convex Hull. Worst case time complexity of Jarvis's Algorithm is $O(n^2)$. Using Graham's scan algorithm, we can find Convex Hull in $O(n \log n)$ time. Following is Graham's algorithm

Let $\text{points}[0..n-1]$ be the input array.

- 1) Find the bottom-most point by comparing y coordinate of all points. If there are two points with same y value, then the point with smaller x coordinate value is considered. Let the bottom-most point be P_0 . Put P_0 at first position in output hull.

2) Consider the remaining $n-1$ points and sort them by polar angle in counterclockwise order around $\text{points}[0]$. If polar angle of two points is same, then put the nearest point first.

3 After sorting, check if two or more points have same angle. If two more points have same angle, then remove all same angle points except the point farthest from P_0 . Let the size of new array be m .

4) If m is less than 3, return (Convex Hull not possible)

5) Create an empty stack ‘ S ’ and push $\text{points}[0]$, $\text{points}[1]$ and $\text{points}[2]$ to S .

6) Process remaining $m-3$ points one by one. Do following for every point ‘ $\text{points}[i]$ ’

4.1) Keep removing points from stack while [orientation](#)of following 3 points is not counterclockwise (or they don’t make a left turn).

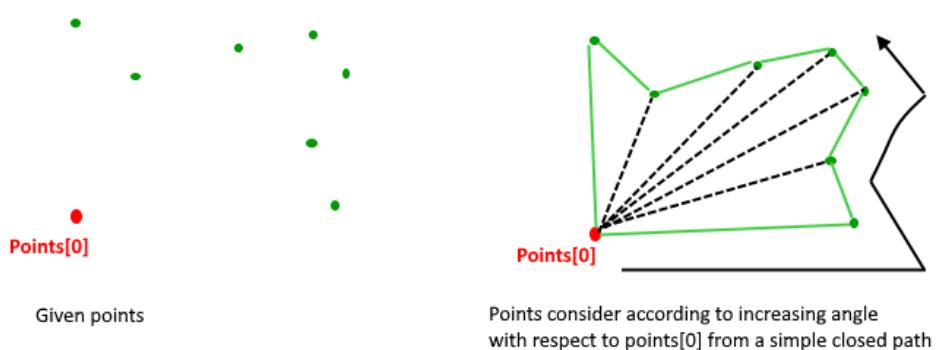
- a) Point next to top in stack
- b) Point at the top of stack
- c) $\text{points}[i]$

4.2) Push $\text{points}[i]$ to S

5) Print contents of S

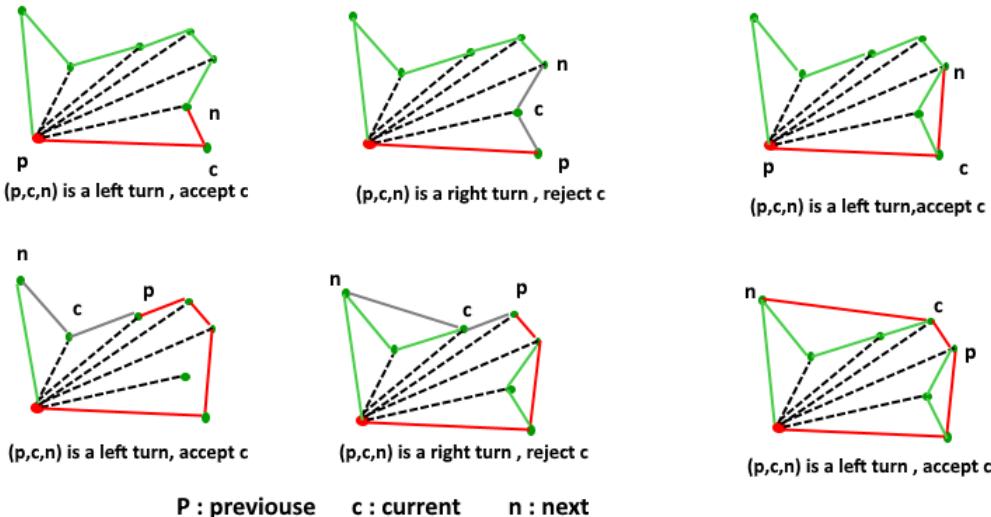
The above algorithm can be divided in two phases.

Phase 1 (Sort points): We first find the bottom-most point. The idea is to pre-process points by sorting them with respect to the bottom-most point. Once the points are sorted, they form a simple closed path (See following diagram).



What should be the sorting criteria? computation of actual angles would be inefficient since trigonometric functions are not simple to evaluate. The idea is to use the orientation to compare angles without actually computing them (See the `compare()` function below)

Phase 2 (Accept or Reject Points): Once we have the closed path, the next step is to traverse the path and remove concave points on this path. How to decide which point to remove and which to keep? Again, [orientation](#)helps here. The first two points in sorted array are always part of Convex Hull. For remaining points, we keep track of recent three points, and find the angle formed by them. Let the three points be $\text{prev}(p)$, $\text{curr}(c)$ and $\text{next}(n)$. If orientation of these points (considering them in same order) is not counterclockwise, we discard c , otherwise we keep it. Following diagram shows step by step process of this phase



In the above algorithm and below code , a stack of points is used to store convex hull points. With reference to the code ,p is next-to-top in stack, c is top of stack and n is point[i]

Following is C++ implementation of the above algorithm.

```

// A C++ program to find convex hull of a set of points. Refer
// https://www.geeksforgeeks.org/orientation-3-ordered-points/
// for explanation of orientation()
#include <iostream>
#include <stack>
#include <stdlib.h>
using namespace std;

struct Point
{
    int x, y;
};

// A globle point needed for sorting points with reference
// to the first point Used in compare function of qsort()
Point p0;

// A utility function to find next to top in a stack
Point nextToTop(stack<Point> &S)
{
    Point p = S.top();
    S.pop();
    Point res = S.top();
    S.push(p);
    return res;
}

```

```

}

// A utility function to swap two points
int swap(Point &p1, Point &p2)
{
    Point temp = p1;
    p1 = p2;
    p2 = temp;
}

// A utility function to return square of distance
// between p1 and p2
int distSq(Point p1, Point p2)
{
    return (p1.x - p2.x)*(p1.x - p2.x) +
           (p1.y - p2.y)*(p1.y - p2.y);
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0; // colinear
    return (val > 0)? 1: 2; // clock or counterclock wise
}

// A function used by library function qsort() to sort an array of
// points with respect to the first point
int compare(const void *vp1, const void *vp2)
{
    Point *p1 = (Point *)vp1;
    Point *p2 = (Point *)vp2;

    // Find orientation
    int o = orientation(p0, *p1, *p2);
    if (o == 0)
        return (distSq(p0, *p2) >= distSq(p0, *p1))? -1 : 1;

    return (o == 2)? -1: 1;
}

// Prints convex hull of a set of n points.

```

```

void convexHull(Point points[], int n)
{
    // Find the bottommost point
    int ymin = points[0].y, min = 0;
    for (int i = 1; i < n; i++)
    {
        int y = points[i].y;

        // Pick the bottom-most or chose the left
        // most point in case of tie
        if ((y < ymin) || (ymin == y &&
            points[i].x < points[min].x))
            ymin = points[i].y, min = i;
    }

    // Place the bottom-most point at first position
    swap(points[0], points[min]);

    // Sort n-1 points with respect to the first point.
    // A point p1 comes before p2 in sorted output if p2
    // has larger polar angle (in counterclockwise
    // direction) than p1
    p0 = points[0];
    qsort(&points[1], n-1, sizeof(Point), compare);

    // If two or more points make same angle with p0,
    // Remove all but the one that is farthest from p0
    // Remember that, in above sorting, our criteria was
    // to keep the farthest point at the end when more than
    // one points have same angle.
    int m = 1; // Initialize size of modified array
    for (int i=1; i<n; i++)
    {
        // Keep removing i while angle of i and i+1 is same
        // with respect to p0
        while (i < n-1 && orientation(p0, points[i],
                                         points[i+1]) == 0)
            i++;

        points[m] = points[i];
        m++; // Update size of modified array
    }

    // If modified array of points has less than 3 points,
    // convex hull is not possible
    if (m < 3) return;
}

```

```

// Create an empty stack and push first three points
// to it.
stack<Point> S;
S.push(points[0]);
S.push(points[1]);
S.push(points[2]);

// Process remaining n-3 points
for (int i = 3; i < m; i++)
{
    // Keep removing top while the angle formed by
    // points next-to-top, top, and points[i] makes
    // a non-left turn
    while (orientation(nextToTop(S), S.top(), points[i]) != 2)
        S.pop();
    S.push(points[i]);
}

// Now stack has the output points, print contents of stack
while (!S.empty())
{
    Point p = S.top();
    cout << "(" << p.x << ", " << p.y << ")" << endl;
    S.pop();
}
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},
                      {0, 0}, {1, 2}, {3, 1}, {3, 3}};
    int n = sizeof(points)/sizeof(points[0]);
    convexHull(points, n);
    return 0;
}

```

Output:

```
(0, 3)
(4, 4)
(3, 1)
(0, 0)
```

Time Complexity: Let n be the number of input points. The algorithm takes $O(n \log n)$ time if we use a $O(n \log n)$ sorting algorithm.

The first step (finding the bottom-most point) takes $O(n)$ time. The second step (sorting

points) takes $O(n \log n)$ time. Third step takes $O(n)$ time. In third step, every element is pushed and popped at most one time. So the sixth step to process points one by one takes $O(n)$ time, assuming that the stack operations take $O(1)$ time. Overall complexity is $O(n) + O(n \log n) + O(n) + O(n)$ which is $O(n \log n)$

References:

Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest
<http://www.dcs.gla.ac.uk/~pat/52233/slides/Hull1x1.pdf>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Source

<https://www.geeksforgeeks.org/convex-hull-set-2-graham-scan/>

Chapter 50

Coordinates of rectangle with given points lie inside

Coordinates of rectangle with given points lie inside - GeeksforGeeks

Given two arrays X[] and Y[] with n-elements, where (Xi, Yi) represent a point on coordinate system, find the smallest rectangle such that all points from given input lie inside that rectangle and sides of rectangle must be parallel to Coordinate axis. Print all four coordinates of obtained rectangle.

Note : n >= 4 (we have at least 4 points as input).

Examples :

```
Input : X[] = {1, 3, 0, 4}, y[] = {2, 1, 0, 2}
Output : {0, 0}, {0, 2}, {4, 2}, {4, 0}
```

```
Input : X[] = {3, 6, 1, 9, 13, 0, 4}, Y[] = {4, 2, 6, 5, 2, 3, 1}
Output : {0, 1}, {0, 6}, {13, 6}, {13, 1}
```

For finding the smallest rectangle you may take two of basic approach :

1. Consider origin of coordinate plane as smallest rectangle and then step by step keep expanding it as per value of coordinates of points if they don't lie inside the current rectangle. This concept requires a little of complex logic to find the exact smallest rectangle.
2. A very simple and efficient logic behind this is to find the smallest as well as largest x & y coordinates among all given points and then the all possible four combinations of these values result the four points of required rectangle as {Xmin, Ymin}, {Xmin, Ymax}, {Xmax, Ymax}, {Xmax, Ymin}.

C++

```
// Program to find smallest rectangle
// to conquer all points
#include <bits/stdc++.h>
using namespace std;

// function to print coordinate of smallest rectangle
void printRect(int X[], int Y[], int n)
{
    // find Xmax and Xmin
    int Xmax = *max_element(X, X + n);
    int Xmin = *min_element(X, X + n);

    // find Ymax and Ymin
    int Ymax = *max_element(Y, Y + n);
    int Ymin = *min_element(Y, Y + n);

    // print all four coordinates
    cout << "{" << Xmin << ", " << Ymin << "}" << endl;
    cout << "{" << Xmin << ", " << Ymax << "}" << endl;
    cout << "{" << Xmax << ", " << Ymax << "}" << endl;
    cout << "{" << Xmax << ", " << Ymin << "}" << endl;
}

// driver program
int main()
{
    int X[] = { 4, 3, 6, 1, -1, 12 };
    int Y[] = { 4, 1, 10, 3, 7, -1 };
    int n = sizeof(X) / sizeof(X[0]);

    printRect(X, Y, n);

    return 0;
}
```

Java

```
// Program to find smallest rectangle
// to conquer all points
import java.util.Arrays;
import java.util.Collections;

class GFG {

    // function to print coordinate of smallest rectangle
    static void printRect(Integer X[], Integer Y[], int n)
    {
```

```
// find Xmax and Xmin
int Xmax = Collections.max(Arrays.asList(X));
int Xmin = Collections.min(Arrays.asList(X));

// find Ymax and Ymin
int Ymax = Collections.max(Arrays.asList(Y));
int Ymin = Collections.min(Arrays.asList(Y));

// print all four coordinates
System.out.println("{ " + Xmin + ", " + Ymin + " }");
System.out.println("{ " + Xmin + ", " + Ymax + " }");
System.out.println("{ " + Xmax + ", " + Ymax + " }");
System.out.println("{ " + Xmax + ", " + Ymin + " }");
}

//Driver code
public static void main (String[] args)
{

    Integer X[] = { 4, 3, 6, 1, -1, 12 };
    Integer Y[] = { 4, 1, 10, 3, 7, -1 };
    int n = X.length;

    printRect(X, Y, n);
}
}

// This code is contributed by Anant Agarwal.
```

Python 3

```
# Program to find smallest rectangle
# to conquer all points

# function to print coordinate of smallest rectangle
def printRect(X, Y, n):

    # find Xmax and Xmin
    Xmax = max(X)
    Xmin = min(X)

    # find Ymax and Ymin
    Ymax = max(Y)
    Ymin = min(Y)

    # print all four coordinates
    print("{,Xmin, ,Ymin,}",sep="")
    print("{,Xmin, ,Ymax,}",sep="")
```

```
print("{",Xmax,", ",Ymax,"}",sep="")
print("{",Xmax,", ",Ymin,"}",sep="")  
  
# driver program
X = [4, 3, 6, 1, -1, 12]
Y = [4, 1, 10, 3, 7, -1]
n = len(X)
printRect(X, Y, n)
# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
// Program to find smallest rectangle
// to conquer all points
using System.Linq;
using System;  
  
public class GFG{  
  
    // function to print coordinate
    // of smallest rectangle
    static void printRect(int[] X,
                          int[] Y, int n)
    {  
  
        // find Xmax and Xmin
        int Xmax = X.Max();
        int Xmin = X.Min();  
  
        // find Ymax and Ymin
        int Ymax = Y.Max();
        int Ymin = Y.Min();  
  
        // print all four coordinates
        Console.WriteLine("{ " + Xmin + ", "
                        + Ymin + "}");
        Console.WriteLine("{ " + Xmin + ", "
                        + Ymax + "}");
        Console.WriteLine("{ " + Xmax + ", "
                        + Ymax + "}");
        Console.WriteLine("{ " + Xmax + ", "
                        + Ymin + "});  
    }  
}
```

```
// Driver code
static public void Main ()
{
    int[] X = { 4, 3, 6, 1, -1, 12 };
    int[] Y = { 4, 1, 10, 3, 7, -1 };
    int n = X.Length;

    printRect(X, Y, n);
}
}

// This code is contributed by Ajit.
```

PHP

```
<?php
// PHP Program to find smallest
// rectangle to conquer all points

// function to print coordinate
// of smallest rectangle
function printRect($X, $Y, $n)
{

    // find Xmax and Xmin
    $Xmax = max($X);
    $Xmin = min($X);

    // find Ymax and Ymin
    $Ymax = max($Y);
    $Ymin = min($Y);

    // print all four coordinates
    echo "{" , $Xmin , " , " , $Ymin , " }","\\n";
    echo "{" , $Xmin , " , " , $Ymax , " }" , "\\n";
    echo "{" , $Xmax , " , " , $Ymax , " }","\\n";
    echo "{" , $Xmax , " , " , $Ymin , " }" ;

}

// Driver Code
$X = array(4, 3, 6, 1, -1, 12);
$Y = array(4, 1, 10, 3, 7, -1);
$n = count($X);
printRect($X, $Y, $n);

// This code is contributed by anuj_67.
?>
```

Output:

```
{-1, -1}  
{-1, 10}  
{12, 10}  
{12, -1}
```

Improved By : [Smitha Dinesh Semwal](#), [jit_t](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/coordinates-rectangle-given-points-lie-inside/>

Chapter 51

Count Integral points inside a Triangle

Count Integral points inside a Triangle - GeeksforGeeks

Given three non-collinear integral points in XY plane, find the number of integral points inside the triangle formed by the three points. (A point in XY plane is said to be integral/lattice point if both its co-ordinates are integral).

Example:

Input: p = (0, 0), q = (0, 5) and r = (5,0)

Output: 6

Explanation: The points (1,1) (1,2) (1,3) (2,1) (2,2) and (3,1) are the integral points inside the triangle.

We can use the [Pick's theorem](#), which states that the following equation holds true for a simple Polygon.

Pick's Theorem:

$$A = I + (B/2) - 1$$

A ==> Area of Polygon

B ==> Number of integral points on edges of polygon

I ==> Number of integral points inside the polygon

Using the above formula, we can deduce,
 $I = (2A - B + 2) / 2$

We can find **A** (area of triangle) using below [Shoelace formula](#).

$$A = 1/2 * \text{abs}(x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2))$$

How to find B (number of integral points on edges of a triangle)?

We can find the number of integral points between any two vertex (V1, V2) of the triangle using the following algorithm.

1. If the edge formed by joining V1 and V2 is parallel to the X-axis, then the number of integral points between the vertices is :
 $\text{abs}(V1.y - V2.y) - 1$
2. Similarly if edge is parallel to the Y-axis, then the number of integral points in between is :
 $\text{abs}(V1.x - V2.y) - 1$
3. Else, we can find the integral points between the vertices using below formula:
 $\text{GCD}(\text{abs}(V1.x - V2.x), \text{abs}(V1.y - V2.y)) - 1$
The above formula is a well known fact and can be verified using simple geometry. (Hint: Shift the edge such that one of the vertex lies at the Origin.)

Please refer below link for detailed explanation.

<https://www.geeksforgeeks.org/number-integral-points-two-points/>

Below is C++ implementation of the above algorithm.

```
// C++ program to find Integral points inside a triangle
#include<bits/stdc++.h>
using namespace std;

// Class to represent an Integral point on XY plane.
class Point
{
public:
    int x, y;
```

```

Point(int a=0, int b=0):x(a),y(b) {}

//utility function to find GCD of two numbers
// GCD of a and b
int gcd(int a, int b)
{
    if (b == 0)
        return a;
    return gcd(b, a%b);
}

// Finds the no. of Integral points between
// two given points.
int getBoundaryCount(Point p, Point q)
{
    // Check if line parallel to axes
    if (p.x==q.x)
        return abs(p.y - q.y) - 1;
    if (p.y == q.y)
        return abs(p.x-q.x) - 1;

    return gcd(abs(p.x-q.x),abs(p.y-q.y))-1;
}

// Returns count of points inside the triangle
int getInternalCount(Point p, Point q, Point r)
{
    // 3 extra integer points for the vertices
    int BoundaryPoints = getBoundaryCount(p, q) +
                        getBoundaryCount(p, r) +
                        getBoundaryCount(q, r) + 3;

    // Calculate 2*A for the triangle
    int doubleArea = abs(p.x*(q.y - r.y) + q.x*(r.y - p.y) +
                         r.x*(p.y - q.y));

    // Use Pick's theorem to calculate the no. of Interior points
    return (doubleArea - BoundaryPoints + 2)/2;
}

// driver function to check the program.
int main()
{
    Point p(0, 0);
    Point q(5, 0);
    Point r(0, 5);
    cout << "Number of integral points inside given triangle is "
}

```

```
    << getInternalCount(p, q, r);  
    return 0;  
}
```

Output:

```
Number of integral points inside given triangle is 6
```

Source

<https://www.geeksforgeeks.org/count-integral-points-inside-a-triangle/>

Chapter 52

Count maximum points on same line

Count maximum points on same line - GeeksforGeeks

Given N point on a 2D plane as pair of (x, y) co-ordinates, we need to find maximum number of point which lie on the same line.

Examples:

```
Input : points[] = {-1, 1}, {0, 0}, {1, 1},
          {2, 2}, {3, 3}, {3, 4}
Output : 4
Then maximum number of point which lie on same
line are 4, those point are {0, 0}, {1, 1}, {2, 2},
{3, 3}
```

We can solve above problem by following approach – For each point p, calculate its slope with other points and use a map to record how many points have same slope, by which we can find out how many points are on same line with p as their one point. For each point keep doing the same thing and update the maximum number of point count found so far.

Some things to note in implementation are:

- 1) if two point are (x_1, y_1) and (x_2, y_2) then their slope will be $(y_2 - y_1) / (x_2 - x_1)$ which can be a double value and can cause precision problems. To get rid of the precision problems, we treat slope as pair $((y_2 - y_1), (x_2 - x_1))$ instead of ratio and reduce pair by their gcd before inserting into map. In below code points which are vertical or repeated are treated separately.
- 2) If we use [unordered_map in c++](#) or [HashMap in Java](#) for storing the slope pair, then total time complexity of solution will be $O(n^2)$

```
/* C/C++ program to find maximum number of point
```

```

which lie on same line */
#include <bits/stdc++.h>
#include <boost/functional/hash.hpp>

using namespace std;

// method to find maximum colinear point
int maxPointOnSameLine(vector< pair<int, int> > points)
{
    int N = points.size();
    if (N < 2)
        return N;

    int maxPoint = 0;
    int curMax, overlapPoints, verticalPoints;

    // here since we are using unordered_map
    // which is based on hash function
    //But by default we don't have hash function for pairs
    //so we'll use hash function defined in Boost library
    unordered_map<pair<int, int>, int, boost::
        hash<pair<int, int> > slopeMap;

    // looping for each point
    for (int i = 0; i < N; i++)
    {
        curMax = overlapPoints = verticalPoints = 0;

        // looping from i + 1 to ignore same pair again
        for (int j = i + 1; j < N; j++)
        {
            // If both point are equal then just
            // increase overlapPoint count
            if (points[i] == points[j])
                overlapPoints++;

            // If x co-ordinate is same, then both
            // point are vertical to each other
            else if (points[i].first == points[j].first)
                verticalPoints++;

            else
            {
                int yDif = points[j].second - points[i].second;
                int xDif = points[j].first - points[i].first;
                int g = __gcd(xDif, yDif);

                // reducing the difference by their gcd

```

```
yDif /= g;
xDif /= g;

// increasing the frequency of current slope
// in map
slopeMap[make_pair(yDif, xDif)]++;
curMax = max(curMax, slopeMap[make_pair(yDif, xDif)]);
}

curMax = max(curMax, verticalPoints);
}

// updating global maximum by current point's maximum
maxPoint = max(maxPoint, curMax + overlapPoints + 1);

// printf("maximum colinear point
// which contains current point
// are : %d\n", curMax + overlapPoints + 1);
slopeMap.clear();
}

return maxPoint;
}

// Driver code
int main()
{
    const int N = 6;
    int arr[N][2] = {{-1, 1}, {0, 0}, {1, 1}, {2, 2},
                     {3, 3}, {3, 4}};

    vector< pair<int, int> > points;
    for (int i = 0; i < N; i++)
        points.push_back(make_pair(arr[i][0], arr[i][1]));

    cout << maxPointOnSameLine(points) << endl;

    return 0;
}
```

Output:

4

Improved By : PortgasDAce

Source

<https://www.geeksforgeeks.org/count-maximum-points-on-same-line/>

Chapter 53

Count of acute, obtuse and right triangles with given sides

Count of acute, obtuse and right triangles with given sides - GeeksforGeeks

Given an array of n positive distinct integers representing lengths of lines that can form triangle. The task is to find the number of acute triangles, obtuse triangles, and right triangles separately that can be formed from the given array.

Examples:

Input : arr[] = { 2, 3, 9, 10, 12, 15 }.

Output :

Acute Triangle: 2

Right Triangle: 1

Obtuse Triangle: 4

Acute triangles that can be formed are {10, 12, 15} and { 9, 10, 12 }.

Right triangles that can be formed are {9, 12, 15}.

Obtuse triangles that can be formed are {2, 9, 10}, {3, 9, 10}, {3, 10, 12} and {9, 10, 15}.

Triangle having edges a, b, c , $a \leq b \leq c$.

If $a^2 + b^2 > c^2$, then it is acute triangle.

If $a^2 + b^2 = c^2$, then it is right triangle.

If $a^2 + b^2 < c^2$, then it is obtuse triangle.

Method 1 (Simple) : A brute force can be, use three loops, one for each side. Check above three conditions if a triangle is possible from three sides.

Method 2 (Efficient): An efficient approach is to first sort the array and run two loops for side a and b ($a < b$). Then find the farthest point q where $a + b > c$. So, from b to q all

triangle are possible.

Also find a farthest point p where $a^2 + b^2 \geq c^2$.

Now, observe if $a^2 + b^2 = c^2$, then increment count of right triangles. Also, acute triangle will be $p - b - 1$ and obtuse triangle will be $q - p$.

```
// C++ program to count of acute, obtuse and right
// triangles in an array
#include <bits/stdc++.h>
using namespace std;

// Find the number of acute, right, obtuse triangle
// that can be formed from given array.
void findTriangle(int a[], int n)
{
    int b[n + 2];

    // Finding the square of each element of array.
    for (int i = 0; i < n; i++)
        b[i] = a[i] * a[i];

    // Sort the sides of array and their squares.
    sort(a, a + n);
    sort(b, b + n);

    // x for acute triangles
    // y for right triangles
    // z for obtuse triangles
    int x=0,y=0,z=0;
    for (int i=0; i<n; i++)
    {
        int p = i+1;
        int q = i+1;

        for (int j=i+1; j<n; j++)
        {
            // Finding the farthest point p where
            //  $a^2 + b^2 \geq c^2$ .
            while (p<n-1 && b[i]+b[j]>=b[p+1])
                p++;

            q = max(q, p);

            // Finding the farthest point q where
            //  $a + b > c$ .
            while (q<n-1 && a[i]+a[j]>a[q+1])
                q++;
        }

        // If point p make right triangle.
    }
}
```

```
if (b[i]+b[j]==b[p])
{
    // All triangle between j and p are acute
    // triangles. So add p - j - 1 in x.
    x += max(p - j - 1, 0);

    // Increment y by 1.
    y++;

    // All triangle between q and p are acute
    // triangles. So add q - p in z.
    z += q - p;
}

// If no right triangle
else
{
    // All triangle between j and p are acute
    // triangles. So add p - j in x.
    x += max(p - j, 0);

    // All triangle between q and p are acute
    // triangles. So add q - p in z.
    z += q - p;
}
}

cout << "Acute Triangle: " << x << endl;
cout << "Right Triangle: " << y << endl;
cout << "Obtuse Triangle: " << z << endl;
}

// Driver Code
int main()
{
    int arr[] = {2, 3, 9, 10, 12, 15};
    int n = sizeof(arr)/sizeof(arr[0]);
    findTriangle(arr, n);
    return 0;
}
```

Output:

```
Acute Triangle: 2
Right Triangle: 1
Obtuse Triangle: 4
```

Source

<https://www.geeksforgeeks.org/count-of-acute-obtuse-and-right-triangles-with-given-sides/>

Chapter 54

Count of different straight lines with total n points with m collinear

Count of different straight lines with total n points with m collinear - GeeksforGeeks

There are ‘n’ points in a plane out of which ‘m’ points are collinear. How many different straight lines can form?

Examples:

Input : n = 3, m = 3

Output : 1

We can form only 1 distinct straight line
using 3 collinear points

Input : n = 10, m = 4

Output : 40

$$\text{Number of distinct Straight lines} = {}^nC_2 - {}^mC_2 + 1$$

How does this formula work?

Consider the second example above. There are 10 points, out of which 4 are collinear. A straight line will be formed by any two of these ten points. Thus forming a straight line amounts to selecting any two of the 10 points. Two points can be selected out of the 10 points in nC_2 ways.

Number of straight line formed by 10 points when no 2 of them are co-linear = ${}^{10}C_2 \dots \dots \text{(i)}$

Similarly, the number of straight lines formed by 4 points when no 2 of them are co-linear = ${}^4C_2 \dots \dots \text{(ii)}$

Since straight lines formed by these 4 points are sane, straight lines formed by them will reduce to only one.

$$\text{Required number of straight lines formed} = {}^{10}\text{C}_2 - {}^4\text{C}_2 + 1 = 45 - 6 + 1 = 40$$

Implementation of the approach is given as:

C++

```
// CPP program to count number of straight lines
// with n total points, out of which m are
// collinear.
#include <bits/stdc++.h>

using namespace std;

// Returns value of binomial coefficient
// Code taken from https://goo.gl/vhy4jp
int nCk(int n, int k)
{
    int C[k+1];
    memset(C, 0, sizeof(C));

    C[0] = 1; // nC0 is 1

    for (int i = 1; i <= n; i++)
    {
        // Compute next row of pascal triangle
        // using the previous row
        for (int j = min(i, k); j > 0; j--)
            C[j] = C[j] + C[j-1];
    }

    return C[k];
}

/* function to calculate number of straight lines
   can be formed */
int count_Straightlines(int n,int m)
{
    return (nCk(n, 2) - nCk(m, 2)+1);
}
```

```
/* driver function*/
int main()
{
    int n = 4, m = 3 ;
    cout << count_Straightlines(n, m);
    return 0;
}

Java

// Java program to count number of straight lines
// with n total points, out of which m are
// collinear.
import java.util.*;
import java.lang.*;

public class GfG {

    // Returns value of binomial coefficient
    // Code taken from https://goo.gl/vhy4jp
    public static int nCk(int n, int k)
    {
        int[] C = new int[k + 1];
        C[0] = 1; // nC0 is 1

        for (int i = 1; i <= n; i++)  {

            // Compute next row of pascal triangle
            // using the previous row
            for (int j = Math.min(i, k); j > 0; j--)
                C[j] = C[j] + C[j - 1];
        }

        return C[k];
    }

    /* function to calculate number of straight lines
    can be formed */
    public static int count_Straightlines(int n, int m)
    {
        return (nCk(n, 2) - nCk(m, 2) + 1);
    }
}
```

```
// Driver function
public static void main(String argc[])
{
    int n = 4, m = 3;
    System.out.println(count_Straightlines(n, m));
}

// This code is contributed by Sagar Shukla
}
```

Python

```
# Python program to count number of straight lines
# with n total points, out of which m are
# collinear.

# Returns value of binomial coefficient
# Code taken from https://goo.gl/vhy4jp
def nCk(n, k):

    C = [0]*(k+1)

    C[0] = 1 # nC0 is 1

    for i in range(1, n+1):

        # Compute next row of pascal triangle
        # using the previous row
        j = min(i, k)

        while(j>0):

            C[j] = C[j] + C[j-1]
            j = j - 1

    return C[k]

#function to calculate number of straight lines
# can be formed
def count_Straightlines(n, m):

    return (nCk(n, 2) - nCk(m, 2)+1)

# Driven code
n = 4
m = 3
```

```
print( count_Straightlines(n, m) );  
  
# This code is contributed by "rishabh_jain".  
  
C#  
  
// C# program to count number of straight  
// lines with n total points, out of  
// which m are collinear.  
using System;  
  
public class GfG {  
  
    // Returns value of binomial coefficient  
    // Code taken from https://goo.gl/vhy4jp  
    public static int nCk(int n, int k)  
    {  
        int[] C = new int[k + 1];  
  
        // nC0 is 1  
        C[0] = 1;  
  
        for (int i = 1; i <= n; i++)  
        {  
  
            // Compute next row of pascal triangle  
            // using the previous row  
            for (int j = Math.Min(i, k); j > 0; j--)  
                C[j] = C[j] + C[j - 1];  
        }  
  
        return C[k];  
    }  
  
    // Function to calculate number of  
    // straight lines can be formed  
    public static int count_Straightlines(int n, int m)  
    {  
        return (nCk(n, 2) - nCk(m, 2) + 1);  
    }  
  
    // Driver Code  
    public static void Main(String []args)  
    {  
        int n = 4, m = 3;  
        Console.WriteLine(count_Straightlines(n, m));  
    }  
}
```

```
    }  
  
}  
  
// This code is contributed by vt_m.
```

PHP

```
<?php  
// PHP program to count number of straight lines  
// with n total points, out of which m are  
// collinear.  
  
// Returns value of binomial coefficient  
function nCk($n, $k)  
{  
    $C = array_fill(0, $k + 1, NULL);  
  
    $C[0] = 1; // nC0 is 1  
  
    for ($i = 1; $i <= $n; $i++)  
    {  
  
        // Compute next row of pascal triangle  
        // using the previous row  
        for ($j = min($i, $k); $j > 0; $j--)  
            $C[$j] = $C[$j] + $C[$j-1];  
    }  
    return $C[$k];  
}  
  
// function to calculate  
// number of straight lines  
// can be formed  
function count_Straightlines($n, $m)  
{  
  
    return (nCk($n, 2) - nCk($m, 2) + 1);  
}  
  
// Driver Code  
$n = 4;  
$m = 3;  
echo(count_Straightlines($n, $m));
```

```
// This code is contributed  
// by Prasad Kshirsagar  
?>
```

Output:

4

Improved By : [vt_m](#), [Prasad_Kshirsagar](#), [NishantBaranwalSomy](#)

Source

<https://www.geeksforgeeks.org/count-different-straight-lines-total-n-points-m-collinear/>

Chapter 55

Count of obtuse angles in a circle with 'k' equidistant points between 2 given points

Count of obtuse angles in a circle with 'k' equidistant points between 2 given points - GeeksforGeeks

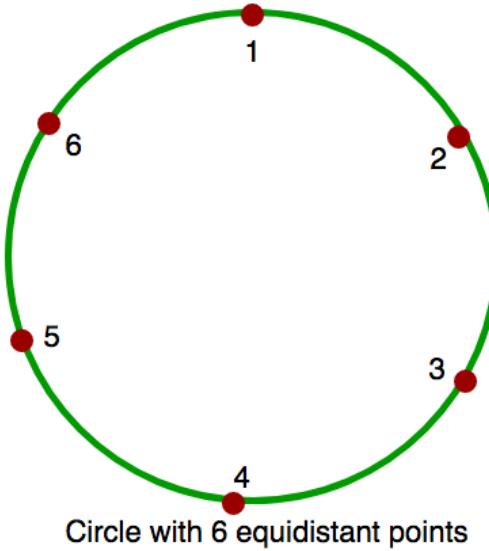
A circle is given with k equidistant points on its circumference. 2 points A and B are given in the circle. Find the count of all obtuse angles (angles larger than 90 degree) formed from $\angle ACB$, where C can be any point in circle other than A or B.

Note :

A and B are not equal.

$A < B$.

Points are between 1 and K(both inclusive).



Examples :

Input : K = 6, A = 1, B = 3.

Output : 1

Explanation : In the circle with 6 equidistant points, when C = 2 i.e. /_123, we get obtuse angle.

Input : K = 6, A = 1, B = 4.

Output : 0

Explanation : In this circle, there is no such C that form an obtuse angle.

It can be observed that if A and B have equal elements in between them, there can't be any C such that ACB is obtuse. Also, the number of possible obtuse angles are the smaller arc between A and B.

Below is the implementation :

C++

```
// C++ program to count number of obtuse
// angles for given two points.
#include <bits/stdc++.h>
using namespace std;
```

```
int countObtuseAngles(int a, int b, int k)
{
    // There are two arcs connecting a
    // and b. Let us count points on
    // both arcs.
    int c1 = (b - a) - 1;
    int c2 = (k - b) + (a - 1);

    // Both arcs have same number of
    // points
    if (c1 == c2)
        return 0;

    // Points on smaller arc is answer
    return min(c1, c2);
}

// Driver code
int main()
{
    int k = 6, a = 1, b = 3;
    cout << countObtuseAngles(a, b, k);
    return 0;
}
```

Java

```
// Java program to count number of obtuse
// angles for given two points
class GFG {

    static int countObtuseAngles(int a,
                                int b, int k)
    {

        // There are two arcs connecting a
        // and b. Let us count points on
        // both arcs.
        int c1 = (b - a) - 1;
        int c2 = (k - b) + (a - 1);

        // Both arcs have same number of
        // points
        if (c1 == c2)
            return 0;

        // Points on smaller arc is answer
        return min(c1, c2);
    }
}
```

```
}

// Driver Program to test above function
public static void main(String arg[])
{
    int k = 6, a = 1, b = 3;
    System.out.print(countObtuseAngles(a, b, k));
}
}

// This code is contributed by Anant Agarwal.
```

Python

```
# C++ program to count number of obtuse
# angles for given two points.

def countObtuseAngles( a, b, k):
    # There are two arcs connecting a
    # and b. Let us count points on
    # both arcs.
    c1 = (b - a) - 1
    c2 = (k - b) + (a - 1)

    # Both arcs have same number of
    # points
    if (c1 == c2):
        return 0

    # Points on smaller arc is answer
    return min(c1, c2)

# Driver code
k, a, b = 6, 1, 3
print countObtuseAngles(a, b, k)

# This code is contributed by Sachin Bisht
```

C#

```
// C# program to count number of obtuse
// angles for given two points
using System;

class GFG {
```

```
static int countObtuseAngles(int a,
                             int b, int k)
{
    // There are two arcs connecting
    // a and b. Let us count points
    // on both arcs.
    int c1 = (b - a) - 1;
    int c2 = (k - b) + (a - 1);

    // Both arcs have same number
    // of points
    if (c1 == c2)
        return 0;

    // Points on smaller arc is
    // answer
    return Math.Min(c1, c2);
}

// Driver Program to test above
// function
public static void Main()
{
    int k = 6, a = 1, b = 3;

    Console.WriteLine(
        countObtuseAngles(a, b, k));
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to count number
// of obtuse angles for given
// two points.

function countObtuseAngles($a, $b, $k)
{
    // There are two arcs connecting a
    // and b. Let us count points on
    // both arcs.
    $c1 = ($b - $a) - 1;
    $c2 = ($k - $b) + ($a - 1);
```

```
// Both arcs have same number of
// points
if ($c1 == $c2)
    return 0;

// Points on smaller arc is answer
return min($c1, $c2);
}

// Driver code
$k = 6; $a = 1; $b = 3;
echo countObtuseAngles($a, $b, $k);

// This code is contributed by aj_36
?>
```

Output :

1

Improved By : [vt_m, jit_t](#)

Source

<https://www.geeksforgeeks.org/count-obtuse-angles-circle-k-equidistant-points-2-given-points/>

Chapter 56

Count of parallelograms in a plane

Count of parallelograms in a plane - GeeksforGeeks

Given some points on a plane, which are distinct and no three of them lie on the same line. We need to find number of Parallelograms with the vertices as the given points.

Examples:

```
Input : points[] = {(0, 0), (0, 2), (2, 2), (4, 2),
                     (1, 4), (3, 4)}
Output : 2
Two Parallelograms are possible by choosing above
given point as vertices, which are shown in below
diagram.
```

We can solve this problem by using a special property of parallelograms that diagonals of a parallelogram intersect each other in the middle. So if we get such a middle point which is middle point of more than one line segment, then we can conclude that a parallelogram exists, more accurately if a middle point occurs x times, then diagonals of possible parallelograms can be chosen in ${}^x C_2$ ways, i.e. there will be $x*(x-1)/2$ parallelograms corresponding to this particular middle point with a frequency x . So we iterate over all pair of points and we calculate their middle point and increase frequency of middle point by 1. At the end, we count number of parallelograms according to the frequency of each distinct middle point as explained above. As we just need frequency of middle point, division by 2 is ignored while calculating middle point for simplicity.

```
// C++ program to get number of Parallelograms we
// can make by given points of the plane
```

```
#include <bits/stdc++.h>
using namespace std;

// Returns count of Parallelograms possible
// from given points
int countOfParallelograms(int x[], int y[], int N)
{
    // Map to store frequency of mid points
    map<pair<int, int>, int> cnt;
    for (int i=0; i<N; i++)
    {
        for (int j=i+1; j<N; j++)
        {
            // division by 2 is ignored, to get
            // rid of doubles
            int midX = x[i] + x[j];
            int midY = y[i] + y[j];

            // increase the frequency of mid point
            cnt[make_pair(midX, midY)]++;
        }
    }

    // Iterating through all mid points
    int res = 0;
    for (auto it = cnt.begin(); it != cnt.end(); it++)
    {
        int freq = it->second;

        // Increase the count of Parallelograms by
        // applying function on frequency of mid point
        res += freq*(freq - 1)/2
    }

    return res;
}

// Driver code to test above methods
int main()
{
    int x[] = {0, 0, 2, 4, 1, 3};
    int y[] = {0, 2, 2, 2, 4, 4};
    int N = sizeof(x) / sizeof(int);

    cout << countOfParallelograms(x, y, N) << endl;

    return 0;
}
```

Output:

2

Source

<https://www.geeksforgeeks.org/count-parallelograms-plane/>

Chapter 57

Count ways to divide circle using N non-intersecting chords

Count ways to divide circle using N non-intersecting chords - GeeksforGeeks

Given a number N, find the number of ways you can draw N chords in a circle with 2^N points such that no 2 chords intersect.

Two ways are different if there exists a chord which is present in one way and not in other.

Examples:

```
Input : N = 2
Output : 2
Explanation: If points are numbered 1 to 4 in
clockwise direction, then different ways to
draw chords are:
{(1-2), (3-4)} and {(1-4), (2-3)}
```

```
Input : N = 1
Output : 1
Explanation: Draw a chord between points 1 and 2.
```

If we draw a chord between any two points, can you observe the current set of points getting broken into two smaller sets S_1 and S_2 . If we draw a chord from a point in S_1 to a point in S_2 , it will surely intersect the chord we've just drawn.

So, we can arrive at a recurrence that $\text{Ways}(n) = \sum[i = 0 \text{ to } n-1] \{ \text{Ways}(i) * \text{Ways}(n-i-1) \}$.

Here we iterate over i, assuming that size of one of the sets is i and size of another set automatically is $(n-i-1)$ since we've already used a pair of points and i pair of points in one set.

C++

```
// cpp code to count ways
// to divide circle using
// N non-intersecting chords.
#include <bits/stdc++.h>
using namespace std;

int chordCnt( int A){

    // n = no of points required
    int n = 2 * A;

    // dp array containing the sum
    int dpArray[n + 1]={ 0 };
    dpArray[0] = 1;
    dpArray[2] = 1;
    for (int i=4;i<=n;i+=2){
        for (int j=0;j<i-1;j+=2){

            dpArray[i] +=
                (dpArray[j]*dpArray[i-2-j]);
        }
    }

    // returning the required number
    return dpArray[n];
}

// Driver function
int main()
{

    int N;
    N = 2;
    cout<<chordCnt( N)<<'\n';
    N = 1;
    cout<<chordCnt( N)<<'\n';
    N = 4;
    cout<<chordCnt( N)<<'\n';
    return 0;
}

// This code is contributed by Gitanjali.
```

Java

```
// Java code to count ways
// to divide circle using
// N non-intersecting chords.
import java.io.*;
```

```
class GFG {
    static int chordCnt(int A)
    {

        // n = no of points required
        int n = 2 * A;

        // dp array containing the sum
        int[] dpArray = new int[n + 1];
        dpArray[0] = 1;
        dpArray[2] = 1;
        for (int i = 4; i <= n; i += 2) {
            for (int j = 0; j < i - 1; j += 2)
            {
                dpArray[i] += (dpArray[j] *
                               dpArray[i - 2 - j]);
            }
        }

        // returning the required number
        return dpArray[n];
    }
    public static void main(String[] args)
    {
        int N;
        N = 2;
        System.out.println(chordCnt(N));
        N = 1;
        System.out.println(chordCnt(N));
        N = 4;
        System.out.println(chordCnt(N));
    }
}

// This code is contributed by Gitanjali.
```

Python 3

```
# python code to count ways to divide
# circle using N non-intersecting chords.
def chordCnt( A):

    # n = no of points required
    n = 2 * A

    # dp array containing the sum
    dpArray = [0]*(n + 1)
```

```
dpArray[0] = 1
dpArray[2] = 1
for i in range(4, n + 1, 2):
    for j in range(0, i-1, 2):
        dpArray[i] += (dpArray[j]*dpArray[i-2-j])

# returning the required number
return int(dpArray[n])

# driver code
N = 2
print(chordCnt( N))
N = 1
print(chordCnt( N))
N = 4
print(chordCnt( N))
```

C#

```
// C# code to count ways to divide
// circle using N non-intersecting chords.
using System;

class GFG {

    static int chordCnt(int A)
    {
        // n = no of points required
        int n = 2 * A;

        // dp array containing the sum
        int[] dpArray = new int[n + 1];
        dpArray[0] = 1;
        dpArray[2] = 1;

        for (int i = 4; i <= n; i += 2)
        {
            for (int j = 0; j < i - 1; j += 2)
            {
                dpArray[i] += (dpArray[j] * dpArray[i - 2 - j]);
            }
        }

        // returning the required number
        return dpArray[n];
    }

    // Driver code
```

```
public static void Main()
{
    int N;
    N = 2;
    Console.WriteLine(chordCnt(N));
    N = 1;
    Console.WriteLine(chordCnt(N));
    N = 4;
    Console.WriteLine(chordCnt(N));
}
// This code is contributed by vt_m.
```

Output:

```
2
1
14
```

Source

<https://www.geeksforgeeks.org/count-ways-divide-circle-using-n-non-intersecting-chords/>

Chapter 58

Deleting points from Convex Hull

Deleting points from Convex Hull - GeeksforGeeks

Given a fixed set of points. We need to find convex hull of given set. We also need to find convex hull when a point is removed from the set.

Example:

```
Initial Set of Points: (-2, 8) (-1, 2) (0, 1) (1, 0)
                      (-3, 0) (-1, -9) (2, -6) (3, 0)
                      (5, 3) (2, 5)
Initial convex hull:- (-2, 8) (-3, 0) (-1, -9) (2, -6)
                      (5, 3)
Point to remove from the set : (-2, 8)
Final convex hull: (2, 5) (-3, 0) (-1, -9) (2, -6) (5, 3)
```

Prerequisite :[Convex Hull \(Simple Divide and Conquer Algorithm\)](#)

The algorithm for solving the above problem is very easy. We simply check whether the point to be removed is a part of the convex hull. If it is, then we have to remove that point from the initial set and then make the convex hull again (refer [Convex hull \(divide and conquer\)](#)).

And if not then we already have the solution (the convex hull will not change).

```
// C++ program to demonstrate delete operation
// on Convex Hull.
#include<bits/stdc++.h>
using namespace std;

// stores the center of polygon (It is made
```

```

// global because it is used in compare function)
pair<int, int> mid;

// determines the quadrant of a point
// (used in compare())
int quad(pair<int, int> p)
{
    if (p.first >= 0 && p.second >= 0)
        return 1;
    if (p.first <= 0 && p.second >= 0)
        return 2;
    if (p.first <= 0 && p.second <= 0)
        return 3;
    return 4;
}

// Checks whether the line is crossing the polygon
int orientation(pair<int, int> a, pair<int, int> b,
                pair<int, int> c)
{
    int res = (b.second-a.second)*(c.first-b.first) -
              (c.second-b.second)*(b.first-a.first);

    if (res == 0)
        return 0;
    if (res > 0)
        return 1;
    return -1;
}

// compare function for sorting
bool compare(pair<int, int> p1, pair<int, int> q1)
{
    pair<int, int> p = make_pair(p1.first - mid.first,
                                  p1.second - mid.second);
    pair<int, int> q = make_pair(q1.first - mid.first,
                                  q1.second - mid.second);

    int one = quad(p);
    int two = quad(q);

    if (one != two)
        return (one < two);
    return (p.second*q.first < q.second*p.first);
}

// Finds upper tangent of two polygons 'a' and 'b'
// represented as two vectors.

```

```

vector<pair<int, int> > merger(vector<pair<int, int> > a,
                                vector<pair<int, int> > b)
{
    // n1 -> number of points in polygon a
    // n2 -> number of points in polygon b
    int n1 = a.size(), n2 = b.size();

    int ia = 0, ib = 0;
    for (int i=1; i<n1; i++)
        if (a[i].first > a[ia].first)
            ia = i;

    // ib -> leftmost point of b
    for (int i=1; i<n2; i++)
        if (b[i].first < b[ib].first)
            ib=i;

    // finding the upper tangent
    int inda = ia, indb = ib;
    bool done = 0;
    while (!done)
    {
        done = 1;
        while (orientation(b[indb], a[inda], a[(inda+1)%n1]) >=0)
            inda = (inda + 1) % n1;

        while (orientation(a[inda], b[indb], b[(n2+indb-1)%n2]) <=0)
        {
            indb = (n2+indb-1)%n2;
            done = 0;
        }
    }

    int uppera = inda, upperb = indb;
    inda = ia, indb=ib;
    done = 0;
    int g = 0;
    while (!done)//finding the lower tangent
    {
        done = 1;
        while (orientation(a[inda], b[indb], b[(indb+1)%n2])>=0)
            indb=(indb+1)%n2;

        while (orientation(b[indb], a[inda], a[(n1+inda-1)%n1])<=0)
        {
            inda=(n1+inda-1)%n1;
            done=0;
        }
    }
}

```

```

}

int lowera = inda, lowerb = indb;
vector<pair<int, int> > ret;

//ret contains the convex hull after merging the two convex hulls
//with the points sorted in anti-clockwise order
int ind = uppera;
ret.push_back(a[uppera]);
while (ind != lowera)
{
    ind = (ind+1)%n1;
    ret.push_back(a[ind]);
}

ind = lowerb;
ret.push_back(b[lowerb]);
while (ind != upperb)
{
    ind = (ind+1)%n2;
    ret.push_back(b[ind]);
}
return ret;
}

// Brute force algorithm to find convex hull for a set
// of less than 6 points
vector<pair<int, int> > bruteHull(vector<pair<int, int> > a)
{
    // Take any pair of points from the set and check
    // whether it is the edge of the convex hull or not.
    // if all the remaining points are on the same side
    // of the line then the line is the edge of convex
    // hull otherwise not
    set<pair<int, int> >s;

    for (int i=0; i<a.size(); i++)
    {
        for (int j=i+1; j<a.size(); j++)
        {
            int x1 = a[i].first, x2 = a[j].first;
            int y1 = a[i].second, y2 = a[j].second;

            int a1 = y1-y2;
            int b1 = x2-x1;
            int c1 = x1*y2-y1*x2;
            int pos = 0, neg = 0;

```

```

        for (int k=0; k<a.size(); k++)
        {
            if (a1*a[k].first+b1*a[k].second+c1 <= 0)
                neg++;
            if (a1*a[k].first+b1*a[k].second+c1 >= 0)
                pos++;
        }
        if (pos == a.size() || neg == a.size())
        {
            s.insert(a[i]);
            s.insert(a[j]);
        }
    }
}

vector<pair<int, int>> ret;
for (auto e : s)
    ret.push_back(e);

// Sorting the points in the anti-clockwise order
mid = {0, 0};
int n = ret.size();
for (int i=0; i<n; i++)
{
    mid.first += ret[i].first;
    mid.second += ret[i].second;
    ret[i].first *= n;
    ret[i].second *= n;
}
sort(ret.begin(), ret.end(), compare);
for (int i=0; i<n; i++)
    ret[i] = make_pair(ret[i].first/n, ret[i].second/n);

return ret;
}

// Returns the convex hull for the given set of points
vector<pair<int, int>> findHull(vector<pair<int, int>> a)
{
    // If the number of points is less than 6 then the
    // function uses the brute algorithm to find the
    // convex hull
    if (a.size() <= 5)
        return bruteHull(a);

    // left contains the left half points
    // right contains the right half points
    vector<pair<int, int>> left, right;

```

```

for (int i=0; i<a.size()/2; i++)
    left.push_back(a[i]);
for (int i=a.size()/2; i<a.size(); i++)
    right.push_back(a[i]);

// convex hull for the left and right sets
vector<pair<int, int>>left_hull = findHull(left);
vector<pair<int, int>>right_hull = findHull(right);

// merging the convex hulls
return merger(left_hull, right_hull);
}

// Returns the convex hull for the given set of points after
// removing a point p.
vector<pair<int, int>> removePoint(vector<pair<int, int>> a,
                                         vector<pair<int, int>> hull,
                                         pair<int, int> p)
{
    // checking whether the point is a part of the
    // convex hull or not.
    bool found = 0;
    for (int i=0; i < hull.size() && !found; i++)
        if (hull[i].first == p.first &&
            hull[i].second == p.second)
            found = 1;

    // If point is not part of convex hull
    if (found == 0)
        return hull;

    // if it is the part of the convex hull then
    // we remove the point and again make the convex hull
    // and if not, we print the same convex hull.
    for (int i=0; i<a.size(); i++)
    {
        if (a[i].first==p.first && a[i].second==p.second)
        {
            a.erase(a.begin()+i);
            break;
        }
    }

    sort(a.begin(), a.end());
    return findHull(a);
}

// Driver code

```

```
int main()
{
    vector<pair<int, int> > a;
    a.push_back(make_pair(0, 0));
    a.push_back(make_pair(1, -4));
    a.push_back(make_pair(-1, -5));
    a.push_back(make_pair(-5, -3));
    a.push_back(make_pair(-3, -1));
    a.push_back(make_pair(-1, -3));
    a.push_back(make_pair(-2, -2));
    a.push_back(make_pair(-1, -1));
    a.push_back(make_pair(-2, -1));
    a.push_back(make_pair(-1, 1));

    int n = a.size();

    // sorting the set of points according
    // to the x-coordinate
    sort(a.begin(), a.end());
    vector<pair<int, int> > hull = findHull(a);

    cout << "Convex hull:\n";
    for (auto e : hull)
        cout << e.first << " "
            << e.second << endl;

    pair<int, int> p = make_pair(-5, -3);
    removePoint(a, hull, p);

    cout << "\nModified Convex Hull:\n";
    for (auto e:hull)
        cout << e.first << " "
            << e.second << endl;

    return 0;
}
```

Output:

```
convex hull:
-3 0
-1 -9
2 -6
5 3
2 5
```

Time Complexity:

It is simple to see that the maximum time taken per query is the time taken to construct the convex hull which is $O(n \log n)$. So, the overall complexity is $O(q \cdot n \log n)$, where q is the number of points to be deleted.

Source

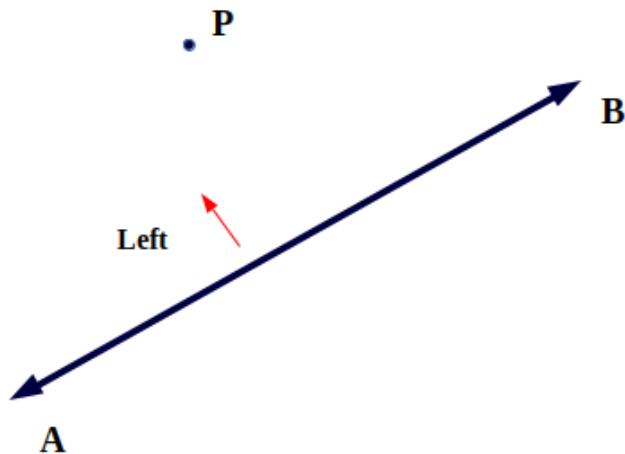
<https://www.geeksforgeeks.org/deleting-points-convex-hull/>

Chapter 59

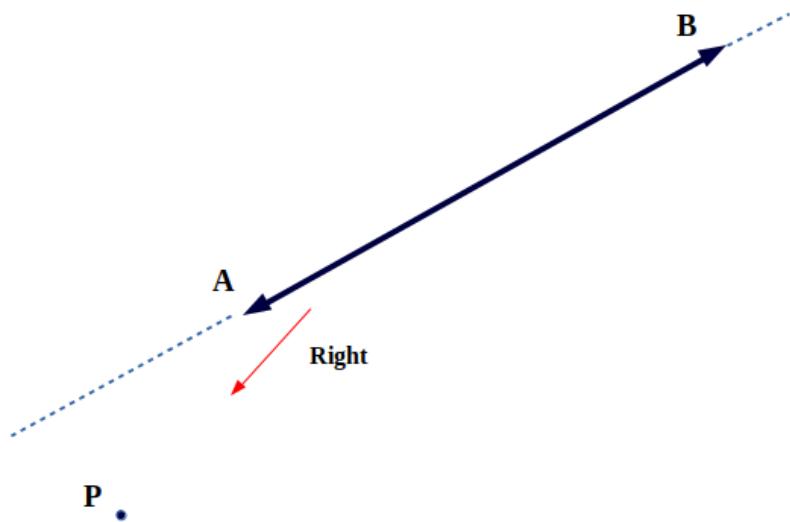
Direction of a Point from a Line Segment

Direction of a Point from a Line Segment - GeeksforGeeks

Direction of given point P from a line segment simply means given the co-ordinates of a point P and line segment (say AB), and we have to determine the direction of point P from the line segment. That is whether the Point lies to the Right of Line Segment or to the Left of Line Segment.



The point might lie behind the line segment, in that case we assume imaginary line by extending the line segment and determine the direction of point.



* There are only three cases, either the point is on left side, or right side or on the line segment itself.

This is a very fundamental Problem and is commonly encountered for directions in **online map**,

Example : Suppose a user A has to go to Point C in the following map, the user first reaches point B but after that how does user A know that whether he has to make a right turn or left turn.



Knowing the direction of a point from a line segment also acts a building block to solve more complicated problem such as :

Line segment Intersection : finding if two line segment intersect

Convex Hull of a set of Points.

Right Direction

Source

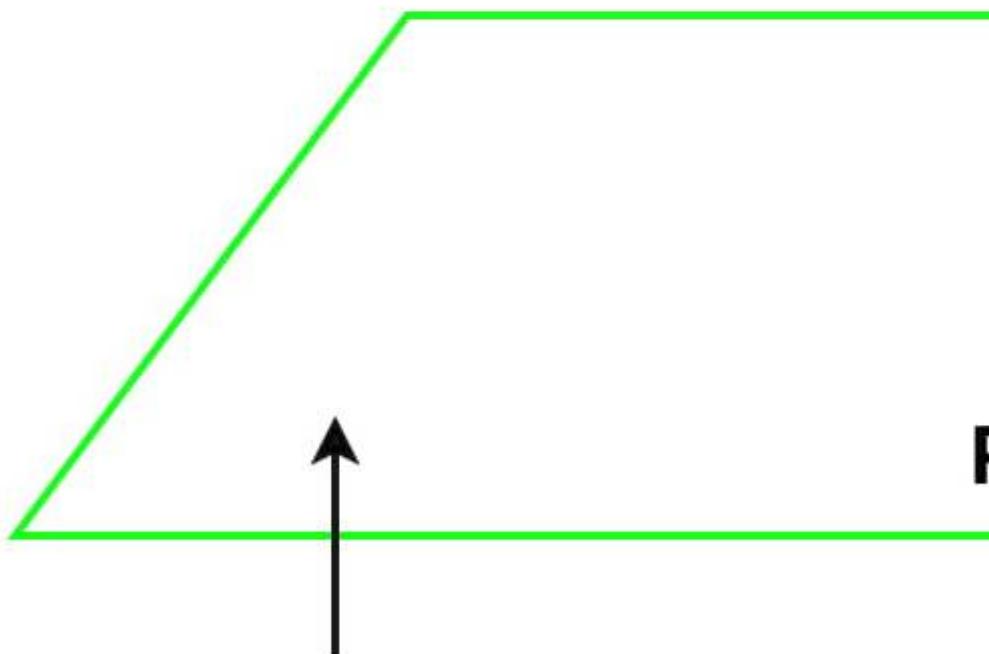
<https://www.geeksforgeeks.org/direction-point-line-segment/>

Chapter 60

Distance between a point and a Plane in 3 D

Distance between a point and a Plane in 3 D - GeeksforGeeks

You are given a points (x_1, y_1, z_1) and a plane $a * x + b * y + c * z + d = 0$. The task is to find the perpendicular(shortest) distance between that point and the given Plane.



$$a^*x + b^*y + c^*z + d = 0$$

Examples :

Input: $x_1 = 4, y_1 = -4, z_1 = 3, a = 2, b = -2, c = 5, d = 8$

Output: Perpendicular distance is 6.78902858227

Input: $x_1 = 2, y_1 = 8, z_1 = 5, a = 1, b = -2, c = -2, d = -1$

Output: Perpendicular distance is 8.33333333333

Approach: The perpendicular distance (i.e shortest distance) from a given point to a Plane is the perpendicular distance from that point to the given plane. Let the co-ordinate of the given point be (x_1, y_1, z_1) and equation of the plane be given by the equation $a * x + b * y + c * z + d = 0$, where a, b and c are real constants.

The formula for distance between a point and Plane in 3-D is given by:

```
Distance = (| a*x1 + b*y1 + c*z1 + d |) / (sqrt( a*a + b*b + c*c))
```

Below is the implementation of the above formulae:

C

```
// C program to find the Perpendicular(shortest)
// distance between a point and a Plane in 3 D.

#include<stdio.h>
#include<math.h>

// Function to find distance
void shortest_distance(float x1, float y1, float z1,
                      float a, float b, float c, float d)
{
    d = fabs((a * x1 + b * y1 + c * z1 + d));
    float e = sqrt(a * a + b * b + c * c);
    printf("Perpendicular distance is %f", d/e);
    return;
}

// Driver Code
int main()
{
    float x1 = 4;
    float y1 = -4;
    float z1 = 3;
    float a = 2;
    float b = -2;
    float c = 5;
```

```
float d = 8;

// Function call
shortest_distance(x1, y1, z1, a, b, c, d);
}

// This code is contributed
// by Amber_Saxena.
```

Java

```
// Java program to find the
// Perpendicular(shortest)
// distance between a point
// and a Plane in 3 D.
import java .io.*;

class GFG
{

    // Function to find distance
    static void shortest_distance(float x1, float y1,
                                  float z1, float a,
                                  float b, float c,
                                  float d)
    {
        d = Math.abs((a * x1 + b *
                     y1 + c * z1 + d));
        float e = (float)Math.sqrt(a * a + b *
                                   b + c * c);
        System.out.println("Perpendicular distance " +
                           "is " + d / e);
    }

    // Driver code
    public static void main(String[] args)
    {
        float x1 = 4;
        float y1 = -4;
        float z1 = 3;
        float a = 2;
        float b = -2;
        float c = 5;
        float d = 8;

        // Function call
        shortest_distance(x1, y1, z1,
                          a, b, c, d);
    }
}
```

}

```
// This code is contributed  
// by Amber_Saxena.
```

Python

```
# Python program to find the Perpendicular(shortest)  
# distance between a point and a Plane in 3 D.  
  
import math  
  
# Function to find distance  
def shortest_distance(x1, y1, z1, a, b, c, d):  
  
    d = abs((a * x1 + b * y1 + c * z1 + d))  
    e = (math.sqrt(a * a + b * b + c * c))  
    print("Perpendicular distance is"), d/e  
  
# Driver Code  
x1 = 4  
y1 = -4  
z1 = 3  
a = 2  
b = -2  
c = 5  
d = 8  
  
# Function call  
shortest_distance(x1, y1, z1, a, b, c, d)
```

C#

```
// C# program to find the  
// Perpendicular(shortest)  
// distance between a point  
// and a Plane in 3 D.  
using System;  
  
class GFG  
{  
    // Function to find distance  
    static void shortest_distance(float x1, float y1,  
        float z1, float a,  
        float b, float c,  
        float d)  
    {
```

```
d = Math.Abs((a * x1 + b *  
y1 + c * z1 + d));  
float e = (float)Math.Sqrt(a * a + b *  
b + c * c);  
Console.WriteLine("Perpendicular distance " +  
"is " + d / e);  
}  
  
// Driver code  
public static void Main()  
{  
    float x1 = 4;  
    float y1 = -4;  
    float z1 = 3;  
    float a = 2;  
    float b = -2;  
    float c = 5;  
    float d = 8;  
  
    // Function call  
    shortest_distance(x1, y1, z1,  
a, b, c, d);  
}
```

// This code is contributed
// by ChitraNayal

PHP

```
<?php  
// PHP program to find the  
// Perpendicular(shortest)  
// distance between a point  
// and a Plane in 3 D.  
  
// Function to find distance  
function shortest_distance($x1, $y1, $z1,  
                           $a, $b, $c, $d)  
{  
    $d = abs(($a * $x1 + $b * $y1 +  
              $c * $z1 + $d));  
    $e = sqrt($a * $a + $b *  
              $b + $c * $c);  
    echo "Perpendicular distance is ". $d / $e;  
}  
  
// Driver Code  
$x1 = 4;  
$y1 = -4;
```

```
$z1 = 3;  
$a = 2;  
$b = -2;  
$c = 5;  
$d = 8;  
  
// function call  
shortest_distance($x1, $y1, $z1,  
                  $a, $b, $c, $d);  
  
// This code is contributed  
// by Amber_Saxena.  
?>
```

Output:

Perpendicular distance is 6.78902858227

Improved By : [Amber_Saxena](#), [ChitraNayal](#)

Source

<https://www.geeksforgeeks.org/distance-between-a-point-and-a-plane-in-3-d/>

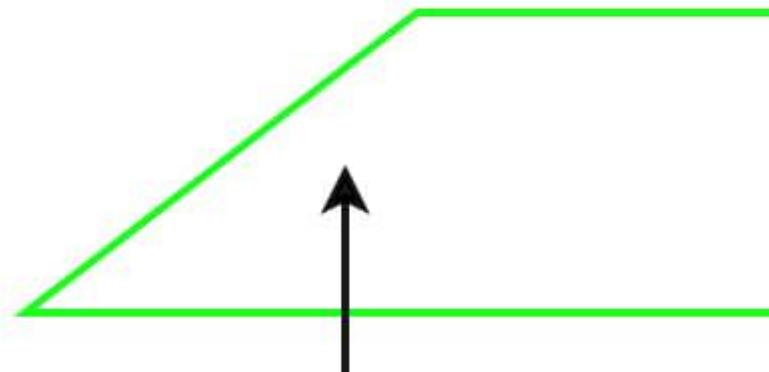
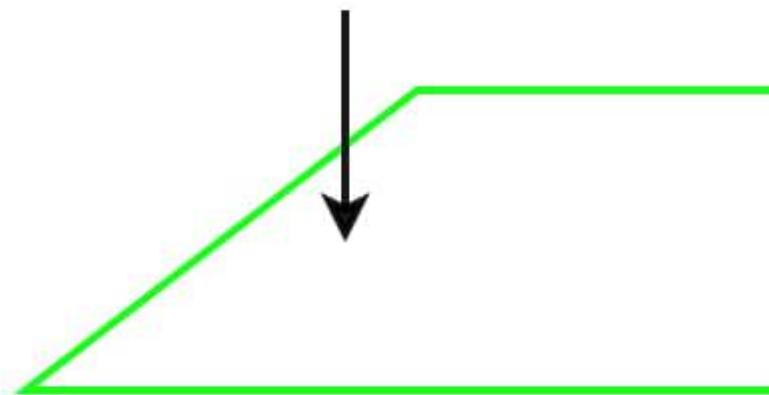
Chapter 61

Distance between two parallel Planes in 3-D

Distance between two parallel Planes in 3-D - GeeksforGeeks

You are given two planes **P1**: $a1 * x + b1 * y + c1 * z + d1 = 0$ and **P2**: $a2 * x + b2 * y + c2 * z + d2 = 0$. The task is to write a program to find distance between these two Planes.

$$a_2^*x + b_2^*y + c_2^*$$



$$a_1^*x + b_1^*y + c_1^*$$

Examples :

Input: $a_1 = 1, b_1 = 2, c_1 = -1, d_1 = 1, a_2 = 3, b_2 = 6, c_2 = -3, d_2 = -4$
Output: Distance is 0.952579344416

Input: $a_1 = 1, b_1 = 2, c_1 = -1, d_1 = 1, a_2 = 1, b_2 = 6, c_2 = -3, d_2 = -4$
Output: Planes are not parallel

Approach : Consider two planes are given by the equations:-

P1 : $a_1 * x + b_1 * y + c_1 * z + d_1 = 0$, where a_1, b_1 and c_1, d_1 are real constants and

P2 : $a_2 * x + b_2 * y + c_2 * z + d_2 = 0$, where a_2, b_2 and c_2, d_2 are real constants.

The condition for two planes to be parallel is:-

$$\Rightarrow a_1 / a_2 = b_1 / b_2 = c_1 / c_2$$

Find a point in any one plane such that the distance from that point to the other plane that will be the distance between those two planes. The distance can be calculated by using the formulae:

$$\text{Distance} = (| a_1*x_1 + b_1*y_1 + c_1*z_1 + d_1 |) / (\sqrt{a_1^2 + b_1^2 + c_1^2})$$

Let a point in Plane P1 be P(x1, y1, z1),

put $x = y = 0$ in equation $a_1 * x + b_1 * y + c_1 * z + d_1 = 0$ and find z.

$$\Rightarrow z = -d_1 / c_1$$

Now we have coordinates of P(0, 0, z) = P(x1, y1, z1).

Distance of point P to Plane P2 will be:-

$$\text{Distance} = (| a_2*x_1 + b_2*y_1 + c_2*z_1 + d_2 |) / (\sqrt{a_2^2 + b_2^2 + c_2^2})$$

$$= (| a_2*0 + b_2*0 + c_2*z_1 + d_2 |) / (\sqrt{a_2^2 + b_2^2 + c_2^2})$$

$$= (| c_2*z_1 + d_2 |) / (\sqrt{a_2^2 + b_2^2 + c_2^2})$$

Below is the implementation of the above formulae:

C

```
// C program to find the Distance between  
// two parallel Planes in 3 D.
```

```
#include <stdio.h>  
#include<math.h>
```

```
// Function to find distance
void distance(float a1, float b1, float c1,
              float d1, float a2, float b2,
              float c2, float d2)
{
    float x1,y1,z1,d;
    if (a1 / a2 == b1 / b2 && b1 / b2 == c1 / c2)
    {
        x1 = y1 = 0;
        z1 = -d1 / c1;
        d = fabs((c2 * z1 + d2)) / (sqrt(a2 * a2 + b2 * b2 + c2 * c2));
        printf("Perpendicular distance is %f\n", d);
    }
    else
        printf("Planes are not parallel");
    return;
}

// Driver Code
int main()
{
    float a1 = 1;
    float b1 = 2;
    float c1 = -1;
    float d1 = 1;
    float a2 = 3;
    float b2 = 6;
    float c2 = -3;
    float d2 = -4;
    distance(a1, b1, c1, d1, a2, b2, c2, d2);      // Fxn cal
    return 0;
}

// This code is contributed
// by Amber_Saxena.
```

Java

```
// Java program to find the Distance
// between two parallel Planes in 3 D.
import java.io.*;
import java.lang.Math;

class GFG
{

    // Function to find distance
    static void distance(float a1, float b1, float c1,
```

```
        float d1, float a2, float b2,
        float c2, float d2)
{

    float x1,y1,z1,d;
    if (a1 / a2 == b1 / b2 &&
        b1 / b2 == c1 / c2)
    {
        x1 = y1 = 0;
        z1 = -d1 / c1;
        d = Math.abs((c2 * z1 + d2)) /
            (float)(Math.sqrt(a2 * a2 + b2 *
                b2 + c2 * c2));
        System.out.println("Perpendicular distance is "+ d);
    }
    else
        System.out.println("Planes are not parallel");
}

// Driver code
public static void main(String[] args)
{
    float a1 = 1;
    float b1 = 2;
    float c1 = -1;
    float d1 = 1;
    float a2 = 3;
    float b2 = 6;
    float c2 = -3;
    float d2 = -4;
    distance(a1, b1, c1, d1,
              a2, b2, c2, d2); // Fxn cal
}
}

// This code is contributed
// by Amber_Saxena.
```

Python

```
# Python program to find the Distance between
# two parallel Planes in 3 D.

import math

# Function to find distance
def distance(a1, b1, c1, d1, a2, b2, c2, d2):
```

```
if (a1 / a2 == b1 / b2 and b1 / b2 == c1 / c2):
    x1 = y1 = 0
    z1 = -d1 / c1
    d = abs((c2 * z1 + d2)) / (math.sqrt(a2 * a2 + b2 * b2 + c2 * c2))
    print("Perpendicular distance is"), d
else:
    print("Planes are not parallel")

# Driver Code
a1 = 1
b1 = 2
c1 = -1
d1 = 1
a2 = 3
b2 = 6
c2 = -3
d2 = -4
distance(a1, b1, c1, d1, a2, b2, c2, d2)      # Fxn cal
```

C#

```
// C# program to find the Distance
// between two parallel Planes in 3 D.
using System;

class GFG
{
    // Function to find distance
    static void distance(float a1, float b1,
    float c1, float d1,
    float a2, float b2,
    float c2, float d2)
    {
        float z1, d;
        if (a1 / a2 == b1 / b2 &&
        b1 / b2 == c1 / c2)
        {
            z1 = -d1 / c1;
            d = Math.Abs((c2 * z1 + d2)) /
            (float)(Math.Sqrt(a2 * a2 + b2 *
            b2 + c2 * c2));
            Console.WriteLine("Perpendicular distance is " + d);
        }
        else
            Console.WriteLine("Planes are not parallel");
    }

    // Driver code
    public static void Main()
```

```
{  
float a1 = 1;  
float b1 = 2;  
float c1 = -1;  
float d1 = 1;  
float a2 = 3;  
float b2 = 6;  
float c2 = -3;  
float d2 = -4;  
distance(a1, b1, c1, d1,  
a2, b2, c2, d2); // Fxn cal  
}  
}  
  
// This code is contributed  
// by ChitraNayal
```

PHP

```
<?php  
// PHP program to find the Distance  
// between two parallel Planes in 3 D  
  
// Function to find distance  
function distance($a1, $b1, $c1,  
                  $d1, $a2, $b2,  
                  $c2, $d2)  
{  
    if ($a1 / $a2 == $b1 / $b2 &&  
        $b1 / $b2 == $c1 / $c2)  
    {  
        $x1 = $y1 = 0;  
        $z1 = - $d1 / $c1;  
        $d = abs(($c2 * $z1 + $d2)) /  
            (sqrt($a2 * $a2 + $b2 *  
                   $b2 + $c2 * $c2));  
        echo "Perpendicular distance is ", $d;  
    }  
    else  
        echo "Planes are not parallel";  
}  
  
// Driver Code  
$a1 = 1;  
$b1 = 2;  
$c1 = -1;  
$d1 = 1;  
$a2 = 3;  
$b2 = 6;
```

```
$c2 = -3;  
$d2 = -4;  
distance($a1, $b1, $c1, $d1,  
         $a2, $b2, $c2, $d2);  
  
// This code is contributed  
// by Amber_Saxena.  
?>
```

Output:

Perpendicular distance is 0.952579344416

Improved By : [Amber_Saxena](#), [ChitraNayal](#)

Source

<https://www.geeksforgeeks.org/distance-between-two-parallel-planes-in-3-d/>

Chapter 62

Divide cuboid into cubes such that sum of volumes is maximum

Divide cuboid into cubes such that sum of volumes is maximum - GeeksforGeeks

Given the **length**, **breadth**, **height** of a cuboid. The task is to divide the given cuboid in minimum number of cubes such that size of all cubes is same and sum of volumes of cubes is maximum.

Examples:

```
Input : l = 2, b = 4, h = 6
Output : 2 6
A cuboid of length 2, breadth 4 and
height 6 can be divided into 6 cube
of side equal to 2.
Volume of cubes = 6*(2*2*2) = 6*8 = 48.
Volume of cuboid = 2*4*6 = 48.
```

```
Input : 1 2 3
Output : 1 6
```

First of all, we are not allowed to waste volume of cuboid as we need maximum volume sum. So, each side should be completely divided among all cubes. And since each of three sides of cubes are equal, so each side of the cuboid needs to be divisible by same number, say x , which will go to be the side of the cube. So, we have to maximize this x , which will divide given length, breadth and height. This x will be maximum only if it is greatest common divisor of given length, breadth and height. So, the length of the cube will be GCD of length, breadth and height.

Now, to compute number of cubes, we know total volume of cuboid and can find volume of one cube (since side is already calculated). So, total number of cubes is equal to (volume of cuboid)/(volume of cube) i.e $(l * b * h) / (x * x * x)$.

Below is implementation of this approach:

C++

```
// CPP program to find optimal way to break
// cuboid into cubes.
#include <bits/stdc++.h>
using namespace std;

// Print the maximum side and no of cube.
void maximizecube(int l, int b, int h)
{
    // GCD to find side.
    int side = __gcd(l, __gcd(b, h));

    // dividing to find number of cubes.
    int num = l / side;
    num = (num * b / side);
    num = (num * h / side);

    cout << side << " " << num << endl;
}

// Driver code
int main()
{
    int l = 2, b = 4, h = 6;

    maximizecube(l, b, h);
    return 0;
}
```

Java

```
// JAVA Code for Divide cuboid into cubes
// such that sum of volumes is maximum
import java.util.*;

class GFG {

    static int gcd(int m, int n)
    {
        if(n == 0)
```

```
        return m;
    else if(n > m)
        return gcd(n,m);
    else
        return gcd(n, m % n);
}

// Print the maximum side and no
//      of cube.
static void maximizecube(int l, int b,
                         int h)
{
    // GCD to find side.
    int side = gcd(l, gcd(b, h));

    // dividing to find number of cubes.
    int num = l / side;
    num = (num * b / side);
    num = (num * h / side);

    System.out.println( side + " " + num);
}

/* Driver program */
public static void main(String[] args)
{
    int l = 2, b = 4, h = 6;
    maximizecube(l, b, h);
}
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
# Python3 code to find optimal way to break
# cuboid into cubes.
from fractions import gcd

# Print the maximum side and no of cube.
def maximizecube( l , b , h ):

    # GCD to find side.
    side = gcd(l, gcd(b, h))

    # dividing to find number of cubes.
    num = int(l / side)
    num = int(num * b / side)
```

```
num = int(num * h / side)

print(side, num)

# Driver code
l = 2
b = 4
h = 6

maximizecube(l, b, h)

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# Code for Divide cuboid into cubes
// such that sum of volumes is maximum
using System;

class GFG {

    static int gcd(int m, int n)
    {
        if(n == 0)
            return m;
        else if(n > m)
            return gcd(n,m);
        else
            return gcd(n, m % n);
    }

    // Print the maximum side and no
    // of cube.
    static void maximizecube(int l, int b,
                            int h)
    {
        // GCD to find side.
        int side = gcd(l, gcd(b, h));

        // dividing to find number of cubes.
        int num = l / side;
        num = (num * b / side);
        num = (num * h / side);

        Console.WriteLine( side + " " + num);
    }

    /* Driver program */
```

```
public static void Main()
{
    int l = 2, b = 4, h = 6;
    maximizecube(l, b, h);
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find optimal way
// to break cuboid into cubes.

// Recursive function to
// return gcd of a and b
function __gcd($a, $b)
{

    // Everything divides 0
    if($a == 0 or $b == 0)
        return 0;

    // base case
    if($a == $b)
        return $a;

    // a is greater
    if($a > $b)
        return __gcd($a - $b, $b);

    return __gcd($a, $b - $a);
}

// Print the maximum side and no of cube.
function maximizecube($l, $b, $h)
{

    // GCD to find side.
    $side = __gcd($l, __gcd($b, $h));

    // dividing to find number of cubes.
    $num = $l / $side;
    $num = ($num * $b / $side);
    $num = ($num * $h / $side);
}
```

```
echo $side , " " , $num ;  
}  
  
// Driver code  
$l = 2;  
$b = 4;  
$h = 6;  
maximizecube($l, $b, $h);  
  
// This code is contributed by anuj_67.  
?>
```

Output:

2 6

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/divide-cuboid-cubes-sum-volumes-maximum/>

Chapter 63

Draw geometric shapes on images using OpenCV

Draw geometric shapes on images using OpenCV - GeeksforGeeks

[OpenCV](#) provides many drawing functions to draw geometric shapes and write text on images. Let's see some of the drawing functions and draw geometric shapes on images using OpenCV.

Some of the drawing functions are :

- cv2.line()** : Used to draw line on an image.
- cv2.rectangle()** : Used to draw rectangle on an image.
- cv2.circle()** : Used to draw circle on an image.
- cv2.putText()** : Used to write text on image.

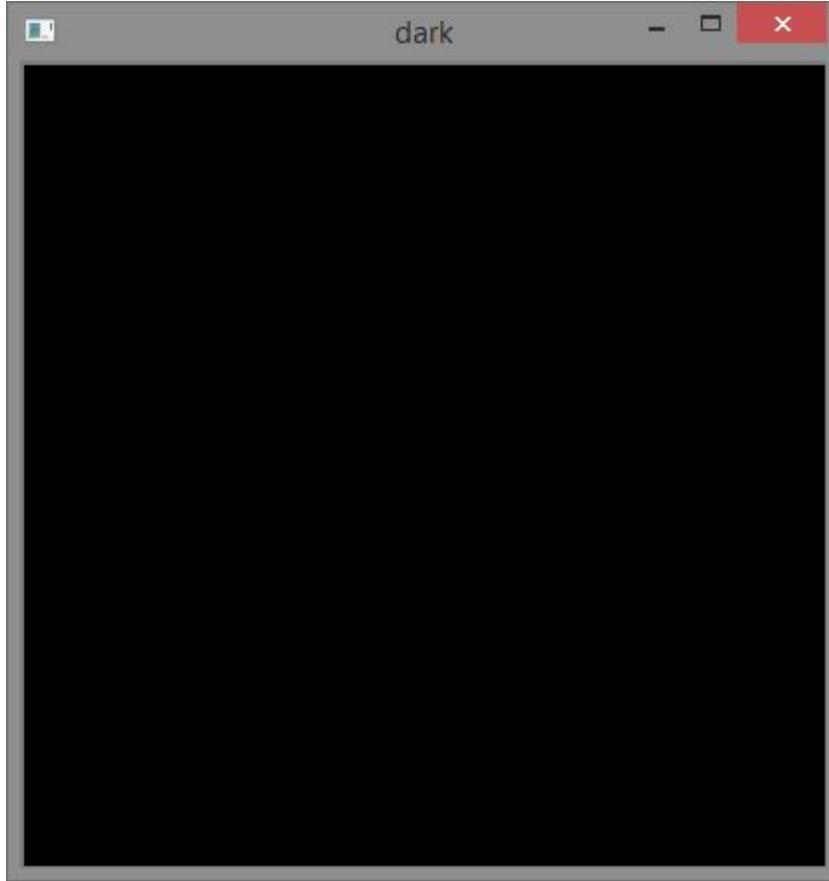
To demonstrate uses of the above-mentioned functions we need an image of size 400 X 400 filled with a solid colour (black in this case). Inorder to do this, We can utilize **numpy.zeros** function to create the required image.

```
# Python3 program to draw solid-colored
# image using numpy.zeros() function
import numpy as np
import cv2

# Creating a black image with 3 channels
# RGB and unsigned int datatype
img = np.zeros((400, 400, 3), dtype = "uint8")
cv2.imshow('dark', img)

# Allows us to see image
# untill closed forcefully
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output :



Now, let's draw some geometric shapes on this solid black image.

Draw a line :

```
cv2.line(imageObjectName,      ('start_coordinates'),      ('end_coordinates'),
         ('color_in_bgr'), 'line_thickness')

# Python3 program to draw line
# shape on solid image
import numpy as np
import cv2

# Creating a black image with 3 channels
# RGB and unsigned int datatype
img = np.zeros((400, 400, 3), dtype = "uint8")

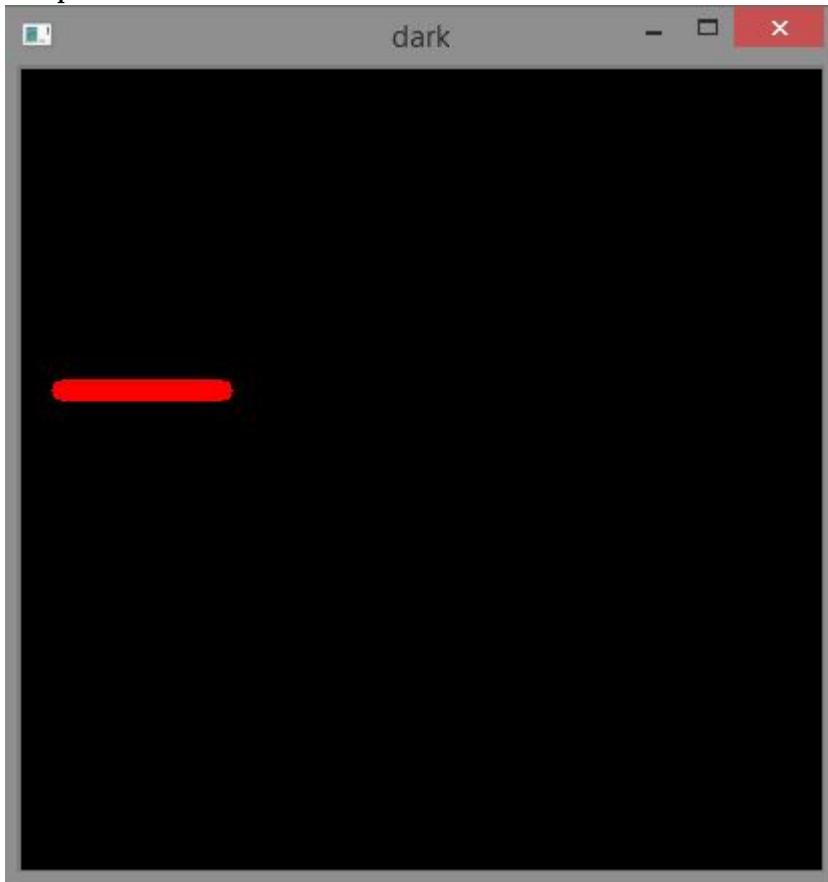
# Creating line
```

```
cv2.line(img, (20, 160), (100, 160), (0, 0, 255), 10)

cv2.imshow('dark', img)

# Allows us to see image
# until closed forcefully
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output :



Draw a rectangle :

```
cv2.rectangle(imageObjectName, ('top_left_vertex_coordinates'), ('lower_right_vertex_coordinates'),
('stroke_color_in_bgr'), 'stroke_thickness')

# Python3 program to draw rectangle
# shape on solid image
```

```
import numpy as np
import cv2

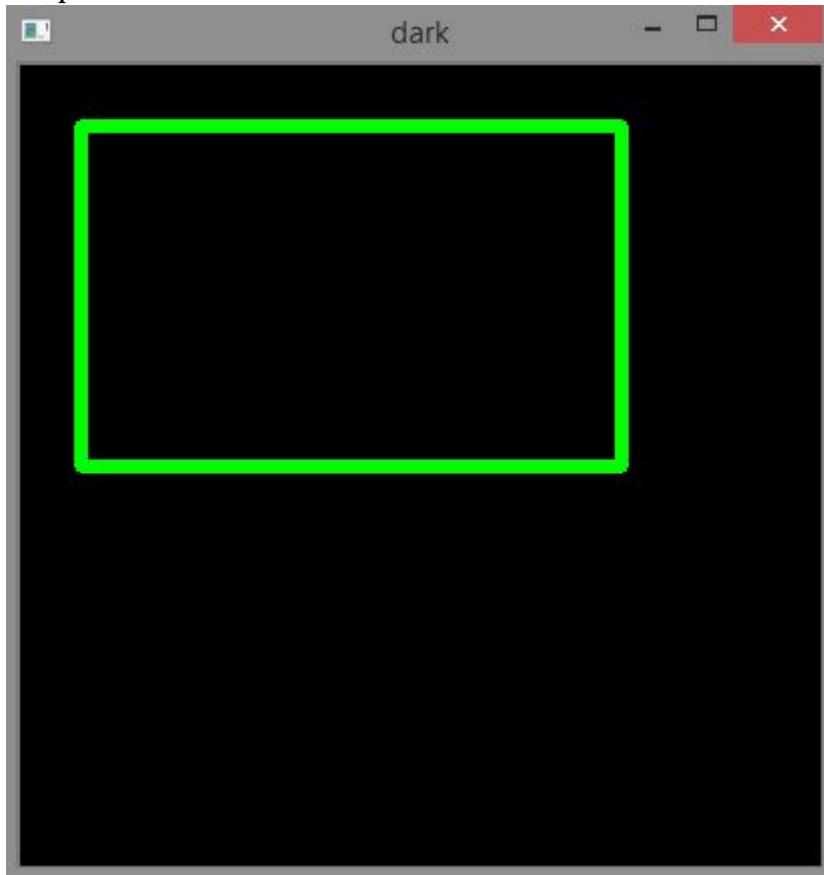
# Creating a black image with 3
# channels RGB and unsigned int datatype
img = np.zeros((400, 400, 3), dtype = "uint8")

# Creating rectangle
cv2.rectangle(img, (30, 30), (300, 200), (0, 255, 0), 5)

cv2.imshow('dark', img)

# Allows us to see image
# untill closed forcefully
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output :



Draw a Circle :

```
cv2.circle(imageObjectName,      ('center_coordinates'),      ('circle_radius'),
           ('color_in_bgr'), 'stroke_thickness')

# Python3 program to draw circle
# shape on solid image
import numpy as np
import cv2

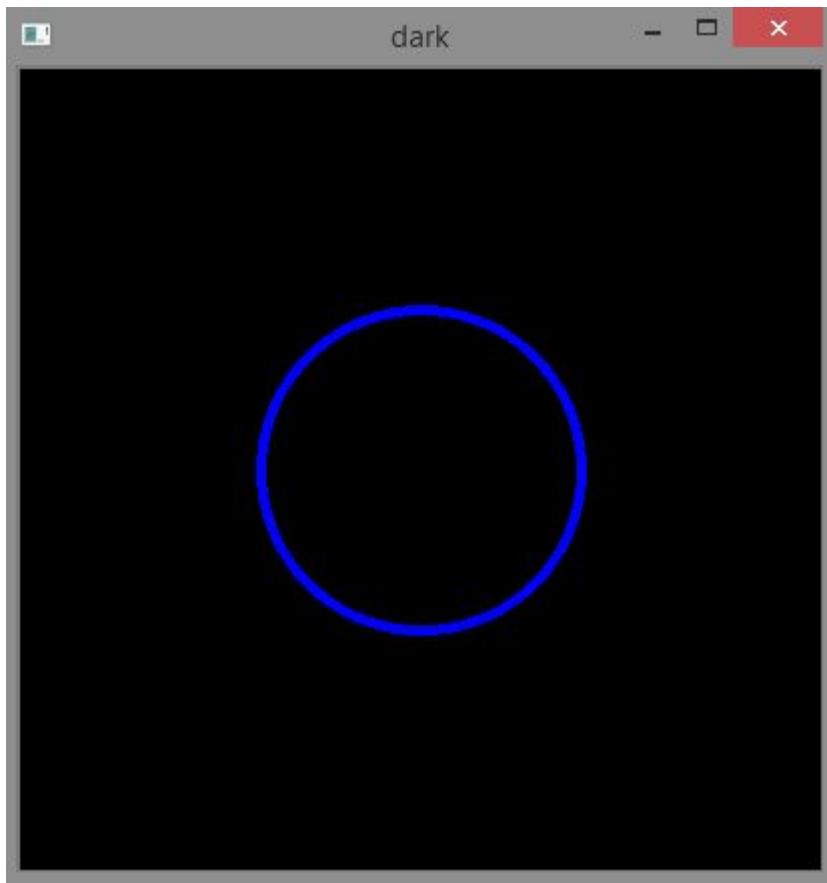
# Creating a black image with 3
# channels RGB and unsigned int datatype
img = np.zeros((400, 400, 3), dtype = "uint8")

# Creating circle
cv2.circle(img, (200, 200), 80, (255, 0, 0), 3)

cv2.imshow('dark', img)

# Allows us to see image
# untill closed forcefully
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output :



Writing text :

```
cv2.putText(imageObjectName, 'TextContent', ('text_starting_point_coordinates'),
           'fontToBeUsed', 'font_size', ('text_color', 'text_thickness', 'line_type')

# Python3 program to write
# text on solid image
import numpy as np
import cv2

# Creating a black image with 3
# channels RGB and unsigned int datatype
img = np.zeros((400, 400, 3), dtype = "uint8")

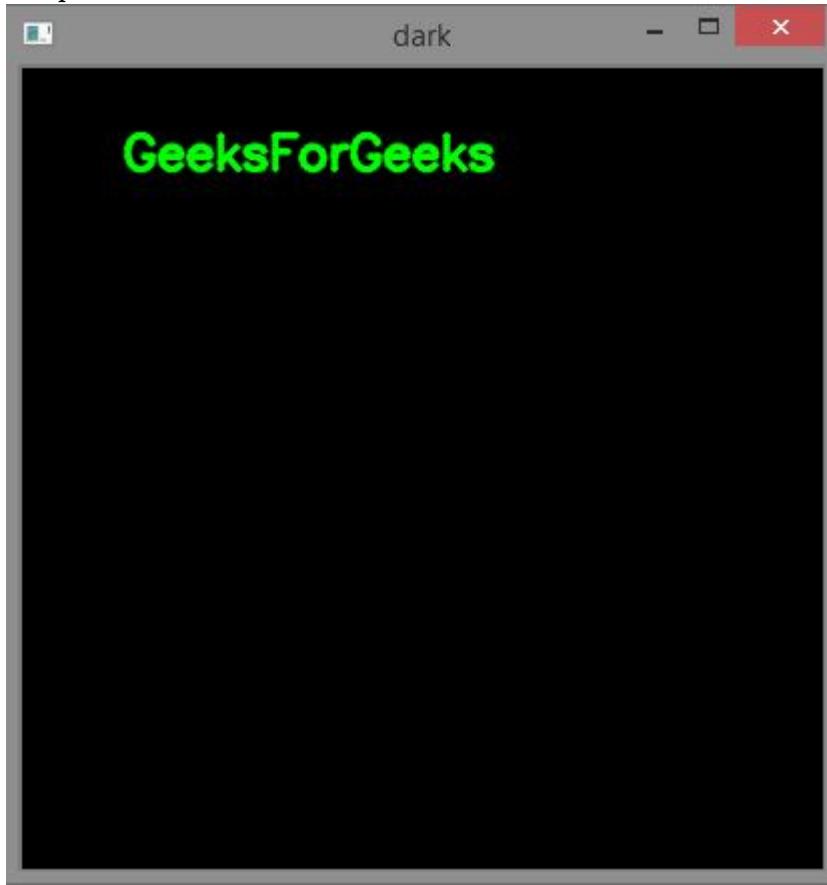
# writing text
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img, 'GeeksForGeeks', (50, 50),
```

```
font, 0.8, (0, 255, 0), 2, cv2.LINE_AA)

cv2.imshow('dark', img)

# Allows us to see image
# until closed forcefully
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output :



Applications of drawing shapes on images :

- Drawing geometrical shapes can help us highlight the particular portions of an image.
- Geometrical shapes like line can help us point or identify particular regions in image.
- Writing text on certain regions of images can add description to that region.

Reference :

https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html

Source

<https://www.geeksforgeeks.org/draw-geometric-shapes-images-using-opencv/>

Chapter 64

Dynamic Convex hull | Adding Points to an Existing Convex Hull

Dynamic Convex hull | Adding Points to an Existing Convex Hull - GeeksforGeeks

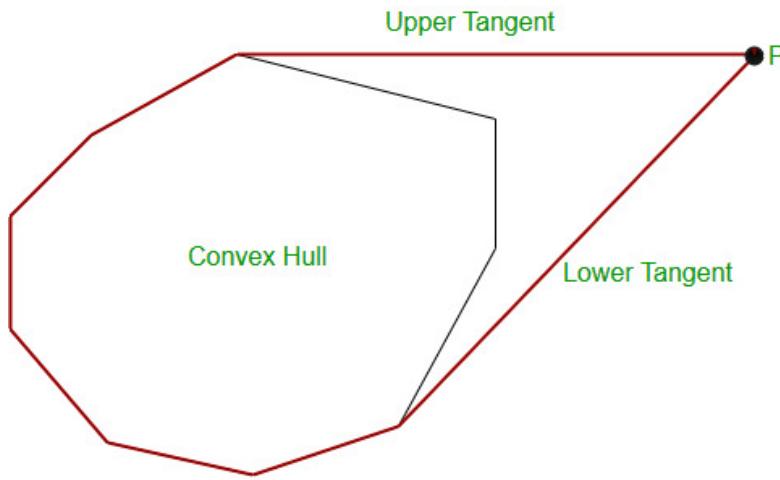
Given a convex hull, we need to add a given number of points to the convex hull and print the convex hull after every point addition. The points should be in anti-clockwise order after addition of every point.

Examples:

```
Input :  
Convex Hull : (0, 0), (3, -1), (4, 5), (-1, 4)  
Point to add : (100, 100)
```

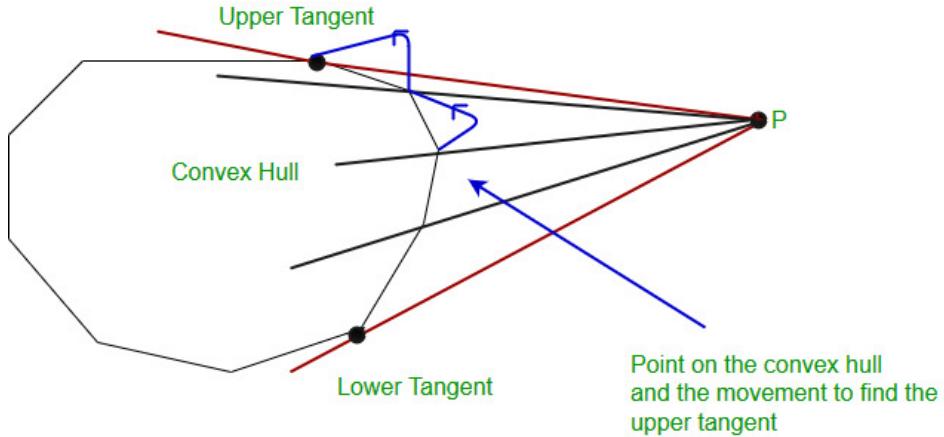
```
Output :  
New convex hull : (-1, 4) (0, 0) (3, -1) (100, 100)
```

We first check whether the point is inside the given convex hull or not. If it is, then nothing has to be done we directly return the given convex hull. If the point is outside the convex hull, we find the lower and upper tangents, and then merge the point with the given convex hull to find the new convex hull, as shown in the figure.



The red outline shows the new convex hull after merging the point and the given convex hull.

To find the upper tangent, we first choose a point on the hull that is nearest to the given point. Then while the line joining the point on the convex hull and the given point crosses the convex hull, we move anti-clockwise till we get the tangent line.



The figure shows the moving of the point on the convex hull for finding the upper tangent.

Note: It is assumed here that the input of the initial convex hull is in the anti-clockwise order, otherwise we have to first sort them in anti-clockwise order then apply the following code.

Code:

```
// C++ program to add given a point p to a given
// convex hull. The program assumes that the
// point of given convex hull are in anti-clockwise
// order.
#include<bits/stdc++.h>
```

```

using namespace std;

// checks whether the point crosses the convex hull
// or not
int orientation(pair<int, int> a, pair<int, int> b,
                pair<int, int> c)
{
    int res = (b.second-a.second)*(c.first-b.first) -
              (c.second-b.second)*(b.first-a.first);

    if (res == 0)
        return 0;
    if (res > 0)
        return 1;
    return -1;
}

// Returns the square of distance between two input points
int sqDist(pair<int, int> p1, pair<int, int> p2)
{
    return (p1.first-p2.first)*(p1.first-p2.first) +
           (p1.second-p2.second)*(p1.second-p2.second);
}

// Checks whether the point is inside the convex hull or not
bool inside(vector<pair<int, int>> a, pair<int, int> p)
{
    // Initialize the centroid of the convex hull
    pair<int, int> mid = {0, 0};

    int n = a.size();

    // Multiplying with n to avoid floating point
    // arithmetic.
    p.first *= n;
    p.second *= n;
    for (int i=0; i<n; i++)
    {
        mid.first += a[i].first;
        mid.second += a[i].second;
        a[i].first *= n;
        a[i].second *= n;
    }

    // if the mid and the given point lies always
    // on the same side w.r.t every edge of the
    // convex hull, then the point lies inside
    // the convex hull
}

```

```

for (int i=0, j; i<n; i++)
{
    j = (i+1)%n;
    int x1 = a[i].first, x2 = a[j].first,
        int y1 = a[i].second, y2 = a[j].second;
    int a1 = y1-y2;
    int b1 = x2-x1;
    int c1 = x1*y2-y1*x2;
    int for_mid = a1*mid.first+b1*mid.second+c1;
    int for_p = a1*p.first+b1*p.second+c1;
    if (for_mid*for_p < 0)
        return false;
}

return true;
}

// Adds a point p to given convex hull a[]
void addPoint(vector<pair<int, int>> &a, pair<int, int> p)
{
    // If point is inside p
    if (inside(a, p))
        return;

    // point having minimum distance from the point p
    int ind = 0;
    int n = a.size();
    for (int i=1; i<n; i++)
        if (sqDist(p, a[i]) < sqDist(p, a[ind]))
            ind = i;

    // Find the upper tangent
    int up = ind;
    while (orientation(p, a[up], a[(up+1)%n])>=0)
        up = (up + 1) % n;

    // Find the lower tangent
    int low = ind;
    while (orientation(p, a[low], a[(n+low-1)%n])<=0)
        low = (n+low - 1) % n;

    // Initialize result
    vector<pair<int, int>>ret;

    // making the final hull by traversing points
    // from up to low of given convex hull.
    int curr = up;
    ret.push_back(a[curr]);
}

```

```
while (curr != low)
{
    curr = (curr+1)%n;
    ret.push_back(a[curr]);
}

// Modify the original vector
ret.push_back(p);
a.clear();
for (int i=0; i<ret.size(); i++)
    a.push_back(ret[i]);
}

// Driver code
int main()
{
    // the set of points in the convex hull
    vector<pair<int, int> > a;
    a.push_back({0, 0});
    a.push_back({3, -1});
    a.push_back({4, 5});
    a.push_back({-1, 4});
    int n = a.size();

    pair<int, int> p = {100, 100};
    addPoint(a, p);

    // Print the modified Convex Hull
    for (auto e : a)
        cout << "(" << e.first << ", "
              << e.second << ") ";
}

return 0;
}
```

Output:

(-1, 4) (0, 0) (3, -1) (100, 100)

Time Complexity:

The time complexity of the above algorithm is $O(n*q)$, where q is the number of points to be added.

Source

<https://www.geeksforgeeks.org/dynamic-convex-hull-adding-points-existing-convex-hull/>

Chapter 65

Enneadecagonal number

Enneadecagonal number - GeeksforGeeks

Given a number n, the task is to find the nth Enneadecagonal number.

An Enneadecagonal number is a nineteen-sided polygon in mathematics. It belongs to a class of figurative number. The number contains the number of dots and the dots are arranged in a pattern or series. An Enneadecagonal number is also known as nonadecagon. The dots have common points and all other dots are arranged in the successive layer.

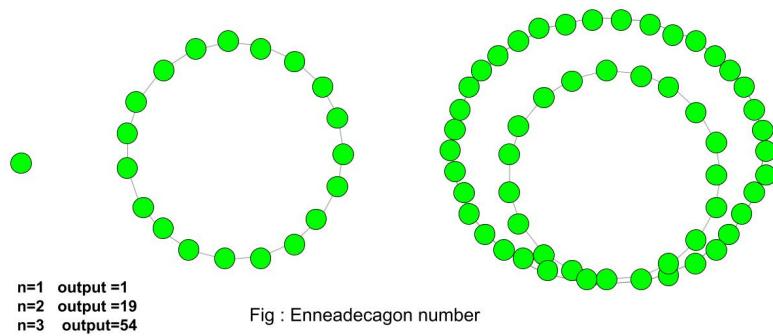
Examples :

Input : 4

Output :106

Input :10

Output :775



Formula to find nth Enneadecagonal number :

C++

```
// C++ program to find
// nth Enneadecagonal number
#include <bits/stdc++.h>
using namespace std;

// Function to calculate
// Enneadecagonal number
int nthEnneadecagonal(long int n)
{
    // Formula for finding
    // nth Enneadecagonal number
    return (17 * n * n - 15 * n) / 2;
}

// Drivers code
int main()
{
    long int n = 6;
    cout << n << "th Enneadecagonal number :" << nthEnneadecagonal(n);
    return 0;
}
```

Java

```
// Java program to find
// nth Enneadecagonal number
import java.io.*;

class GFG {

    // Function to calculate
    // Enneadecagonal number
    static int nthEnneadecagonal(int n)
    {

        // Formula for finding
        // nth Enneadecagonal number
        return (17 * n * n - 15 * n) / 2;
    }

    // Driver Code
    public static void main (String[] args)
```

```
{  
  
    int n = 6;  
    System.out.print(n + "th Enneadecagonal number :");  
  
    System.out.println( nthEnneadecagonal(n));  
}  
}  
  
// This code is contributed by m_kit.
```

Python3

```
# Program to find nth  
# Enneadecagonal number  
  
def nthEnneadecagonal(n) :  
  
    # Formula to calculate nth  
    # Enneadecagonal number  
    return (17 * n * n - 15 * n) // 2  
  
# Driver Code  
if __name__ == '__main__' :  
  
    n = 6  
    print(n,"th Enneadecagonal number :"  
          , nthEnneadecagonal(n))  
  
# This code is contributed by Ajit
```

C#

```
// C# program to find  
// nth Enneadecagonal number  
using System;  
  
class GFG  
{  
    // Function to calculate  
    // Enneadecagonal number  
    static int nthEnneadecagonal(int n)  
    {  
  
        // Formula for finding  
        // nth Enneadecagonal number  
        return (17 * n * n - 15 * n) / 2;
```

```
}

// Driver Code
static public void Main ()
{
    int n = 6;
    Console.WriteLine(n + "th Enneadecagonal number :");

    Console.WriteLine( nthEnneadecagonal(n));
}
}

// This code is contributed by aj_36
```

PHP

```
<?php
// PHP program to find
// nth Enneadecagonal number

// Function to calculate
// Enneadecagonal number
function nthEnneadecagonal($n)
{
    // Formula for finding
    // nth Enneadecagonal number
    return (17 * $n * $n -
           15 * $n) / 2;
}

// Driver Code
$n = 6;
echo $n , "th Enneadecagonal number :" ,
      nthEnneadecagonal($n);

// This code is contributed by ajit
?>
```

Output :

6th Enneadecagonal number :261

Reference: https://en.wikipedia.org/wiki/Polygonal_number

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/enneadecagonal-number/>

Chapter 66

Equable Shapes

Equable Shapes - GeeksforGeeks

A shape is equable if its area is equal to its perimeter. Given ordered coordinates of polygon find whether the shape is equable or not.

Examples :

Input : X[] = {0, 5, 0}
Y[] = {0, 0, 12}
Output : Equable Shape

Input : X[] = {0, 4, 4, 0}
Y[] = {0, 0, 4, 4}
Output : Equable Shape

Input: X[] = {0, 6, 6, 0}
Y[] = {0, 0, 4, 4}
Output: Not Equable Shape

We can find area of polygon using shoelace formula which is described in [Area of a polygon with given n ordered vertices](#). We can also find its perimeter simply by adding distances between adjacent points.

C++

```
// C++ program to find equable shape
#include <bits/stdc++.h>
using namespace std;

// To calculate area of polygon
double polygonArea(double X[], double Y[], int n)
{
```

```
double area = 0.0;

// Calculate value of area using shoelace
// formula
int j = n - 1;
for (int i = 0; i < n; i++) {
    area += (X[j] + X[i]) * (Y[j] - Y[i]);
    j = i; // j is previous vertex to i
}

return abs(area / 2.0);
}

// To calculate perimeter of polygon
double polygonPerimeter(double X[], double Y[],
                        int n)
{
    double perimeter = 0.0;

    // Calculate value of perimeter
    int j = n - 1;
    for (int i = 0; i < n; i++) {
        perimeter += sqrt((X[j] - X[i]) * (X[j] - X[i]) +
                           (Y[j] - Y[i]) * (Y[j] - Y[i]));
        j = i; // j is previous vertex to i
    }

    return perimeter;
}

// To find equable shape
void equableShape(double X[], double Y[], int n)
{
    // Find area and perimeter of polygon if
    // they are equal then it is equable shape
    if (polygonPerimeter(X, Y, n) == polygonArea(X, Y, n))
        cout << "Equable Shape";
    else
        cout << "Not Equable Shape";
}

// Driver program to test above function
int main()
{
    double X[] = { 0, 5, 0 };
    double Y[] = { 0, 0, 12 };

    int n = sizeof(X) / sizeof(X[0]);
```

```
equableShape(X, Y, n);

return 0;
}

Java

// Java program to find equable shape
class equable {

    // To calculate area of polygon
    static double polygonArea(double X[], double Y[], int n)
    {
        double area = 0.0;

        // Calculate value of area using shoelace formula
        int j = n - 1;
        for (int i = 0; i < n; i++) {
            area += (X[j] + X[i]) * (Y[j] - Y[i]);
            j = i; // j is previous vertex to i
        }

        return Math.abs(area / 2.0);
    }

    // To calculate perimeter of polygon
    static double polygonPerimeter(double X[], double Y[], int n)
    {
        double perimeter = 0.0;

        // Calculate value of perimeter
        int j = n - 1;
        for (int i = 0; i < n; i++) {
            perimeter += Math.sqrt((X[j] - X[i]) * (X[j] - X[i)) +
                (Y[j] - Y[i]) * (Y[j] - Y[i]));
            j = i; // j is previous vertex to i
        }

        return perimeter;
    }

    // To find equable shape
    static void equableShape(double X[], double Y[], int n)
    {
        // Find area and perimeter of polygon if
        // they are equal then it is equable shape
        if (polygonPerimeter(X, Y, n) == polygonArea(X, Y, n))

```

```
        System.out.println("Equable Shape");
    else
        System.out.println("Not Equable Shape");
}

// Driver program to test above function
public static void main(String[] args)
{
    double X[] = { 0, 5, 0 };
    double Y[] = { 0, 0, 12 };

    int n = X.length;

    equableShape(X, Y, n);
}
}
```

Python 3

```
# Python 3 program to find equable shape
# To calculate area of polygon

import math
def polygonArea(X, Y, n):
    area = 0.0

    # Calculate value of area
    # using shoelace formula
    j = n - 1
    for i in range(n):
        area += (X[j] + X[i]) * (Y[j] - Y[i])

        # j is previous vertex to i
        j = i
    return abs(area / 2.0)

# To calculate perimeter of polygon
def polygonPerimeter(X, Y, n):
    perimeter = 0.0

    # Calculate value of perimeter
    j = n - 1
    for i in range(n):
        perimeter += math.sqrt((X[j] - X[i]) * (X[j] - X[i]) +
                               (Y[j] - Y[i]) * (Y[j] - Y[i]))

        # j is previous vertex to i
        j = i
```

```
        return perimeter

# To find equable shape
def equableShape(X, Y, n):
    # Find area and perimeter of polygon if
    # they are equal then it is equable shape
    if (polygonPerimeter(X, Y, n) == polygonArea(X, Y, n)):
        print("Equable Shape")
    else:
        print("Not Equable Shape")

# Driver program to test above function
X = [ 0, 5, 0 ]
Y = [ 0, 0, 12 ]
n = len(X)
equableShape(X, Y, n)

# This code is contributed by Azkia Anam.
```

C#

```
// C# program to find equable shape
using System;

class equable {

    // To calculate area of polygon
    static double polygonArea(double []X,
                              double []Y,
                              int n)
    {
        double area = 0.0;

        // Calculate value of area using
        // Shoelace Formula
        int j = n - 1;
        for (int i = 0; i < n; i++)
        {
            area += (X[j] + X[i]) * (Y[j] - Y[i]);
            j = i; // j is previous vertex to i
        }

        return Math.Abs(area / 2.0);
    }

    // To calculate perimeter of polygon
    static double polygonPerimeter(double []X,
```

```
        double []Y,
        int n)
{
    double perimeter = 0.0;

    // Calculate value of perimeter
    int j = n - 1;
    for (int i = 0; i < n; i++) {
        perimeter += Math.Sqrt((X[j] - X[i]) *
                               (X[j] - X[i]) +
                               (Y[j] - Y[i]) *
                               (Y[j] - Y[i]));
        j = i; // j is previous vertex to i
    }

    return perimeter;
}

// To find equable shape
static void equableShape(double []X,
                        double []Y,
                        int n)
{
    // Find area and perimeter of
    // polygon if they are equal
    // then it is equable shape
    if (polygonPerimeter(X, Y, n) ==
        polygonArea(X, Y, n))
        Console.WriteLine("Equable Shape");
    else
        Console.WriteLine("Not Equable Shape");
}

// Driver Code
public static void Main(String []args)
{
    double []X = {0, 5, 0};
    double []Y = {0, 0, 12};

    int n = X.Length;

    // Calling Function
    equableShape(X, Y, n);
}
}

// This Code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find
// equable shape

// To calculate area
// of polygon
function polygonArea($X, $Y, $n)
{
    $area = 0.0;

    // Calculate value of area
    // using shoelace formula
    $j = $n - 1;
    for ($i = 0; $i < $n; $i++)
    {
        $area += ($X[$j] + $X[$i]) *
            ($Y[$j] - $Y[$i]);

        // j is previous vertex to i
        $j = $i;
    }

    return abs($area / 2.0);
}

// To calculate perimeter of polygon
function polygonPerimeter($X, $Y, $n)

{
    $perimeter = 0.0;

    // Calculate value of perimeter
    $j = $n - 1;
    for ($i = 0; $i < $n; $i++)
    {
        $perimeter += sqrt((($X[$j] - $X[$i]) *
            ($X[$j] - $X[$i])) +
            (($Y[$j] - $Y[$i]) *
            ($Y[$j] - $Y[$i])));
    }

    // j is previous vertex to i
    $j = $i;
}

return $perimeter;
}
```

```
// To find equable shape
function equableShape($X, $Y, $n)
{
    // Find area and perimeter of
    // polygon if they are equal
    // then it is equable shape
    if (polygonPerimeter($X, $Y, $n) ==
        polygonArea($X, $Y, $n))
        echo "Equable Shape";
    else
        echo "Not Equable Shape";
}

// Driver Code
$X = array( 0, 5, 0 );
$Y = array( 0, 0, 12 );

$n = sizeof($X);

equableShape($X, $Y, $n);

// This code is contributed by ajit
?>
```

Output :

Equable Shape

Reference:

https://en.wikipedia.org/wiki/Equable_shape

Improved By : [vt_m, jit_t](#)

Source

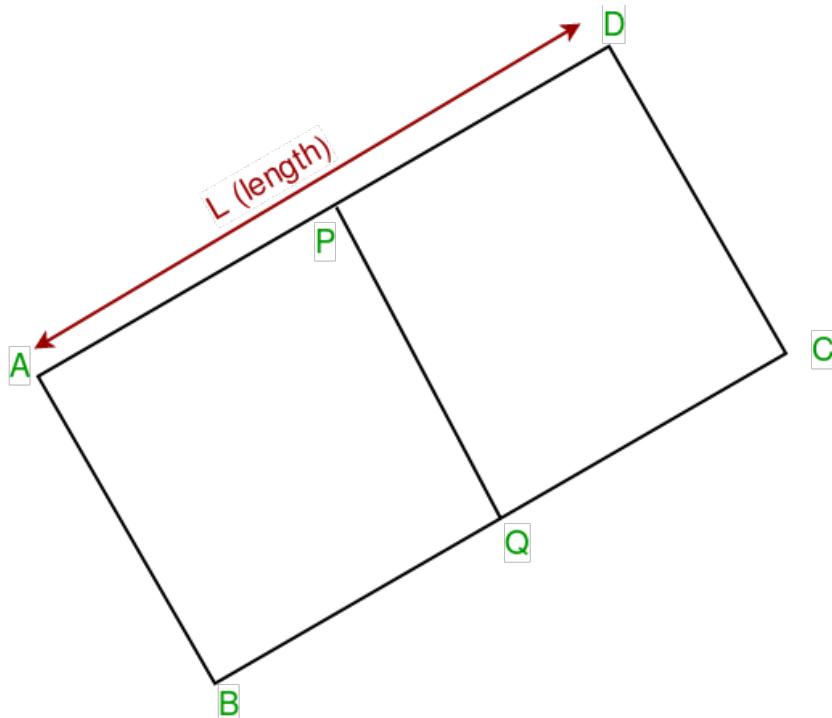
<https://www.geeksforgeeks.org/equable-shapes/>

Chapter 67

Find Corners of Rectangle using mid points

Find Corners of Rectangle using mid points - GeeksforGeeks

Consider a rectangle ABCD, we're given the co-ordinates of the mid points of side AD and BC (p and q respectively) along with their length L (AD = BC = L). Now given the parameters, we need to print the co-ordinates of the 4 points A, B, C and D.



Examples:

```
Input : p = (1, 0)
        q = (1, 2)
        L = 2
Output : (0, 0), (0, 2), (2, 2), (2, 0)
```

Explanation:

The printed points form a rectangle which satisfy the input constraints.

```
Input : p = (1, 1)
        q = (-1, -1)
        L = 2*sqrt(2)
Output : (0, 2), (-2, 0), (0, -2), (2, 0)
```

From the problem statement 3 cases can arise :

1. The Rectangle is horizontal i.e., AD and BC are parallel to X-axis
2. The Rectangle is vertical i.e., AD and BC are parallel to Y-axis
3. The Rectangle is inclined at a certain angle with the axes

The first two cases are trivial and can easily be solved using basic geometry. For the third case we need to apply some mathematical concepts to find the points.

Consider the above diagram for clarity. We have the co-ordinates of p and q. Thus we can find the slope of AD and BC (As pq is perpendicular to AD). Once we have the slope of AD, we can find the equation of straight line passing through AD. Now we can apply distance formula to obtain the displacements along X and Y axes.

If slope of AD = m, then
 $m = (p.x - q.x)/(q.y - p.y)$

and displacement along X axis, $dx = L/(2*sqrt(1+m*m))$

Similarly, $dy = m*L/(2*sqrt(1+m*m))$

Now we can simply find the co-ordinates of 4 corners by simply adding and subtracting the displacements obtained accordingly.

Below is the implementation in C++.

```
// C++ program to find corner points of
// a rectangle using given length and middle
// points.
#include <bits/stdc++.h>
using namespace std;
```

```
// Structure to represent a co-ordinate point
struct Point
{
    float x, y;
    Point()
    {
        x = y = 0;
    }
    Point(float a, float b)
    {
        x = a, y = b;
    }
};

// This function receives two points and length
// of the side of rectangle and prints the 4
// corner points of the rectangle
void printCorners(Point p, Point q, float l)
{
    Point a, b, c, d;

    // horizontal rectangle
    if (p.x == q.x)
    {
        a.x = p.x - (l/2.0);
        a.y = p.y;

        d.x = p.x + (l/2.0);
        d.y = p.y;

        b.x = q.x - (l/2.0);
        b.y = q.y;

        c.x = q.x + (l/2.0);
        c.y = q.y;
    }

    // vertical rectangle
    else if (p.y == q.y)
    {
        a.y = p.y - (l/2.0);
        a.x = p.x;

        d.y = p.y + (l/2.0);
        d.x = p.x;

        b.y = q.y - (l/2.0);
    }
}
```

```
b.x = q.x;
c.y = q.y + (l/2.0);
c.x = q.x;
}

// slanted rectangle
else
{
    // calculate slope of the side
    float m = (p.x-q.x)/float(q.y-p.y);

    // calculate displacements along axes
    float dx = (l / sqrt(1+(m*m))) *0.5 ;
    float dy = m*dx;

    a.x = p.x - dx;
    a.y = p.y - dy;

    d.x = p.x + dx;
    d.y = p.y + dy;

    b.x = q.x - dx;
    b.y = q.y - dy;

    c.x = q.x + dx;
    c.y = q.y + dy;
}

cout << a.x << ", " << a.y << " n"
    << b.x << ", " << b.y << "n";
    << c.x << ", " << c.y << " n"
    << d.x << ", " << d.y << "nn";
}

// Driver code
int main()
{
    Point p1(1, 0), q1(1, 2);
    printCorners(p1, q1, 2);

    Point p(1, 1), q(-1, -1);
    printCorners(p, q, 2*sqrt(2));

    return 0;
}
```

Output:

```
0, 0  
0, 2  
2, 2  
2, 0
```

```
0, 2  
-2, 0  
0, -2  
2, 0
```

Reference:

[StackOverflow](#)

Source

<https://www.geeksforgeeks.org/find-corners-of-rectangle-using-mid-points/>

Chapter 68

Find Simple Closed Path for a given set of points

Find Simple Closed Path for a given set of points - GeeksforGeeks

Given a set of points, connect the dots without crossing.

Example:

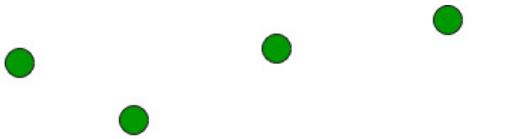
```
Input: points[] = {(0, 3), (1, 1), (2, 2), (4, 4),
                   (0, 0), (1, 2), (3, 1), {3, 3}};
```

```
Output: Connecting points in following order would
        not cause any crossing
        {(0, 0), (3, 1), (1, 1), (2, 2), (3, 3),
         (4, 4), (1, 2), (0, 3)}
```

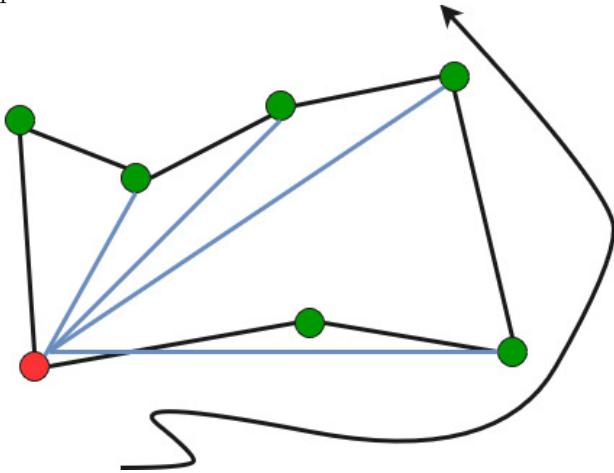
We strongly recommend you to minimize your browser and try this yourself first.

The idea is to use sorting.

1. Find the bottom-most point by comparing y coordinate of all points. If there are two points with same y value, then the point with smaller x coordinate value is considered.
Put the bottom-most point at first position.



2. Consider the remaining $n-1$ points and sort them by polar angle in counterclockwise order around points[0]. If polar angle of two points is same, then put the nearest point first.
3. Traversing the sorted array (sorted in increasing order of angle) yields simple closed path.



How to compute angles?

One solution is to use trigonometric functions.

Observation: We don't care about the actual values of the angles. We just want to sort by angle.

Idea: Use the [orientation](#) to compare angles without actually computing them!

Below is C++ implementation of above idea.

```
// A C++ program to find simple closed path for n points
// for explanation of orientation()
#include <bits/stdc++.h>
using namespace std;

struct Point
{
    int x, y;
};


```

```
// A global point needed for sorting points with reference
// to the first point. Used in compare function of qsort()
Point p0;

// A utility function to swap two points
int swap(Point &p1, Point &p2)
{
    Point temp = p1;
    p1 = p2;
    p2 = temp;
}

// A utility function to return square of distance between
// p1 and p2
int dist(Point p1, Point p2)
{
    return (p1.x - p2.x)*(p1.x - p2.x) +
           (p1.y - p2.y)*(p1.y - p2.y);
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0; // colinear
    return (val > 0)? 1: 2; // clockwise or counterclock wise
}

// A function used by library function qsort() to sort
// an array of points with respect to the first point
int compare(const void *vp1, const void *vp2)
{
    Point *p1 = (Point *)vp1;
    Point *p2 = (Point *)vp2;

    // Find orientation
    int o = orientation(p0, *p1, *p2);
    if (o == 0)
        return (dist(p0, *p2) >= dist(p0, *p1))? -1 : 1;

    return (o == 2)? -1: 1;
}
```

```
}

// Prints simple closed path for a set of n points.
void printClosedPath(Point points[], int n)
{
    // Find the bottommost point
    int ymin = points[0].y, min = 0;
    for (int i = 1; i < n; i++)
    {
        int y = points[i].y;

        // Pick the bottom-most. In case of tie, chose the
        // left most point
        if ((y < ymin) || (ymin == y &&
            points[i].x < points[min].x))
            ymin = points[i].y, min = i;
    }

    // Place the bottom-most point at first position
    swap(points[0], points[min]);

    // Sort n-1 points with respect to the first point.
    // A point p1 comes before p2 in sorted output if p2
    // has larger polar angle (in counterclockwise
    // direction) than p1
    p0 = points[0];
    qsort(&points[1], n-1, sizeof(Point), compare);

    // Now stack has the output points, print contents
    // of stack
    for (int i=0; i<n; i++)
        cout << "(" << points[i].x << ", "
            << points[i].y <<")", " ;
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},
                      {0, 0}, {1, 2}, {3, 1}, {3, 3}};
    int n = sizeof(points)/sizeof(points[0]);
    printClosedPath(points, n);
    return 0;
}
```

Output:

(0, 0), (3, 1), (1, 1), (2, 2), (3, 3),
(4, 4), (1, 2), (0, 3),

Time complexity of above solution is $O(n \log n)$ if we use a $O(n \log n)$ sorting algorithm for sorting points.

Source:

<http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf>

This article is contributed by **Rajeev Agrawal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/find-simple-closed-path-for-a-given-set-of-points/>

Chapter 69

Find all angles of a given triangle

Find all angles of a given triangle - GeeksforGeeks

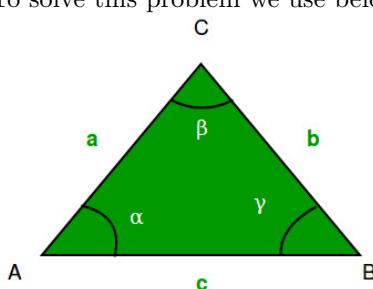
Given coordinates of all three vertices of the triangle in the 2D plane, the task is to find all three angles.

Example:

Input : A = (0, 0),
B = (0, 1),
C = (1, 0)

Output : 90, 45, 45

To solve this problem we use below [Law of cosines](#).



$$c^2 = a^2 + b^2 - 2(a)(b)(\cos \gamma)$$

After re-arranging

```
gamma = acos( ( a^2 + b^2 - c^2 ) / (2ab) )
```

In trigonometry, the law of cosines (also known as the cosine formula or cosine rule) relates the lengths of the sides of a triangle to the cosine of one of its angles.

First, calculate the length of all the sides.
Then apply above formula to get all angles in
radian. Then convert angles from radian into
degrees.

Below is implementation of above steps.

C++

```
// Code to find all three angles
// of a triangle given coordinate
// of all three vertices
#include <iostream>
#include <utility> // for pair
#include <cmath> // for math functions
using namespace std;

#define PI 3.1415926535

// returns square of distance b/w two points
int lengthSquare(pair<int,int> X, pair<int,int> Y)
{
    int xDiff = X.first - Y.first;
    int yDiff = X.second - Y.second;
    return xDiff*xDiff + yDiff*yDiff;
}

void printAngle(pair<int,int> A, pair<int,int> B,
                pair<int,int> C)
{
    // Square of lengths be a2, b2, c2
    int a2 = lengthSquare(B,C);
    int b2 = lengthSquare(A,C);
    int c2 = lengthSquare(A,B);

    // lenght of sides be a, b, c
    float a = sqrt(a2);
    float b = sqrt(b2);
    float c = sqrt(c2);
```

```
// From Cosine law
float alpha = acos((b2 + c2 - a2)/(2*b*c));
float betta = acos((a2 + c2 - b2)/(2*a*c));
float gamma = acos((a2 + b2 - c2)/(2*a*b));

// Converting to degree
alpha = alpha * 180 / PI;
betta = betta * 180 / PI;
gamma = gamma * 180 / PI;

// printing all the angles
cout << "alpha : " << alpha << endl;
cout << "betta : " << betta << endl;
cout << "gamma : " << gamma << endl;
}

// Driver code
int main()
{
    pair<int,int> A = make_pair(0,0);
    pair<int,int> B = make_pair(0,1);
    pair<int,int> C = make_pair(1,0);

    printAngle(A,B,C);

    return 0;
}
```

Java

```
// Java Code to find all three angles
// of a triangle given coordinate
// of all three vertices

import java.awt.Point;
import static java.lang.Math.PI;
import static java.lang.Math.sqrt;
import static java.lang.Math.acos;

class Test
{
    // returns square of distance b/w two points
    static int lengthSquare(Point p1, Point p2)
    {
        int xDiff = p1.x- p2.x;
        int yDiff = p1.y- p2.y;
        return xDiff*xDiff + yDiff*yDiff;
    }
}
```

```
static void printAngle(Point A, Point B,
                      Point C)
{
    // Square of lengths be a2, b2, c2
    int a2 = lengthSquare(B,C);
    int b2 = lengthSquare(A,C);
    int c2 = lengthSquare(A,B);

    // lenght of sides be a, b, c
    float a = (float)sqrt(a2);
    float b = (float)sqrt(b2);
    float c = (float)sqrt(c2);

    // From Cosine law
    float alpha = (float) acos((b2 + c2 - a2)/(2*a*c));
    float betta = (float) acos((a2 + c2 - b2)/(2*a*c));
    float gamma = (float) acos((a2 + b2 - c2)/(2*a*b));

    // Converting to degree
    alpha = (float) (alpha * 180 / PI);
    betta = (float) (betta * 180 / PI);
    gamma = (float) (gamma * 180 / PI);

    // printing all the angles
    System.out.println("alpha : " + alpha);
    System.out.println("betta : " + betta);
    System.out.println("gamma : " + gamma);
}

// Driver method
public static void main(String[] args)
{
    Point A = new Point(0,0);
    Point B = new Point(0,1);
    Point C = new Point(1,0);

    printAngle(A,B,C);
}
```

Output:

```
alpha : 90
betta : 45
gamma : 45
```

Reference :

https://en.wikipedia.org/wiki/Law_of_cosines

Source

<https://www.geeksforgeeks.org/find-angles-given-triangle/>

Chapter 70

Find all angles of a triangle in 3D

Find all angles of a triangle in 3D - GeeksforGeeks

Given coordinates of 3 vertices of a triangle in 3D i.e. A(x₁, y₁, z₁), B(x₂, y₂, z₂), C(x₃, y₃, z₃). The task is to find out all the angles of the triangle formed by above coordinates.

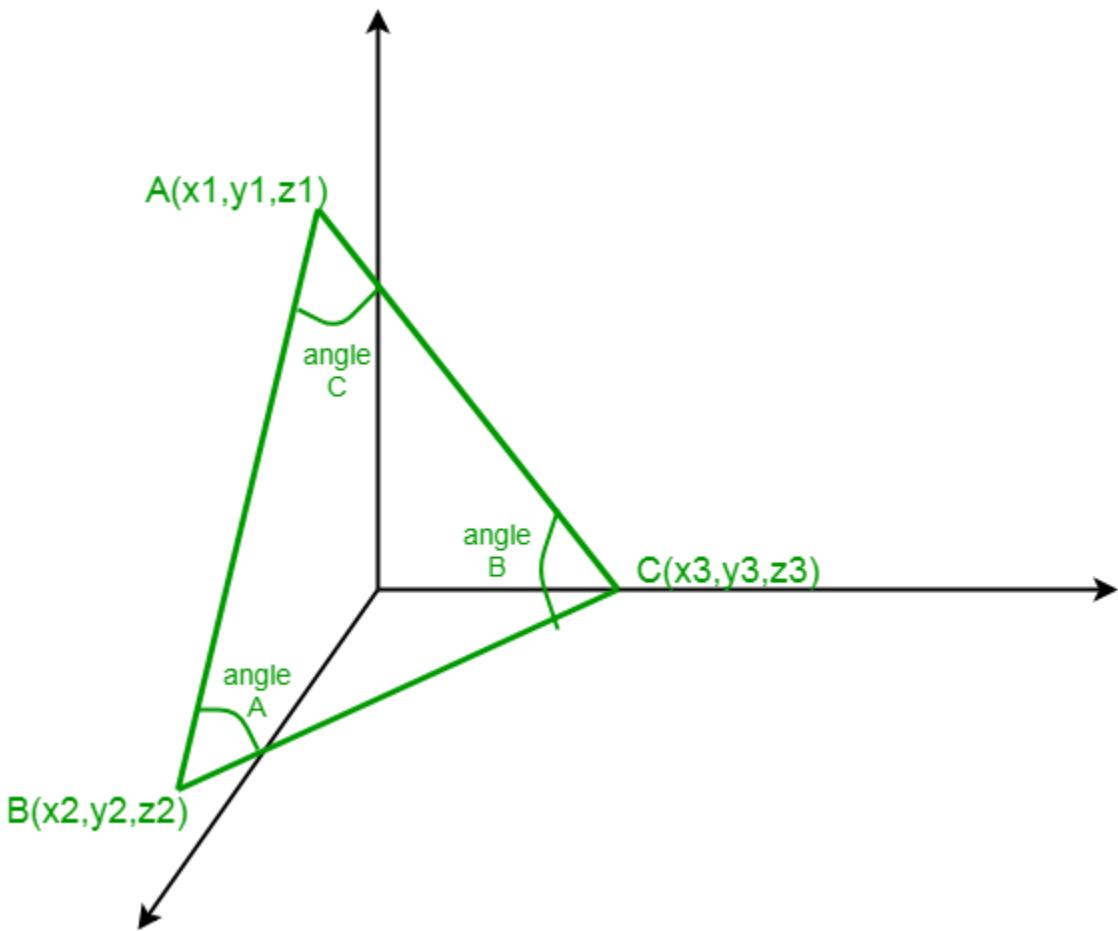
Examples:

Input:

```
x1 = -1, y1 = 3, z1 = 2  
x2 = 2, y2 = 3, z2 = 5  
x3 = 3, y3 = 5, z3 = -2
```

Output:

```
angle A = 90.0 degree  
angle B = 54.736 degree  
angle C = 35.264 degree
```



Approach:

For finding angle A find out direction ratios of AB and AC :

direction ratios of AB = $x_2 - x_1, y_2 - y_1, z_2 - z_1$

direction ratios of AC = $x_3 - x_1, y_3 - y_1, z_3 - z_1$

$$\text{then angle } A = \frac{1}{2} \cos^{-1} \left(\frac{(x_2 - x_1)(x_3 - x_1) + (y_2 - y_1)(y_3 - y_1) + (z_2 - z_1)(z_3 - z_1)}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2 + (z_3 - z_1)^2}} \right)$$

then angle A =

For finding angle B find out direction ratios of BA and BC :

direction ratios of BA = $x_1 - x_2, y_1 - y_2, z_1 - z_2$

direction ratios of BC = $x_3 - x_2, y_3 - y_2, z_3 - z_2$

$$\text{then angle } B = \frac{1}{2} \cos^{-1} \left(\frac{(x_1 - x_2)(x_3 - x_2) + (y_1 - y_2)(y_3 - y_2) + (z_1 - z_2)(z_3 - z_2)}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2 + (z_3 - z_2)^2}} \right)$$

then angle B =

For finding angle C find out direction ratios of CB and CA :

direction ratios of CB = x2-x3, y2-y3, z2-z3
 direction ratios of CA = x1-x3, y1-y3, z1-z3

$$\text{then angle C} = \frac{\sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2 + (z_2 - z_3)^2}}{\sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2 + (z_1 - z_3)^2}}$$

Below is the implementation of above approach:

```
# Python Code for finding all angles of a triangle
import math

# function for finding the angle
def angle_triangle(x1, x2, x3, y1, y2, y3, z1, z2, z3):

    num = (x2-x1)*(x3-x1)+(y2-y1)*(y3-y1)+(z2-z1)*(z3-z1)

    den = math.sqrt((x2-x1)**2+(y2-y1)**2+(z2-z1)**2)*\
          math.sqrt((x3-x1)**2+(y3-y1)**2+(z3-z1)**2)

    angle = math.degrees(math.acos(num / den))

    return round(angle, 3)

# driver code
x1 = -1
y1 = 3
z1 = 2
x2 = 2
y2 = 3
z2 = 5
x3 = 3
y3 = 5
z3 = -2
angle_A = angle_triangle(x1, x2, x3, y1, y2, y3, z1, z2, z3)
angle_B = angle_triangle(x2, x3, x1, y2, y3, y1, z2, z3, z1)
angle_C = angle_triangle(x3, x2, x1, y3, y2, y1, z3, z2, z1)
print("Angles are :")
print("angle A = ", angle_A, "degree")
print("angle B = ", angle_B, "degree")
print("angle C = ", angle_C, "degree")
```

Output:

```
Angles are :
angle A = 90.0 degree
angle B = 54.736 degree
angle C = 35.264 degree
```

Source

<https://www.geeksforgeeks.org/find-all-angles-of-a-triangle-in-3d/>

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Chapter 71

Find all possible coordinates of parallelogram

Find all possible coordinates of parallelogram - GeeksforGeeks

Find the all the possible coordinate from the given three coordinates to make a parallelogram of a non-zero area.

Let's call A,B,C are the three given points. We can have only the three possible situations:

- (1) AB and AC are sides, and BC a diagonal
- (2) AB and BC are sides, and AC a diagonal
- (3) BC and AC are sides, and AB a diagonal

Hence, we can say that only 3 coordinates are possible from which we can generate a parallelogram if three coordinates are given.

To prove that all the three points are different let's suppose it's wrong. Without losing of generality suppose that the points got in cases AD and BC are equal.

Consider the system of two equations for the equality of these points:

$$\begin{aligned}Bx + Cx - Ax &= Ax + Cx - Bx \\By + Cy - Ay &= Ay + Cy - By\end{aligned}$$

It can be simplified as-

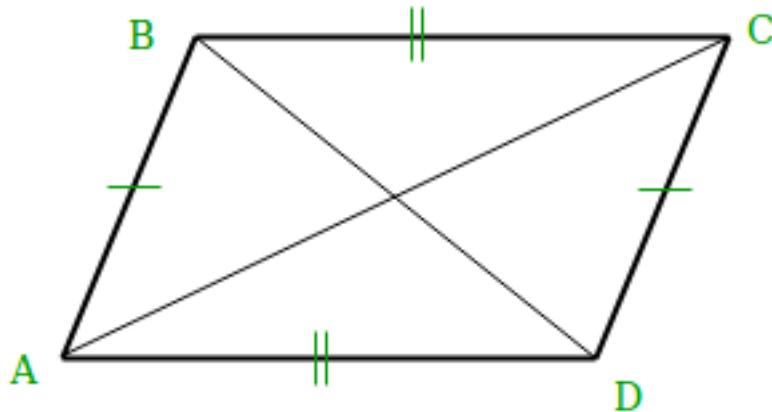
$$\begin{aligned}Ax &= Bx \\Ay &= By\end{aligned}$$

And we got a contradiction, as all the points A, B, C are distinct.

Examples:

```
Input : A = (0 0)
        B = (1 0)
        C = (0 1)
Output : 1 -1
         -1 1
         1 1
```

```
Input : A = (-1 -1)
        B = (0 1)
        C = (1 1)
Output : -2 -1
         0 -1
         2 3
```



Since the opposite sides are equal, $AD = BC$ and $AB = CD$, we can calculate the co-ordinates of the missing point (D) as:

$$\begin{aligned}AD &= BC \\(Dx - Ax, Dy - Ay) &= (Cx - Bx, Cy - By) \\Dx &= Ax + Cx - Bx \\Dy &= Ay + Cy - By\end{aligned}$$

The cases where the diagonals are AD and BC, CD and AB are processed in the same way.

Reference: <https://math.stackexchange.com/questions/1322535/how-many-different-parallelograms-can-be-drawn>

Below is the implementation of above approach:

C++

```
// C++ program to all possible points
// of a parallelogram
#include <bits/stdc++.h>
```

```
using namespace std;

// main method
int main()
{
    int ax = 5, ay = 0; //coordinates of A
    int bx = 1, by = 1; //coordinates of B
    int cx = 2, cy = 5; //coordinates of C
    cout << ax + bx - cx << ", "
        << ay + by - cy << endl;
    cout << ax + cx - bx << ", "
        << ay + cy - by << endl;
    cout << cx + bx - ax << ", "
        << cy + by - ax << endl;
    return 0;
}
```

Java

```
// Java program to all possible
// points of a parallelogram
public class ParallelogramPoints{

    // Driver code
    public static void main(String[] s)
    {
        int ax = 5, ay = 0; //coordinates of A
        int bx = 1, by = 1; //coordinates of B
        int cx = 2, cy = 5; //coordinates of C
        System.out.println(ax + bx - cx + ", "
            + (ay + by - cy));
        System.out.println(ax + cx - bx + ", "
            + (ay + cy - by));
        System.out.println(cx + bx - ax + ", "
            + (cy + by - ax));
    }
}

// This code is contributed by Prerna Saini
```

Python3

```
# Python3 program to find all possible points
# of a parallelogram

ax = 5
ay = 0 #coordinates of A
```

```
bx = 1
by = 1 #coordinates of B
cx = 2
cy = 5 #coordinates of C
print(ax + bx - cx, ", ", ay + by - cy)
print(ax + cx - bx, ", ", ay + cy - by)
print(cx + bx - ax, ", ", cy + by - ax)
```

C#

```
// C# program to all possible
// points of a parallelogram
using System;

public class ParallelogramPoints
{

    // Driver code
    public static void Main()
    {

        //coordinates of A
        int ax = 5, ay = 0;

        //coordinates of B
        int bx = 1, by = 1;

        //coordinates of C
        int cx = 2, cy = 5;

        Console.WriteLine(ax + bx - cx + ", "
                        + (ay + by - cy));
        Console.WriteLine(ax + cx - bx + ", "
                        + (ay + cy - by));
        Console.WriteLine(cx + bx - ax + ", "
                        + (cy + by - ax));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to all
// possible points
// of a parallelogram
```

```
// Driver Code  
  
//coordinates of A  
$ax = 5; $ay = 0;  
  
//coordinates of B  
$bx = 1; $by = 1;  
  
//coordinates of C  
$cx = 2; $cy = 5;  
  
echo $ax + $bx - $cx , " , "  
     , $ay + $by - $cy , "\n";  
echo $ax + $cx - $bx , " , "  
     , $ay + $cy - $by, "\n" ;  
echo $cx + $bx - $ax , " , "  
     , $cy + $by - $ax ;  
  
// This code is contributed by anuj_67.  
?>
```

Output:

```
4, -4  
6, 4  
-2, 1
```

Time Complexity: O(1)

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-possible-coordinates-parallelogram/>

Chapter 72

Find all sides of a right angled triangle from given hypotenuse and area | Set 1

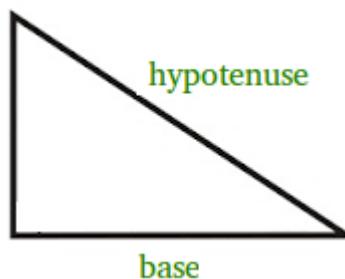
Find all sides of a right angled triangle from given hypotenuse and area | Set 1 - Geeks-forGeeks

Given hypotenuse and area of a right angle triangle, get its base and height and if any triangle with given hypotenuse and area is not possible, print not possible.

Examples:

Input : hypotenuse = 5, area = 6
Output : base = 3, height = 4

Input : hypotenuse = 5, area = 7
Output : No triangle possible with above specification.



We can use a property of right angle triangle for solving this problem, which can be stated as follows,

A right angle triangle with fixed hypotenuse attains maximum area, when it is isosceles i.e. both height and base becomes equal so if hypotenuse is H, then by pythagorean theorem,
 $\text{Base}^2 + \text{Height}^2 = H^2$

For maximum area both base and height should be equal,
 $b^2 + b^2 = H^2$
 $b = \sqrt{H^2/2}$

Above is the length of base at which triangle attains maximum area, given area must be less than this maximum area, otherwise no such triangle will possible.

Now if given area is less than this maximum area, we can do a binary search for length of base, as increasing base will increases area, it is a monotonically increasing function where binary search can be applied easily.

In below code, a method is written for getting area of right angle triangle, recall that for right angle triangle area is $\frac{1}{2} * \text{base} * \text{height}$ and height can be calculated from base and hypotenuse using [pythagorean theorem](#).

```
// C++ program to get right angle triangle, given
// hypotenuse and area of triangle
#include <bits/stdc++.h>
using namespace std;

// limit for float comparison
#define eps 1e-6

// Utility method to get area of right angle triangle,
// given base and hypotenuse
double getArea(double base, double hypotenuse)
{
    double height = sqrt(hypotenuse*hypotenuse - base*base);
    return 0.5 * base * height;
}

// Prints base and height of triangle using hypotenuse
// and area information
void printRightAngleTriangle(int hypotenuse, int area)
{
    int hsquare = hypotenuse * hypotenuse;

    // maximum area will be obtained when base and height
    // are equal (= sqrt(h*h/2))
    double sideForMaxArea = sqrt(hsquare / 2.0);
    double maxArea = getArea(sideForMaxArea, hypotenuse);
```

```
// if given area itself is larger than maxArea then no
// solution is possible
if (area > maxArea)
{
    cout << "Not possible";
    return;
}

double low = 0.0;
double high = sideForMaxArea;
double base;

// binary search for base
while (abs(high - low) > eps)
{
    base = (low + high) / 2.0;
    if (getArea(base, hypotenuse) >= area)
        high = base;
    else
        low = base;
}

// get height by pythagorean rule
double height = sqrt(hsquare - base*base);
cout << base << " " << height << endl;
}

// Driver code to test above methods
int main()
{
    int hypotenuse = 5;
    int area = 6;

    printRightAngleTriangle(hypotenuse, area);
    return 0;
}
```

Output:

3 4

One more solution is discussed in below post.

[Check if right angles possible from given area and hypotenuse](#)

Chapter 72. Find all sides of a right angled triangle from given hypotenuse and area | Set 1

Source

<https://www.geeksforgeeks.org/find-sides-right-angled-triangle-given-hypotenuse-area/>

Chapter 73

Find an Integer point on a line segment with given two ends

Find an Integer point on a line segment with given two ends - GeeksforGeeks

Given two points **pointU** and **pointV** in XY-space, we need to find a point which has integer coordinates and lies on a line going through points pointU and pointV.

Examples:

```
If pointU = (1, -1) and pointV = (-4, 1)
then equation of line which goes
through these two points is,
2X + 5Y = -3
One point with integer co-ordinate which
satisfies above equation is (6, -3)
```

We can see that once we found the equation of line, this problem can be treated as [Extended Euclid algorithm](#) problem, where we know A, B, C in $AX + BY = C$ and we want to find out the value of X and Y from the equation.

In above Extended Euclid equation, C is gcd of A and B, so after finding out the line equation from given two points if C is not a multiple of $\text{gcd}(A, B)$ then we can conclude that there is no possible integer coordinate on the specified line. If C is a multiple of g, then we can scale up the founded X and Y coefficients to satisfy the actual equation, which will be our final answer.

```
// C++ program to get Integer point on a line
#include <bits/stdc++.h>
using namespace std;

// Utility method for extended Euclidean Algorithm
int gcdExtended(int a, int b, int *x, int *y)
```

```
{  
    // Base Case  
    if (a == 0)  
    {  
        *x = 0;  
        *y = 1;  
        return b;  
    }  
  
    int x1, y1; // To store results of recursive call  
    int gcd = gcdExtended(b%a, a, &x1, &y1);  
  
    // Update x and y using results of recursive  
    // call  
    *x = y1 - (b/a) * x1;  
    *y = x1;  
  
    return gcd;  
}  
  
// method prints integer point on a line with two  
// points U and V.  
void printIntegerPoint(int c[], int pointV[])  
{  
    // Getting coefficient of line  
    int A = (pointU[1] - pointV[1]);  
    int B = (pointV[0] - pointU[0]);  
    int C = (pointU[0] * (pointU[1] - pointV[1]) +  
             pointU[1] * (pointV[0] - pointU[0]));  
  
    int x, y; // To be assigned a value by gcdExtended()  
    int g = gcdExtended(A, B, &x, &y);  
  
    // if C is not divisible by g, then no solution  
    // is available  
    if (C % g != 0)  
        cout << "No possible integer point\n";  
  
    else  
  
        // scaling up x and y to satisfy actual answer  
        cout << "Integer Point : " << (x * C/g) << " "  
            << (y * C/g) << endl;  
}  
  
// Driver code to test above methods  
int main()  
{
```

```
int pointU[] = {1, -1};  
int pointV[] = {-4, 1};  
  
printIntegerPoint(pointU, pointV);  
return 0;  
}
```

Output:

```
Integer Point : 6 -3
```

Source

<https://www.geeksforgeeks.org/find-integer-point-line-segment-given-two-ends/>

Chapter 74

Find area of parallelogram if vectors of two adjacent sides are given

Find area of parallelogram if vectors of two adjacent sides are given - GeeksforGeeks

Given two vectors in form of $(xi + yj + zk)$ of two adjacent sides of a parallelogram. The task is to find out the area of a parallelogram.

Example:

Input:

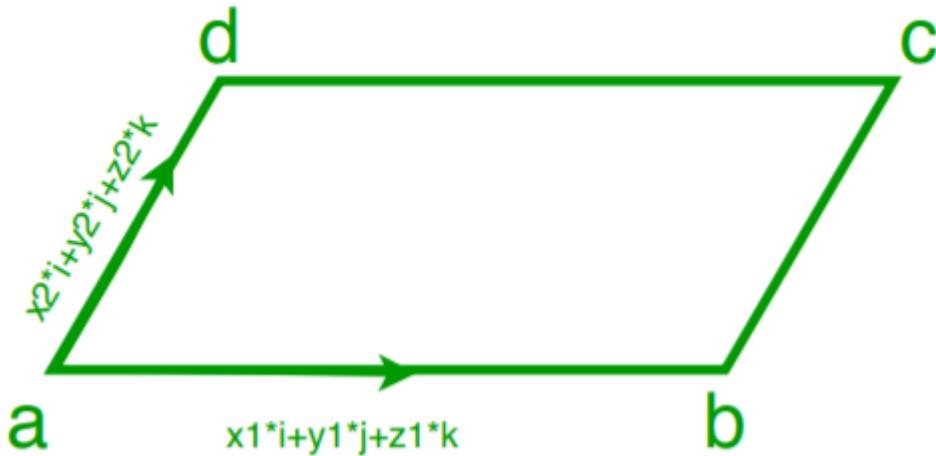
$x_1 = 3, y_1 = 1, z_1 = -2$
 $x_2 = 1, y_2 = -3, z_2 = 4$

Output: Area = 17.3205081

Input:

$x_1 = 1, y_1 = 3, z_1 = 2$
 $x_2 = 1, y_2 = -3, z_2 = 4$

Output: Area = 19.078784028338912



Approach: Suppose we have two vectors $a(x_1\mathbf{i}+y_1\mathbf{j}+z_1\mathbf{k})$ and $b(x_2\mathbf{i}+y_2\mathbf{j}+z_2\mathbf{k})$ and we know that area of parallelogram is given by :

Area of parallelogram = magnitude of cross product of vectors a and b i.e $|axb|$
And we know $a \times b = (y_1z_2 - y_2z_1)\mathbf{i} - (x_1z_2 - x_2z_1)\mathbf{j} + (x_1y_2 - x_2y_1)\mathbf{k}$

Then area =

$$\sqrt{(y_1z_2 - y_2z_1)^2 + (x_1z_2 - x_2z_1)^2 + (x_1y_2 - x_2y_1)^2}$$

Python Code:

```
# Python code to calculate area of
# parallelogram if vectors of
# 2 adjacent sides are given

import math

# to calculate area of parallelogram
def area(x1, y1, z1, x2, y2, z2):
    area = math.sqrt((y1 * z2 - y2 * z1) ** 2
                    + (x1 * z2 - x2 * z1) ** 2 +
                    (x1 * y2 - x2 * y1) ** 2)
    return area

# main function
def main():
    x1 = 3
    y1 = 1
    z1 = -2
```

```
x2 = 1
y2 = -3
z2 = 4
a = area(x1, y1, z1, x2, y2, z2)
print("Area = ", a)

# driver code
if __name__ == "__main__":
    main()
```

Output:

```
Area = 17.320508075688775
```

Source

<https://www.geeksforgeeks.org/find-area-of-parallelogram-if-vectors-of-two-adjacent-sides-are-given/>

Chapter 75

Find if it's possible to rotate the page by an angle or not.

Find if it's possible to rotate the page by an angle or not. - GeeksforGeeks

You are given three points a, b, c on a page. Find if it's possible to rotate the page around the point by an angle, such that the new position of 'a' is same as the old position of 'b', and the new position of 'b' is same as the old position of 'c'. If such angle exists print "Yes", else "No".

Examples:

```
Input : a1 = 0, a2 = 1, b1 = 1, b2 =  1,  
        c1 = 1, c2 = 0  
Output : Yes  
Explanation : Rotate the page by 90 degree.
```

```
Input : a1 = 1, a2 = 1, b1 = 0, b2 = 0,  
        c1 = 1000, c2 = 1000  
Output : No
```

Rotation of page by some angle is only possible if the distance between points 'a' and 'b' is equal to distance between points 'b' and 'c'. But if the points are on same line, there is no rotation at point 'b'. The problem has no solution when 'a', 'b', 'c' are in the same line or $\text{dis}(a, b) \neq \text{dis}(b, c)$

C++

```
// C++ program to find if its possible  
// to rotate page or not  
#include<bits/stdc++.h>
```

```
using namespace std;

// function to find if it's possible
// to rotate page or not
void possibleOrNot(long long a1, long long a2,
                    long long b1, long long b2,
                    long long c1, long long c2){

    // Calculating distance b/w points
    long long dis1 = pow(b1 - a1, 2) +
                     pow(b2 - a2, 2);
    long long dis2 = pow(c1 - b1, 2) +
                     pow(c2 - b2, 2);

    // If distance is not equal
    if(dis1 != dis2)
        cout << "No";

    // If the points are in same line
    else if (b1 == ((a1 + c1) / 2.0) && b2 ==
              ((a2 + c2) / 2.0))
        cout << "No";
    else
        cout << "Yes";
}

// Driver Code
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    // Points a, b, and c
    long long a1 = 1, a2 = 0, b1 = 2,
             b2 = 0, c1 = 3, c2 = 0;
    possibleOrNot(a1, a2, b1, b2, c1, c2);
    return 0;
}
```

Java

```
// java program to find if its possible
// to rotate page or not
import java.util.Arrays;

class GFG {

    // function to find if it's possible
```

```
// to rotate page or not
static void possibleOrNot(long a1,long a2,
                           long b1,long b2,
                           long c1,long c2)
{
    // Calculating distance b/w points
    long dis1 = (long)Math.pow(b1 - a1, 2) +
                (long) Math.pow(b2 - a2, 2);

    long dis2 = (long)Math.pow(c1 - b1, 2) +
                (long)Math.pow(c2 - b2, 2);

    // If distance is not equal
    if(dis1 != dis2)
        System.out.print("No");

    // If the points are in same line
    else if (b1 == ((a1 + c1) / 2.0) &&
              b2 == ((a2 + c2) / 2.0))
        System.out.print("No");
    else
        System.out.print("Yes");
}

// Driver method
public static void main(String[] args)
{
    // Points a, b, and c
    long a1 = 1, a2 = 0, b1 = 2,
          b2 = 0, c1 = 3, c2 = 0;

    possibleOrNot(a1, a2, b1, b2, c1, c2);
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 program to fill an
# array with frequencies.

# Function to find if it's possible
# to rotate page or not
def possibleOrNot(a1, a2, b1, b2, c1, c2):
```

```
# Calculating distance b/w points
dis1 = (pow(b1 - a1, 2) +
        pow(b2 - a2, 2))
dis2 = (pow(c1 - b1, 2) +
        pow(c2 - b2, 2))

# If distance is not equal
if(dis1 != dis2):
    print("No")

# If the points are in same line
elif (b1 == ((a1 + c1) // 2.0) and
      b2 == ((a2 + c2) // 2.0)):
    print("No")
else:
    print("Yes")

# Driver Code

# Points a, b, and c
a1, b1, c1 = 1, 2, 3
a2 = b2 = c2 = 0
possibleOrNot(a1, a2, b1, b2, c1, c2)

# This code is contributed by Anant Agarwal.
```

C#

```
// C# program to fill an array with frequencies.
using System;

class GFG
{
    // function to find if it's possible
    // to rotate page or not
    static void possibleOrNot(long a1,long a2,
                             long b1,long b2,
                             long c1,long c2)
    {

        // Calculating distance b/w points
        long dis1 = (long)Math.Pow(b1 - a1, 2) +
                    (long) Math.Pow(b2 - a2, 2);
        long dis2 = (long)Math.Pow(c1 - b1, 2) +
                    (long)Math.Pow(c2 - b2, 2);

        // If distance is not equal
        if(dis1 != dis2)
```

```
Console.WriteLine("No");

// If the points are in same line
else if (b1 == ((a1 + c1) / 2.0) && b2 ==
          ((a2 + c2) / 2.0))
    Console.WriteLine("No");
else
    Console.WriteLine("Yes");
}

// Driver method
public static void Main()
{
    // Points a, b, and c
    long a1 = 1, a2 = 0, b1 = 2,
          b2 = 0, c1 = 3, c2 = 0;
    possibleOrNot(a1, a2, b1, b2, c1, c2);
}
}

// This code is contributed by Anant Agarwal.
```

PHP

```
<?php
// PHP program to fill
// an array with frequencies.

// function to find if
// it's possible to
// rotate page or not
function possibleOrNot($a1, $a2, $b1,
                      $b2, $c1, $c2)
{

    // Calculating distance
    // b/w points
    $dis1 = pow($b1 - $a1, 2) +
            pow($b2 - $a2, 2);
    $dis2 = pow($c1 - $b1, 2) +
            pow($c2 - $b2, 2);

    // If distance
    // is not equal
    if($dis1 != $dis2)
        echo "No";

    // If the points
    // are in same line
```

```
else if ($b1 == (($a1 + $c1) / 2.0) &&
         $b2 == (($a2 + $c2) / 2.0))
    echo "No";
else
    echo "Yes";
}

// Driver Code

// Points a, b, and c
$a1 = 1;
$a2 = 0;
$b1 = 2;
$b2 = 0;
$c1 = 3;
$c2 = 0;
possibleOrNot($a1, $a2, $b1,
               $b2, $c1, $c2);

// This code is contributed by ajit
?>
```

Output:

No

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/find-possible-rotate-page-angle-not/>

Chapter 76

Find if two rectangles overlap

Find if two rectangles overlap - GeeksforGeeks

Given two rectangles, find if the given two rectangles overlap or not.

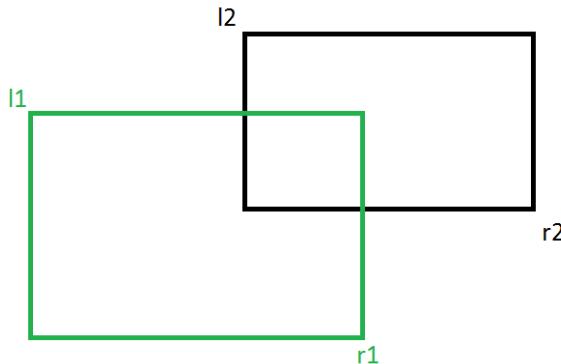
Note that a rectangle can be represented by two coordinates, top left and bottom right. So mainly we are given following four coordinates.

l1: Top Left coordinate of first rectangle.

r1: Bottom Right coordinate of first rectangle.

l2: Top Left coordinate of second rectangle.

r2: Bottom Right coordinate of second rectangle.



We need to write a function `bool doOverlap(l1, r1, l2, r2)` that returns true if the two given rectangles overlap.

Note : It may be assumed that the rectangles are parallel to the coordinate axis.

One solution is to one by one pick all points of one rectangle and [see if the point lies inside the other rectangle or not](#). This can be done using the algorithm discussed [here](#).

Following is a simpler approach. Two rectangles **do not** overlap if one of the following conditions is true.

- 1) One rectangle is above top edge of other rectangle.
- 2) One rectangle is on left side of left edge of other rectangle.

We need to check above cases to find out if given rectangles overlap or not. Following is C++ implementation of the above approach.

```
#include<bits/stdc++.h>

struct Point
{
    int x, y;
};

// Returns true if two rectangles (l1, r1) and (l2, r2) overlap
bool doOverlap(Point l1, Point r1, Point l2, Point r2)
{
    // If one rectangle is on left side of other
    if (l1.x > r2.x || l2.x > r1.x)
        return false;

    // If one rectangle is above other
    if (l1.y < r2.y || l2.y < r1.y)
        return false;

    return true;
}

/* Driver program to test above function */
int main()
{
    Point l1 = {0, 10}, r1 = {10, 0};
    Point l2 = {5, 5}, r2 = {15, 0};
    if (doOverlap(l1, r1, l2, r2))
        printf("Rectangles Overlap");
    else
        printf("Rectangles Don't Overlap");
    return 0;
}
```

Output:

```
Rectangles Overlap
```

Time Complexity of above code is $O(1)$ as the code doesn't have any loop or recursion.

This article is compiled by **Aman Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/find-two-rectangles-overlap/>

Chapter 77

Find intersection point of lines inside a section

Find intersection point of lines inside a section - GeeksforGeeks

Given N lines in two-dimensional space in $y = mx + b$ form and a vertical section. We need to find out whether there is an intersection point inside the given section or not.

Examples:

In below diagram four lines are there,
L1 : $y = x + 2$
L2 : $y = -x + 7$
L3 : $y = -3$
L4 : $y = 2x - 7$
and vertical section is given from $x = 2$ to $x = 4$

We can see that in above diagram, the intersection point of line L1 and L2 lies between the section.

We can solve this problem using sorting. First, we will calculate intersection point of each line with both the boundaries of vertical section and store that as a pair. We just need to store y-coordinates of intersections as a pair because x-coordinates are equal to boundary itself. Now we will sort these pairs on the basis of their intersection with left boundary. After that, we will loop over these pairs one by one and if for any two consecutive pairs, the second value of the current pair is less than that of the second value of the previous pair then there must be an intersection in the given vertical section.

The possible orientation of two consecutive pairs can be seen in above diagram for L1 and L2. We can see that when the second value is less, intersection lies in vertical section.

Total time complexity of solution will be $O(n \log n)$

```
// C++ program to check an intersection point
// inside a given vertical section
#include <bits/stdc++.h>
using namespace std;

// structure to represent a line
struct line {
    int m, b;
    line() { }
    line(int m, int b) : m(m), b(b) { }
};

// Utility method to get Y-coordinate
// corresponding to x in line l
int getYFromLine(line l, int x)
{
    return (l.m * x + l.b);
}

// method returns true if two line cross
// each other between xL and xR range
bool isIntersectionPointInsideSection(line lines[],
                                         int xL, int xR, int N)
{
    pair<int, int> yBoundary[N];

    // first calculating y-values and putting
    // in boundary pair
    for (int i = 0; i < N; i++)
        yBoundary[i] = make_pair(getYFromLine(lines[i], xL),
                                 getYFromLine(lines[i], xR));

    // sorting the pair on the basis of first
    // boundary intersection
    sort(yBoundary, yBoundary + N);

    // looping over sorted pairs for comparison
    for (int i = 1; i < N; i++) {

        // if current pair's second value is smaller
        // than previous pair's then return true
        if (yBoundary[i].second < yBoundary[i - 1].second)
            return true;
    }

    return false;
}
```

```
// Driver code to test above methods
int main()
{
    int N = 4;
    int m[] = { 1, -1, 0, 2 };
    int b[] = { 2, 7, -3, -7 };

    // copy values in line struct
    line lines[N];
    for (int i = 0; i < N; i++) {
        lines[i] = line(m[i], b[i]);

        int xL = 2;
        int xR = 4;

        if (isIntersectionPointInsideSection(lines, xL, xR, N)) {
            cout << "Intersection point lies between "
                << xL << " and " << xR << endl;
        } else {
            cout << "No Intersection point lies between "
                << xL << " and " << xR << endl;
        }
    }
}
```

Output:

```
Intersection point lies between 2 and 4
```

Source

<https://www.geeksforgeeks.org/find-intersection-point-lines-inside-section/>

Chapter 78

Find maximum and minimum distance between magnets

Find maximum and minimum distance between magnets - GeeksforGeeks

Given coordinates of two pivot points (x_0, y_0) & (x_1, y_1) in coordinates plane. Along with each pivot, two different magnets are tied with the help of a string of length r_1 and r_2 respectively. Find the distance between both magnets when they repelling each other and when they are attracting each other.

Examples :

Input : $x_1=0, y_1=0, x_2=5, y_2=0, r_1=2, r_2=2$

Output : Distance while repulsion = 9, Distance while attraction = 1

Input : $x_1=0, y_1=0, x_2=5, y_2=0, r_1=3, r_2=3$

Output : Distance while repulsion = 11, Distance while attraction = 0

As we all know about the properties of magnet that they repel each other when they are facing each other with the same pole and attract each other when they are facing each other with opposite pole. Also, the force of attraction, as well as repulsion, always work in a straight line.

We have two pivots points on coordinates, so distance between these points are $D = ((x_1 - x_2)^2 + (y_1 - y_2)^2)^{1/2}$.

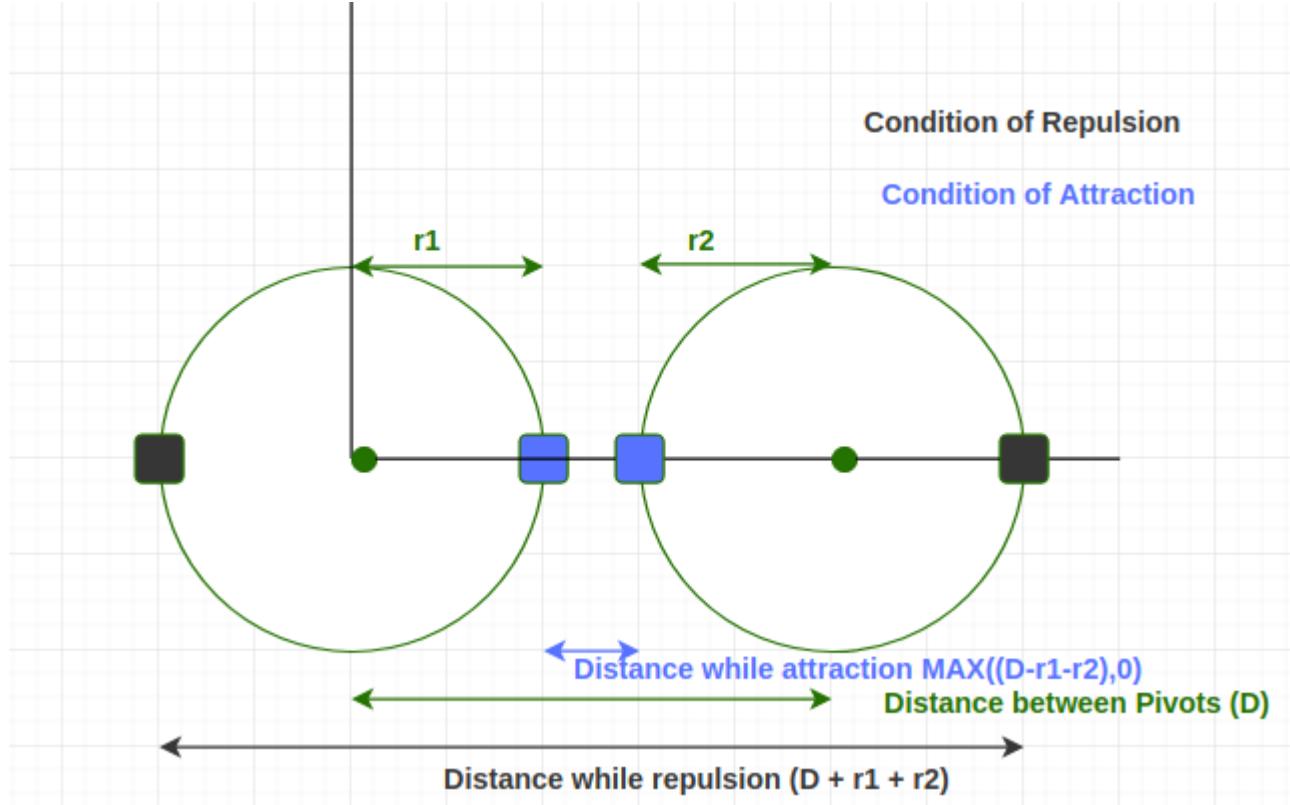
Also, we can conclude that distance between magnet is maximum while repulsion and that too should be the distance between pivots + sum of the length of both strings.

In case of attraction we have two cases to take care of:

Either the minimum distance is the distance between pivots – the sum of the length of both strings

Or minimum distance should be zero in case if the sum of the length of strings is greater than the distance between pivot points.

Illustration with help of diagram:



C++

```
// C++ program for max and min distance
#include <bits/stdc++.h>
using namespace std;

// Function for finding distance between pivots
int pivotDis(int x0, int y0, int x1, int y1)
{
    return sqrt((x1 - x0) * (x1 - x0) +
                (y1 - y0) * (y1 - y0));
}

// Function for minimum distance
int minDis(int D, int r1, int r2)
{
    return max((D - r1 - r2), 0);
}

// Function for maximum distance
```

```
int maxDis(int D, int r1, int r2)
{
    return D + r1 + r2;
}

// Drivers code
int main()
{
    int x0 = 0, y0 = 0, x1 = 8, y1 = 0, r1 = 4, r2 = 5;
    int D = pivotDis(x0, y0, x1, y1);
    cout << "Distance while repulsion = " << maxDis(D, r1, r2);
    cout << "\nDistance while attraction = " << minDis(D, r1, r2);
    return 0;
}
```

Java

```
// Java program for max
// and min distance
import java.io.*;

class GFG
{

    // Function for finding
    // distance between pivots
    static int pivotDis(int x0, int y0,
                        int x1, int y1)
    {
        return (int)Math.sqrt((x1 - x0) *
                             (x1 - x0) +
                             (y1 - y0) *
                             (y1 - y0));
    }

    // Function for
    // minimum distance
    static int minDis(int D, int r1, int r2)
    {
        return Math.max((D - r1 - r2), 0);
    }

    // Function for
    // maximum distance
    static int maxDis(int D, int r1, int r2)
    {
        return D + r1 + r2;
    }
}
```

```
// Driver Code
public static void main (String[] args)
{
    int x0 = 0, y0 = 0, x1 = 8,
        y1 = 0, r1 = 4, r2 = 5;
    int D = pivotDis(x0, y0, x1, y1);
    System.out.print( "Distance while " +
                      "repulsion = " +
                      maxDis(D, r1, r2));
    System.out.print("\nDistance while " +
                      "attraction = " +
                      minDis(D, r1, r2));
}
}

// This code is contributed by anuj_67.
```

Python3

```
# Python 3 program for max and min
# distance
import math

# Function for finding distance between
# pivots
def pivotDis(x0, y0, x1, y1):

    return math.sqrt((x1 - x0) * (x1 - x0)
                    + (y1 - y0) * (y1 - y0))

# Function for minimum distance
def minDis( D, r1, r2):

    return max((D - r1 - r2), 0)

# Function for maximum distance
def maxDis( D, r1, r2):

    return D + r1 + r2

# Drivers code
x0 = 0
y0 = 0
x1 = 8
y1 = 0
r1 = 4
r2 = 5
```

```
D = pivotDis(x0, y0, x1, y1)
print("Distance while repulsion = ",
      int(maxDis(D, r1, r2)))

print("Distance while attraction = ",
      minDis(D, r1, r2))

# This code is contributed by Smitha

C#

// C# program for max and min distance
using System;

class GFG {

    // Function for finding
    // distance between pivots
    static int pivotDis(int x0, int y0,
                        int x1, int y1)
    {
        return (int)Math.Sqrt((x1 - x0) *
                              (x1 - x0) +
                              (y1 - y0) *
                              (y1 - y0));
    }

    // Function for
    // minimum distance
    static int minDis(int D, int r1, int r2)
    {
        return Math.Max((D - r1 - r2), 0);
    }

    // Function for
    // maximum distance
    static int maxDis(int D, int r1, int r2)
    {
        return D + r1 + r2;
    }

    // Driver Code
    public static void Main ()
    {
        int x0 = 0, y0 = 0, x1 = 8,
            y1 = 0, r1 = 4, r2 = 5;
        int D = pivotDis(x0, y0, x1, y1);
```

```
Console.WriteLine("Distance while " +
                  "repulsion = " +
                  maxDis(D, r1, r2));
Console.WriteLine("Distance while " +
                  "attraction = " +
                  minDis(D, r1, r2));
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP program for max and
// min distance

// Function for finding
// distance between pivots
function pivotDis($x0, $y0,
                  $x1, $y1)
{
    return sqrt(($x1 - $x0) *
                ($x1 - $x0) +
                ($y1 - $y0) *
                ($y1 - $y0));
}

// Function for minimum distance
function minDis( $D, $r1, $r2)
{
    return max(( $D - $r1 - $r2), 0);
}

// Function for maximum distance
function maxDis( $D, $r1, $r2)
{
    return $D + $r1 + $r2;
}

// Driver code
$x0 = 0; $y0 = 0;
$x1 = 8; $y1 = 0;
$r1 = 4; $r2 = 5;
$D = pivotDis($x0, $y0,
              $x1, $y1);
echo "Distance while repulsion = "
     , maxDis($D, $r1, $r2);
```

```
echo "\nDistance while attraction = "
      , minDis($D, $r1, $r2);

// This code is contributed by anuj_67.
?>
```

Output:

```
Distance while repulsion = 17
Distance while attraction = 0
```

Improved By : [vt_m](#), Smitha Dinesh Semwal

Source

<https://www.geeksforgeeks.org/find-maximum-minimum-distance-magnets/>

Chapter 79

Find minimum radius such that atleast k point lie inside the circle

Find minimum radius such that atleast k point lie inside the circle - GeeksforGeeks

Given a positive integer K, a circle center at (0, 0) and coordinates of some points. The task is to find minimum radius of the circle so that at-least k points lie inside the circle. Output the square of the minimum radius.

Examples:

```
Input : (1, 1), (-1, -1), (1, -1),
        k = 3
Output : 2
We need a circle of radius at least 2
to include 3 points.
```

```
Input : (1, 1), (0, 1), (1, -1),
        k = 2
Output : 1
We need a circle of radius at least 1
to include 2 points. The circle around
(0, 0) of radius 1 would include (1, 1)
and (0, 1).
```

The idea is to find square of [Euclidean Distance](#) of each point from origin (0, 0). Now, sort these distance in increasing order. Now the k^{th} element of distance is the required minimum radius.

Below is the implementation of this approach:

C++

```
// C++ program to find minimum radius
// such that atleast k point lie inside
// the circle
#include<bits/stdc++.h>
using namespace std;

// Return minumum distance required so that
// atleast k point lie inside the circle.
int minRadius(int k, int x[], int y[], int n)
{
    int dis[n];

    // Finding distance between of each
    // point from origin
    for (int i = 0; i < n; i++)
        dis[i] = x[i] * x[i] + y[i] * y[i];

    // Sorting the distance
    sort(dis, dis + n);

    return dis[k - 1];
}

// Driven Program
int main()
{
    int k = 3;
    int x[] = { 1, -1, 1 };
    int y[] = { 1, -1, -1 };
    int n = sizeof(x)/sizeof(x[0]);

    cout << minRadius(k, x, y, n) << endl;

    return 0;
}
```

Java

```
// Java program to find minimum radius
// such that atleast k point lie inside
// the circle
import java.util.Arrays;
```

```
class GFG
{

    // Return minumum distance required so that
    // aleast k point lie inside the circle.
    static int minRadius(int k, int[] x, int[] y,
                         int n)
    {
        int[] dis=new int[n];

        // Finding distance between of each
        // point from origin
        for (int i = 0; i < n; i++)
            dis[i] = x[i] * x[i] + y[i] * y[i];

        // Sorting the distance
        Arrays.sort(dis);

        return dis[k - 1];
    }

    // Driven Program
    public static void main (String[] args) {

        int k = 3;
        int[] x = { 1, -1, 1 };
        int[] y = { 1, -1, -1 };
        int n = x.length;

        System.out.println(minRadius(k, x, y, n));
    }
}

/* This code is contributed by Mr. Somesh Awasthi */
```

Python3

```
# Python3 program to find minimum radius
# such that atleast k point lie inside
# the circle

# Return minumum distance required so
# that aleast k point lie inside the
# circle.
def minRadius(k, x, y, n):
    dis = [0] * n
```

```
# Finding distance between of each
# point from origin

for i in range(0, n):
    dis[i] = x[i] * x[i] + y[i] * y[i]

# Sorting the distance
dis.sort()

return dis[k - 1]

# Driver Program
k = 3
x = [1, -1, 1]
y = [1, -1, -1]
n = len(x)

print(minRadius(k, x, y, n))

# This code is contributed by
# Prasad Kshirsagar
```

C#

```
// C# program to find minimum radius
// such that atleast k point lie inside
// the circle
using System;

class GFG {

    // Return minumum distance required
    // so that aleast k point lie inside
    // the circle.
    static int minRadius(int k, int []x,
                         int[] y, int n)
    {
        int[] dis = new int[n];

        // Finding distance between of
        // each point from origin
        for (int i = 0; i < n; i++)
            dis[i] = x[i] * x[i] +
                    y[i] * y[i];

        // Sorting the distance
        Array.Sort(dis);
```

```
        return dis[k - 1];
    }

    // Driven Program
    public static void Main ()
    {
        int k = 3;
        int[] x = { 1, -1, 1 };
        int[] y = { 1, -1, -1 };
        int n = x.Length;

        Console.WriteLine(
            minRadius(k, x, y, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find minimum radius
// such that atleast k point lie inside
// the circle

// Return minumum distance required
// so that atleast k point lie
// inside the circle.
function minRadius($k, $x, $y, $n)
{
    $dis =array();

    // Finding distance between
    // of each point from origin
    for ($i = 0; $i < $n; $i++)
        $dis[$i] = $x[$i] * $x[$i] +
                    $y[$i] * $y[$i];

    // Sorting the distance
    sort($dis);

    return $dis[$k - 1];
}

// Driver Code
$k = 3;
$x = array(1, -1, 1);
```

```
$y = array(1, -1, -1);  
$n = count($x);  
  
echo minRadius($k, $x, $y, $n) ;  
  
// This code is contributed by anuj_67.  
?>
```

Output:

2

Improved By : [vt_m](#), [Prasad_Kshirsagar](#)

Source

<https://www.geeksforgeeks.org/find-minimum-radius-atleast-k-point-lie-inside-circle/>

Chapter 80

Find mirror image of a point in 2-D plane

Find mirror image of a point in 2-D plane - GeeksforGeeks

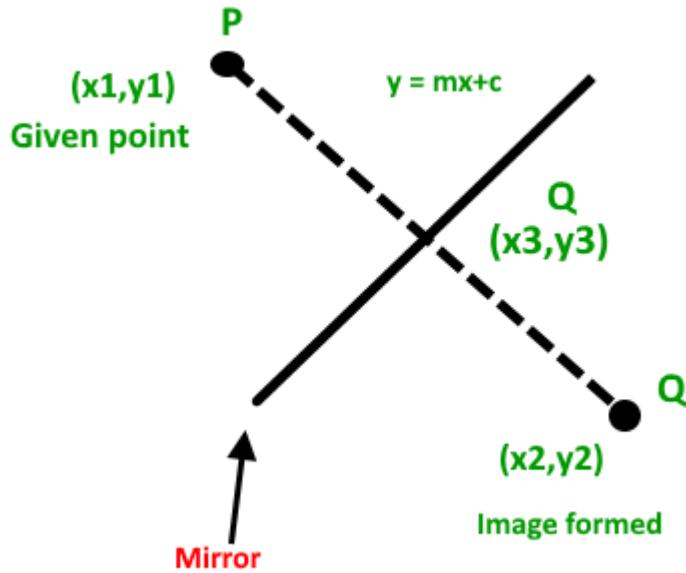
Given a point P in 2-D plane and equation of mirror, the task is to find the image of that point Q formed due to the mirror.

Equation of mirror is in form $ax + by + c = 0$

Examples:

Input : $P = (1, 0)$, $a = -1$, $b = 1$, $c = 0$
Output : $Q = (0, 1)$

Input : $P = (3, 3)$, $a = 0$, $b = 1$, $c = -2$
Output : $Q = (3, 1)$



Let coordinate of P(given point) be (x_1, y_1)

Let coordinate of Q(image point) be (x_2, y_2)

Let coordinate of R(point on mirror) be (x_3, y_3)

Since object and image are equidistant from mirror, R must be middle point of P and Q

Since equation of mirror is given to be: $ax + by + c = 0$. Equation of line passing through P and Q is perpendicular to mirror. Therefore equation of line passing through P and Q becomes $ay - bx + d = 0$, Also P passes through line passing through P and Q, so we put coordinate of P in above equation,

$$a*y_1 - b*x_1 + d = 0$$

$$d = b*x_1 - a*y_1$$

Also R is intersection of mirror and line passing through P and Q. So we find solution of
 $ax + by + c = 0$
 $ay - bx + d = 0$

Since a, b, c, d all are known we can find x and y here. Since coordinates of R is known now ie.. x_3, y_3 are known now.

Since R is middle point of PQ,

$$(x_3, y_3) = ((x_1+x_2)/2, (y_1+y_2)/2)$$

Since x_1, y_1, x_3, y_3 are known, we get below equation where (x, y) are coordinates of Q(image point)

$$\begin{aligned} x &= \frac{x_1 + x_3}{2} = \frac{x_1 + ((x_1+x_2)/2)}{2} = \frac{3x_1 + x_2}{4} \\ y &= \frac{y_1 + y_3}{2} = \frac{y_1 + ((y_1+y_2)/2)}{2} = \frac{3y_1 + y_2}{4} \end{aligned}$$

We use above formula to find mirror image of point $P(x_1, y_1)$ with respect to mirror of equation $ax + by + c$

```
// C++ code to find mirror image
#include <iostream>
using namespace std;

// C++ function which finds coordinates
// of mirror image.
// This function return a pair of double
pair<double, double> mirrorImage(
    double a, double b, double c,
    double x1, double y1)
{
    double temp = -2 * (a * x1 + b * y1 + c) /
                  (a * a + b * b);
    double x = temp * a + x1;
    double y = temp * b + y1;
    return make_pair(x, y);
}

// Driver code to test above function
int main()
{
    double a = -1.0;
    double b = 1.0;
    double c = 0.0;
    double x1 = 1.0;
    double y1 = 0.0;

    pair<double, double> image = mirrorImage(a, b, c, x1, y1);
    cout << "Image of point (" << x1 << ", " << y1 << ") ";
    cout << "by mirror (" << a << ")x + (" << b
         << ")y + (" << c << ") = 0, is :";
    cout << "(" << image.first << ", " << image.second
         << ")" << endl;

    return 0;
}
```

Output:

```
Image of point (1, 0) by mirror
(-1)x + (1)y + (0) = 0, is :(0, 1)
```

Source

<https://www.geeksforgeeks.org/find-mirror-image-point-2-d-plane/>

Chapter 81

Find number of diagonals in n sided convex polygon

Find number of diagonals in n sided convex polygon - GeeksforGeeks

Given $n > 3$, find number of diagonals in n sided convex polygon.

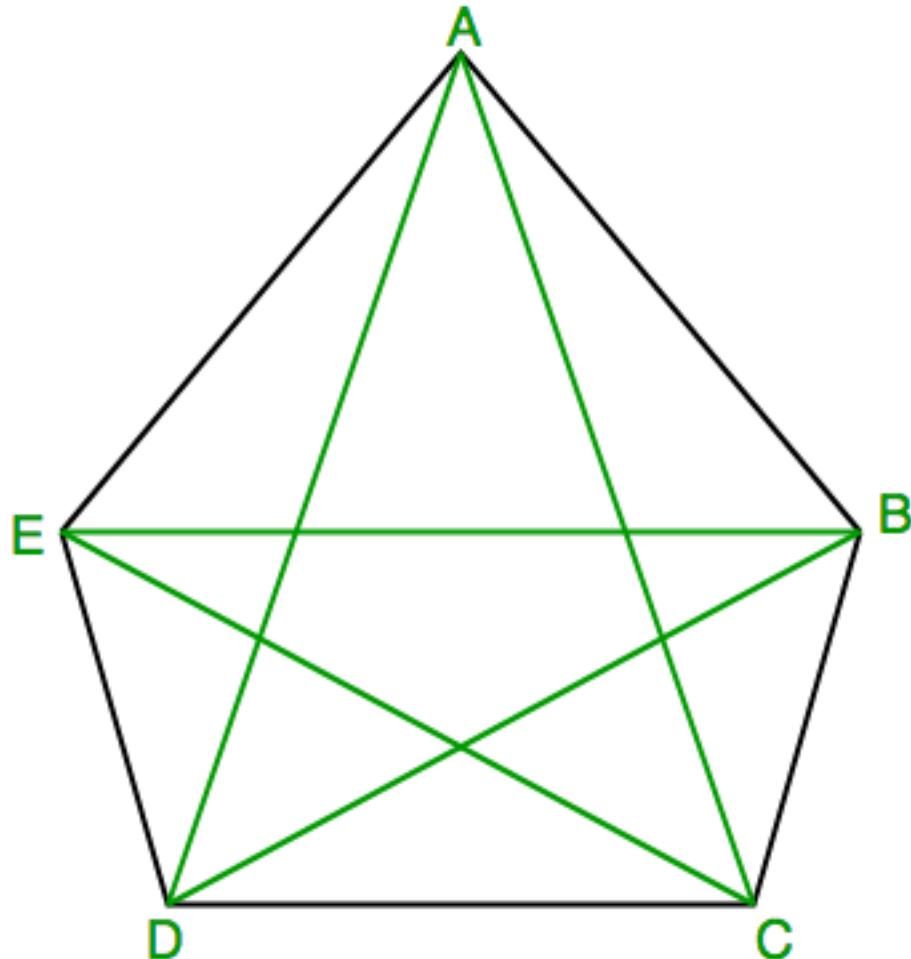
According to [Wikipedia](#), In geometry, a diagonal is a line segment joining two vertices of a polygon or polyhedron, when those vertices are not on the same edge. Informally, any sloping line is called diagonal.

Examples :

Input : 5

Output : 5

Explanation: Five possible diagonals are : AC, AD, BD, BE, CE



Since for an n-sided convex polygon, from each vertex, we can draw $n-3$ diagonals leaving two adjacent vertices and itself. Following this way for n -vertices, there will be $n*(n-3)$ diagonals but then we will be calculating each diagonal twice so total number of diagonals become $n*(n-3)/2$

Here is code for above formula.

C++

```
#include <iostream>
using namespace std;

// C++ function to find number of diagonals
// in n sided convex polygon
```

```
int numberOfDiagonals(int n)
{
    return n * (n - 3) / 2;
}

// driver code to test above function
int main()
{
    int n = 5;
    cout << n << " sided convex polygon have ";
    cout << numberOfDiagonals(n) << " diagonals";
    return 0;
}
```

Java

```
// Java function to find number of diagonals
// in n sided convex polygon

public class Diagonals {

    static int numberOfDiagonals(int n)
    {
        return n * (n - 3) / 2;
    }

    // driver code to test above function
    public static void main(String[] args)
    {
        int n = 5;
        System.out.print(n + " sided convex polygon have ");
        System.out.println(numberOfDiagonals(n) + " diagonals");
    }
}

// This code is contributed by Saket Kumar
```

C#

```
// C# function to find number of diagonals
// in n sided convex polygon
using System;

class GFG {

    static int numberOfDiagonals(int n)
    {
```

```
        return n * (n - 3) / 2;
    }

    // driver code to test above function
    public static void Main()
    {
        int n = 5;
        Console.Write(n + " sided convex polygon have ");

        Console.WriteLine(numberOfDiagonals(n) +
                           " diagonals");
    }
}

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP function to find number
// of diagonals in n sided
// convex polygon
function numberOfDiagonals($n)
{
    return $n * ($n - 3) / 2;
}

// Driver Code
$n = 5;
echo $n , " sided convex polygon have ";
echo numberOfDiagonals($n) ,
                           " diagonals";

// This code is contributed by aj_36
?>
```

Output :

```
5 sided convex polygon have 5 diagonals
```

Improved By : [Sam007, jit_t](#)

Source

<https://www.geeksforgeeks.org/find-number-diagonals-n-sided-convex-polygon/>

Chapter 82

Find perimeter of shapes formed with 1s in binary matrix

Find perimeter of shapes formed with 1s in binary matrix - GeeksforGeeks

Given a matrix of **N** rows and **M** columns, consist of 0's and 1's. The task is to find the perimeter of subfigure consisting only 1's in the matrix. Perimeter of single 1 is 4 as it can be covered from all 4 side. Perimeter of double 11 is 6.

| 1 | | 1 1 |

Examples:

```
Input : mat[][] =
{
    1, 0,
    1, 1,
}
Output : 8
Cell (1,0) and (1,1) making a L shape whose perimeter is 8.

Input : mat[][] =
{
    0, 1, 0, 0, 0,
    1, 1, 1, 0, 0,
    1, 0, 0, 0, 0
}
Output : 12
```

The idea is to traverse the matrix, find all ones and find their contribution in perimeter. The maximum contribution of a 1 is four if it is surrounded by all 0s. The contribution reduces by one with 1 around it.

Algorithm for solving this problem:

1. Traverse the whole matrix and find the cell having value equal to 1.
2. Calculate the number of closed side for that cell and add, 4 – number of closed side to the total perimeter.

Below is the implementation of this approach:

C++

```
// C++ program to find perimeter of area coverede by
// 1 in 2D matrix consists of 0's and 1's.
#include<bits/stdc++.h>
using namespace std;
#define R 3
#define C 5

// Find the number of covered side for mat[i][j].
int numofneighbour(int mat[][][C], int i, int j)
{
    int count = 0;

    // UP
    if (i > 0 && mat[i - 1][j])
        count++;

    // LEFT
    if (j > 0 && mat[i][j - 1])
        count++;

    // DOWN
    if (i < R-1 && mat[i + 1][j])
        count++;

    // RIGHT
    if (j < C-1 && mat[i][j + 1])
        count++;

    return count;
}

// Returns sum of perimeter of shapes formed with 1s
int findperimeter(int mat[R][C])
{
```

```
int perimeter = 0;

// Traversing the matrix and finding ones to
// calculate their contribution.
for (int i = 0; i < R; i++)
    for (int j = 0; j < C; j++)
        if (mat[i][j])
            perimeter += (4 - numofneighbour(mat, i, j));

return perimeter;
}

// Driven Program
int main()
{
    int mat[R][C] =
    {
        0, 1, 0, 0, 0,
        1, 1, 1, 0, 0,
        1, 0, 0, 0, 0,
    };

    cout << findperimeter(mat) << endl;

    return 0;
}
```

Java

```
// Java program to find perimeter of area
// covered by 1 in 2D matrix consists
// of 0's and 1's
class GFG {

    static final int R = 3;
    static final int C = 5;

    // Find the number of covered side
    // for mat[i][j].
    static int numofneighbour(int mat[][],
                               int i, int j)
    {

        int count = 0;

        // UP
        if (i > 0 && mat[i - 1][j] == 1)
            count++;

        // DOWN
        if (i < R - 1 && mat[i + 1][j] == 1)
            count++;

        // LEFT
        if (j > 0 && mat[i][j - 1] == 1)
            count++;

        // RIGHT
        if (j < C - 1 && mat[i][j + 1] == 1)
            count++;

        return count;
    }
}
```

```
// LEFT
if (j > 0 && mat[i][j - 1] == 1)
    count++;

// DOWN
if (i < R - 1 && mat[i + 1][j] == 1)
    count++;

// RIGHT
if (j < C - 1 && mat[i][j + 1] == 1)
    count++;

return count;
}

// Returns sum of perimeter of shapes
// formed with 1s
static int findperimeter(int mat[][])
{

    int perimeter = 0;

    // Traversing the matrix and
    // finding ones to calculate
    // their contribution.
    for (int i = 0; i < R; i++)
        for (int j = 0; j < C; j++)
            if (mat[i][j] == 1)
                perimeter += (4 -
numofneighbour(mat, i, j));

    return perimeter;
}

// Driver code
public static void main(String[] args)
{
    int mat[][] = {{0, 1, 0, 0, 0},
                   {1, 1, 1, 0, 0},
                   {1, 0, 0, 0, 0}};

    System.out.println(findperimeter(mat));
}
}

// This code is contributed by Anant Agarwal.
```

Output:

12

Time Complexity : $O(RC)$.

Source

<https://www.geeksforgeeks.org/find-perimeter-shapes-formed-1s-binary-matrix/>

Chapter 83

Find points at a given distance on a line of given slope

Find points at a given distance on a line of given slope - GeeksforGeeks

Given the co-ordinates of a 2-dimensional point $p(x_0, y_0)$. Find the points at a distance L away from it, such that the line formed by joining these points has a slope of M .

Examples:

```
Input : p = (2, 1)
        L = sqrt(2)
        M = 1
Output :3, 2
         1, 0
```

Explanation:

The two points are $\sqrt{2}$ distance away
from the source and have the required slope
 $m = 1$.

```
Input : p = (1, 0)
        L = 5
        M = 0
Output : 6, 0
         -4, 0
```

We need to find two points that are L distance from given point, on a line with slope M .

The idea has been introduced in below post.

[Find Corners of Rectangle using mid points](#)

Based on the input slope, the problem can be classified into 3 categories.

1. If slope is zero, we just need to adjust the x coordinate of the source point
2. If slope is infinite, the we need to adjust the y coordinate
3. For other values of slope, we can use the following equations to find the points

Now using the above formula we can find the required points.

```
// C++ program to find the points on a line of
// slope M at distance L
#include <bits/stdc++.h>
using namespace std;

// structure to represent a co-ordinate
// point
struct Point {

    float x, y;
    Point()
    {
        x = y = 0;
    }
    Point(float a, float b)
    {
        x = a, y = b;
    }
};

// Function to print pair of points at
// distance 'l' and having a slope 'm'
// from the source
void printPoints(Point source, float l,
                 int m)
{
    // m is the slope of line, and the
    // required Point lies distance l
    // away from the source Point
    Point a, b;

    // slope is 0
    if (m == 0) {
        a.x = source.x + l;
        a.y = source.y;

        b.x = source.x - l;
```

```
        b.y = source.y;
    }

    // if slope is infinite
    else if (m == std::numeric_limits<float>
                  ::max()) {
        a.x = source.x;
        a.y = source.y + 1;

        b.x = source.x;
        b.y = source.y - 1;
    }
    else {
        float dx = (l / sqrt(1 + (m * m)));
        float dy = m * dx;
        a.x = source.x + dx;
        a.y = source.y + dy;
        b.x = source.x - dx;
        b.y = source.y - dy;
    }

    // print the first Point
    cout << a.x << ", " << a.y << endl;

    // print the second Point
    cout << b.x << ", " << b.y << endl;
}

// driver function
int main()
{
    Point p(2, 1), q(1, 0);
    printPoints(p, sqrt(2), 1);
    cout << endl;
    printPoints(q, 5, 0);
    return 0;
}
```

Output:

```
3, 2
1, 0

6, 0
-4, 0
```

Source

<https://www.geeksforgeeks.org/find-points-at-a-given-distance-on-a-line-of-given-slope/>

Chapter 84

Find the Missing Point of Parallelogram

Find the Missing Point of Parallelogram - GeeksforGeeks

Given three coordinate points A, B and C, find the missing point D such that ABCD can be a parallelogram.

Examples :

Input : A = (1, 0)
B = (1, 1)
C = (0, 1)

Output : 0, 0

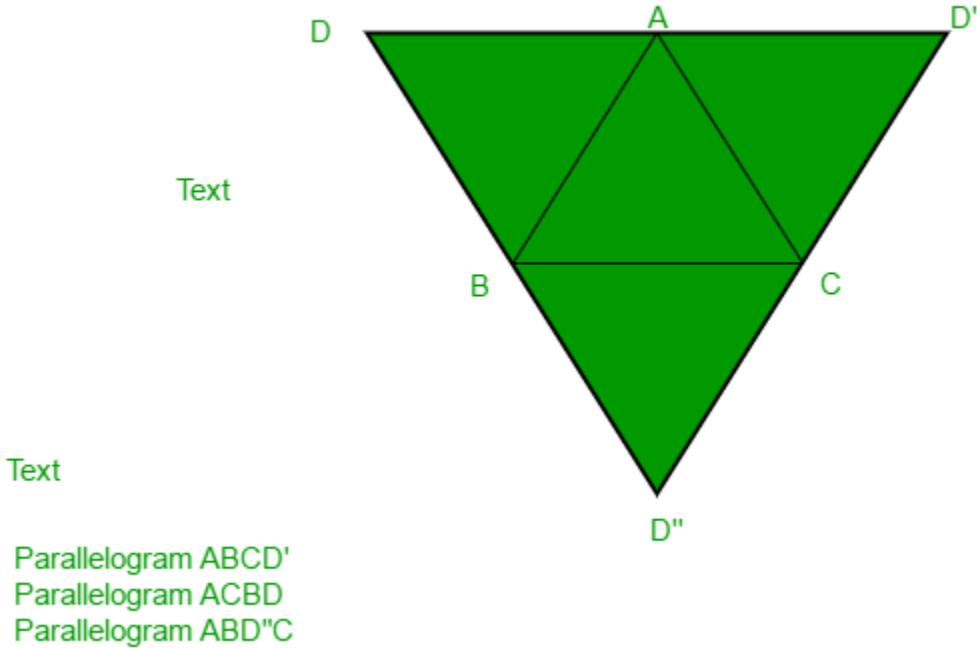
Explanation:

The three input points form a unit square with the point (0, 0)

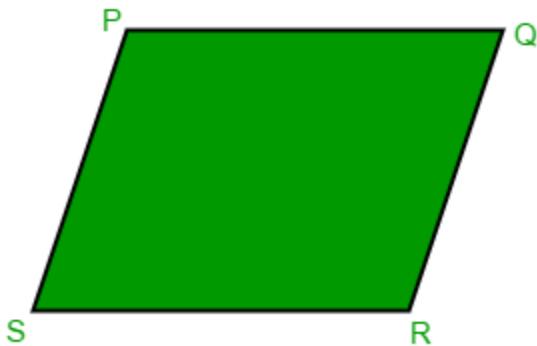
Input : A = (5, 0)
B = (1, 1)
C = (2, 5)

Output : 6, 4

As shown in below diagram, there can be multiple possible outputs, we need to print any one of them.



A quadrilateral is said to be a parallelogram if its opposite sides are parallel and equal in length.



As we're given three points of the parallelogram, we can find the slope of the missing sides as well as their lengths.

The algorithm can be explained as follows

Let R be the missing point. Now from definition, we have

- Length of PR = Length of QS = L1 (Opposite sides are equal)
- Slope of PR = Slope of QS = M1 (Opposite sides are parallel)
- Length of PQ = Length of RS = L2 (Opposite sides are equal)

- Slope of PQ = Slope of RS = M2 (Opposite sides are parallel)

Thus we can find the points at a distance L1 from P having slope M1 as mentioned in below article :

[Find points at a given distance on a line of given slope.](#)

Now one of the points will satisfy the above conditions which can easily be checked (using either condition 3 or 4).

Below is the program for the same.

C++

```
// C++ program to find missing point of a
// parallelogram
#include <bits/stdc++.h>
using namespace std;

// struct to represent a co-ordinate point
struct Point {
    float x, y;
    Point()
    {
        x = y = 0;
    }
    Point(float a, float b)
    {
        x = a, y = b;
    }
};

// given a source point, slope(m) of line
// passing through it this function calculates
// and return two points at a distance l away
// from the source
pair<Point, Point> findPoints(Point source,
                                float m, float l)
{
    Point a, b;

    // slope is 0
    if (m == 0) {
        a.x = source.x + l;
        a.y = source.y;

        b.x = source.x - l;
        b.y = source.y;
    }
}
```

```

// slope if infinity
else if (m == std::numeric_limits<float>::max()) {
    a.x = source.x;
    a.y = source.y + 1;

    b.x = source.x;
    b.y = source.y - 1;
}

// normal case
else {
    float dx = (1 / sqrt(1 + (m * m)));
    float dy = m * dx;
    a.x = source.x + dx, a.y = source.y + dy;
    b.x = source.x - dx, b.y = source.y - dy;
}

return pair<Point, Point>(a, b);
}

// given two points, this function calculates
// the slope of the line/ passing through the
// points
float findSlope(Point p, Point q)
{
    if (p.y == q.y)
        return 0;
    if (p.x == q.x)
        return std::numeric_limits<float>::max();
    return (q.y - p.y) / (q.x - p.x);
}

// calculates the distance between two points
float findDistance(Point p, Point q)
{
    return sqrt(pow((q.x - p.x), 2) + pow((q.y - p.y), 2));
}

// given three points, it prints a point such
// that a parallelogram is formed
void findMissingPoint(Point a, Point b, Point c)
{
    // calculate points originating from a
    pair<Point, Point> d = findPoints(a, findSlope(b, c),
                                         findDistance(b, c));

    // now check which of the two points satisfy
    // the conditions
}

```

```

if (findDistance(d.first, c) == findDistance(a, b))
    cout << d.first.x << ", " << d.first.y << endl;
else
    cout << d.second.x << ", " << d.second.y << endl;
}

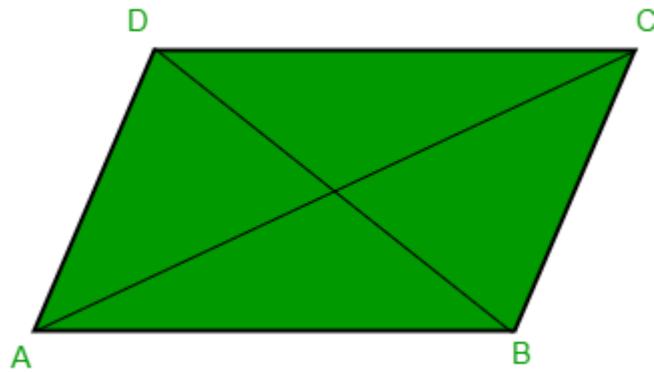
// Driver code
int main()
{
    findMissingPoint(Point(1, 0), Point(1, 1), Point(0, 1));
    findMissingPoint(Point(5, 0), Point(1, 1), Point(2, 5));
    return 0;
}

```

Output :

0, 0
6, 4

Alternative Approach:



Since the opposite sides are equal, $AD = BC$ and $AB = CD$, we can calculate the co-ordinates of the missing point (D) as:

$$\begin{aligned}
 AD &= BC \\
 (Dx - Ax, Dy - Ay) &= (Cx - Bx, Cy - By) \\
 Dx &= Ax + Cx - Bx \\
 Dy &= Ay + Cy - By
 \end{aligned}$$

References: <https://math.stackexchange.com/questions/887095/find-the-4th-vertex-of-the-parallelogram>

Below is the implementation of above approach:

C++

```
// C++ program to find missing point
// of a parallelogram
#include <bits/stdc++.h>
using namespace std;

// main method
int main()
{
    int ax = 5, ay = 0; //coordinates of A
    int bx = 1, by = 1; //coordinates of B
    int cx = 2, cy = 5; //coordinates of C
    cout << ax + cx - bx << ", "
        << ay + cy - by;
    return 0;
}
```

Java

```
// Java program to
// find missing point
// of a parallelogram
import java.io.*;

class GFG
{
public static void main (String[] args)
{

    int ax = 5, ay = 0; //coordinates of A
    int bx = 1, by = 1; //coordinates of B
    int cx = 2, cy = 5; //coordinates of C
    System.out.println(ax + (cx - bx) + ", " +
                       ay + (cy - by));
}
}

// This code is contributed by m_kit
```

Python 3

```
# Python 3 program to find missing point
# of a parallelogram
```

```
# Main method
if __name__ == "__main__":
    # coordinates of A
    ax, ay = 5, 0
    # coordinates of B
    bx, by = 1, 1
    # coordinates of C
    cx, cy = 2, 5
    print(ax + cx - bx, ", " , ay + cy - by)

# This code is contributed by Smitha
```

C#

```
// C# program to
// find missing point
// of a parallelogram
using System;

class GFG
{
    static public void Main ()
    {
        int ax = 5, ay = 0; //coordinates of A
        int bx = 1, by = 1; //coordinates of B
        int cx = 2, cy = 5; //coordinates of C
        Console.WriteLine(ax + (cx - bx) + ", " +
                          ay + (cy - by));
    }
}

// This code is contributed by ajit
```

PHP

```
<?php
// PHP program to find missing
// point of a parallelogram
// Driver Code
```

```
$ax = 5;  
$ay = 0; //coordinates of A  
$bx = 1;  
$by = 1; //coordinates of B  
$cx = 2;  
$cy = 5; //coordinates of C  
echo $ax + $cx - $bx , ", ",  
      $ay + $cy - $by;  
  
// This code is contributed by aj_36  
?>
```

Output:

6, 4

Improved By : [jit_t](#), Smitha Dinesh Semwal

Source

<https://www.geeksforgeeks.org/find-missing-point-parallelogram/>

Chapter 85

Find the Surface area of a 3D figure

Find the Surface area of a 3D figure - GeeksforGeeks

Given a $N \times M$ matrix $A[][]$ representing a 3D figure. The height of the building at (i, j) is $A[i][j]$. Find the surface area of the figure.

Examples :

Input : $N = 1, M = 1$ $A[][] = \{ \{1\} \}$
Output : 6

Explanation :
The total surface area is 6 i.e 6 side of the figure and each are of height 1.

Input : $N = 3, M = 3$ $A[][] = \{ \{1, 3, 4\}, \{2, 2, 3\}, \{1, 2, 4\} \}$
Output : 60

Approach : To find the surface area we need to consider the contribution of all the six sides of the given 3D figure. We will solve the questions in part to make it easy. The base of the Figure will always contribute $N \times M$ to the total surface area of the figure, and same $N \times M$ area will be contributed by the top of the figure. Now, to calculate the area contributed by the walls, we will take out the absolute difference between the height of two adjacent wall. The difference will be the contribution in the total surface area.

Below is the implementation of the above idea :

C++

```
// CPP program to find the Surface area of a 3D figure
#include <bits/stdc++.h>
using namespace std;

// Declaring the size of the matrix
const int M = 3;
const int N = 3;

// Absolute Difference between the height of
// two consecutive blocks
int contribution_height(int current, int previous)
{
    return abs(current - previous);
}

// Function To calculate the Total surfaceArea.
int surfaceArea(int A[N][M])
{
    int ans = 0;

    // Traversing the matrix.
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {

            /* If we are traveling the topmost row in the
            matrix, we declare the wall above it as 0
            as there is no wall above it. */
            int up = 0;

            /* If we are traveling the leftmost column in the
            matrix, we declare the wall left to it as 0
            as there is no wall left it. */
            int left = 0;

            // If its not the topmost row
            if (i > 0)
                up = A[i - 1][j];

            // If its not the leftmost column
            if (j > 0)
                left = A[i][j - 1];

            // Summing up the contribution of by
            // the current block
            ans += contribution_height(A[i][j], up)
                + contribution_height(A[i][j], left);
        }
    }
}
```

```
/* If its the rightmost block of the matrix
   it will contribute area equal to its height
   as a wall on the right of the figure */
if (i == N - 1)
    ans += A[i][j];

/* If its the lowest block of the matrix it will
   contribute area equal to its height as a wall
   on the bottom of the figure */
if (j == M - 1)
    ans += A[i][j];
}
}

// Adding the contribution by the base and top of the figure
ans += N * M * 2;
return ans;
}

// Driver program
int main()
{
    int A[N][M] = { { 1, 3, 4 },
                    { 2, 2, 3 },
                    { 1, 2, 4 } };
    cout << surfaceArea(A) << endl;
    return 0;
}
```

Java

```
// Java program to find the Surface
// area of a 3D figure

class GFG
{
    // Declaring the size of the matrix
    static final int M=3;
    static final int N=3;

    // Absolute Difference between the height of
    // two consecutive blocks
    static int contribution_height(int current, int previous)
    {
        return Math.abs(current - previous);
    }
```

```
// Function To calculate the Total surfaceArea.  
static int surfaceArea(int A[] [])  
{  
    int ans = 0;  
  
    // Traversing the matrix.  
    for (int i = 0; i < N; i++)  
    {  
        for (int j = 0; j < M; j++) {  
  
            /* If we are traveling the topmost  
            row in the matrix, we declare the  
            wall above it as 0 as there is no  
            wall above it. */  
            int up = 0;  
  
            /* If we are traveling the leftmost  
            column in the matrix, we declare the  
            wall left to it as 0 as there is no  
            wall left it. */  
            int left = 0;  
  
            // If its not the topmost row  
            if (i > 0)  
                up = A[i - 1][j];  
  
            // If its not the leftmost column  
            if (j > 0)  
                left = A[i][j - 1];  
  
            // Summing up the contribution of by  
            // the current block  
            ans += contribution_height(A[i][j], up)  
                  + contribution_height(A[i][j], left);  
  
            /* If its the rightmost block of the matrix  
            it will contribute area equal to its height  
            as a wall on the right of the figure */  
            if (i == N - 1)  
                ans += A[i][j];  
  
            /* If its the lowest block of the  
            matrix it will contribute area equal  
            to its height as a wall on  
            the bottom of the figure */  
            if (j == M - 1)  
                ans += A[i][j];  
        }  
    }  
}
```

```
}

// Adding the contribution by
// the base and top of the figure
ans += N * M * 2;
return ans;
}

// Driver code
public static void main (String[] args)
{
    int A[][] = {{ 1, 3, 4 },
                  { 2, 2, 3 },
                  { 1, 2, 4 } };
    System.out.println(surfaceArea(A));
}
}

// This code is contributed By Anant Agarwal.
```

C#

```
// C# program to find the
// Surface area of a 3D figure
using System;

class GFG
{
    // Declaring the size of the matrix
    static int M=3;
    static int N=3;

    // Absolute Difference between the
    // height of two consecutive blocks
    static int contribution_height(int current, int previous)
    {
        return Math.Abs(current - previous);
    }

    // Function To calculate the
    // Total surfaceArea.
    static int surfaceArea(int [,]A)
    {
        int ans = 0;

        // Traversing the matrix.
        for (int i = 0; i < N; i++)
        {
```

```
for (int j = 0; j < M; j++) {  
  
    // If we are traveling the topmost  
    // row in the matrix, we declare the  
    // wall above it as 0 as there is no  
    // wall above it.  
    int up = 0;  
  
    // If we are traveling the leftmost  
    // column in the matrix, we declare  
    // the wall left to it as 0 as there  
    // is no wall left it.  
    int left = 0;  
  
    // If its not the topmost row  
    if (i > 0)  
        up = A[i - 1, j];  
  
    // If its not the leftmost column  
    if (j > 0)  
        left = A[i, j - 1];  
  
    // Summing up the contribution  
    // of by the current block  
    ans += contribution_height(A[i, j], up)  
        + contribution_height(A[i, j], left);  
  
    // If its the rightmost block of the  
    // matrix it will contribute area equal  
    // to its height as a wall on the right  
    // of the figure  
    if (i == N - 1)  
        ans += A[i, j];  
  
    // If its the lowest block of the  
    // matrix it will contribute area  
    // equal to its height as a wall  
    // on the bottom of the figure  
    if (j == M - 1)  
        ans += A[i, j];  
    }  
}  
  
// Adding the contribution by the  
// base and top of the figure  
ans += N * M * 2;  
return ans;  
}
```

```
// Driver code
public static void Main ()
{
    int [,]A = {{ 1, 3, 4 },
                 { 2, 2, 3 },
                 { 1, 2, 4 } };
    Console.WriteLine(surfaceArea(A));
}
}

// This code is contributed By vt_m.
```

PHP

```
<?php
// PHP program to find the
// Surface area of a 3D figure

// Declaring the size
// of the matrix
$M = 3;
$N = 3;

// Absolute Difference
// between the height of
// two consecutive blocks
function contribution_height($current,
                             $previous)
{
    return abs($current - $previous);
}

// Function To calculate
// the Total surfaceArea.
function surfaceArea($A)
{
    global $M;
    global $N;
    $ans = 0;

    // Traversing the matrix.
    for ($i = 0; $i < $N; $i++)
    {
        for ($j = 0; $j < $M; $j++)
        {
```

```

/* If we are traveling the
topmost row in the matrix,
we declare the wall above it
as 0 as there is no wall
above it. */
$up = 0;

/* If we are traveling the
leftmost column in the
matrix, we declare the wall
left to it as 0 as there is
no wall left it. */
$left = 0;

// If its not the topmost row
if ($i > 0)
    $up = $A[$i - 1][$j];

// If its not the
// leftmost column
if ($j > 0)
    $left = $A[$i][$j - 1];

// Summing up the
// contribution of by
// the current block
$ans += contribution_height($A[$i][$j], $up) +
        contribution_height($A[$i][$j], $left);

/* If its the rightmost block
of the matrix it will contribute
area equal to its height as a
wall on the right of the figure */
if ($i == $N - 1)
    $ans += $A[$i][$j];

/* If its the lowest block
of the matrix it will
contribute area equal to
its height as a wall on
the bottom of the figure */
if ($j == $M - 1)
    $ans += $A[$i][$j];
}

}

// Adding the contribution by
// the base and top of the figure

```

```
$ans += $N * $M * 2;  
return $ans;  
}  
  
// Driver Code  
$A = array(array(1, 3, 4),  
           array(2, 2, 3),  
           array(1, 2, 4));  
echo surfaceArea($A);  
  
// This code is contributed By mits  
?>
```

Output :

60

Improved By : [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/find-surface-area-3d-figure/>

Chapter 86

Find the center of the circle using endpoints of diameter

Find the center of the circle using endpoints of diameter - GeeksforGeeks

Given two endpoint of diameter of a circle (x_1, y_1) and (x_2, y_2) find out the center of a circle.

Examples :

Input : $x_1 = -9$, $y_1 = 3$, and
 $x_2 = 5$, $y_2 = -7$
Output : $-2, -2$

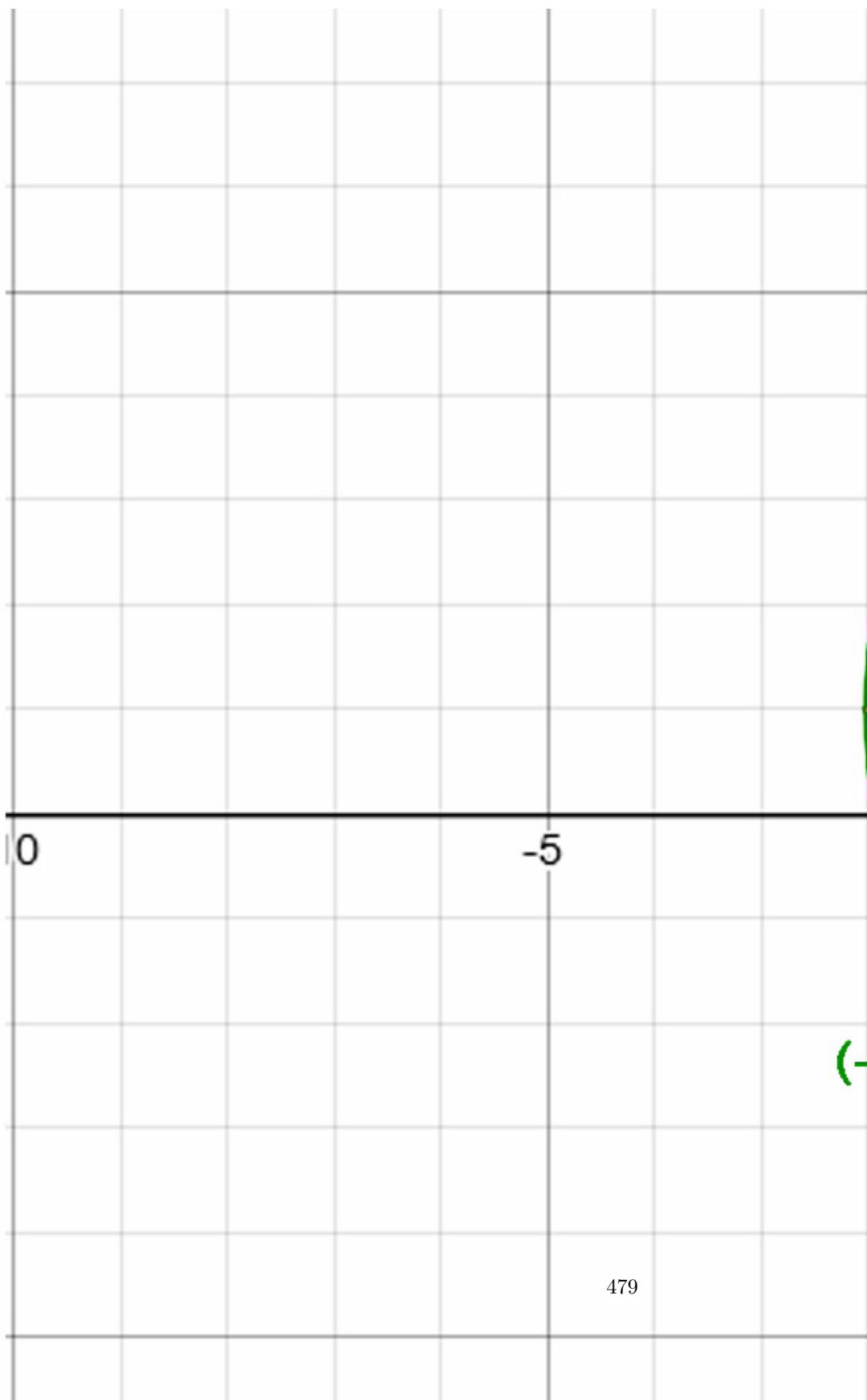
Input : $x_1 = 5$, $y_1 = 3$ and
 $x_2 = -10$ $y_2 = 4$
Output : $-2.5, 3.5$

Midpoint Formula:

The midpoint of two points, (x_1, y_2) and (x_2, y_2) is : $M = ((x_1 + x_2) / 2, (y_1 + y_2) / 2)$

The center of the circle is the mid point of its diameter so we calculate the mid point of its diameter by using midpoint formula.

Chapter 86. Find the center of the circle using endpoints of diameter



C++

```
// C++ program to find the
// center of the circle
#include <iostream>
using namespace std;

// function to find the
// center of the circle
void center(int x1, int x2,
            int y1, int y2)
{
    cout << (float)(x1 + x2) / 2 <<
        ", " << (float)(y1 + y2) / 2;
}

// Driven Program
int main()
{
    int x1 = -9, y1 = 3, x2 = 5, y2 = -7;
    center(x1, x2, y1, y2);
    return 0;
}
```

Java

```
// Java program to find the
// center of the circle
class GFG {

    // function to find the
    // center of the circle
    static void center(int x1, int x2,
                       int y1, int y2)
    {
        System.out.print((float)(x1 + x2) / 2
                        + ", " + (float)(y1 + y2) / 2);
    }

    // Driver Program to test above function
    public static void main(String arg[])
    {
        int x1 = -9, y1 = 3, x2 = 5, y2 = -7;
        center(x1, x2, y1, y2);
    }
}
```

```
        }
    }

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 program to find
# the center of the circle

# Function to find the
# center of the circle
def center(x1, x2, y1, y2) :

    print(int((x1 + x2) / 2), end= "")
    print(",", int((y1 + y2) / 2) )

# Driver Code
x1 = -9; y1 = 3; x2 = 5; y2 = -7
center(x1, x2, y1, y2)

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
// C# program to find the
// center of the circle
using System;

class GFG {

    // function to find the
    // center of the circle
    static void center(int x1, int x2,
                       int y1, int y2)
    {

        Console.WriteLine((float)(x1 + x2) / 2
                        + ", " + (float)(y1 + y2) / 2);
    }

    // Driver Program to test above function
    public static void Main() {

        int x1 = -9, y1 = 3, x2 = 5, y2 = -7;
        center(x1, x2, y1, y2);
    }
}
```

```
}
```

```
// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find the
// center of the circle

// function to find the
// center of the circle
function center($x1, $x2, $y1, $y2)
{
    echo((float)($x1 + $x2) / 2 . ", " .
          (float)($y1 + $y2) / 2);
}

// Driven Code
$x1 = -9; $y1 = 3; $x2 = 5; $y2 = -7;
center($x1, $x2, $y1, $y2);

// This code is contributed by Ajit.
?>
```

Output :

-2, -2

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/find-center-circle-using-endpoints-diameter/>

Chapter 87

Find the dimensions of Right angled triangle

Find the dimensions of Right angled triangle - GeeksforGeeks

Given H (Hypotenuse) and A (area) of a right angled triangle, find the dimensions of right angled triangle such that the hypotenuse is of length H and its area is A. If no such triangle exists, print “Not Possible”.

Examples:

```
Input : H = 10, A = 24
Output : P = 6.00, B = 8.00
```

```
Input : H = 13, A = 36
Output : Not Possible
```

Approach:

Before moving to exact solution, let's do some of mathematical calculations related to properties of Right angled triangle.

Suppose **H** = Hypotenuse, **P** = Perpendicular, **B** = Base and **A** = Area of right angled triangle.

We have some sort of equations as :

$$\begin{aligned}P^2 + B^2 &= H^2 \\P * B &= 2 * A \\(P+B)^2 &= P^2 + B^2 + 2*P*B = H^2 + 4*A \\(P+B) &= \sqrt{H^2 + 4*A} \quad \text{----1} \\(P-B)^2 &= P^2 + B^2 - 2*P*B = H^2 - 4*A \\mod(P-B) &= \sqrt{H^2 - 4*A} \quad \text{----2}\end{aligned}$$

from equation (2) we can conclude that if $H^2 < 4*A$ then no solution is possible.

Further from (1)+(2) and (1)-(2) we have :
 $P = (\sqrt{H^2 + 4*A} + \sqrt{H^2 - 4*A}) / 2$
 $B = (\sqrt{H^2 + 4*A} - \sqrt{H^2 - 4*A}) / 2$

Below is the implementation of above approach:

C++

```
// CPP program to find dimensions of
// Right angled triangle
#include <bits/stdc++.h>
using namespace std;

// function to calculate dimension
void findDimen(int H, int A)
{
    // P^2+B^2 = H^2
    // P*B = 2*A
    // (P+B)^2 = P^2+B^2+2*P*B = H^2+4*A
    // (P-B)^2 = P^2+B^2-2*P*B = H^2-4*A
    // P+B = sqrt(H^2+4*A)
    // |P-B| = sqrt(H^2-4*A)

    if (H * H < 4 * A) {
        cout << "Not Possible\n";
        return;
    }

    // sqrt value of H^2 + 4A and H^2- 4A
    double apb = sqrt(H * H + 4 * A);
    double asb = sqrt(H * H - 4 * A);

    // Set precision
    cout.precision(2);

    cout << "P = " << fixed
        << (apb - asb) / 2.0 << "\n";
    cout << "B = " << (apb + asb) / 2.0;
}

// driver function
int main()
{
    int H = 5;
```

```
    int A = 6;
    findDimen(H, A);
    return 0;
}
```

Java

```
// Java program to find dimensions of
// Right angled triangle
class GFG {

    // function to calculate dimension
    static void findDimen(int H, int A)
    {

        // P^2+B^2 = H^2
        // P*B = 2*A
        // (P+B)^2 = P^2+B^2+2*P*B = H^2+4*A
        // (P-B)^2 = P^2+B^2-2*P*B = H^2-4*A
        // P+B = sqrt(H^2+4*A)
        // |P-B| = sqrt(H^2-4*A)
        if (H * H < 4 * A) {
            System.out.println("Not Possible");
            return;
        }

        // sqrt value of H^2 + 4A and H^2- 4A
        double apb = Math.sqrt(H * H + 4 * A);
        double asb = Math.sqrt(H * H - 4 * A);

        System.out.println("P = " + Math.round(((apb - asb) / 2.0) * 100.0) / 100.0);
        System.out.print("B = " + Math.round(((apb + asb) / 2.0) * 100.0) / 100.0);
    }

    // Driver function
    public static void main(String[] args)
    {
        int H = 5;
        int A = 6;

        findDimen(H, A);
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python code to find dimensions
# of Right angled triangle

# importing the math package
# to use sqrt function
from math import sqrt

# function to find the dimensions
def findDimen( H, A):

    # P ^ 2 + B ^ 2 = H ^ 2
    # P * B = 2 * A
    # (P + B)^2 = P ^ 2 + B ^ 2 + 2 * P*B = H ^ 2 + 4 * A
    # (P-B)^2 = P ^ 2 + B ^ 2 - 2 * P*B = H ^ 2 - 4 * A
    # P + B = sqrt(H ^ 2 + 4 * A)
    # |P-B| = sqrt(H ^ 2 - 4 * A)
    if H * H < 4 * A:
        print("Not Possible")
        return

    # sqrt value of H ^ 2 + 4A and H ^ 2 - 4A
    apb = sqrt(H * H + 4 * A)
    asb = sqrt(H * H - 4 * A)

    # printing the dimensions
    print("P = ", "%.2f" %((apb - asb) / 2.0))
    print("B = ", "%.2f" %((apb + asb) / 2.0))

# driver code
H = 5 # assigning value to H
A = 6 # assigning value to A
findDimen(H, A) # calling function

# This code is contributed by "Abhishek Sharma 44"
```

C#

```
// C# program to find dimensions of
// Right angled triangle
using System;

class GFG {

    // function to calculate dimension
    static void findDimen(int H, int A)
    {
```

```
// P^2+B^2 = H^2
// P*B = 2*A
// (P+B)^2 = P^2+B^2+2*P*B = H^2+4*A
// (P-B)^2 = P^2+B^2-2*P*B = H^2-4*A
// P+B = sqrt(H^2+4*A)
// |P-B| = sqrt(H^2-4*A)
if (H * H < 4 * A) {
    Console.WriteLine("Not Possible");
    return;
}

// sqrt value of H^2 + 4A and H^2- 4A
double apb = Math.Sqrt(H * H + 4 * A);
double asb = Math.Sqrt(H * H - 4 * A);

Console.WriteLine("P = " + Math.Round(
    ((apb - asb) / 2.0) * 100.0) / 100.0);

Console.WriteLine("B = " + Math.Round(
    ((apb + asb) / 2.0) * 100.0) / 100.0);
}

// Driver function
public static void Main()
{
    int H = 5;
    int A = 6;

    findDimen(H, A);
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find dimensions
// of Right angled triangle

// function to calculate dimension
function findDimen($H, $A)
{

    // P^2+B^2 = H^2
    // P*B = 2*A
    // (P+B)^2 = P^2+B^2+2*P*B = H^2+4*A
    // (P-B)^2 = P^2+B^2-2*P*B = H^2-4*A
```

```
// P+B = sqrt(H^2+4*A)
// |P-B| = sqrt(H^2-4*A)
if ($H * $H < 4 * $A)
{
    echo "Not Possible\n";
    return;
}

// sqrt value of H^2 + 4A and
// H^2- 4A
$apb = sqrt($H * $H + 4 * $A);
$asb = sqrt($H * $H - 4 * $A);

echo "P = " , $fixed
      , ($apb - $asb) / 2.0 , "\n";
echo "B = " , ($apb + $asb) / 2.0;
}

// Driver Code
$H = 5;
$A = 6;
findDimen($H, $A);

// This code is contributed by vt_m.
?>
```

Output:

```
P = 3.00
B = 4.00
```

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-dimensions-right-angled-triangle/>

Chapter 88

Find the other end point of a line with given one end and mid

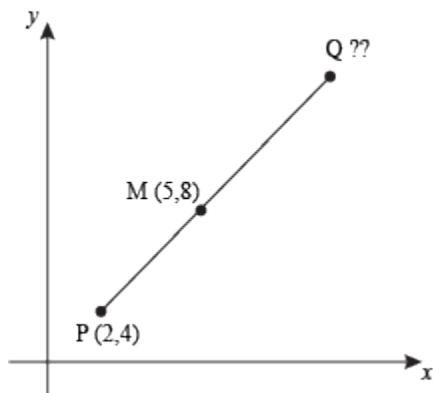
Find the other end point of a line with given one end and mid - GeeksforGeeks

Given a midpoint of line(m_1, m_2) and one coordinate of a line (x_1, y_1), find the other end point(x_2, y_2) of a line

Examples:

Input : $x_1 = -1, y_1 = 2$, and
 $m_1 = 3, m_2 = -6$
Output : $x_2 = 7, y_2 = 10$

Input : $x_1 = 6.4, y_1 = 3$ and
 $m_1 = -10.7, m_2 = 4$
Output : $x_2 = 3, y_2 = 4$



The Midpoint Formula: The midpoint of two points, (x_1, y_1) and (x_2, y_2) is the point M

found by using:

$$M = ((x_1 + x_2)/2, (y_1 + y_2)/2),$$

We have need of a (x_2, y_2) so we modifies the formula

$$\begin{aligned}m_1 &= ((x_1 + x_2)/2), \quad m_2 = ((y_1 + y_2)/2) \\2*m_1 &= (x_1 + x_2), \quad 2*m_2 = (y_1 + y_2) \\x_2 &= (2*m_1 - x_1), \quad y_2 = (2*m_2 - y_1)\end{aligned}$$

C++

```
// CPP program to find the end point of a line
#include <iostream>
using namespace std;

// CPP function to find the end point of a line
void otherEndPoint(int x1, int y1, int m1, int m2)
{
    // find end point for x coordinates
    float x2 = (float)(2 * m1 - x1);

    // find end point for y coordinates
    float y2 = (float)(2 * m2 - y1);

    cout << "x2 = " << x2 << ", "
        << "y2 = " << y2;
}

// Driven Program
int main()
{
    int x1 = -4, y1 = -1, m1 = 3, m2 = 5;
    otherEndPoint(x1, y1, m1, m2);
    return 0;
}
```

Java

```
// Java program to find the end point of a line
class GFG {

    // CPP function to find the end point
    // of a line
    static void otherEndPoint(int x1, int y1,
                              int m1, int m2)
    {
        // find end point for x coordinates
        float x2 = (float)(2 * m1 - x1);
```

```
// find end point for y coordinates
float y2 = (float)(2 * m2 - y1);

System.out.println("x2 = " + x2 + ", "
+ "y2 = " + y2);
}

// Driven Program
public static void main(String args[])
{
    int x1 = -4, y1 = -1, m1 = 3, m2 = 5;
    otherEndPoint(x1, y1, m1, m2);
}
}

// This code is contributed by JaideepPyne.
```

Python3

```
# Python3 program to find the end
# point of a line

# function to find the end point
# of a line
def otherEndPoint(x1, y1, m1, m2):

    # find end point for x coordinates
    x2 = (2 * m1 - x1)

    # find end point for y coordinates
    y2 = (2 * m2 - y1)

    print ("x2 = {}, y2 = {}"
           . format(x2, y2))

# Driven Program
x1 = -4
y1 = -1
m1 = 3
m2 = 5
otherEndPoint(x1, y1, m1, m2)

# This code is contributed by
# Manish Shaw (manishshaw1)
```

C#

```
// C# program to find the
// end point of a line
using System;

class GFG {

    // function to find the
    // end point of a line
    static void otherEndPoint(int x1, int y1,
                               int m1, int m2)
    {

        // find end point for x coordinates
        float x2 = (float)(2 * m1 - x1);

        // find end point for y coordinates
        float y2 = (float)(2 * m2 - y1);

        Console.WriteLine("x2 = " + x2 + ", "
                          + "y2 = " + y2);
    }

    // Driver Program
    public static void Main(String []args)
    {
        int x1 = -4, y1 = -1, m1 = 3, m2 = 5;
        otherEndPoint(x1, y1, m1, m2);
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// php program to find the end point of a line

// PHP function to find the end point of a line
function otherEndPoint($x1, $y1, $m1, $m2)
{

    // find end point for x coordinates
    $x2 = (2 * $m1 - $x1);

    // find end point for y coordinates
    $y2 = (2 * $m2 - $y1);

    echo "x2 = " . $x2 . ", y2 = " . $y2 ;
}
```

```
}
```

```
// Driven Program
$x1 = -4; $y1 = -1; $m1 = 3; $m2 = 5;
otherEndPoint($x1, $y1, $m1, $m2);
```

```
// This code is contributed by nitin mittal.
?>
```

Output:

x2 = 10, y2 = 11

Improved By : [jaideeppyne1997](#), [nitin mittal](#), [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/find-end-point-line-given-one-end-mid/>

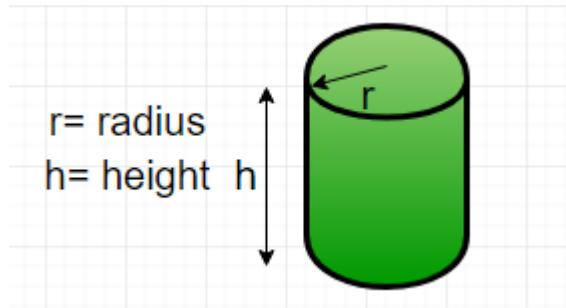
Chapter 89

Find the perimeter of a cylinder

Find the perimeter of a cylinder - GeeksforGeeks

Given diameter and height, find the perimeter of a cylinder.

Perimeter is the length of the outline of a two – dimensional shape. A cylinder is a three – dimensional shape. So, technically we cannot find the perimeter of a cylinder but we can find the perimeter of the cross-section of the cylinder. This can be done by creating the projection on its base, thus, creating the projection on its side, then the shape would be reduced to a rectangle.



Formula :

$$\text{Perimeter of cylinder (P)} = \left(\frac{\pi d}{2} + 2h \right) \cdot 2 = (\pi d + 4h)$$

here d is the diameter of the cylinder
h is the height of the cylinder

Examples :

Input : diameter = 5, height = 10
Output : Perimeter = 30

Input : diameter = 50, height = 150
Output : Perimeter = 400

C++

```
// CPP program to find
// perimeter of cylinder
#include <iostream>
using namespace std;

// Function to calculate perimeter
int perimeter(int diameter, int height)
{
    return 2 * (diameter + height);
}

// Driver function
int main()
{
    int diameter = 5;
    int height = 10;

    cout << "Perimeter = ";
    cout<< perimeter(diameter, height);
    cout<<" units\n";

    return 0;
}
```

Java

```
// Java program to find
// perimeter of cylinder
import java.io.*;

class GFG {

    // Function to calculate perimeter
    static int perimeter(int diameter, int height)
    {
        return 2 * (diameter + height);
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int diameter = 5;
        int height = 10;
        System.out.println("Perimeter = " +
                           perimeter(diameter, height))
    }
}
```

```
        + " units\n");
    }
}

// This code is contributed by Gitanjali.
```

Python

```
# Function to calculate
# the perimeter of a cylinder
def perimeter( diameter, height ) :
    return 2 * ( diameter + height )

# Driver function
diameter = 5 ;
height = 10 ;
print ("Perimeter = ",
       perimeter(diameter, height))
```

C#

```
// C# program to find perimeter of cylinder
using System;

class GFG {

    // Function to calculate perimeter
    static int perimeter(int diameter, int height)
    {
        return 2 * (diameter + height);
    }

    /* Driver program to test above function */
    public static void Main(String[] args)
    {
        int diameter = 5;
        int height = 10;
        Console.Write("Perimeter = " +
                      perimeter(diameter, height)
                      + " units\n");
    }
}

// This code is contributed by parashar...
```

PHP

```
<?php
// PHP program to find
// perimeter of cylinder

// Function to calculate perimeter
function perimeter($diameter, $height)
{
    return 2 * ($diameter + $height);
}

// Driver Code
$diameter = 5;
$height = 10;

echo("Perimeter = ");
echo(perimeter($diameter, $height));
echo(" units");

// This code is contributed by vt_m.
?>
```

Output :

Perimeter = 30 units

Improved By : [parashar](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-perimeter-cylinder/>

Chapter 90

Finding Quadrant of a Coordinate with respect to a Circle

Finding Quadrant of a Coordinate with respect to a Circle - GeeksforGeeks

Given the radius and coordinates of the centre of a circle. Find the quadrant in which another given coordinate (X, Y) lies with respect to the centre of circle if the point lies inside the circle. Else print an error “Lies outside the circle”.

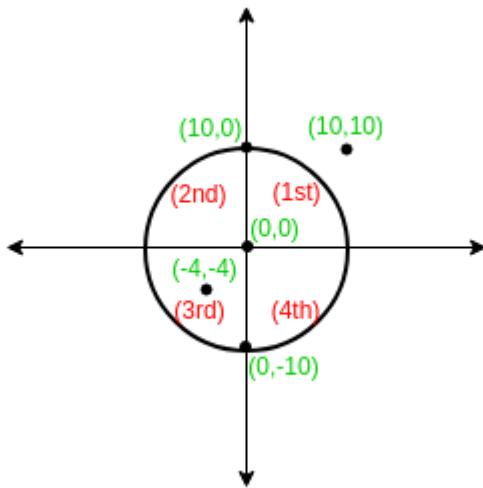
If the point lies at the centre of circle output 0 or if the point lies on any of the axes and inside the circle output the next quadrant in anti-clock direction.

Examples:

Input : Centre = (0, 0), Radius = 10

(X, Y) = (10, 10)

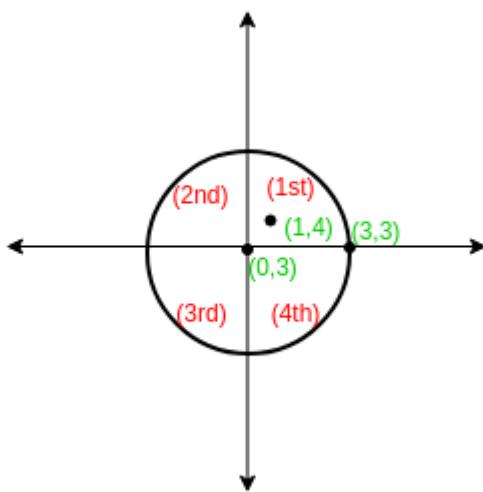
Output : Lies Outside the Circle



Input : Centre = (0, 3), Radius = 2

(X, Y) = (1, 4)

Output : 1 (I quadrant)



Approach:

Let center be (x', y')

Equation of circle is $(x - x')^2 + (y - y')^2 = r^2$ (Eq. 1)
According to this equation,

If $(x - x')^2 + (y - y')^2 > r^2$ point (x, y) lies outside of circle

If $(x - x')^2 + (y - y')^2 = r^2$ point (x, y) lies on the circle

If $(x - x_c)^2 + (y - y_c)^2 < r^2$ point (x, y) lies inside of circle

To check position of point with respect to circle:-

1. Put given coordinates in equation 1.
2. If it is greater than 0 coordinate lies outside circle.
3. If point lies inside circle find the quadrant within the circle. Check the point with respect to centre of circle.

Below is the implementation of above idea :

C++

```
// CPP Program to find the quadrant of
// a given coordinate with respect to the
// centre of a circle
#include <bits/stdc++.h>

using namespace std;

// Thus function returns the quadrant number
int getQuadrant(int X, int Y, int R, int PX, int PY)
{
    // Coincides with center
    if (PX == X && PY == Y)
        return 0;

    int val = pow((PX - X), 2) + pow((PY - Y), 2);

    // Outside circle
    if (val > pow(R, 2))
        return -1;

    // 1st quadrant
    if (PX > X && PY >= Y)
        return 1;

    // 2nd quadrant
    if (PX <= X && PY > Y)
        return 2;

    // 3rd quadrant
    if (PX < X && PY <= Y)
        return 3;
}
```

```
// 4th quadrant
if (PX >= X && PY < Y)
    return 4;
}

// Driver Code
int main()
{
    // Coordinates of centre
    int X = 0, Y = 3;

    // Radius of circle
    int R = 2;

    // Coordinates of the given point
    int PX = 1, PY = 4;

    int ans = getQuadrant(X, Y, R, PX, PY);
    if (ans == -1)
        cout << "Lies Outside the circle" << endl;
    else if (ans == 0)
        cout << "Coincides with centre" << endl;
    else
        cout << ans << " Quadrant" << endl;
    return 0;
}
```

Java

```
// Java Program to find the quadrant of
// a given coordinate with respect to the
// centre of a circle
import java.io.*;
class GFG {

    // Thus function returns
    // the quadrant number
    static int getQuadrant(int X, int Y,
                           int R, int PX,
                           int PY)
    {

        // Coincides with center
        if (PX == X && PY == Y)
            return 0;

        int val = (int)Math.pow((PX - X), 2) +
                  (int)Math.pow((PY - Y), 2);
```

```
// Outside circle
if (val > Math.pow(R, 2))
    return -1;

// 1st quadrant
if (PX > X && PY >= Y)
    return 1;

// 2nd quadrant
if (PX <= X && PY > Y)
    return 2;

// 3rd quadrant
if (PX < X && PY <= Y)
    return 3;

// 4th quadrant
if (PX >= X && PY < Y)
    return 4;
    return 0;
}

// Driver Code
public static void main (String[] args)
{
    // Coordinates of centre
    int X = 0, Y = 3;

    // Radius of circle
    int R = 2;

    // Coordinates of the given point
    int PX = 1, PY = 4;

    int ans = getQuadrant(X, Y, R, PX, PY);
    if (ans == -1)
        System.out.println( "Lies Outside the circle");
    else if (ans == 0)
        System.out.println( "Coincides with centre");
    else
        System.out.println( ans +" Quadrant");
}
}

// This code is contributed by anuj_67.
```

Python3

```
# Python3 Program to find the
# quadrant of a given coordinate
# w.r.t. the centre of a circle
import math

# Thus function returns the
# quadrant number
def getQuadrant(X, Y, R, PX, PY):

    # Coincides with center
    if (PX == X and PY == Y):
        return 0;

    val = (math.pow((PX - X), 2) +
           math.pow((PY - Y), 2));

    # Outside circle
    if (val > pow(R, 2)):
        return -1;

    # 1st quadrant
    if (PX > X and PY >= Y):
        return 1;

    # 2nd quadrant
    if (PX <= X and PY > Y):
        return 2;

    # 3rd quadrant
    if (PX < X and PY <= Y): return 3; # 4th quadrant if (PX >= X and PY < Y): return
    4; # Driver Code # Coordinates of centre X = 0; Y = 3; # Radius of circle R = 2; # Coordinates of the given po PX = 1; PY = 4; ans = getQuadrant(X, Y, R, PX, PY); if (ans == -1) : print("Lies Outside the circle"); elif (ans == 0) : print("Coincides with centre"); else:print(ans, "Quadrant"); # This code is contributed by mits [tabby title="C#"]

    // C# Program to find the quadrant of
    // a given coordinate with respect to
    // the centre of a circle
    using System;

    class GFG {

        // Thus function returns
        // the quadrant number
        static int getQuadrant(int X, int Y,
                              int R, int PX, int PY)
        {

            // Coincides with center
            if (PX == X && PY == Y)
```

```
        return 0;

    int val = (int)Math.Pow((PX - X), 2)
        + (int)Math.Pow((PY - Y), 2);

    // Outside circle
    if (val > Math.Pow(R, 2))
        return -1;

    // 1st quadrant
    if (PX > X && PY >= Y)
        return 1;

    // 2nd quadrant
    if (PX <= X && PY > Y)
        return 2;

    // 3rd quadrant
    if (PX < X && PY <= Y)
        return 3;

    // 4th quadrant
    if (PX >= X && PY < Y)
        return 4;
    return 0;
}

// Driver Code
public static void Main ()
{
    // Coordinates of centre
    int X = 0, Y = 3;

    // Radius of circle
    int R = 2;

    // Coordinates of the given point
    int PX = 1, PY = 4;

    int ans =
        getQuadrant(X, Y, R, PX, PY);
    if (ans == -1)
        Console.WriteLine( "Lies Outside"
            + " the circle");
    else if (ans == 0)
        Console.WriteLine( "C coincides "
            + "with centre");
}
```

```
        else
            Console.WriteLine( ans +
                " Quadrant");
    }
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP Program to find the quadrant of
// a given coordinate with respect to the
// centre of a circle

// Thus function returns the
// quadrant number
function getQuadrant($X, $Y, $R,
                      $PX, $PY)
{

    // Coincides with center
    if ($PX == $X and $PY == $Y)
        return 0;

    $val = pow(($PX - $X), 2) +
           pow(($PY - $Y), 2);

    // Outside circle
    if ($val > pow($R, 2))
        return -1;

    // 1st quadrant
    if ($PX > $X and $PY >= $Y)
        return 1;

    // 2nd quadrant
    if ($PX <= $X and $PY > $Y)
        return 2;

    // 3rd quadrant
    if ($PX < $X and $PY <= $Y)
        return 3;

    // 4th quadrant
    if ($PX >= $X and $PY < $Y)
        return 4;
}
```

```
// Driver Code
// Coordinates of centre
$X = 0; $Y = 3;

// Radius of circle
$R = 2;

// Coordinates of the given point
$PX = 1;
$PY = 4;

$ans = getQuadrant($X, $Y, $R,
                     $PX, $PY);
if ($ans == -1)
    echo "Lies Outside the circle" ;
else if ($ans == 0)
    echo "Coincides with centre" ;
else
    echo $ans , " Quadrant" ;

// This code is contributed by anuj_67.
?>
```

Output:

1 Quadrant

Improved By : [vt_m](#), Mithun Kumar

Source

<https://www.geeksforgeeks.org/finding-quadrant-coordinate-respect-circle/>

Chapter 91

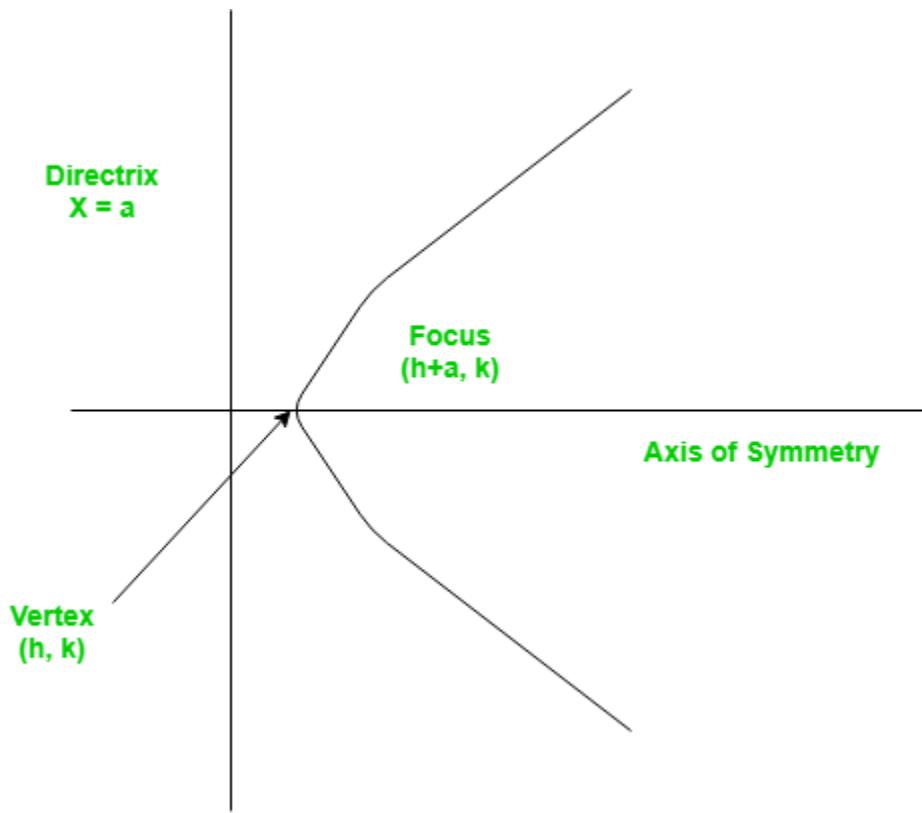
Finding the vertex, focus and directrix of a parabola

Finding the vertex, focus and directrix of a parabola - GeeksforGeeks

Problem – Find the vertex, focus and directrix of a parabola when the coefficients of its equation are given.

A set of points on a plain surface that forms a curve such that any point on that curve is equidistant from the focus is a **parabola**.

Vertex of a parabola is the coordinate from which it takes the sharpest turn whereas a is the straight line used to generate the curve.



The standard form of a parabola equation is $y^2 - 4ax - 4bx + 4ah + 4k^2 - 4k + b^2 = 0$. Given the values of a , b and c ; our task is to find the coordinates of vertex, focus and the equation of the directrix.

Example –

```

Input : 5 3 2
Output : Vertex:(-0.3, 1.55)
          Focus: (-0.3, 1.6)
          Directrix: y=-198
Consult the formula below for explanation.
  
```

This problem is a simple example of implementations of formulae. Given below are the required set of formulae which will help us tackle the problem.

For a parabola in the form

Vertex:

Focus:

Directrix:

C++

```
#include <iostream>
using namespace std;

// Function to calculate Vertex, Focus and Directrix
void parabola(float a, float b, float c)
{
    cout << "Vertex: (" << (-b / (2 * a)) << ", "
          << (((4 * a * c) - (b * b)) / (4 * a))
          << ")" << endl;
    cout << "Focus: (" << (-b / (2 * a)) << ", "
          << (((4 * a * c) - (b * b) + 1) / (4 * a))
          << ")" << endl;
    cout << "Directrix: y="
          << c - ((b * b) + 1) * 4 * a << endl;
}

// Driver Function
int main()
{
    float a = 5, b = 3, c = 2;
    parabola(a, b, c);
    return 0;
}
```

Java

```
// Java program to find the vertex,
// focus and directrix of a parabola

class GFG {

    // Function to calculate Vertex,
    // Focus and Directrix
    static void parabola(float a,
                         float b, float c)
    {

        System.out.println("Vertex: (" +
                           (-b / (2 * a)) + ", " +
                           (((4 * a * c) - (b * b)) /
                           (4 * a)) + ")");
        System.out.println("Focus: (" +
                           (-b / (2 * a)) + ", " +
                           (((4 * a * c) - (b * b) + 1) /
                           (4 * a)) + ")");
    }
}
```

```
(4 * a) + ")");  
  
System.out.println("Directrix: " + " y=" +  
                    (int)(c - ((b * b) + 1) *  
                    4 * a));  
}  
  
// Driver Function  
public static void main(String[] args)  
{  
    float a = 5, b = 3, c = 2;  
  
    // Function calling  
    parabola(a, b, c);  
}  
}  
  
// This code is contributed by  
// Smitha Dinesh Semwal
```

Python 3

```
# Function to calculate Vertex,  
# Focus and Directrix  
def parabola(a, b, c):  
  
    print("Vertex: (" , (-b / (2 * a)),  
          ", ", (((4 * a * c) - (b * b))  
          / (4 * a)), ")", sep = "")  
  
    print("Focus: (" , (-b / (2 * a)),  
          ", ", (((4 * a * c) - (b * b) + 1)  
          / (4 * a)), ")", sep = "")  
  
    print("Directrix: y=", c - ((b * b)  
        + 1) * 4 * a, sep = "")  
  
# Driver Function  
a = 5  
b = 3  
c = 2  
parabola(a, b, c)
```

This code is contributed by Smitha.

C#

```
// C# program to find the vertex,
```

```
// focus and directrix of a parabola
using System;

class GFG {

    // Function to calculate Vertex,
    // Focus and Directrix
    static void parabola(float a,
                         float b, float c)
    {
        Console.WriteLine("Vertex: (" +
                          (-b / (2 * a)) + ", " +
                          (((4 * a * c) - (b * b)) /
                           (4 * a)) + ")");
        Console.WriteLine("Focus: (" +
                          (-b / (2 * a)) + ", " +
                          (((4 * a * c) - (b * b) + 1) /
                           (4 * a)) + ")");
        Console.Write("Directrix: " + " y=" +
                     (int)(c - ((b * b) + 1) * 4 * a));
    }

    // Driver Function
    public static void Main()
    {
        float a = 5, b = 3, c = 2;

        // Function calling
        parabola(a, b, c);
    }
}

// This code is contributed by nitin mittal
```

PHP

```
<?php
// PHP program to Find the vertex,
// focus and directrix of a parabola

// Function to calculate Vertex,
// Focus and Directrix
function parabola($a, $b, $c)
{
    echo "Vertex: (" , (-$b / (2 * $a)) , " , " ,
```

```
((($a * $c) - ($b * $b)) / (4 * $a)),  
    ")", "\n" ;  
echo "Focus: (" , (-$b / (2 * $a)) , " , "  
    ((($a * $c) - ($b * $b) + 1) / (4 * $a))  
    , ")" , "\n" ;  
echo "Directrix: y=",  
    $c - (($b * $b) + 1) * 4 * $a ;  
}  
  
// Driver Code  
$a = 5; $b = 3; $c = 2;  
parabola($a, $b, $c);  
  
// This code is contributed by vt_m.  
?>
```

Output –

```
Vertex: (-0.3, 1.55)  
Focus: (-0.3, 1.6)  
Directrix: y=-198
```

Improved By : [Smitha Dinesh Semwal, nitin mittal, vt_m](#)

Source

<https://www.geeksforgeeks.org/finding-vertex-focus-directrix-parabola/>

Chapter 92

Given equation of a circle as string, find area

Given equation of a circle as string, find area - GeeksforGeeks

Given an equation of the circle $X^2 + Y^2 = R^2$ whose center at origin (0, 0) and the radius is R. The task is to find area of circle.

Examples :

```
Input :  
X*X + Y*Y = 25  
Output :  
The area of circle centered at origin is : 78.55
```

```
Input :  
X*X + Y*Y = 64  
Output :  
The area of circle centered at origin is : 201.088
```

- Given an equation $X^2 + Y^2 = R^2$ and store it into string ‘str’.
- Count length of string and store it into ‘len’.
- Start loop from 0 to len – 1 and check if $\text{str}[i] == '='$.
- Store characters after ‘=’ into string variable st.
- Convert string ‘st’ into digits and take square root and store it into ‘radius’.
- Use formula $\text{Pi} * \text{R}^2$ to find area of circle.

C++

```
// C++ program to find area of circle  
// when equation of circle give.
```

```
#include <bits/stdc++.h>
#define PI 3.142
using namespace std;

// Function to find the area
double findArea(double radius)
{
    return PI * pow(radius, 2);
}

// Function to return the value of radius
static double findradius(string str)
{

    // Intialization
    double radius = 0;

    // For storing the value of R*R
    string st = "";

    // For counting the number of
    // spaces in the string
    int c = 0;

    // calculate the length of the
    int len = str.length();

    // getting the value of R*R
    // After = sign
    for (int i = 0; i < len; i++) {
        if (str[i] == '=') {
            c = 1;
        }
        else if (c == 1 && str[i] != ' ') {
            st = st + str[i];
        }
    }

    // Converting the digits into integer
    // Taking square root of that number
    radius = (double)sqrt(stoi(st));
    return radius;
}

int main()
{
```

```
// Static input for equation
// of circle
string str = "X*X + Y*Y = 100";

// calling the Function
double radius = findradius(str);

// Display the result
cout << "The area of circle " <<
"centered at origin is : " <<
findArea(radius) << endl;

return 0;
}
```

Java

```
// Java program to find area of circle
// when equation of circle give.
import java.io.*;

public class GFG {

    static double PI = 3.142;

    // Function to find the area
    static double findArea(double radius)
    {
        return PI * Math.pow(radius, 2);
    }

    // Function to return the value of radius
    static double findradius(String str)
    {
        double radius = 0;

        // For storing the value of radius * radius
        String st = "";

        // For counting the number of spaces
        // in the string
        int c = 0;

        // calculate the length of the
        int len = str.length();

        // getting the value of radius * radius
        // After = sign
```

```
for (int i = 0; i < len; i++) {
    if (str.charAt(i) == '=') {
        c = 1;
    }
    else if (c == 1 && str.charAt(i) != ' ')
    {
        st = st + str.charAt(i);
    }
}

// Converting the digits into integer
// Taking square root of that number
if (c == 1)
    radius = (double)Math.sqrt(
        Integer.parseInt(st));
return radius;
}

public static void main(String[] args)
{
    // Static input for equation of circle
    String str = "X*X + Y*Y = 100";

    // calling the Function
    double radius = findradius(str);

    // Display the result
    System.out.println("The area of circle"
        + " centered at origin is : "
        + findArea(radius));
}
}
```

Python3

```
# python program to find area of circle
# when equation of circle give.

import math

# Function to find the area
def findArea(radius):

    return math.pi * pow(radius, 2)

# Function to return the value of radius
def findradius(str):
```

```
#Initialization
radius = 0

# For storing the value of R*R
st = ""

# For counting the number of
# spaces in the string
c = 0

# calculate the length of the
Len = len(str)

# getting the value of R*R
# After = sign
for i in range(0, Len):
    if (str[i] == '='):
        c = 1
    elif (c == 1 and str[i] != ' '):
        st = st + str[i]

# Converting the digits into integer
# Taking square root of that number
radius = float(math.sqrt(float(st)))
return radius

# Static input for equation
# of circle
str = "X*X + Y*Y = 100"

# calling the Function
radius = findradius(str)

# Display the result
print( "The area of circle " ,
"centered at origin is : " ,
(fndArea(radius)))

# This code is contributed by Sam007.
```

C#

```
// C# program to find area of circle
// when equation of circle give.
using System;
```

```
class GFG
{
    // Function to find the area
    static double findArea(double radius)
    {
        return Math.PI * Math.Pow(radius, 2);
    }

    // Function to return the value of radius
    static double findradius(string str)
    {
        double radius = 0;

        // For storing the value
        // of radius * radius
        String st = "";

        // For counting the number
        // of spaces in the string
        int c = 0;

        // calculate the length of the
        int len = str.Length;

        // getting the value of radius * radius
        // After = sign
        for (int i = 0; i < len; i++)
        {
            if (str[i] == '=')
            {
                c = 1;
            }
            else if (c == 1 && str[i] != ' ')
            {
                st = st + str[i];
            }
        }

        // Converting the digits into integer
        // Taking square root of that number
        if (c == 1)
            radius = (double)Math.Sqrt(
                int.Parse(st));
        return radius;
    }

    // Driver code
    public static void Main()
    {
```

```
{  
    // Static input for equation of circle  
    string str = "X*X + Y*Y = 100";  
  
    // calling the Function  
    double radius = findradius(str);  
  
    // Display the result  
    Console.WriteLine("The area of circle" +  
                      " centered at origin is : " +  
                      System.Math.Round(findArea(radius),1));  
}  
}  
  
// This code is contributed by Sam007
```

Output:

The area of circle centered at origin is : 314.2

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/given-equation-circle-find-area-circle/>

Chapter 93

Given n line segments, find if any two segments intersect

Given n line segments, find if any two segments intersect - GeeksforGeeks

We have discussed the problem to detect if two given line segments intersect or not. In this post, we extend the problem. Here we are given n line segments and we need to find out if any two line segments intersect or not.

Naive Algorithm A naive solution to solve this problem is to check every pair of lines and check if the pair intersects or not. We can check two line segments in $O(1)$ time. Therefore, this approach takes $O(n^2)$.

Sweep Line Algorithm: We can solve this problem in $O(n \log n)$ time using Sweep Line Algorithm. The algorithm first sorts the end points along the x axis from left to right, then it passes a vertical line through all points from left to right and checks for intersections. Following are detailed steps.

1) Let there be n given lines. There must be 2n end points to represent the n lines. Sort all points according to x coordinates. While sorting maintain a flag to indicate whether this point is left point of its line or right point.

2) Start from the leftmost point. Do following for every point

....a) If the current point is a left point of its line segment, check for intersection of its line segment with the segments just above and below it. And add its line to *active* line segments (line segments for which left end point is seen, but right end point is not seen yet). Note that we consider only those neighbors which are still active.

....b) If the current point is a right point, remove its line segment from active list and check whether its two active neighbors (points just above and below) intersect with each other.

The step 2 is like passing a vertical line from all points starting from the leftmost point to the rightmost point. That is why this algorithm is called Sweep Line Algorithm. The Sweep Line technique is useful in many other geometric algorithms like calculating the 2D Voronoi diagram

What data structures should be used for efficient implementation?

In step 2, we need to store all active line segments. We need to do following operations efficiently:

- a) Insert a new line segment
- b) Delete a line segment
- c) Find predecessor and successor according to y coordinate values

The obvious choice for above operations is Self-Balancing Binary Search Tree like AVL Tree, Red Black Tree. With a Self-Balancing BST, we can do all of the above operations in $O(\log n)$ time.

Also, in step 1, instead of sorting, we can use min heap data structure. Building a min heap takes $O(n)$ time and every extract min operation takes $O(\log n)$ time (See [this](#)).

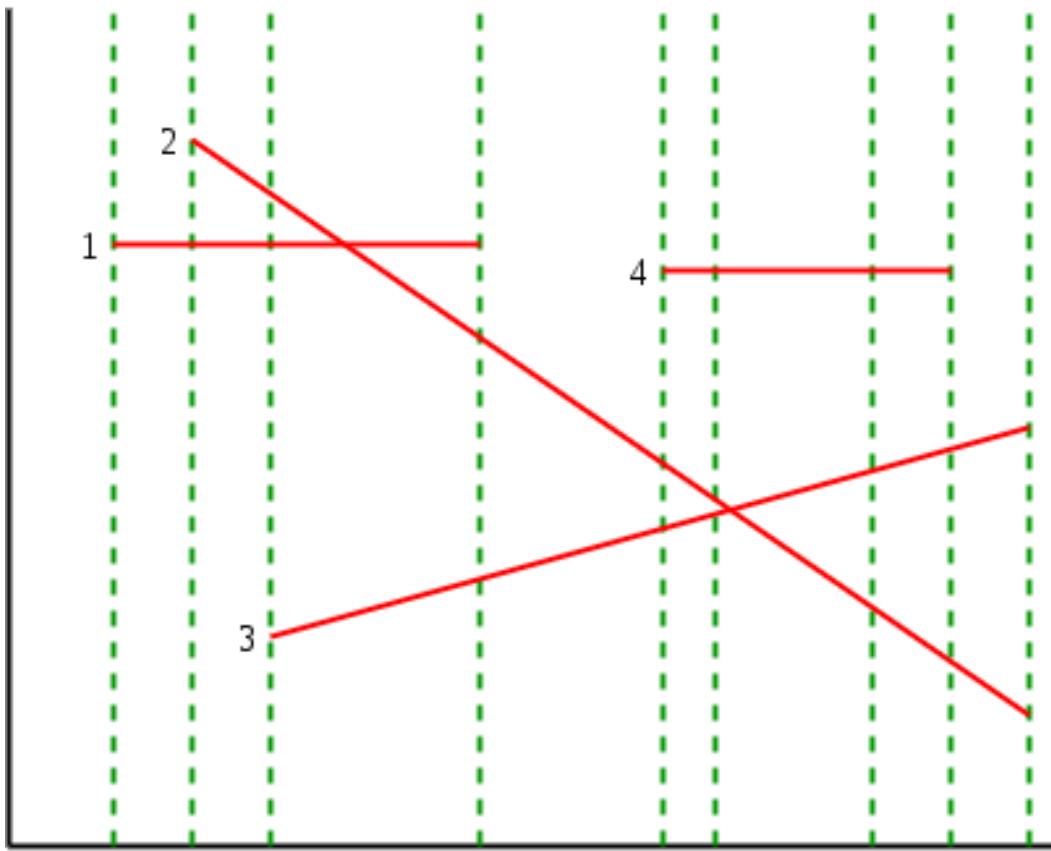
PseudoCode:

The following pseudocode doesn't use heap. It simply sort the array.

```
sweepLineIntersection(Points[0..2n-1]):  
1. Sort Points[] from left to right (according to x coordinate)  
  
2. Create an empty Self-Balancing BST T. It will contain all active line  
Segments ordered by y coordinate.  
  
// Process all 2n points  
3. for i = 0 to 2n-1  
  
    // If this point is left end of its line  
    if (Points[i].isLeft)  
        T.insert(Points[i].line()) // Insert into the tree  
  
    // Check if this points intersects with its predecessor and successor  
    if ( doIntersect(Points[i].line(), T.pred(Points[i].line())) )  
        return true  
    if ( doIntersect(Points[i].line(), T.succ(Points[i].line())) )  
        return true  
  
    else // If it's a right end of its line  
        // Check if its predecessor and successor intersect with each other  
        if ( doIntersect(T.pred(Points[i].line()), T.succ(Points[i].line())) )  
            return true  
        T.delete(Points[i].line()) // Delete from tree  
  
4. return False
```

Example:

Let us consider the following example taken from [here](#). There are 5 line segments 1, 2, 3, 4 and 5. The dotted green lines show sweep lines.



Sweep Lines

Following are steps followed by the algorithm. All points from left to right are processed one by one. We maintain a self-balancing binary search tree.

Left end point of line segment 1 is processed: 1 is inserted into the Tree. The tree contains
1. No intersection.

Left end point of line segment 2 is processed: Intersection of 1 and 2 is checked. 2 is inserted into the Tree. No intersection. The tree contains 1, 2.

Left end point of line segment 3 is processed: Intersection of 3 with 1 is checked. No intersection. 3 is inserted into the Tree. The tree contains 2, 1, 3.

Right end point of line segment 1 is processed: 1 is deleted from the Tree. Intersection of 2 and 3 is checked. Intersection of 2 and 3 is reported. The tree contains 2, 3. Note that the above pseudocode returns at this point. We can continue from here to report all intersection points.

Left end point of line segment 4 is processed: Intersections of line 4 with lines 2 and 3 are checked. No intersection. 4 is inserted into the Tree. The tree contains 2, 4, 3.

Left end point of line segment 5 is processed: Intersection of 5 with 3 is checked. No intersection. 5 is inserted into the Tree. The tree contains 2, 4, 3, 5.

Right end point of line segment 5 is processed: 5 is deleted from the Tree. The tree contains 2, 4, 3.

Right end point of line segment 4 is processed: 4 is deleted from the Tree. The tree contains 2, 4, 3. Intersection of 2 with 3 is checked. Intersection of 2 with 3 is reported. The tree contains 2, 3. Note that the intersection of 2 and 3 is reported again. We can add some logic to check for duplicates.

Right end point of line segment 2 and 3 are processed: Both are deleted from tree and tree becomes empty.

Time Complexity: The first step is sorting which takes $O(n \log n)$ time. The second step process $2n$ points and for processing every point, it takes $O(\log n)$ time. Therefore, overall time complexity is $O(n \log n)$

References:

- <http://www.cs.uiuc.edu/~jeffe/teaching/373/notes/x06-sweepline.pdf>
- <http://courses.csail.mit.edu/6.006/spring11/lectures/lec24.pdf>
- <http://www.youtube.com/watch?v=dePDHVovJIE>
- <http://www.eecs.wsu.edu/~cook/aa/lectures/l25/node10.html>

Source

<https://www.geeksforgeeks.org/given-a-set-of-line-segments-find-if-any-two-segments-intersect/>

Chapter 94

Haversine formula to find distance between two points on a sphere

Haversine formula to find distance between two points on a sphere - GeeksforGeeks

The **Haversine** formula calculates the shortest distance between two points on a sphere using their latitudes and longitudes measured along the surface. It is important for use in navigation. The haversine can be expressed in trigonometric function as:

$$\text{Haversine}(\theta) = \sin^2\left(\frac{\theta}{2}\right)$$

The haversine of the central angle (which is d/r) is calculated by the following formula:

$$\text{Haversine}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \sin^2\left(\frac{\pi}{r} d\right)$$

where r is the radius of earth(6371 km), d is the distance between two points, φ_1, φ_2 is latitude of the two points and λ_1, λ_2 is longitude of the two points respectively.

Solving d by applying the inverse haversine or by using the inverse sine function, we get:

$$d = r \text{haversine}^{-1}(v) = 2r \sin^{-1}(\sqrt{v})$$

or

$$d = r \sqrt{2 \left[1 - \cos\left(\frac{\pi}{r} d\right)\right]} = r \sqrt{2 \left[1 - \cos\left(\frac{\pi}{r} \sqrt{2r \sin^{-1}(\sqrt{v})}\right)\right]}$$

The distance between Big Ben in London (51.5007° N, 0.1246° W) and The Statue of Liberty in New York (40.6892° N, 74.0445° W) is 5574.8 km. This is not the exact measurement because the formula assumes that the Earth is a perfect sphere when in fact it is an oblate spheroid.

Below is the implementation of the above formulae:

C++

```
// C++ program for the haversine formula
// C++ program for the
// haversine formula
#include <iostream>
#include <cmath>
using namespace std;

static double haversine(double lat1, double lon1,
                       double lat2, double lon2)
{
    // distance between latitudes
    // and longitudes
    double dLat = (lat2 - lat1) *
                  M_PI / 180.0;
    double dLon = (lon2 - lon1) *
                  M_PI / 180.0;

    // convert to radians
    lat1 = (lat1) * M_PI / 180.0;
    lat2 = (lat2) * M_PI / 180.0;

    // apply formulae
    double a = pow(sin(dLat / 2), 2) +
               pow(sin(dLon / 2), 2) *
               cos(lat1) * cos(lat2);
    double rad = 6371;
    double c = 2 * asin(sqrt(a));
    return rad * c;
}

// Driver code
int main()
{
    double lat1 = 51.5007;
    double lon1 = 0.1246;
    double lat2 = 40.6892;
    double lon2 = 74.0445;

    cout << haversine(lat1, lon1,
```

```
        lat2, lon2) << " K.M.";  
    return 0;  
}  
  
// This code is contributed  
// by Mahadev.
```

Java

```
// Java program for the haversine formula  
public class Haversine {  
  
    static double haversine(double lat1, double lon1,  
                           double lat2, double lon2)  
    {  
        // distance between latitudes and longitudes  
        double dLat = Math.toRadians(lat2 - lat1);  
        double dLon = Math.toRadians(lon2 - lon1);  
  
        // convert to radians  
        lat1 = Math.toRadians(lat1);  
        lat2 = Math.toRadians(lat2);  
  
        // apply formulae  
        double a = Math.pow(Math.sin(dLat / 2), 2) +  
                  Math.pow(Math.sin(dLon / 2), 2) *  
                  Math.cos(lat1) *  
                  Math.cos(lat2);  
        double rad = 6371;  
        double c = 2 * Math.asin(Math.sqrt(a));  
        return rad * c;  
    }  
  
    // Driver Code  
    public static void main(String[] args)  
    {  
        double lat1 = 51.5007;  
        double lon1 = 0.1246;  
        double lat2 = 40.6892;  
        double lon2 = 74.0445;  
        System.out.println(haversine(lat1, lon1, lat2, lon2) + " K.M.");  
    }  
}
```

PHP

```
<?php
```

```
// PHP program for the
// haversine formula

function haversine($lat1, $lon1,
                   $lat2, $lon2)
{
    // distance between latitudes
    // and longitudes
    $dLat = ($lat2 - $lat1) *
            M_PI / 180.0;
    $dLon = ($lon2 - $lon1) *
            M_PI / 180.0;

    // convert to radians
    $lat1 = ($lat1) * M_PI / 180.0;
    $lat2 = ($lat2) * M_PI / 180.0;

    // apply formulae
    $a = pow(sin($dLat / 2), 2) +
         pow(sin($dLon / 2), 2) *
         cos($lat1) * cos($lat2);
    $rad = 6371;
    $c = 2 * asin(sqrt($a));
    return $rad * $c;
}

// Driver code
$lat1 = 51.5007;
$lon1 = 0.1246;
$lat2 = 40.6892;
$lon2 = 74.0445;

echo haversine($lat1, $lon1,
                $lat2, $lon2) .
                 " K.M. ";

// This code is contributed
// by Akanksha Rai(Addy_akku)
?>
```

Output:

5574.840456848555 K.M.

Improved By : [Mahadev99](#), [Abby_akku](#)

Source

<https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/>

Chapter 95

Hendecagonal number

Hendecagonal number - GeeksforGeeks

Given a number n, the task is to find the nth Hendecagonal number.

A Hendecagonal number is a figurate number that extends the concept of triangular and square numbers to the decagon (Eleven -sided polygon). The nth hendecagonal number counts the number of dots in a pattern of n nested decagons, all sharing a common corner, where the ith hendecagon in the pattern has sides made of i dots spaced one unit apart from each other.

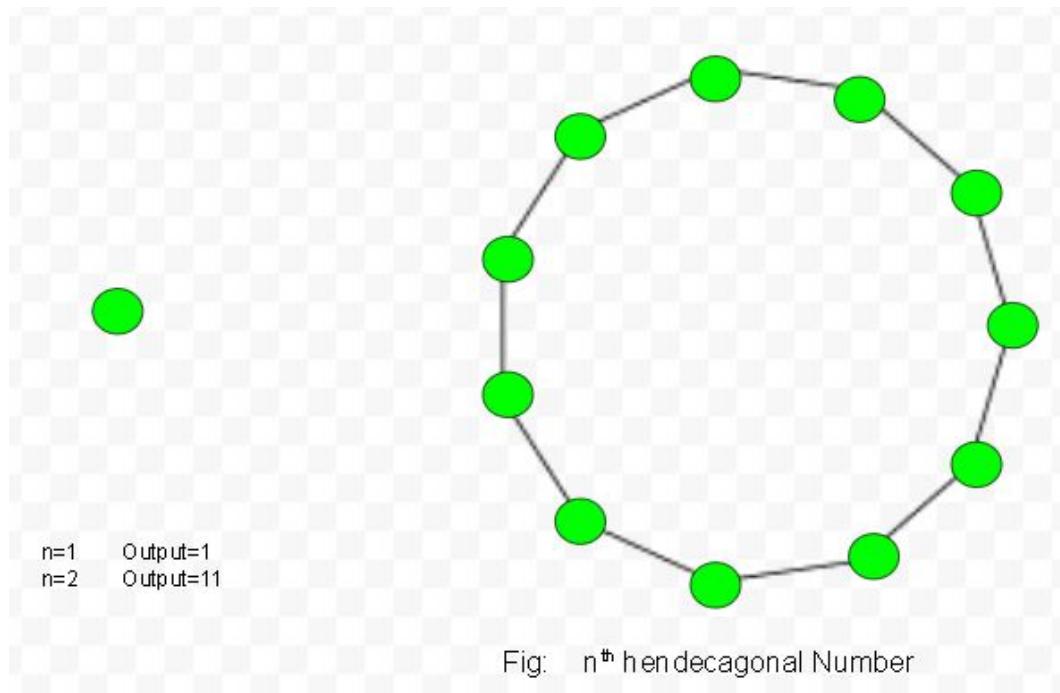
Examples:

Input : 2

Output :11

Input :6

Output :141



Formula for nth hendecagonal number :

C++

```
// C++ program to find nth
// Hendecagonal number
#include <bits/stdc++.h>
using namespace std;

// Function to find
// Hendecagonal number
int hendecagonal_num(int n)
{
    // Formula to calculate nth
    // Hendecagonal number
    return (9 * n * n - 7 * n) / 2;
}

// Driver Code
int main()
{
    int n = 3;
```

```
    cout << n << "rd Hendecagonal number: ";
    cout << hendecagonal_num(n);
    cout << endl;
    n = 10;
    cout << n << "th Hendecagonal number: ";
    cout << hendecagonal_num(n);

    return 0;
}
```

Java

```
// Java program to find nth
// Hendecagonal number
import java.io.*;

class GFG
{

    // Function to find
    // Hendecagonal number
    static int hendecagonal_num(int n)
    {
        // Formula to calculate nth
        // Hendecagonal number
        return (9 * n * n -
               7 * n) / 2;
    }

    // Driver Code
    public static void main (String[] args)
    {
        int n = 3;
        System.out.print(n + "rd Hendecagonal " +
                         "number: ");
        System.out.println(hendecagonal_num(n));

        n = 10;
        System.out.print(n + "th Hendecagonal " +
                         "number: ");
        System.out.println(hendecagonal_num(n));
    }
}

// This code is contributed by ajit
```

Python3

```
# Program to find nth
# Hendecagonal number

# Function of Hendecagonal
# number
def hendecagonal_num(n) :

    # Formula to calculate nth
    # Hendecagonal number &
    # return it into main function.

    return (9 * n * n -
           7 * n) // 2

# Driver Code
if __name__ == '__main__' :

    n = 3
    print(n,"rd Hendecagonal number : " ,
          hendecagonal_num(n))

    n = 10
    print(n,"th Hendecagonal number : " ,
          hendecagonal_num(n))

# This code is contributed by ajit
```

C#

```
// C# program to find nth
// Hendecagonal number
using System;

class GFG
{
// Function to find
// Hendecagonal number
static int hendecagonal_num(int n)
{
    // Formula to calculate nth
    // Hendecagonal number
    return (9 * n * n - 7 * n) / 2;
}

// Driver Code
static public void Main ()
{
    int n = 3;
```

```
Console.WriteLine(n +
                  "rd Hendecagonal number: ");
Console.WriteLine( hendecagonal_num(n));

n = 10;
Console.WriteLine(n +
                  "th Hendecagonal number: ");
Console.WriteLine( hendecagonal_num(n));
}

}

// This code is contributed by aj_36
```

PHP

```
<?php
// PHP program to find nth
// Hendecagonal number

// Function to find
// Hendecagonal number

function hendecagonal_num($n)
{

    // Formula to calculate nth
    // Hendecagonal number
    return (9 * $n * $n - 7 * $n) / 2;
}

// Driver Code
$n = 3;
echo $n , "th Hendecagonal number: ";
echo hendecagonal_num($n);
echo "\n";

$n = 10;
echo $n , "th Hendecagonal number: ";
echo hendecagonal_num($n);

// This code is contributed by m_kit
?>
```

Output :

```
3th Hendecagonal number: 30
10th Hendecagonal number: 415
```

Reference: https://en.wikipedia.org/wiki/Polygonal_number

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/hendecagonal-number/>

Chapter 96

Heptadecagonal number

Heptadecagonal number - GeeksforGeeks

Given a number n, the task is to find the nth heptadecagonal number .

A heptadecagonal number is class of figurate number. It has seventeen sided polygon called heptadecagon. The n-th heptadecagonal number count's the seventeen number of dots and all others dots are surrounding with a common sharing corner and make a pattern.

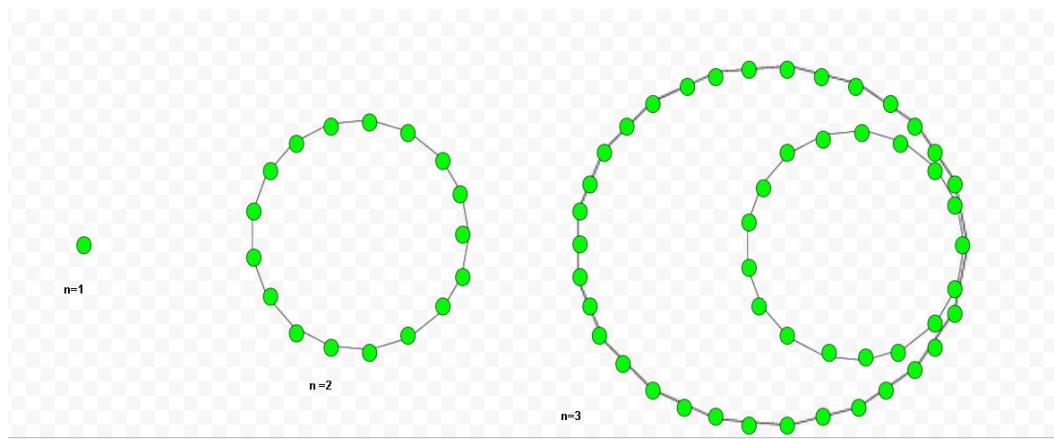
Examples:

Input : 5

Output :155

Input :9

Output :549



Formula to calculate nth heptadecagonal number:

C++

```
// C++ program to find Nth
// heptadecagonal number
#include <iostream>
using namespace std;

// Function to calculate heptadecagonal
// number
int heptadecagonalNum(long int n)
{
    return ((15 * n * n) - 13 * n) / 2;
}

// Driver Code
int main()
{
    long int n = 3;
    cout << n << "th Heptadecagonal number : ";
    cout << heptadecagonalNum(n);
    cout << endl;
    n = 8;
    cout << n << "th Heptadecagonal number : ";
    cout << heptadecagonalNum(n);

    return 0;
}
```

Java

```
// Java program to find Nth heptadecagonal number
import java.io.*;

class GFG {

    // Function to calculate heptadecagonal
    // number
    static long heptadecagonalNum(long n)
    {
        return ((15 * n * n) - 13 * n) / 2;
    }

    // Driver Code
    public static void main (String[] args)
    {
        long n = 3;
        System.out.print( n + "th Heptadecagonal"
```

```
        + " number : ");
System.out.println( heptadecagonalNum(n));

n = 8;
System.out.print( n + "th Heptadecagonal"
                  + " number : ");
System.out.print( heptadecagonalNum(n));
}
}

// This code is contributed by anuj_67.
```

Python3

```
# Python program to find Nth
# heptadecagonal number

# Function to calculate
# heptadecagonal number
def heptadecagonalNum(n):

    # Formula to calculate nth
    # heptadecagonal number
    return ((15 * n * n) - 13 * n) // 2

# Driver Code
n = 3
print("%sth Heptadecagonal number : " %n,
      heptadecagonalNum(n))
n = 8
print("%sth Heptadecagonal number: " %n,
      heptadecagonalNum(n))

# This code is contributed by ajit
```

C#

```
// C# program to find Nth
// heptadecagonal number
using System;
class GFG {

    // Function to calculate
    // heptadecagonal number
    static long heptadecagonalNum(long n)
```

```
{  
    return ((15 * n * n) -  
           13 * n) / 2;  
}  
  
// Driver Code  
public static void Main ()  
{  
    long n = 3;  
    Console.Write( n + "th Heptadecagonal"  
                  + " number : ");  
    Console.WriteLine( heptadecagonalNum(n));  
  
    n = 8;  
    Console.Write( n + "th Heptadecagonal"  
                  + " number : ");  
    Console.WriteLine( heptadecagonalNum(n));  
}  
}  
  
// This code is contributed by anuj_67.
```

PHP

```
<?php  
// PHP program to find Nth  
// heptadecagonal number  
  
// Function to calculate heptadecagonal  
// number  
function heptadecagonalNum( $n)  
{  
    return ((15 * $n * $n) -  
           13 * $n) / 2;  
}  
  
// Driver Code  
$n = 3;  
echo $n , "th Heptadecagonal number : ";  
echo heptadecagonalNum($n);  
echo "\n";  
$n = 8;  
echo $n , "th Heptadecagonal number : ";  
echo heptadecagonalNum($n);  
  
// This code is contributed by anuj_67.  
?>
```

Output

```
3th Heptadecagonal number : 48
8th Heptadecagonal number : 428
```

Reference: https://en.wikipedia.org/wiki/Polygonal_number

Improved By : [vt_m, jit_t](#)

Source

<https://www.geeksforgeeks.org/heptadecagonal-number/>

Chapter 97

Heptagonal number

Heptagonal number - GeeksforGeeks

Given a number n, the task is to find Nth heptagonal number. A Heptagonal number represents heptagon and belongs to a figurative number. Heptagonal has seven angles, seven vertices, and seven-sided polygon.

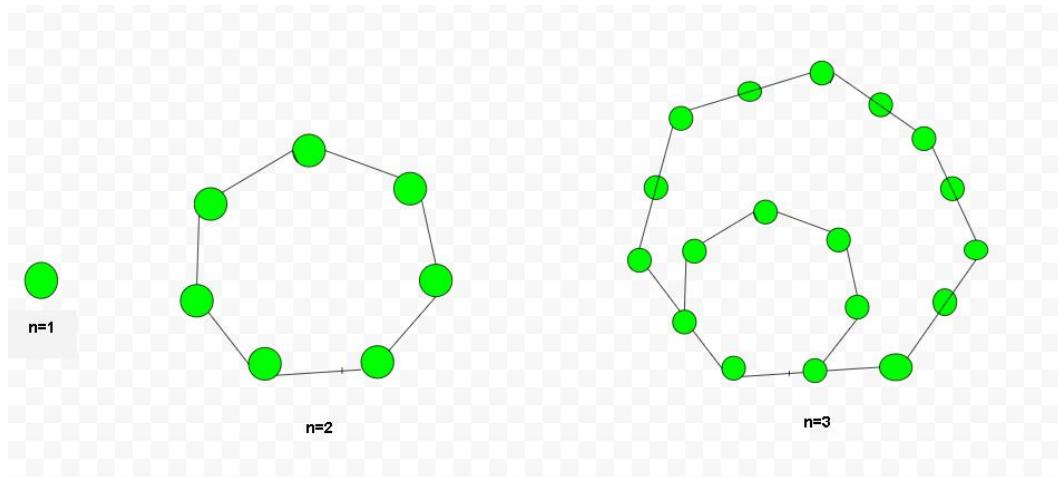
Examples :

Input : 2

Output :7

Input :15

Output :540



Few Heptagonal numbers are :

1, 7, 18, 34, 55, 81, 112, 148, 189, 235.....

A formula to calculate Nth Heptagonal number:

C++

```
// C++ program to find the
// nth Heptagonal number
#include <iostream>
using namespace std;

// Function to return Nth Heptagonal
// number
int heptagonalNumber(int n)
{
    return ((5 * n * n) - (3 * n)) / 2;
}

// Drivers Code
int main()
{

    int n = 2;
    cout << heptagonalNumber(n) << endl;
    n = 15;
    cout << heptagonalNumber(n) << endl;

    return 0;
}
```

Java

```
// Java program to find the
// nth Heptagonal number
import java.io.*;

class GFG
{
// Function to return
// Nth Heptagonal number
static int heptagonalNumber(int n)
{
    return ((5 * n * n) - (3 * n)) / 2;
}

// Driver Code
public static void main (String[] args)
{
```

```
int n = 2;
System.out.println(heptagonalNumber(n));
n = 15;
System.out.println(heptagonalNumber(n));
}
}

// This code is contributed by anuj_67.
```

Python3

```
# Program to find nth
# Heptagonal number

# Function to find
# nth Heptagonal number
def heptagonalNumber(n) :

    # Formula to calculate
    # nth Heptagonal number
    return ((5 * n * n) -
            (3 * n)) // 2

# Driver Code
if __name__ == '__main__':
    n = 2
    print(heptagonalNumber(n))
    n = 15
    print(heptagonalNumber(n))

# This code is contributed
# by ajit
```

C#

```
// C# program to find the
// nth Heptagonal number
using System;

class GFG
{
// Function to return
// Nth Heptagonal number
static int heptagonalNumber(int n)
{
    return ((5 * n * n) -
            (3 * n)) / 2;
```

```
}

// Driver Code
public static void Main ()
{
    int n = 2;
    Console.WriteLine(heptagonalNumber(n));
    n = 15;
    Console.WriteLine(heptagonalNumber(n));
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP program to find the
// nth Heptagonal number

// Function to return Nth
// Heptagonal number
function heptagonalNumber($n)
{
    return ((5 * $n * $n) -
            (3 * $n)) / 2;
}

// Driver Code
$n = 2;
echo heptagonalNumber($n), "\n";
$n = 15;
echo heptagonalNumber($n);

// This code is contributed
// by anuj_67.
?>
```

Output :

7
540

Reference: https://en.wikipedia.org/wiki/Heptagonal_number

Improved By : [vt_m, jit_t](#)

Source

<https://www.geeksforgeeks.org/heptagonal-number/>

Chapter 98

Hexadecagonal number

Hexadecagonal number - GeeksforGeeks

Given a number n, the task is to find the nth hexadecagonal number.

A Hexadecagonal number is class of figurate number and a perfect squares. It has sixteen sided polygon called hexadecagon or hexakaidecagon. The n-th hexadecagonal number count's the sixteen number of dots and all others dots are surrounding to its successive layer.

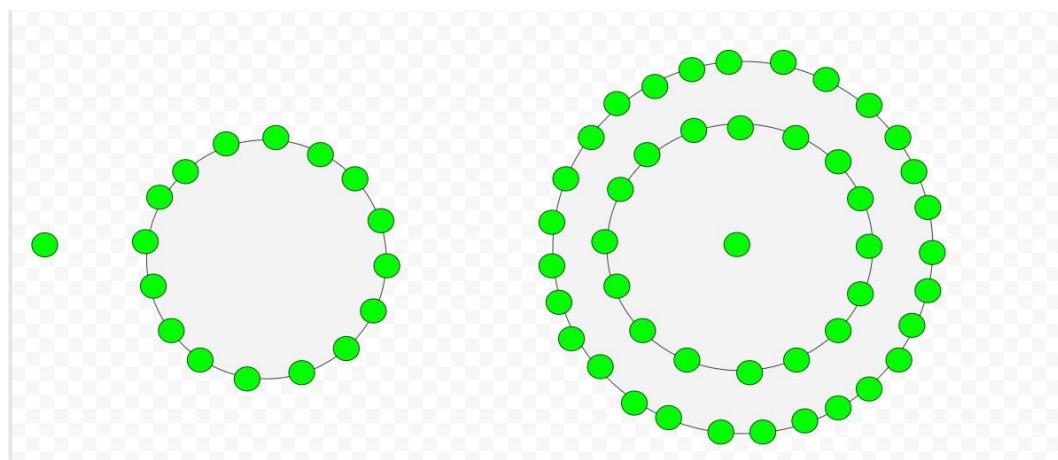
Examples :

Input : 2

Output :16

Input :7

Output :301



Formula to calculate hexadecagonal number:

C++

```
// C++ program to find Nth
// hexadecagon number
#include <bits/stdc++.h>
using namespace std;

// Function to calculate hexadecagonal number
int hexadecagonalNum(long int n)
{
    return ((14 * n * n) - 12 * n) / 2;
}

// Drivers Code
int main()
{
    long int n = 5;
    cout << n << "th Hexadecagonal number : ";
    cout << hexadecagonalNum(n);
    cout << endl;
    n = 9;
    cout << n << "th Hexadecagonal number : ";
    cout << hexadecagonalNum(n);

    return 0;
}
```

Java

```
// Java program to find Nth hexadecagon
// number
import java.io.*;

class GFG {

    // Function to calculate hexadecagonal
    // number
    static long hexadecagonalNum(long n)
    {
        return ((14 * n * n) - 12 * n) / 2;
    }

    // Drivers Code
    public static void main (String[] args)
```

```
{  
    long n = 5;  
    System.out.println( n + "th "  
        + "Hexadecagonal number : "  
        + hexadecagonalNum(n));  
  
    n = 9;  
    System.out.println( n + "th "  
        + "Hexadecagonal number : "  
        + hexadecagonalNum(n));  
}  
}  
  
// This code contribued by anuj_67.
```

Python3

```
# Python program to find Nth  
# hexadecagon number  
  
# Function to calculate  
# hexadecagonal number  
def hexadecagonalNum(n):  
  
    # Formula to calculate nth  
    # Centered heptagonal number  
    return ((14 * n * n) - 12 * n) // 2  
  
# Driver Code  
n = 5  
print("%sth Hexadecagonal number : " %n,  
      hexadecagonalNum(n))  
n = 9  
print("%sth Hexadecagonal number : " %n,  
      hexadecagonalNum(n))  
  
# This code is contributed by ajit
```

C#

```
// C# program to find Nth hexadecagon  
// number  
using System;  
class GFG {  
  
    // Function to calculate hexadecagonal  
    // number
```

```
static long hexadecagonalNum(long n)
{
    return ((14 * n * n) - 12 * n) / 2;
}

// Drivers Code
public static void Main ()
{
    long n = 5;
    Console.WriteLine( n + "th "
        + "Hexadecagonal number : "
        + hexadecagonalNum(n));

    n = 9;
    Console.WriteLine( n + "th "
        + "Hexadecagonal number : "
        + hexadecagonalNum(n));
}
}

// This code contribued by anuj_67.
```

PHP

```
<?php
// PHP program to find Nth
// hexadecagon number

// Function to calculate
// hexadecagonal number

function hexadecagonalNum($n)
{
    return ((14 * $n * $n) - 12 * $n) / 2;
}

// Driver Code
$n = 5;
echo $n , "th Hexadecagonal number : ";
echo hexadecagonalNum($n);
echo "\n";

$n = 9;
echo $n , "th Hexadecagonal number : ";
echo hexadecagonalNum($n);

// This code is contributed bu m_kit
?>
```

Output :

```
5th Hexadecagonal number : 145
9th Hexadecagonal number : 513
```

Reference: https://en.wikipedia.org/wiki/Polygonal_number

Improved By : [vt_m, jit_t](#)

Source

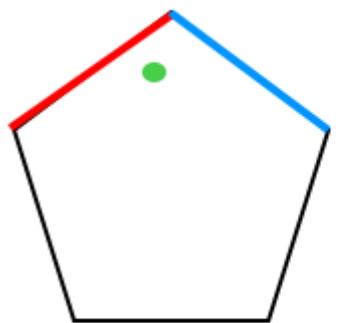
<https://www.geeksforgeeks.org/hexadecagonal-number/>

Chapter 99

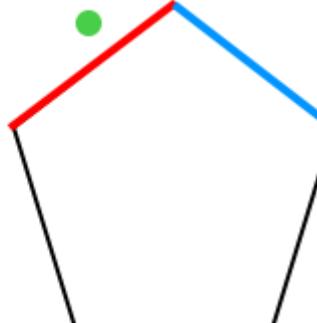
How to check if a given point lies inside or outside a polygon?

How to check if a given point lies inside or outside a polygon? - GeeksforGeeks

Given a polygon and a point ‘p’, find if ‘p’ lies inside the polygon or not. The points lying on the border are considered inside.



Inside example



outside example

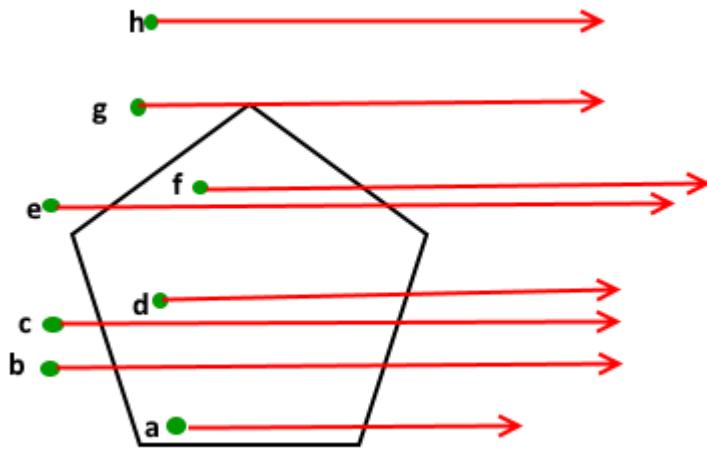
We strongly recommend to see the following post first.

[How to check if two given line segments intersect?](#)

Following is a simple idea to check whether a point is inside or outside.

- 1) Draw a horizontal line to the right of each point and extend it to infinity

- 1) Count the number of times the line intersects with polygon edges.
- 2) A point is inside the polygon if either count of intersections is odd or point lies on an edge of polygon. If none of the conditions is true, then point lies outside.



How to handle point 'g' in the above figure?

Note that we should return true if the point lies on the line or same as one of the vertices of the given polygon. To handle this, after checking if the line from 'p' to extreme intersects, we check whether 'p' is colinear with vertices of current line of polygon. If it is colinear, then we check if the point 'p' lies on current side of polygon, if it lies, we return true, else false.

Following is C++ implementation of the above idea.

```
// A C++ program to check if a given point lies inside a given polygon
// Refer https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/
// for explanation of functions onSegment(), orientation() and doIntersect()
#include <iostream>
using namespace std;

// Define Infinite (Using INT_MAX caused overflow problems)
#define INF 10000

struct Point
{
    int x;
    int y;
};


```

```
// Given three colinear points p, q, r, the function checks if
// point q lies on line segment 'pr'
bool onSegment(Point p, Point q, Point r)
{
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
        q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
        return true;
    return false;
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0; // colinear
    return (val > 0)? 1: 2; // clock or counterclock wise
}

// The function that returns true if line segment 'p1q1'
// and 'p2q2' intersect.
bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
    // Find the four orientations needed for general and
    // special cases
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    // General case
    if (o1 != o2 && o3 != o4)
        return true;

    // Special Cases
    // p1, q1 and p2 are colinear and p2 lies on segment p1q1
    if (o1 == 0 && onSegment(p1, p2, q1)) return true;

    // p1, q1 and p2 are colinear and q2 lies on segment p1q1
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;

    // p2, q2 and p1 are colinear and p1 lies on segment p2q2
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;
```

```
// p2, q2 and q1 are colinear and q1 lies on segment p2q2
if (o4 == 0 && onSegment(p2, q1, q2)) return true;

return false; // Doesn't fall in any of the above cases
}

// Returns true if the point p lies inside the polygon[] with n vertices
bool isInside(Point polygon[], int n, Point p)
{
    // There must be at least 3 vertices in polygon[]
    if (n < 3) return false;

    // Create a point for line segment from p to infinite
    Point extreme = {INF, p.y};

    // Count intersections of the above line with sides of polygon
    int count = 0, i = 0;
    do
    {
        int next = (i+1)%n;

        // Check if the line segment from 'p' to 'extreme' intersects
        // with the line segment from 'polygon[i]' to 'polygon[next]'
        if (doIntersect(polygon[i], polygon[next], p, extreme))
        {
            // If the point 'p' is colinear with line segment 'i-next',
            // then check if it lies on segment. If it lies, return true,
            // otherwise false
            if (orientation(polygon[i], p, polygon[next]) == 0)
                return onSegment(polygon[i], p, polygon[next]);

            count++;
        }
        i = next;
    } while (i != 0);

    // Return true if count is odd, false otherwise
    return count&1; // Same as (count%2 == 1)
}

// Driver program to test above functions
int main()
{
    Point polygon1[] = {{0, 0}, {10, 0}, {10, 10}, {0, 10}};
    int n = sizeof(polygon1)/sizeof(polygon1[0]);
    Point p = {20, 20};
    isInside(polygon1, n, p)? cout << "Yes \n": cout << "No \n";
}
```

```
p = {5, 5};  
isInside(polygon1, n, p)? cout << "Yes \n": cout << "No \n";  
  
Point polygon2[] = {{0, 0}, {5, 5}, {5, 0}};  
p = {3, 3};  
n = sizeof(polygon2)/sizeof(polygon2[0]);  
isInside(polygon2, n, p)? cout << "Yes \n": cout << "No \n";  
  
p = {5, 1};  
isInside(polygon2, n, p)? cout << "Yes \n": cout << "No \n";  
  
p = {8, 1};  
isInside(polygon2, n, p)? cout << "Yes \n": cout << "No \n";  
  
Point polygon3[] = {{0, 0}, {10, 0}, {10, 10}, {0, 10}};  
p = {-1,10};  
n = sizeof(polygon3)/sizeof(polygon3[0]);  
isInside(polygon3, n, p)? cout << "Yes \n": cout << "No \n";  
  
return 0;  
}
```

Output:

```
No  
Yes  
Yes  
Yes  
No  
No
```

Time Complexity: O(n) where n is the number of vertices in the given polygon.

Source:

<http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf>

Source

<https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/>

Chapter 100

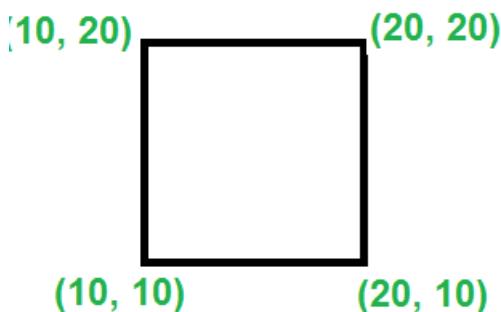
How to check if given four points form a square

How to check if given four points form a square - GeeksforGeeks

Given coordinates of four points in a plane, find if the four points form a square or not.

To check for square, we need to check for following.

- a) All four sides formed by points are same.
- b) The angle between any two sides is 90 degree. (This condition is required as [Quadrilateral](#) also has same sides.)
- c) Check both the diagonals have same distance



The idea is to pick any point and calculate its distance from rest of the points. Let the picked point be 'p'. To form a square, distance of two points must be same from 'p', let this distance be d. The distance from one point must be different from that d and must be equal to $\sqrt{2}$ times d. Let this point with different distance be 'q'.

The above condition is not good enough as the point with different distance can be on the other side. We also need to check that q is at same distance from 2 other points and this distance is same as d.

Below is C++ implementation of above idea.

```
// A C++ program to check if four given points form a square or not.
#include <iostream>
using namespace std;

// Structure of a point in 2D space
struct Point {
    int x, y;
};

// A utility function to find square of distance
// from point 'p' to point 'q'
int distSq(Point p, Point q)
{
    return (p.x - q.x) * (p.x - q.x) + (p.y - q.y) * (p.y - q.y);
}

// This function returns true if (p1, p2, p3, p4) form a
// square, otherwise false
bool isSquare(Point p1, Point p2, Point p3, Point p4)
{
    int d2 = distSq(p1, p2); // from p1 to p2
    int d3 = distSq(p1, p3); // from p1 to p3
    int d4 = distSq(p1, p4); // from p1 to p4

    // If lengths of (p1, p2) and (p1, p3) are same, then
    // following conditions must met to form a square.
    // 1) Square of length of (p1, p4) is same as twice
    // the square of (p1, p2)
    // 2) Square of length of (p2, p3) is same as twice the square of (p1, p2)

    if (d2 == d3 && 2 * d2 == d4 && 2 * d2 == distSq(p2, p3)) {
        int d = distSq(p2, p4);
        return (d == distSq(p3, p4) && d == d2);
    }

    // The below two cases are similar to above case
    if (d3 == d4 && 2 * d3 == d2 && 2 * d3 == distSq(p3, p4)) {
        int d = distSq(p2, p3);
        return (d == distSq(p2, p4) && d == d3);
    }
    if (d2 == d4 && 2 * d2 == d3 && 2 * d2 == distSq(p2, p4)) {
        int d = distSq(p2, p3);
        return (d == distSq(p3, p4) && d == d2);
    }

    return false;
}
```

```
// Driver program to test above function
int main()
{
    Point p1 = { 20, 10 }, p2 = { 10, 20 },
        p3 = { 20, 20 }, p4 = { 10, 10 };
    isSquare(p1, p2, p3, p4) ? cout << "Yes" : cout << "No";
    return 0;
}
```

Output:

Yes

Extended Problem:

[Check if four segments form a rectangle](#)

This article is contributed by **Anuj**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [Sukanta_it](#)

Source

<https://www.geeksforgeeks.org/check-given-four-points-form-square/>

Chapter 101

How to check if two given line segments intersect?

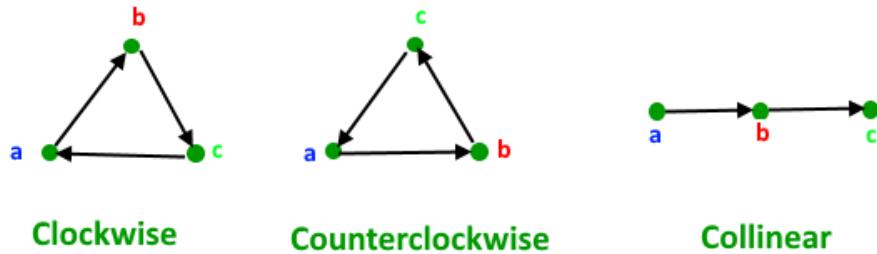
How to check if two given line segments intersect? - GeeksforGeeks

Given two line segments (p_1, q_1) and (p_2, q_2) , find if the given line segments intersect with each other.

Before we discuss solution, let us define notion of **orientation**. Orientation of an ordered triplet of points in the plane can be

- counterclockwise
- clockwise
- collinear

The following diagram shows different possible orientations of (a, b, c)



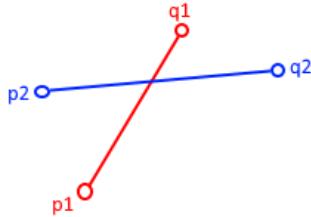
How is Orientation useful here?

Two segments (p_1, q_1) and (p_2, q_2) intersect if and only if one of the following two conditions is verified

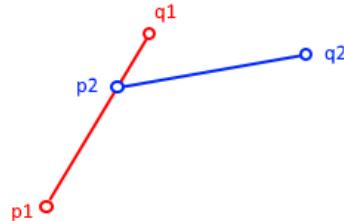
1. General Case:

- (p_1, q_1, p_2) and (p_1, q_1, q_2) have different orientations and
- (p_2, q_2, p_1) and (p_2, q_2, q_1) have different orientations.

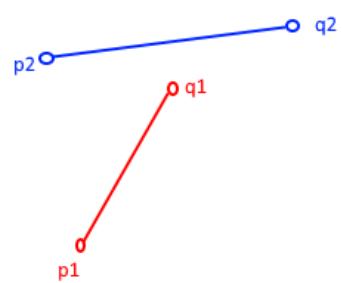
Examples:



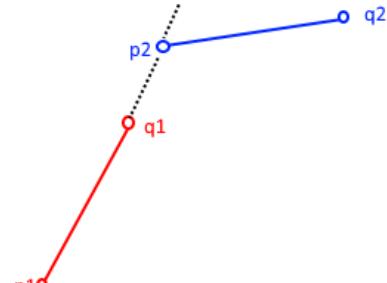
Example : Orientations of (p_1, q_1, p_2) and (p_1, q_1, q_2) are different. Orientations of (p_2, q_2, p_1) and (p_2, q_2, q_1) also different.



Example: Orientations of (p_1, q_1, p_2) and (p_1, q_1, q_2) are different. Orientations of (p_2, q_2, p_1) and (p_2, q_2, q_1) also different



Example: Orientations of (p_1, q_1, p_2) and (p_1, q_1, q_2) are different. Orientations of (p_2, q_2, p_1) and (p_2, q_2, q_1) are same

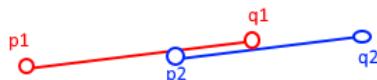


Example: Orientations of (p_1, q_1, p_2) and (p_1, q_1, q_2) are different. Orientations of (p_2, q_2, p_1) and (p_2, q_2, q_1) are same.

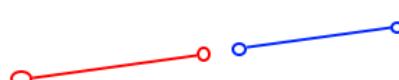
2. Special Case

- (p_1, q_1, p_2) , (p_1, q_1, q_2) , (p_2, q_2, p_1) , and (p_2, q_2, q_1) are all collinear and
- the x-projections of (p_1, q_1) and (p_2, q_2) intersect
- the y-projections of (p_1, q_1) and (p_2, q_2) intersect

Examples:



Example: All points are collinear. The x-projections of (p_1, q_1) and (p_2, q_2) intersect. The y-projection of (p_1, q_1) and (p_2, q_2) also intersect



Example: All points are collinear. The x-projections of (p_1, q_1) and (p_2, q_2) do not intersect. The y-projection of (p_1, q_1) and (p_2, q_2) do not intersect

Following is C++ implementation based on above idea.

```
// A C++ program to check if two given line segments intersect
#include <iostream>
using namespace std;

struct Point
```

```
{  
    int x;  
    int y;  
};  
  
// Given three colinear points p, q, r, the function checks if  
// point q lies on line segment 'pr'  
bool onSegment(Point p, Point q, Point r)  
{  
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&  
        q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))  
        return true;  
  
    return false;  
}  
  
// To find orientation of ordered triplet (p, q, r).  
// The function returns following values  
// 0 --> p, q and r are colinear  
// 1 --> Clockwise  
// 2 --> Counterclockwise  
int orientation(Point p, Point q, Point r)  
{  
    // See https://www.geeksforgeeks.org/orientation-3-ordered-points/  
    // for details of below formula.  
    int val = (q.y - p.y) * (r.x - q.x) -  
              (q.x - p.x) * (r.y - q.y);  
  
    if (val == 0) return 0; // colinear  
  
    return (val > 0)? 1: 2; // clock or counterclock wise  
}  
  
// The main function that returns true if line segment 'p1q1'  
// and 'p2q2' intersect.  
bool doIntersect(Point p1, Point q1, Point p2, Point q2)  
{  
    // Find the four orientations needed for general and  
    // special cases  
    int o1 = orientation(p1, q1, p2);  
    int o2 = orientation(p1, q1, q2);  
    int o3 = orientation(p2, q2, p1);  
    int o4 = orientation(p2, q2, q1);  
  
    // General case  
    if (o1 != o2 && o3 != o4)  
        return true;  
}
```

```
// Special Cases
// p1, q1 and p2 are colinear and p2 lies on segment p1q1
if (o1 == 0 && onSegment(p1, p2, q1)) return true;

// p1, q1 and q2 are colinear and q2 lies on segment p1q1
if (o2 == 0 && onSegment(p1, q2, q1)) return true;

// p2, q2 and p1 are colinear and p1 lies on segment p2q2
if (o3 == 0 && onSegment(p2, p1, q2)) return true;

// p2, q2 and q1 are colinear and q1 lies on segment p2q2
if (o4 == 0 && onSegment(p2, q1, q2)) return true;

return false; // Doesn't fall in any of the above cases
}

// Driver program to test above functions
int main()
{
    struct Point p1 = {1, 1}, q1 = {10, 1};
    struct Point p2 = {1, 2}, q2 = {10, 2};

    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";

    p1 = {10, 0}, q1 = {0, 10};
    p2 = {0, 0}, q2 = {10, 10};
    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";

    p1 = {-5, -5}, q1 = {0, 0};
    p2 = {1, 1}, q2 = {10, 10};
    doIntersect(p1, q1, p2, q2)? cout << "Yes\n": cout << "No\n";

    return 0;
}
```

Output:

```
No
Yes
No
```

Sources:

<http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf>
Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

Source

<https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/>

Chapter 102

Icosagonal number

Icosagonal number - GeeksforGeeks

Given a number n, the task is to find the nth Icosagonal number.

An Icosagonal number is the 20-gon is a twenty-sided polygon. The number derived from the figurative class. There are different pattern series number in this number. The dots are countable, arrange in a specific way of position and create a diagram. All the dots have a common dots points, all others dots are connected to this points and except this common point the dots connected to their i-th dots with their respective successive layer.

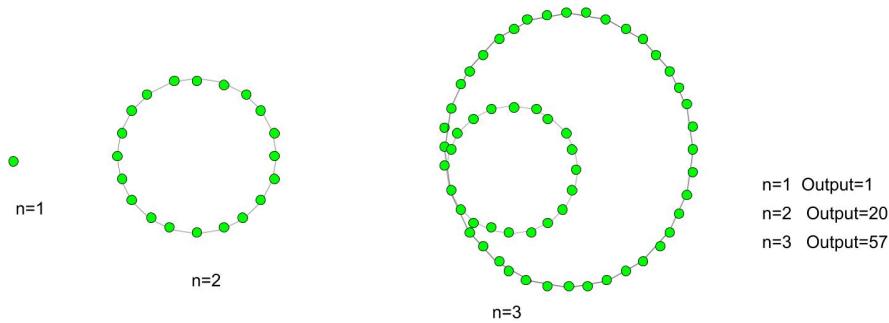
Examples :

Input : 3

Output :57

Input :8

Output :512



Formula for nth icosagonal number:

C++

```
// C++ program to find
// nth Icosagonal number
#include <bits/stdc++.h>
using namespace std;

// Function to calculate Icosagonal number
int icosagonal_poly(long int n)
{
    // Formula for finding
    // nth Icosagonal number
    return (18 * n * n - 16 * n) / 2;
}

// Drivers code
int main()
{
    long int n = 7;
    cout << n << "th Icosagonal number :"
        << icosagonal_poly(n);

    return 0;
}
```

Java

```
// Java program to find
// nth Icosagonal number

import java.io.*;

class GFG {

    // Function to calculate Icosagonal number

    static int icosagonal_poly(int n)
    {
        // Formula for finding
        // nth Icosagonal number
        return (18 * n * n - 16 * n) / 2;
    }

    // Drivers code

    public static void main (String[] args) {
```

```
int n = 7;

System.out.print (n + "th Icosagonal number :");
System.out.println(icosagonal_poly(n));
}

}

// This code is contributed by aj_36
```

Python 3

```
# Python 3 program to find
# nth Icosagonal number

# Function to calculate
# Icosagonal number
def icosagonal_poly(n) :

    # Formula for finding
    # nth Icosagonal number
    return (18 * n * n -
           16 * n) // 2

# Driver Code
if __name__ == '__main__':
    n = 7
    print(n,"th Icosagonal number :",
          icosagonal_poly(n))

# This code is contributed m_kit
```

C#

```
// C# program to find
// nth Icosagonal number
using System;

class GFG
{

    // Function to calculate
    // Icosagonal number
    static int icosagonal_poly(int n)
    {
        // Formula for finding
        // nth Icosagonal number
        return (18 * n * n -
               16 * n) / 2;
    }
}
```

```
}

// Driver code
static public void Main ()
{
    int n = 7;

    Console.WriteLine(n + "th Icosagonal " +
                      "number :");
    Console.WriteLine(icosagonal_poly(n));
}
}

// This code is contributed by ajit
```

PHP

```
<?php
// PHP program to find
// nth Icosagonal number

// Function to calculate
// Icosagonal number
function icosagonal_poly($n)
{
    // Formula for finding
    // nth Icosagonal number
    return (18 * $n *
            $n - 16 * $n) / 2;
}

// Driver Code
$n = 7;
echo $n , "th Icosagonal number :",
      icosagonal_poly($n);

// This code is contributed by ajit
?>
```

Output :

7th Icosagonal number :385

Reference: https://en.wikipedia.org/wiki/Polygonal_number

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/icosagonal-number/>

Chapter 103

Icosahedral Number

Icosahedral Number - GeeksforGeeks

Given a number n, find the n-th icosahedral number. The **Icosahedral Number** is class of figurative number that represents an icosahedron(a polyhedron with 20 faces) Source : [Wiki](#).

The first few Icosahedral Numbers are 1, 12, 48, 124, 255, 456, 742, 1128, 1629.....

Examples :

Input : 5
Output :255

Input :10
Output :2260

n-th term of Icosahedral Number is given by:

Basic implementation of the above idea:

C++

```
// Icosahedral number to find
// n-th term in C++
#include <bits/stdc++.h>
using namespace std;

// Function to find
// Icosahedral number
```

```
int icosahedralnum(int n)
{
    // Formula to calculate nth
    // Icosahedral number &
    // return it into main function.
    return (n * (5 * n * n - 5 * n + 2)) / 2;
}

// Driver Code
int main()
{
    int n = 7;

    cout << icosahedralnum(n);
    return 0;
}
```

Java

```
// Icosahedral number to find
// n-th term in Java
import java.io.*;

class GFG {

    // Function to find
    // Icosahedral number
    static int icosahedralnum(int n)
    {
        // Formula to calculate nth
        // Icosahedral number &
        // return it into main function.

        return (n * (5 * n * n - 5 *
                    n + 2)) / 2;
    }

    // Driver Code
    public static void main (String[] args)
    {
        int n = 7;
        System.out.println(
            icosahedralnum(n));
    }
}

// This code is contributed by aj_36.
```

Python3

```
# Python 3 Program to find
# nth Icosahedral number

# Icosahedral number
# number function
def icosahedralnum(n) :

    # Formula to calculate nth
    # Icosahedral number
    # return it into main function.
    return (n * (5 * n * n -
                  5 * n + 2)) // 2

# Driver Code
if __name__ == '__main__' :

    n = 7
    print(icosahedralnum(n))

# This code is contributed aj_36
```

C#

```
// Icosahedral number to
// find n-th term in C#
using System;

class GFG
{

    // Function to find
    // Icosahedral number
    static int icosahedralnum(int n)
    {
        // Formula to calculate
        // nth Icosahedral number
        // & return it into main
        // function.
        return (n * (5 * n * n -
                      5 * n + 2)) / 2;
    }

    // Driver Code
    static public void Main ()

```

```
{  
    int n = 7;  
    Console.WriteLine(icosahedralnum(n));  
}  
}  
  
// This code is contributed by ajit
```

PHP

```
<?php  
// Icosahedral number to  
// find n-th term in PHP  
  
// Function to find  
// Icosahedral number  
function icosahedralnum($n)  
{  
    // Formula to calculate nth  
    // Icosahedral number &  
    // return it into main function.  
    return ($n * (5 * $n * $n -  
        5 * $n + 2)) / 2;  
}  
  
// Driver Code  
$n = 7;  
  
echo icosahedralnum($n);  
  
// This code is contributed by m_kit  
?>
```

Output

742

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/icosahedral-number/>

Chapter 104

Icosidigonal number

Icosidigonal number - GeeksforGeeks

Given a number n, the task is to find the nth Icosidigonal number (Isdn).

The polygon has the many gons, depends on their gonal number series. In mathematics, there is a number of gonal numbers and the icosidigonal number is one of them and these numbers have 22 -sided polygon (icosidigon). An icosidigonal number belong to the class of figurative number. They have one common dots points and other dots pattern is arranged in an n-th nested icosidigon pattern.

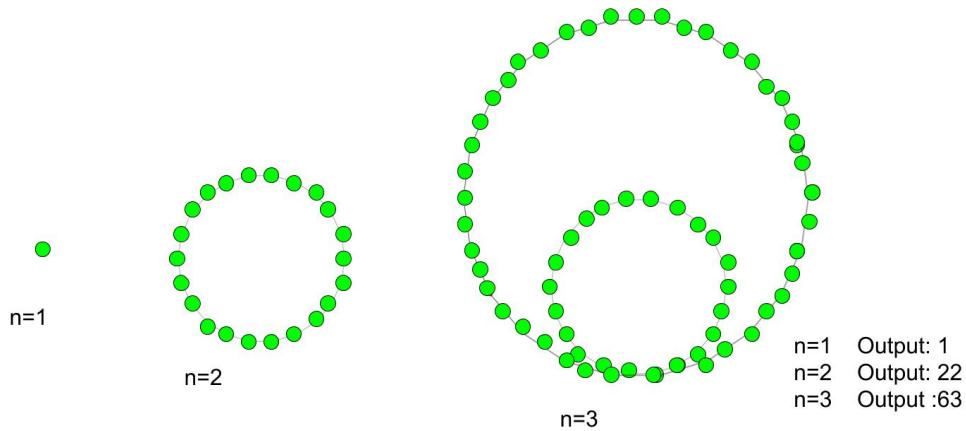
Examples :

Input : 2

Output :22

Input :6

Output :306



Formula for nth Icosidigonal number:

C++

```
// C++ program to find nth Icosidigonal
// number
#include <bits/stdc++.h>
using namespace std;

// Function to calculate Icosidigonal number
int icosidigonal_num(long int n)
{
    // Formula for finding
    // nth Icosidigonal number
    return (20 * n * n - 18 * n) / 2;
}

// Driver function
int main()
{
    long int n = 4;
    cout << n << "th Icosidigonal number :" << icosidigonal_num(n);
    cout << endl;
    n = 8;
    cout << n << "th Icosidigonal number:" << icosidigonal_num(n);
    return 0;
}
```

Java

```
// Java program to find nth
// Icosidigonal number
import java.io.*;

class GFG
{

    // Function to calculate
    // Icosidigonal number
    static int icosidigonal_num(int n)
    {

        // Formula for finding
        // nth Icosidigonal number
```

```
    return (20 * n * n - 18 * n) / 2;
}

// Driver Code
public static void main (String[] args)
{
    int n = 4;
    System.out.print (n + "th Icosidigonal number :");
    System.out.println (icosidigonal_num(n));

    n = 8;
    System.out.print (n + "th Icosidigonal number :");
    System.out.println (icosidigonal_num(n));
}
}

// This code is contributed by ajit
```

Python 3

```
# python 3 program to find
# nth Icosidigonal number

# Function to calculate
# Icosidigonal number
def icosidigonal_num(n) :

    # Formula for finding
    # nth Icosidigonal number
    return (20 * n * n -
           18 * n) // 2

# Driver Code
if __name__ == '__main__' :

    n = 4
    print(n,"th Icosidigonal " +
          "number : ",
          icosidigonal_num(n))
    n = 8
    print(n,"th Icosidigonal " +
          "number : ",
          icosidigonal_num(n))

# This code is contributed m_kit
```

C#

```
// C# program to find nth
// Icosidigonal number
using System;

class GFG
{

    // Function to calculate
    // Icosidigonal number
    static int icosidigonal_num(int n)
    {

        // Formula for finding
        // nth Icosidigonal number
        return (20 * n * n -
            18 * n) / 2;
    }

    // Driver Code
    static public void Main ()
    {
        int n = 4;
        Console.Write(n + "th Icosidigonal " +
                      "number :");
        Console.WriteLine(icosidigonal_num(n));

        n = 8;
        Console.Write (n + "th Icosidigonal "+
                      "number :");
        Console.WriteLine(icosidigonal_num(n));
    }
}

// This code is contributed by ajit
```

PHP

```
<?php
// PHP program to find nth
// Icosidigonal number

// Function to calculate
// Icosidigonal number
function icosidigonal_num($n)
{
```

```
// Formula for finding
// nth Icosidigonal number
return (20 * $n * $n - 18 * $n) / 2;
}

// Driver Code
$n = 4;
echo $n , "th Icosidigonal number : ",
       icosidigonal_num($n);
echo "\n";
$n = 8;
echo $n , "th Icosidigonal number : ",
       icosidigonal_num($n);

// This code is contributed by m_kit
?>
```

Output :

```
4th Icosidigonal number :124
8th Icosidigonal number:568
```

Reference: https://en.wikipedia.org/wiki/Polygonal_number

Improved By : [jit_t](#)

Source

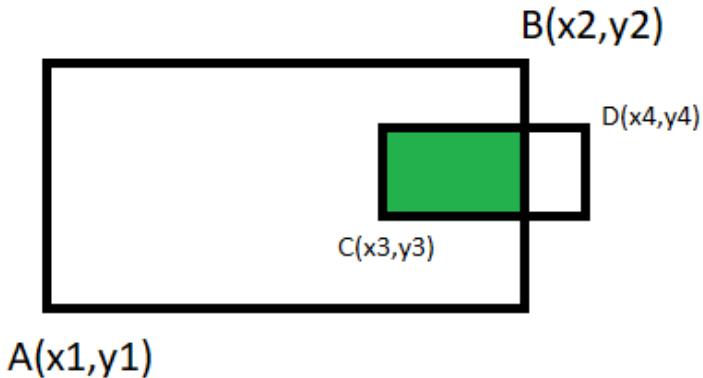
<https://www.geeksforgeeks.org/icosidigonal-number/>

Chapter 105

Intersecting rectangle when bottom-left and top-right corners of two rectangles are given

Intersecting rectangle when bottom-left and top-right corners of two rectangles are given - GeeksforGeeks

Given coordinates of 4 points, bottom-left and top-right corners of two rectangles. The task is to find the coordinates of the intersecting rectangle formed by the given two rectangles.



Examples:

Input:

rec1: bottom-left(0, 0), top-right(10, 8),
rec2: bottom-left(2, 3), top-right(7, 9)

Output: (2, 3) (7, 8) (2, 8) (7, 3)

Input:

rec1: bottom-left(0, 0), top-right(3, 3),
rec2: bottom-left(1, 1), top-right(2, 2)

Output: (1, 1) (2, 2) (1, 2) (2, 1)

Approach :

As two given points are diagonals of a rectangle. so, $x_1 < x_2, y_1 < y_2$. similarly $x_3 < x_4, y_3 < y_4$.

so, bottom-left and top-right points of intersection rectangle can be found by using formula.

```
x5 = max(x1, x3);
y5 = max(y1, y3);
x6 = min(x2, x4);
y6 = min(y2, y4);
```

In case of no intersection, x5 and y5 will always exceed x6 and y5 respectively. The other two points of the rectangle can be found by using simple geometry.

Below is the implementation of the above approach:

C++

```
// CPP program to find intersection
// rectangle of given two rectangles.
#include <bits/stdc++.h>
using namespace std;

// function to find intersection rectangle of given two rectangles.
void FindPoints(int x1, int y1, int x2, int y2,
                 int x3, int y3, int x4, int y4)
{
    // gives bottom-left point
    // of intersection rectangle
    int x5 = max(x1, x3);
    int y5 = max(y1, y3);

    // gives top-right point
    // of intersection rectangle
    int x6 = min(x2, x4);
    int y6 = min(y2, y4);

    // no intersection
    if (x5 > x6 || y5 > y6) {
        cout << "No intersection";
        return;
    }

    cout << "(" << x5 << ", " << y5 << ") ";
    cout << "(" << x6 << ", " << y6 << ") ";

    // gives top-left point
    // of intersection rectangle
    int x7 = x5;
    int y7 = y6;

    cout << "(" << x7 << ", " << y7 << ") ";

    // gives bottom-right point
    // of intersection rectangle
    int x8 = x6;
    int y8 = y5;

    cout << "(" << x8 << ", " << y8 << ") ";
}

// Driver code
int main()
{
    // bottom-left and top-right
```

```
// corners of first rectangle
int x1 = 0, y1 = 0, x2 = 10, y2 = 8;

// bottom-left and top-right
// corners of first rectangle
int x3 = 2, y3 = 3, x4 = 7, y4 = 9;

// function call
FindPoints(x1, y1, x2, y2, x3, y3, x4, y4);

return 0;
}
```

Java

```
// Java program to find intersection
// rectangle of given two rectangles.
class GFG
{

    // function to find intersection
    // rectangle of given two rectangles.
    static void FindPoints(int x1, int y1,
                           int x2, int y2,
                           int x3, int y3,
                           int x4, int y4)
    {
        // gives bottom-left point
        // of intersection rectangle
        int x5 = Math.max(x1, x3);
        int y5 = Math.max(y1, y3);

        // gives top-right point
        // of intersection rectangle
        int x6 = Math.min(x2, x4);
        int y6 = Math.min(y2, y4);

        // no intersection
        if (x5 > x6 || y5 > y6)
        {
            System.out.println("No intersection");
            return;
        }

        System.out.print("(" + x5 + ", " +
                      y5 + ") ");

        System.out.print("(" + x6 + ", " +
                      y6 + ") ");
    }
}
```

```
y6 + ") "));

// gives top-left point
// of intersection rectangle
int x7 = x5;
int y7 = y6;

System.out.print("(" + x7 + ", " +
                  y7 + ") ");

// gives bottom-right point
// of intersection rectangle
int x8 = x6;
int y8 = y5;

System.out.print("(" + x8 + ", " +
                  y8 + ") ");
}

// Driver code
public static void main(String args[])
{
    // bottom-left and top-right
    // corners of first rectangle
    int x1 = 0, y1 = 0,
        x2 = 10, y2 = 8;

    // bottom-left and top-right
    // corners of second rectangle
    int x3 = 2, y3 = 3,
        x4 = 7, y4 = 9;

    // function call
    FindPoints(x1, y1, x2, y2,
               x3, y3, x4, y4);
}
}

// This code is contributed
// by Arnab Kundu
```

Output:

(2, 3) (7, 8) (2, 8) (7, 3)

Time Complexity: O(1)

Improved By : [andrew1234](#)

Source

<https://www.geeksforgeeks.org/intersecting-rectangle-when-bottom-left-and-top-right-corners-of-two-rectangles-are-given>

Chapter 106

Klee's Algorithm (Length Of Union Of Segments of a line)

Klee's Algorithm (Length Of Union Of Segments of a line) - GeeksforGeeks

Given starting and ending positions of segments on a line, the task is to take the union of all given segments and find length covered by these segments.

Examples:

```
Input : segments[] = {{2, 5}, {4, 8}, {9, 12}}
Output : 9
segment 1 = {2, 5}
segment 2 = {4, 8}
segment 3 = {9, 12}
If we take the union of all the line segments,
we cover distances [2, 8] and [9, 12]. Sum of
these two distances is 9 (6 + 3)
```

The algorithm was proposed by Klee in 1977. The time complexity of the algorithm is $O(N \log N)$. It has been proven that this algorithm is the fastest (asymptotically) and this problem can not be solved with a better complexity.

```
// C++ program to implement Klee's algorithm
#include<bits/stdc++.h>
using namespace std;

// Returns sum of lengths covered by union of given
// segments
int segmentUnionLength(const vector <pair <int,int> > &seg)
{
    int n = seg.size();
```

```
// Create a vector to store starting and ending
// points
vector <pair <int, bool> > points(n * 2);
for (int i = 0; i < n; i++)
{
    points[i*2]      = make_pair(seg[i].first, false);
    points[i*2 + 1] = make_pair(seg[i].second, true);
}

// Sorting all points by point value
sort(points.begin(), points.end());

int result = 0; // Initialize result

// To keep track of counts of current open segments
// (Starting point is processed, but ending point
// is not)
int Counter = 0;

// Traverse through all points
for (unsigned i=0; i<n*2; i++)
{
    // If there are open points, then we add the
    // difference between previous and current point.
    // This is interesting as we don't check whether
    // current point is opening or closing,
    if (Counter)
        result += (points[i].first - points[i-1].first);

    // If this is an ending point, reduce, count of
    // open points.
    (points[i].second)? Counter-- : Counter++;
}
return result;
}

// Driver program for the above code
int main()
{
    vector< pair <int,int> > segments;
    segments.push_back(make_pair(3, 15));
    segments.push_back(make_pair(2, 5));
    segments.push_back(make_pair(4, 8));
    segments.push_back(make_pair(9, 12));
    cout << "Length of Union of All segments = ";
    cout << segmentUnionLength(segments) << endl;
    return 0;
}
```

}

Output:

```
Length of Union of All segments = 9
```

Description :

- 1) Put all the coordinates of all the segments in an auxiliary array points[].
- 2) Sort it on the value of the coordinates.
- 3) An additional condition for sorting – if there are equal coordinates, insert the one which is the left coordinate of any segment instead of a right one.
- 4) Now go through the entire array, with the counter “count” of overlapping segments.
- 5) If count is greater than zero, then the result is added to the difference between the points[i] – points[i-1].
- 6) If the current element belongs to the left end, we increase “count”, otherwise reduce it.

Illustration:

Lets take the example :

```
segment 1 : (2,5)
segment 2 : (4,8)
segment 3 : (9,12)
```

```
Counter = result = 0;
n = number of segments = 3;

for i=0, points[0] = {2, false}
    points[1] = {5, true}
for i=1, points[2] = {4, false}
    points[3] = {8, true}
for i=2, points[4] = {9, false}
    points[5] = {12, true}
```

Therefore :

```
points = {2, 5, 4, 8, 9, 12}
        {f, t, f, t, f, t}
```

after applying sorting :

```
points = {2, 4, 5, 8, 9, 12}
        {f, f, t, t, f, t}
```

Now,

```
for i=0, result = 0;
    Counter = 1;

for i=1, result = 2;
    Counter = 2;
```

```
for i=2, result = 3;  
    Counter = 1;  
  
for i=3, result = 6;  
    Counter = 0;  
  
for i=4, result = 6;  
    Counter = 1;  
  
for i=5, result = 9;  
    Counter = 0;  
  
Final answer = 9;
```

Time Complexity : $O(n * \log n)$

Source

<https://www.geeksforgeeks.org/klees-algorithm-length-union-segments-line/>

Chapter 107

LS3/NS3 sphere generation algorithm and its implementation

LS3/NS3 sphere generation algorithm and its implementation - GeeksforGeeks

Given center of the sphere and its radius. your task is to store efficiently all the integer points required to show a sphere on the computer screen of pixels and This algorithm will generate naive sphere by creating voxels by its naive arcs. NS3 stands for naive sphere by sum of squares and LS3 stands for lattice sphere by sum of squares.

Examples:

```
Input : Center(0, 0, 0) Radius 1
Output : (-1, 0, 0), (0, 0, 1), (0, 0, -1), (0, 1, 0),
          (0, -1, 0), (1, 0, 0)
          6(Total no. of points)
```

```
Input : Center(0, 0, 0) Radius 2
Output :(-2, 0, 0)(-1, 1, 1)(-1, -1, 1)(-1, 1, -1)
          (-1, -1, -1)(0, 0, 2)(0, 0, -2)(0, 2, 0)
          (0, -2, 0)(0, 1, 2)(0, 2, 1)(0, -1, 2)
          (0, 1, -2)(0, -1, -2)(0, -2, 1)(0, 2, -1)
          (0, -2, -1)(1, 2, 0)(1, 0, 2)(1, -2, 0)
          (1, 2, 0)(1, -2, 0)(1, 0, -2)(1, 0, 2)
          (1, 0, -2)(1, 1, 1)(1, -1, 1)(1, 1, -1)
          (1, -1, -1)(2, 0, 0)(2, 0, 1)(2, 0, 1)
          (2, 0, -1)(2, 0, -1)(2, 1, 0)(2, -1, 0)
          (2, 1, 0)(2, -1, 0)
          38 (Total no. of points)
```

I strongly suggest you that if you do not have any prerequisites in this field than go to below link. This will help you to gain some prerequisites and concepts which are required to understand the following implementation and algorithm.

Algorithm LS3 primarily focuses on generating the voxels of **prima quadraginta**. prima quadraginta is formed when a discrete sphere is partitioned into 48 parts such that all 48 parts having some symmetrical relation and they all can be generated by using any one of its parts. that one part is called as prima quadraginta (see figure 2). similarly, quadraginta octant defined as same as prima qudraginta except that it is partitioned into 8 equal parts instead of 48. quadaginta octant contains 6 prima quadraginta which are clearly shown in the figure below. Q denotes to prima quadraginta (q-octant) and C denotes to quadraginta octant(c-octant).a naive arc are those set of voxels which are having same x coordinate in prima quadraginta (in above figure, the red voxels are making a naive arc).

By dint of 48 symmetric property, a whole discrete sphere can be developed from its prima quadraginta. **the limitation of using 48 symmetry is that some of the voxels of q-octant will be repeated because they are sharing an edge or voxels with other adjacent prima quadraginta. for example, In above figure, the voxels shown in grey belongs to two or more q-octants.**

Our algorithm will generate some repeated voxels therefore, we need to make some restrictions on input voxels.

In order to do this, I have included some functions in my implementation which are:

All these methods principally focuses on grey voxels (See above figure) expect one putpixelall() which is going to produce all voxels except grey voxels by using 48 symmetry without any restrictions.

putpixeledge1 : This function will produce all voxels which has exactly one coordinate zero

and if we proceed to find another voxel using symmetry and take an image with respect to that zero coordinate

axis (this (0, 2, -1) voxel will have the same image if we take an image with respect to the x-axis) then it comes out to be the same voxel.

putpixeledge2 : These set of voxels having x and y coordinate are same (see prima quadraginta figure 2).In order to store these voxels we have to look for a relation between q1 and q2 octants(see above table). you can achieve the general coordinate of any voxel in q2 from q1 by swapping first (x, y) and then (y, z).

putpixelsingle : These pixels is very less in count.they are situated in the middle of quadraginta octant (c octant) and shared with all the q-octants which are lying in the same c-octant. to obtained these voxels we can see the above table and write all the voxels corresponding to its c-octant.

putpixeldouble : These set of voxels having y and z coordinate are same (see prima qudraginta figure 2).In order to store these voxels we need to find a relation between q1 and q6 octants(see table). you can achieve the general coordinate of any voxel in q6 from q1 by swapping first (y, z) and then (x, y).

putpixelmiddle : These set of voxels are located in those points where exactly two coordinate

value is NULL which can help us to find to obtain the coordinates of these voxels(you can write the coordinates of these voxels manually).

We need an efficient storage of all the integer points (voxels) which will be formed by this algorithm. In order to store these points, we will use hashmap and key value of hashmap will be x coordinate of point and data value will be linked list of pair of y and z coordinate.

```
#include <iostream>
#include <cmath>

// this header file contains get<0> and get<1> function
#include <utility>

#include <list>
#include <map>
using namespace std;

// map to store the voxels
map<int, list<pair<int, int> > > mymap;
map<int, list<pair<int, int> > >::iterator itr;

// this function will store single voxel
void putpixelone(int m, int n, int p)
{
    mymap[m].push_back(make_pair(n, p));
}

// function to store some particular type of voxel
void putpixelmiddle(int a, int b, int c, int x, int y, int z)
{
    putpixelone(a + x, b + y, c + z);
    putpixelone(a + x, b + y, -c + z);
    putpixelone(a + x, c + y, b + z);
    putpixelone(a + x, -c + y, b + z);
    putpixelone(c + x, a + y, b + z);
    putpixelone(-c + x, a + y, b + z);
}

// This program will generate 24 voxels
void putpixeldouble(int a, int b, int c, int x, int y, int z)
{
    for (int j = 0; j < 3; j++) {
        putpixelone(a + x, b + y, c + z);
        swap(b, c);
        swap(a, b);
    }
    for (int j = 0; j < 3; j++) {
        putpixelone(-a + x, b + y, c + z);
    }
}
```

```

        swap(b, c);
        swap(a, b);
    }
    for (int j = 0; j < 3; j++) {
        putpixelone(a + x, -b + y, c + z);
        swap(b, c);
        swap(a, b);
    }
    for (int j = 0; j < 3; j++) {
        putpixelone(-a + x, -b + y, c + z);
        swap(b, c);
        swap(a, b);
    }
    for (int j = 0; j < 3; j++) {
        putpixelone(a + x, b + y, -c + z);
        swap(b, c);
        swap(a, b);
    }
    for (int j = 0; j < 3; j++) {
        putpixelone(-a + x, b + y, -c + z);
        swap(b, c);
        swap(a, b);
    }
    for (int j = 0; j < 3; j++) {
        putpixelone(a + x, -b + y, -c + z);
        swap(b, c);
        swap(a, b);
    }
    for (int j = 0; j < 3; j++) {
        putpixelone(-a + x, -b + y, -c + z);
        swap(b, c);
        swap(a, b);
    }
}

// Explained above
void putpixelsingle(int a, int b, int c, int x,
                     int y, int z)
{
    putpixelone(a + x, b + y, c + z);
    putpixelone(-a + x, b + y, c + z);
    putpixelone(a + x, -b + y, c + z);
    putpixelone(-a + x, -b + y, c + z);
    putpixelone(a + x, b + y, -c + z);
    putpixelone(-a + x, b + y, -c + z);
    putpixelone(a + x, -b + y, -c + z);
    putpixelone(-a + x, -b + y, -c + z);
}

```

```
// This program will generate 24 voxels.  
void putpixeledge2(int a, int b, int c, int x,  
                    int y, int z)  
{  
  
    for (int j = 0; j < 3; j++) {  
        putpixelone(a + x, b + y, c + z);  
        swap(a, b);  
        swap(b, c);  
    }  
    for (int j = 0; j < 3; j++) {  
        putpixelone(-a + x, b + y, c + z);  
        swap(a, b);  
        swap(b, c);  
    }  
    for (int j = 0; j < 3; j++) {  
        putpixelone(a + x, -b + y, c + z);  
        swap(a, b);  
        swap(b, c);  
    }  
    for (int j = 0; j < 3; j++) {  
        putpixelone(-a + x, -b + y, c + z);  
        swap(a, b);  
        swap(b, c);  
    }  
    for (int j = 0; j < 3; j++) {  
        putpixelone(a + x, b + y, -c + z);  
        swap(a, b);  
        swap(b, c);  
    }  
    for (int j = 0; j < 3; j++) {  
        putpixelone(-a + x, b + y, -c + z);  
        swap(a, b);  
        swap(b, c);  
    }  
    for (int j = 0; j < 3; j++) {  
        putpixelone(a + x, -b + y, -c + z);  
        swap(a, b);  
        swap(b, c);  
    }  
    for (int j = 0; j < 3; j++) {  
        putpixelone(-a + x, -b + y, -c + z);  
        swap(a, b);  
        swap(b, c);  
    }  
}  
// after excluding the repeated pixels from the set all 48  
// pixels we will left with only 24 pixels so, we have
```

```

// defined all the voxels given below.
void putpixeledge1(int a, int b, int c, int x, int y, int z)
{
    putpixelone(a + x, b + y, c + z);
    putpixelone(a + x, c + y, b + z);
    putpixelone(a + x, -b + y, c + z);
    putpixelone(a + x, b + y, -c + z);
    putpixelone(a + x, -b + y, -c + z);
    putpixelone(a + x, -c + y, b + z);
    putpixelone(a + x, c + y, -b + z);
    putpixelone(a + x, -c + y, -b + z);
    putpixelone(b + x, c + y, a + z);
    putpixelone(b + x, a + y, c + z);
    putpixelone(b + x, -c + y, a + z);
    putpixelone(b + x, c + y, -a + z);
    putpixelone(b + x, -c + y, -a + z);
    putpixelone(b + x, a + y, -c + z);
    putpixelone(b + x, -a + y, c + z);
    putpixelone(b + x, -a + y, -c + z);
    putpixelone(c + x, a + y, b + z);
    putpixelone(c + x, -a + y, b + z);
    putpixelone(c + x, a + y, -b + z);
    putpixelone(c + x, -a + y, -b + z);
    putpixelone(c + x, b + y, a + z);
    putpixelone(c + x, -b + y, a + z);
    putpixelone(c + x, b + y, -a + z);
    putpixelone(c + x, -b + y, -a + z);

}

// voxel formation by 48 symmetry.
void putpixelall(int a, int b, int c, int x,
                 int y, int z)
{
    putpixelone(a + x, b + y, c + z);
    putpixelone(b + x, a + y, c + z);
    putpixelone(c + x, b + y, a + z);
    putpixelone(a + x, c + y, b + z);
    putpixelone(c + x, a + y, b + z);
    putpixelone(b + x, c + y, a + z);
    putpixelone(-a + x, -b + y, c + z);
    putpixelone(-b + x, -a + y, c + z);
    putpixelone(c + x, -b + y, -a + z);
    putpixelone(-a + x, c + y, -b + z);
    putpixelone(c + x, -a + y, -b + z);
    putpixelone(-b + x, c + y, -a + z);
    putpixelone(a + x, -b + y, -c + z);
    putpixelone(-b + x, a + y, -c + z);
    putpixelone(-c + x, -b + y, a + z);

```

```

putpixelone(a + x, -c + y, -b + z);
putpixelone(-c + x, a + y, -b + z);
putpixelone(-b + x, -c + y, a + z);
putpixelone(-a + x, b + y, -c + z);
putpixelone(b + x, -a + y, -c + z);
putpixelone(-c + x, b + y, -a + z);
putpixelone(-a + x, -c + y, b + z);
putpixelone(-c + x, -a + y, b + z);
putpixelone(b + x, -c + y, -a + z);
putpixelone(-a + x, b + y, c + z);
putpixelone(b + x, -a + y, c + z);
putpixelone(c + x, b + y, -a + z);
putpixelone(-a + x, c + y, b + z);
putpixelone(c + x, -a + y, b + z);
putpixelone(b + x, c + y, -a + z);
putpixelone(a + x, -b + y, c + z);
putpixelone(-b + x, a + y, c + z);
putpixelone(c + x, -b + y, a + z);
putpixelone(a + x, c + y, -b + z);
putpixelone(c + x, a + y, -b + z);
putpixelone(-b + x, c + y, a + z);
putpixelone(a + x, b + y, -c + z);
putpixelone(b + x, a + y, -c + z);
putpixelone(-c + x, b + y, a + z);
putpixelone(a + x, -c + y, b + z);
putpixelone(-c + x, a + y, b + z);
putpixelone(b + x, -c + y, a + z);
putpixelone(-a + x, -b + y, -c + z);
putpixelone(-b + x, -a + y, -c + z);
putpixelone(-c + x, -b + y, -a + z);
putpixelone(-a + x, -c + y, -b + z);
putpixelone(-c + x, -a + y, -b + z);
putpixelone(-b + x, -c + y, -a + z);

}

// detailed explanation of this algorithm is
// given in above link.
void drawsphere(int x, int y, int z, int r)
{
    int i = 0, j = 0;
    int k0 = r;
    int k = k0;
    int s0 = 0;
    int s = s0;
    int v0 = r - 1;
    int v = v0;
    int l0 = 2 * v0;
    int l = l0;
}

```

```

// this while loop will form naive arcs
while (i <= k) {

    // this while will form voxels in naive arcs
    while (j <= k) {
        if (s > v) {
            k = k - 1;
            v = v + 1;
            l = l - 2;
        }
        if ((j <= k) && ((s != v) || (j != k))) {

            // this if, else and else if condition
            // can easily build using figure 2, 3
            if (i == 0 && j != 0)

                // First naive arc pixels and each
                // pixel is shared with two q-octants
                putpixeledge1(i, j, k, x, y, z);

            // voxels shared between q1 and q2
            else if (i == j && j != k && i != 0)
                putpixeledge2(i, j, k, x, y, z);

            // center voxel of c-octants
            else if (i == j && j == k)
                putpixelsingle(i, j, k, x, y, z);

            // voxels shared between q1 and q6
            else if (j == k && i < k && i < j)
                putpixeldouble(i, j, k, x, y, z);

            // starting voxel of q-octant
            else if (i == 0 && j == 0)
                putpixelmiddle(i, j, k, x, y, z);

            // voxels inside the q-octant
            else
                putpixelall(i, j, k, x, y, z);
        }
        s = s + 2 * j + 1;
        j = j + 1;
    }
    s0 = s0 + 4 * i + 2;
    i = i + 1;

    while ((s0 > v0) && (i <= k0)) {

```

```

        k0 = k0 - 1;
        v0 = v0 + 10;
        l0 = l0 - 2;
    }
    j = i;
    k = k0;
    v = v0;
    l = l0;
    s = s0;
}
}

// Print out all the voxels of discrete sphere
void showallpoints(map<int, list<pair<int, int> > & mymap)
{
    int count = 0;

    for (itr = mymap.begin(); itr != mymap.end(); ++itr) {
        list<pair<int, int> > l = itr->second;
        list<pair<int, int> ::iterator it;
        for (it = l.begin(); it != l.end(); ++it) {
            cout << itr->first << ", " << get<0>(*it)
                << ", " << get<1>(*it) << endl;
            count += 1;
        }
    }
    cout << endl;
    cout << count << endl;
}
// Driver program
int main()
{
    int cx, cy, cz;
    cin >> cx >> cy >> cz;
    int r;
    cin >> r;
    drawsphere(cx, cy, cz, r);
    showallpoints(mymap);
}

```

Input : Center(0, 0, 0) Radius 3

Output:

```

(-3, 0, 0)(-3, 1, 1)(-3, -1, 1)(-3, 1, -1)(-3, -1, -1)
(-2, 1, 2)(-2, 2, 1)(-2, -1, 2)(-2, -2, 1)(-2, 1, -2)
(-2, 2, -1)(-2, -1, -2)(-2, -2, -1)(-1, 1, 3)(-1, 3, 1)
(-1, -1, 3)(-1, -3, 1)(-1, 1, -3)(-1, 3, -1)(-1, -1, -3)
(-1, -3, -1)(-1, 2, 2)(-1, -2, 2)(-1, 2, -2)(-1, -2, -2)

```

(0, 0, 3)(0, 0, -3)(0, 3, 0)(0, -3, 0)(0, 1, 3)(0, 3, 1)
(0, -1, 3)(0, 1, -3)(0, -1, -3)(0, -3, 1)(0, 3, -1)
(0, -3, -1)(0, 2, 2)(0, 2, 2)(0, -2, 2)(0, 2, -2)
(0, -2, -2)(0, -2, 2)(0, 2, -2)(0, -2, -2)(1, 3, 0)
(1, 0, 3)(1, -3, 0)(1, 3, 0)(1, -3, 0)(1, 0, -3)(1, 0, 3)
(1, 0, -3)(1, 1, 3)(1, 3, 1)(1, -1, 3)(1, -3, 1)(1, 1, -3)
(1, 3, -1)(1, -1, -3)(1, -3, -1)(1, 2, 2)(1, -2, 2)(1, 2, -2)
(1, -2, -2)(2, 2, 0)(2, 0, 2)(2, -2, 0)(2, 2, 0)(2, -2, 0)
(2, 0, -2)(2, 0, 2)(2, 0, -2)(2, 0, 2)(2, 0, 2)(2, 0, -2)
(2, 0, -2)(2, 2, 0)(2, -2, 0)(2, 2, 0)(2, -2, 0)(2, 1, 2)
(2, 2, 1)(2, -1, 2)(2, -2, 1)(2, 1, -2)(2, 2, -1)(2, -1, -2)
(2, -2, -1)(3, 0, 0)(3, 0, 1)(3, 0, 1)(3, 0, -1)(3, 0, -1)
(3, 1, 0)(3, -1, 0)(3, 1, 0)(3, -1, 0)(3, 1, 1)(3, -1, 1)
(3, 1, -1)(3, -1, -1)
102 (Total no. of points)

References :

<http://www.sciencedirect.com/science/article/pii/S0304397515010178#tl0010>

Source

<https://www.geeksforgeeks.org/ls3ns3-sphere-generation-algorithm-implementation/>

Chapter 108

Lexicographically Kth smallest way to reach given coordinate from origin

Lexicographically Kth smallest way to reach given coordinate from origin - GeeksforGeeks

Given a coordinate (x, y) on a 2D plane. We have to reach (x, y) from the current position which is at origin i.e $(0, 0)$. In each step, we can either move vertically or horizontally on the plane. While moving horizontally each step we write 'H' and while moving vertically each step we write 'V'. So, there can be possibly many strings containing 'H' and 'V' which represents a path from $(0, 0)$ to (x, y) . The task is to find the lexicographically Kth smallest string among all the possible strings.

Examples:

Input: $x = 2, y = 2, k = 2$

Output: HVHV

Explanation: There are 6 ways to reach $(2, 2)$ from $(0, 0)$. The possible list of strings in lexicographically sorted order: ["HHVV", "HVHV", "HVVH", "VHHV", "VHVV", "VVHH"]. Hence, the lexicographically 2nd smallest string is HVHV.

Input : $x = 2, y = 2, k = 3$

Output : VHHV

Prerequisites: [Ways to Reach a Point from Origin](#)

Approach: The idea is to use recursion to solve the problem. Number of ways to reach (x, y) from origin is ${}^x + {}^y C_x$.

Now observe, the number of ways to reach (x, y) from $(1, 0)$ will be $(x + y - 1, x - 1)$ because we have already made a step in the horizontal direction, so 1 is subtracted from x. Also, the number of ways to reach (x, y) from $(0, 1)$ will be $(x + y - 1, y - 1)$ because we

have already made a step in the vertical direction, so 1 is subtracted from y. Since ‘H’ is lexicographically smaller than ‘V’, so among all strings starting strings will contain ‘H’ in the beginning i.e initial movements will be Horizontal.

So, if $K \leq x + y - 1 C_{x-1}$, we will take ‘H’ as first step else we will take ‘V’ as first step and solve for number of goings to (x, y) from (1, 0) will be $K = K - x + y - 1 C_{x-1}$.

Below is the implementation of this approach:

C++

```
// CPP Program to find Lexicographically Kth
// smallest way to reach given coordinate from origin
#include <bits/stdc++.h>
using namespace std;

// Return (a+b)!/a!b!
int factorial(int a, int b)
{
    int res = 1;

    // finding (a+b)!
    for (int i = 1; i <= (a + b); i++)
        res = res * i;

    // finding (a+b)!/a!
    for (int i = 1; i <= a; i++)
        res = res / i;

    // finding (a+b)!/b!
    for (int i = 1; i <= b; i++)
        res = res / i;

    return res;
}

// Return the Kth smallest way to reach given coordinate from origin
void Ksmallest(int x, int y, int k)
{
    // if at origin
    if (x == 0 && y == 0)
        return;

    // if on y-axis
    else if (x == 0) {
        // decrement y.
        y--;

        // Move vertical
```

```
cout << "V";  
  
    // recursive call to take next step.  
    Ksmallest(x, y, k);  
}  
  
// If on x-axis  
else if (y == 0) {  
    // decrement x.  
    x--;  
  
    // Move horizontal.  
    cout << "H";  
  
    // recursive call to take next step.  
    Ksmallest(x, y, k);  
}  
else {  
    // If x + y C x is greater than K  
    if (factorial(x - 1, y) > k) {  
        // Move Horizontal  
        cout << "H";  
  
        // recursive call to take next step.  
        Ksmallest(x - 1, y, k);  
    }  
    else {  
        // Move vertical  
        cout << "V";  
  
        // recursive call to take next step.  
        Ksmallest(x, y - 1, k - factorial(x - 1, y));  
    }  
}
```

// Driven Program
int main()
{
 int x = 2, y = 2, k = 2;

 Ksmallest(x, y, k);

 return 0;
}

Java

```
// Java Program to find
// Lexicographically Kth
// smallest way to reach
// given coordinate from origin
import java.io.*;

class GFG
{

    // Return (a+b)!/a!b!
    static int factorial(int a,
                         int b)
    {
        int res = 1;

        // finding (a+b)-
        for (int i = 1;
             i <= (a + b); i++)
            res = res * i;

        // finding (a+b)!/a!
        for (int i = 1; i <= a; i++)
            res = res / i;

        // finding (a+b)!/b!
        for (int i = 1; i <= b; i++)
            res = res / i;

        return res;
    }

    // Return the Kth smallest
    // way to reach given
    // coordinate from origin
    static void Ksmallest(int x,
                          int y, int k)
    {
        // if at origin
        if (x == 0 && y == 0)
            return;

        // if on y-axis
        else if (x == 0)
        {
            // decrement y.
            y--;

            // Move vertical
    
```

```
System.out.print("V");

// recursive call to
// take next step.
Ksmallest(x, y, k);
}

// If on x-axis
else if (y == 0)
{
    // decrement x.
    x--;

    // Move horizontal.
    System.out.print("H");

    // recursive call to
    // take next step.
    Ksmallest(x, y, k);
}
else
{
    // If x + y < x is
    // greater than K
    if (factorial(x - 1, y) > k)
    {
        // Move Horizontal
        System.out.print( "H");

        // recursive call to
        // take next step.
        Ksmallest(x - 1, y, k);
    }
    else
    {
        // Move vertical
        System.out.print("V");

        // recursive call to
        // take next step.
        Ksmallest(x, y - 1, k -
                  factorial(x - 1, y));
    }
}

// Driver Code
public static void main (String[] args)
```

```
{  
    int x = 2, y = 2, k = 2;  
  
    Ksmallest(x, y, k);  
}  
}  
  
// This code is contributed  
// by anuj_67.
```

Output

HVWH

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/lexicographically-kth-smallest-way-reach-given-coordinate-origin/>

Chapter 109

Line Clipping | Set 1 (Cohen–Sutherland Algorithm)

Line Clipping | Set 1 (Cohen–Sutherland Algorithm) - GeeksforGeeks

Given a set of lines and a rectangular area of interest, the task is to remove lines which are outside the area of interest and clip the lines which are partially inside the area.

```
Input : Rectangular area of interest (Defined by
        below four values which are coordinates of
        bottom left and top right)
        x_min = 4, y_min = 4, x_max = 10, y_max = 8

        A set of lines (Defined by two corner coordinates)
        line 1 : x1 = 5, y1 = 5, x2 = 7, y2 = 7
        Line 2 : x1 = 7, y1 = 9, x2 = 11, y2 = 4
        Line 2 : x1 = 1, y1 = 5, x2 = 4, y2 = 1

Output : Line 1 : Accepted from (5, 5) to (7, 7)
        Line 2 : Accepted from (7.8, 8) to (10, 5.25)
        Line 3 : Rejected
```

Cohen-Sutherland algorithm divides a two-dimensional space into 9 regions and then efficiently determines the lines and portions of lines that are inside the given rectangular area.

The algorithm can be outlined as follows:-

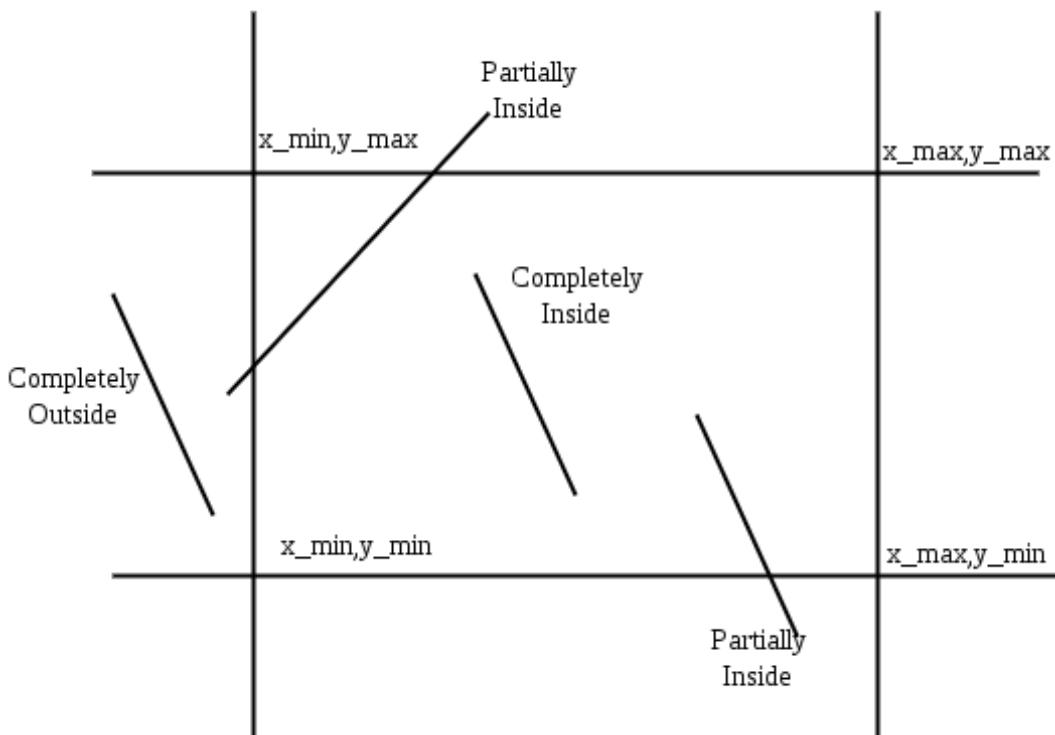
Nine regions are created, eight "outside" regions and one "inside" region.

For a given line extreme point (x, y), we can quickly find its region's four bit code. Four bit code can be computed by comparing x and y with four values ($x_{\min}, x_{\max}, y_{\min}$ and y_{\max}).

If x is less than x_{\min} then bit number 1 is set.
If x is greater than x_{\max} then bit number 2 is set.
If y is less than y_{\min} then bit number 3 is set.
If y is greater than y_{\max} then bit number 4 is set

There are three possible cases for any given line.

1. **Completely inside the given rectangle :** Bitwise OR of region of two end points of line is 0 (Both points are inside the rectangle)
2. **Completely outside the given rectangle :** Both endpoints share at least one outside region which implies that the line does not cross the visible region. (bitwise AND of endpoints != 0).
3. **Partially inside the window :** Both endpoints are in different regions. In this case, the algorithm finds one of the two points that is outside the rectangular region. The intersection of the line from outside point and rectangular window becomes new corner point and the algorithm repeats



Pseudo Code:

```

Step 1 : Assign a region code for two endpoints of given line.
Step 2 : If both endpoints have a region code 0000
         then given line is completely inside.
Step 3 : Else, perform the logical AND operation for both region codes.
        Step 3.1 : If the result is not 0000, then given line is completely
                   outside.
        Step 3.2 : Else line is partially inside.
                    Step 3.2.1 : Choose an endpoint of the line
                                  that is outside the given rectangle.
                    Step 3.2.2 : Find the intersection point of the
                                  rectangular boundary (based on region code).
                    Step 3.2.3 : Replace endpoint with the intersection point
                                  and update the region code.
                    Step 3.2.4 : Repeat step 2 until we find a clipped line either
                                  trivially accepted or trivially rejected.
Step 4 : Repeat step 1 for other lines
    
```

Below is implementation of above steps.

C++

```
// C++ program to implement Cohen Sutherland algorithm
```

```

// for line clipping.
#include <iostream>
using namespace std;

// Defining region codes
const int INSIDE = 0; // 0000
const int LEFT = 1; // 0001
const int RIGHT = 2; // 0010
const int BOTTOM = 4; // 0100
const int TOP = 8; // 1000

// Defining x_max, y_max and x_min, y_min for
// clipping rectangle. Since diagonal points are
// enough to define a rectangle
const int x_max = 10;
const int y_max = 8;
const int x_min = 4;
const int y_min = 4;

// Function to compute region code for a point(x, y)
int computeCode(double x, double y)
{
    // initialized as being inside
    int code = INSIDE;

    if (x < x_min) // to the left of rectangle
        code |= LEFT;
    else if (x > x_max) // to the right of rectangle
        code |= RIGHT;
    if (y < y_min) // below the rectangle
        code |= BOTTOM;
    else if (y > y_max) // above the rectangle
        code |= TOP;

    return code;
}

// Implementing Cohen-Sutherland algorithm
// Clipping a line from P1 = (x2, y2) to P2 = (x2, y2)
void cohenSutherlandClip(double x1, double y1,
                         double x2, double y2)
{
    // Compute region codes for P1, P2
    int code1 = computeCode(x1, y1);
    int code2 = computeCode(x2, y2);

    // Initialize line as outside the rectangular window
    bool accept = false;

```

```
while (true)
{
    if ((code1 == 0) && (code2 == 0))
    {
        // If both endpoints lie within rectangle
        accept = true;
        break;
    }
    else if (code1 & code2)
    {
        // If both endpoints are outside rectangle,
        // in same region
        break;
    }
    else
    {
        // Some segment of line lies within the
        // rectangle
        int code_out;
        double x, y;

        // At least one endpoint is outside the
        // rectangle, pick it.
        if (code1 != 0)
            code_out = code1;
        else
            code_out = code2;

        // Find intersection point;
        // using formulas y = y1 + slope * (x - x1),
        // x = x1 + (1 / slope) * (y - y1)
        if (code_out & TOP)
        {
            // point is above the clip rectangle
            x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1);
            y = y_max;
        }
        else if (code_out & BOTTOM)
        {
            // point is below the rectangle
            x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1);
            y = y_min;
        }
        else if (code_out & RIGHT)
        {
            // point is to the right of rectangle
            y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1);
        }
    }
}
```

```

        x = x_max;
    }
    else if (code_out & LEFT)
    {
        // point is to the left of rectangle
        y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1);
        x = x_min;
    }

    // Now intersection point x,y is found
    // We replace point outside rectangle
    // by intersection point
    if (code_out == code1)
    {
        x1 = x;
        y1 = y;
        code1 = computeCode(x1, y1);
    }
    else
    {
        x2 = x;
        y2 = y;
        code2 = computeCode(x2, y2);
    }
}

if (accept)
{
    cout <<"Line accepted from " << x1 << ", "
        << y1 << " to "<< x2 << ", " << y2 << endl;
    // Here the user can add code to display the rectangle
    // along with the accepted (portion of) lines
}
else
    cout << "Line rejected" << endl;
}

// Driver code
int main()
{
    // First Line segment
    // P11 = (5, 5), P12 = (7, 7)
    cohenSutherlandClip(5, 5, 7, 7);

    // Second Line segment
    // P21 = (7, 9), P22 = (11, 4)
    cohenSutherlandClip(7, 9, 11, 4);
}

```

```
// Third Line segment  
// P31 = (1, 5), P32 = (4, 1)  
cohenSutherlandClip(1, 5, 4, 1);  
  
    return 0;  
}
```

Python

```
# Python program to implement Cohen Sutherland algorithm  
# for line clipping.  
  
# Defining region codes  
INSIDE = 0  #0000  
LEFT = 1    #0001  
RIGHT = 2   #0010  
BOTTOM = 4  #0100  
TOP = 8    #1000  
  
# Defining x_max,y_max and x_min,y_min for rectangle  
# Since diagonal points are enough to define a rectangle  
x_max = 10.0  
y_max = 8.0  
x_min = 4.0  
y_min = 4.0  
  
# Function to compute region code for a point(x,y)  
def computeCode(x, y):  
    code = INSIDE  
    if x < x_min:      # to the left of rectangle  
        code |= LEFT  
    elif x > x_max:    # to the right of rectangle  
        code |= RIGHT  
    if y < y_min:      # below the rectangle  
        code |= BOTTOM  
    elif y > y_max:    # above the rectangle  
        code |= TOP  
  
    return code  
  
# Implementing Cohen-Sutherland algorithm  
# Clipping a line from P1 = (x1, y1) to P2 = (x2, y2)  
def cohenSutherlandClip(x1, y1, x2, y2):  
  
    # Compute region codes for P1, P2  
    code1 = computeCode(x1, y1)
```

```
code2 = computeCode(x2, y2)
accept = False

while True:

    # If both endpoints lie within rectangle
    if code1 == 0 and code2 == 0:
        accept = True
        break

    # If both endpoints are outside rectangle
    elif (code1 & code2) != 0:
        break

    # Some segment lies within the rectangle
    else:

        # Line Needs clipping
        # At least one of the points is outside,
        # select it
        x = 1.0
        y = 1.0
        if code1 != 0:
            code_out = code1
        else:
            code_out = code2

        # Find intersection point
        # using formulas y = y1 + slope * (x - x1),
        # x = x1 + (1 / slope) * (y - y1)
        if code_out & TOP:

            # point is above the clip rectangle
            x = x1 + (x2 - x1) * \
                (y_max - y1) / (y2 - y1)
            y = y_max

        elif code_out & BOTTOM:

            # point is below the clip rectangle
            x = x1 + (x2 - x1) * \
                (y_min - y1) / (y2 - y1)
            y = y_min

        elif code_out & RIGHT:

            # point is to the right of the clip rectangle
            y = y1 + (y2 - y1) * \
```

```
(x_max - x1) / (x2 - x1)
x = x_max

elif code_out & LEFT:

    # point is to the left of the clip rectangle
    y = y1 + (y2 - y1) * \
        (x_min - x1) / (x2 - x1)
    x = x_min

    # Now intersection point x,y is found
    # We replace point outside clipping rectangle
    # by intersection point
    if code_out == code1:
        x1 = x
        y1 = y
        code1 = computeCode(x1,y1)

    else:
        x2 = x
        y2 = y
        code2 = computeCode(x2, y2)

if accept:
    print ("Line accepted from %.2f,%.2f to %.2f,%.2f" % (x1,y1,x2,y2))

    # Here the user can add code to display the rectangle
    # along with the accepted (portion of) lines

else:
    print("Line rejected")

# Driver Script
# First Line segment
# P11 = (5, 5), P12 = (7, 7)
cohenSutherlandClip(5, 5, 7, 7)

# Second Line segment
# P21 = (7, 9), P22 = (11, 4)
cohenSutherlandClip(7, 9, 11, 4)

# Third Line segment
# P31 = (1, 5), P32 = (4, 1)
cohenSutherlandClip(1, 5, 4, 1)
```

Output:

```
Line accepted from 5.00,5.00 to 7.00,7.00
Line accepted from 7.80,8.00 to 10.00,5.25
Line rejected
```

The Cohen–Sutherland algorithm can be used only on a rectangular clip window. For other convex polygon clipping windows, Cyrus–Beck algorithm is used. We will be discussing Cyrus–Beck Algorithm in next set.

Related Post :

[Polygon Clipping | Sutherland–Hodgman Algorithm](#)
[Point Clipping Algorithm in Computer Graphics](#)

Reference:

https://en.wikipedia.org/wiki/Cohen–Sutherland_algorithm

Source

<https://www.geeksforgeeks.org/line-clipping-set-1-cohen-sutherland-algorithm/>

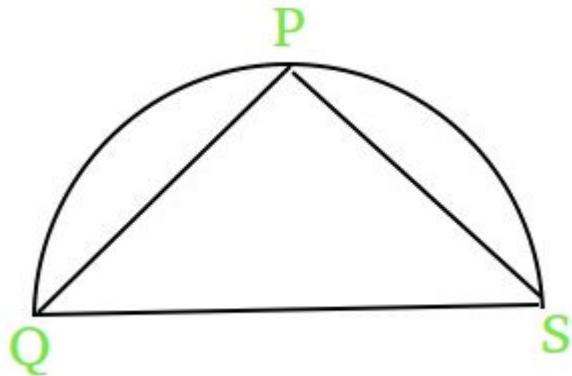
Chapter 110

Maximize a value for a semicircle of given radius

Maximize a value for a semicircle of given radius - GeeksforGeeks

We are given a **semi circle** with radius **R**. We can take any point on the circumference let it be **P**. Now, from that point **P** draw two lines to the two sides of diameter. Let the lines be **PQ** and **PS**.

The task is to find the maximum value of expression $\mathbf{PS}^2 + \mathbf{PQ}^2$ for a given **R**.



Examples :

Input : R = 1
Output : 4.25
 $(4*1^2 + 0.25) = 4.25$

Input : R = 2
Output : 16.25

$$(4 * 2^2 + 0.25) = 16.25$$

Let $F = PS^2 + PQ$. We know $QS = 2R$ (Diameter of the semicircle)

-> We also know that triangle PQS will always be right angled triangle irrespective of the position of point P on the circumference of circle

1.) $QS^2 = PQ^2 + PS^2$ (Pythagorean Theorem)

2.) Adding and Subtracting PQ on the RHS

$$QS^2 = PQ^2 + PS^2 + PQ - PQ$$

3.) Since $QS = 2R$

$$4*R^2 + PQ - PQ^2 = PS^2 + PQ$$

$$\Rightarrow 4*R^2 + PQ - PQ^2 = F$$

4.) Using the concept of maxima and minima

differentiating F with respect to PQ and equating it to 0 to get the point of maxima for F i.e.

$$1 - 2 * PQ = 0$$

$$\Rightarrow PQ = 0.5$$

5.) Now F will be maximum at $F = 4*R^2 + 0.25$

C++

```
// C++ program to find
// the maximum value of F
#include <bits/stdc++.h>
using namespace std;

// Function to find the
// maximum value of F
double maximumValueOfF (int R)
{
    // using the formula derived for
    // getting the maximum value of F
    return 4 * R * R + 0.25;
}

// Drivers code
int main()
{
    int R = 3;
    printf("%.2f", maximumValueOfF(R));
    return 0;
}
```

JAVA

```
// JAVA program to find
// the maximum value of F
import java.io.*;

class GFG
{

    // Function to find the
    // maximum value of F
    static double maximumValueOfF (int R)
    {

        // using the formula derived for
        // getting the maximum value of F
        return 4 * R * R + 0.25;
    }

    // Driver code
    public static void main (String[] args)
    {
        int R = 3;
        System.out.println(maximumValueOfF(R));
    }
}

// This code is contributed
// by anuj_67.
```

Python3

```
# python program to find
# the maximum value of F

# Function to find the
# maximum value of F
def maximumValueOfF (R):

    # using the formula derived for
    # getting the maximum value of F
    return 4 * R * R + 0.25


# Drivers code
R = 3
```

```
print(maximumValueOfF(R))  
# This code is contributed by Sam007.
```

C#

```
// C# program to find the  
// maximum value of F  
using System;  
  
class GFG  
{  
  
    // Function to find the  
    // maximum value of F  
    static double maximumValueOfF (int R)  
    {  
  
        // using the formula derived for  
        // getting the maximum value of F  
        return 4 * R * R + 0.25;  
    }  
  
    // Driver code  
    public static void Main ()  
    {  
        int R = 3;  
        Console.WriteLine(maximumValueOfF(R));  
    }  
  
}  
  
// This code is contributed by Sam007.
```

PHP

```
<?php  
// PHP program to find the  
// maximum value of F  
  
// Function to find the  
// maximum value of F  
function maximumValueOfF ($R)  
{  
  
    // using the formula derived  
    // for getting the maximum
```

```
// value of F  
return 4 * $R * $R + 0.25;  
}  
  
// Drivers code  
$R = 3;  
echo maximumValueOfF($R);  
  
// This code is contributed  
// by anuj_67.  
?>
```

Output :

36.25

Improved By : [vt_m](#), [Sam007](#)

Source

<https://www.geeksforgeeks.org/maximize-a-value-for-a-semicircle-of-given-radius/>

Chapter 111

Maximize volume of cuboid with given sum of sides

Maximize volume of cuboid with given sum of sides - GeeksforGeeks

We are given the sum of **length**, **breadth** and **height**, say S, of a cuboid. The task is to find the maximum volume that can be achieved so that sum of side is S.

Volume of a cuboid = length * breadth * height

Examples :

```
Input : s = 4
Output : 2
Only possible dimensions are some combination of 1, 1, 2.
```

```
Input : s = 8
Output : 18
All possible edge dimensions:
[1, 1, 6], volume = 6
[1, 2, 5], volume = 10
[1, 3, 4], volume = 12
[2, 2, 4], volume = 16
[2, 3, 3], volume = 18
```

Method 1: (Brute Force)

The idea is to run three nested loops, one for length, one for breadth and one for height. For each iteration, calculate the volume and compare with maximum volume.

Below is the implementation of this approach:

C++

```
#include <bits/stdc++.h>
using namespace std;

// Return the maximum volume.
int maxvolume(int s)
{
    int maxvalue = 0;

    // for length
    for (int i = 1; i <= s - 2; i++) {

        // for breadth
        for (int j = 1; j <= s - 1; j++) {

            // for height
            int k = s - i - j;

            // calculating maximum volume.
            maxvalue = max(maxvalue, i * j * k);
        }
    }

    return maxvalue;
}

// Driven Program
int main()
{
    int s = 8;
    cout << maxvolume(s) << endl;
    return 0;
}
```

Java

```
// Java code to Maximize volume of
// cuboid with given sum of sides

class GFG
{

    // Return the maximum volume.
    static int maxvolume(int s)
    {
        int maxvalue = 0;

        // for length
        for (int i = 1; i <= s - 2; i++)
```

```
{\n\n    // for breadth\n    for (int j = 1; j <= s - 1; j++){\n\n        // for height\n        int k = s - i - j;\n\n        // calculating maximum volume.\n        maxvalue = Math.max(maxvalue, i * j * k);\n    }\n}\n\nreturn maxvalue;\n}\n// Driver function\npublic static void main (String[] args)\n{\n    int s = 8;\n    System.out.println(maxvolume(s));\n}\n}\n\n// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 code to Maximize volume of\n# cuboid with given sum of sides\n\n# Return the maximum volume.\ndef maxvolume (s):\n    maxvalue = 0\n\n    # for length\n    i = 1\n    for i in range(s - 1):\n        j = 1\n\n        # for breadth\n        for j in range(s):\n\n            # for height\n            k = s - i - j\n\n            # calculating maximum volume.\n            maxvalue = max(maxvalue, i * j * k)
```

```
        return maxvalue

# Driven Program
s = 8
print(maxvolume(s))

# This code is contributed by "Sharad_Bhardwaj".

C#

// C# code to Maximize volume of
// cuboid with given sum of sides
using System;

class GFG
{

    // Return the maximum volume.
    static int maxvolume(int s)
    {
        int maxvalue = 0;

        // for length
        for (int i = 1; i <= s - 2; i++)
        {

            // for breadth
            for (int j = 1; j <= s - 1; j++)
            {

                // for height
                int k = s - i - j;

                // calculating maximum volume.
                maxvalue = Math.Max(maxvalue, i * j * k);
            }
        }

        return maxvalue;
    }

    // Driver function
    public static void Main ()
    {
        int s = 8;
        Console.WriteLine(maxvolume(s));
    }
}
```

```
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP code to Maximize volume of
// cuboid with given sum of sides

// Return the maximum volume.
function maxvolume( $s)
{
    $maxvalue = 0;

    // for length
    for ( $i = 1; $i <= $s - 2; $i++)
    {

        // for breadth
        for ( $j = 1; $j <= $s - 1; $j++)
        {

            // for height
            $k = $s - $i - $j;

            // calculating maximum volume.
            $maxvalue = max($maxvalue,
                            $i * $j * $k);
        }
    }

    return $maxvalue;
}

// Driver Code
$s = 8;
echo(maxvolume($s));

// This code is contributed by vt_m.
?>
```

Output :

Time Complexity: $O(n^2)$

Method 2: (Efficient approach)

The idea is to divide edges as equally as possible.

So,

$$\text{length} = \text{floor}(s/3)$$

$$\text{width} = \text{floor}((s - \text{length})/2) = \text{floor}((s - \text{floor}(s/3))/2)$$

$$\text{height} = s - \text{length} - \text{width} = s - \text{floor}(s/3) - \text{floor}((s - \text{floor}(s/3))/2)$$

Below is the implementation of this approach:

C++

```
#include <bits/stdc++.h>
using namespace std;

// Return the maximum volume.
int maxvolume(int s)
{
    // finding length
    int length = s / 3;

    s -= length;

    // finding breadth
    int breadth = s / 2;

    // finding height
    int height = s - breadth;

    return length * breadth * height;
}
```

```
// Driven Program
int main()
{
    int s = 8;
    cout << maxvolume(s) << endl;
    return 0;
}
```

Java

```
// Java code to Maximize volume of
// cuboid with given sum of sides
import java.io.*;
```

```
class GFG
{
    // Return the maximum volume.
    static int maxvolume(int s)
    {
        // finding length
        int length = s / 3;

        s -= length;

        // finding breadth
        int breadth = s / 2;

        // finding height
        int height = s - breadth;

        return length * breadth * height;
    }

    // Driven Program
    public static void main (String[] args)
    {
        int s = 8;
        System.out.println ( maxvolume(s));

    }
}

// This code is contributed by vt_m.
```

Python3

```
# Python3 code to Maximize volume of
# cuboid with given sum of sides

# Return the maximum volume.
def maxvolume( s ):

    # finding length
    length = int(s / 3)

    s -= length

    # finding breadth
    breadth = s / 2

    # finding height
```

```
height = s - breadth

return int(length * breadth * height)

# Driven Program
s = 8
print( maxvolume(s) )

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# code to Maximize volume of
// cuboid with given sum of sides
using System;

class GFG
{
    // Return the maximum volume.
    static int maxvolume(int s)
    {
        // finding length
        int length = s / 3;

        s -= length;

        // finding breadth
        int breadth = s / 2;

        // finding height
        int height = s - breadth;

        return length * breadth * height;
    }

    // Driven Program
    public static void Main ()
    {
        int s = 8;
        Console.WriteLine( maxvolume(s));

    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// Return the maximum volume.
function maxvolume($s)
{
    // finding length
    $length = (int)($s / 3);

    $s -= $length;

    // finding breadth
    $breadth = (int)($s / 2);

    // finding height
    $height = $s - $breadth;

    return $length * $breadth * $height;
}

// Driven Code
$s = 8;
echo(maxvolume($s));

// This code is contributed by Ajit.
?>
```

Output :

18

Time Complexity: O(1)

How does this work?

We basically need to maximize product of three numbers, x, y and z whose sum is given.

$$\begin{aligned} \text{Given } s &= x + y + z \\ \text{Maximize } P &= x * y * z \\ &= x * y * (s - x - y) \\ &= x^2y^2s - x^3y - x^2y^2 \end{aligned}$$

We get $dp/dx = sy - 2xy - y^2$
and $dp/dy = sx - 2xy - x^2$

We get $dp/dx = 0$ and $dp/dy = 0$ when
 $x = s/3$, $y = s/3$

So $z = s - x - y = s/3$

Improved By : [vt_m, jit_t](#)

Source

<https://www.geeksforgeeks.org/maximize-volume-cuboid-given-sum-sides/>

Chapter 112

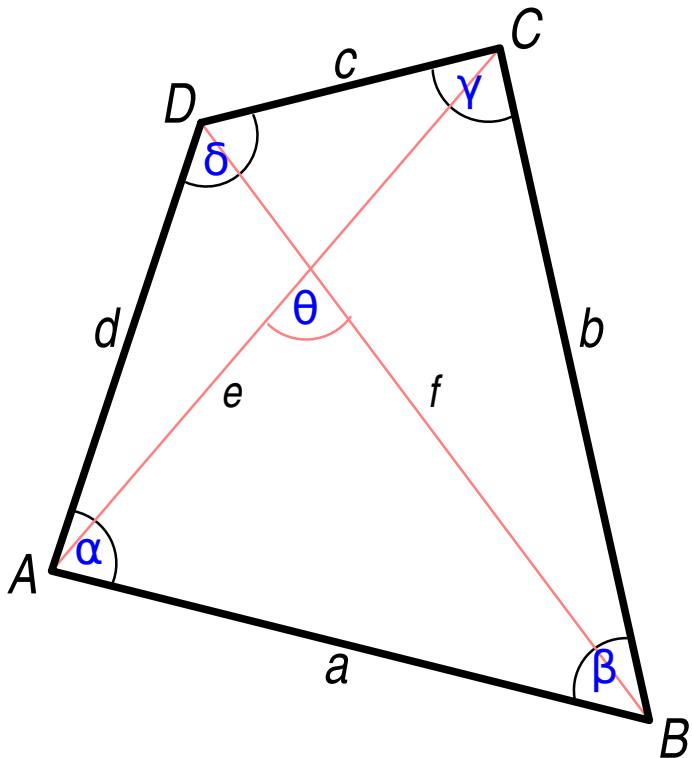
Maximum area of quadrilateral

Maximum area of quadrilateral - GeeksforGeeks

Given four sides of quadrilateral a, b, c, d, find the maximum area of the quadrilateral possible from the given sides .

Examples:

```
Input : 1 2 1 2
Output : 2.00
It is optimal to construct a rectangle for maximum area .
```



According to **Bretschneider's formula**, the area of a general quadrilateral is given by

$$K = \sqrt{(s-a)(s-b)(s-c)(s-d) - abcd \cos^2 \theta}$$

Here a, b, c, d are the sides of a quadrilateral, s is the semiperimeter of a quadrilateral and angles are two opposite angles.

So, this formula is maximized only when opposite angles sum to $\pi(180)$ then we can use a simplified form of Bretschneider's formula to get the (maximum) area K .

$$K = \sqrt{(s-a)(s-b)(s-c)(s-d)}$$

This formula is called as **Brahmagupta's formula**.

Below is the implementation of given approach

C++

```
// CPP program to find maximum are of a
// quadrilateral
#include <bits/stdc++.h>
using namespace std;

double maxArea(double a, double b,
               double c, double d)
```

```
{  
    // Calculating the semi-perimeter  
    // of the given quadrilateral  
    double semiperimeter = (a + b + c + d) / 2;  
  
    // Applying Brahmagupta's formula to  
    // get maximum area of quadrilateral  
    return sqrt((semiperimeter - a) *  
               (semiperimeter - b) *  
               (semiperimeter - c) *  
               (semiperimeter - d));  
}  
  
// Driver code  
int main()  
{  
    double a = 1, b = 2, c = 1, d = 2;  
    printf("%.2f\n", maxArea(a, b, c, d));  
    return 0;  
}
```

Java

```
// Java program to find maximum are of a  
// quadrilateral  
import java.io.*;  
  
class GFG  
{  
    static double maxArea(double a, double b,  
                          double c, double d)  
    {  
        // Calculating the semi-perimeter  
        // of the given quadrilateral  
        double semiperimeter = (a + b + c + d) / 2;  
  
        // Applying Brahmagupta's formula to  
        // get maximum area of quadrilateral  
        return Math.sqrt((semiperimeter - a) *  
                       (semiperimeter - b) *  
                       (semiperimeter - c) *  
                       (semiperimeter - d));  
    }  
  
    // Driver code  
    public static void main (String[] args)  
    {  
        double a = 1, b = 2, c = 1, d = 2;
```

```
        System.out.println(maxArea(a, b, c, d));
    }
}

// This code is contributed by sunnysingh
```

Python3

```
# Python3 program to find maximum
# area of a quadrilateral
import math

def maxArea (a , b , c , d ):

    # Calculating the semi-perimeter
    # of the given quadrilateral
    semiperimeter = (a + b + c + d) / 2

    # Applying Brahmagupta's formula to
    # get maximum area of quadrilateral
    return math.sqrt((semiperimeter - a) *
                    (semiperimeter - b) *
                    (semiperimeter - c) *
                    (semiperimeter - d))

# Driver code
a = 1
b = 2
c = 1
d = 2
print("%.2f"%maxArea(a, b, c, d))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# program to find maximum area of a
// quadrilateral
using System;

class GFG {

    static double maxArea(double a, double b,
                          double c, double d)
    {

        // Calculating the semi-perimeter
```

```
// of the given quadrilateral
double semiperimeter = (a + b + c + d) / 2;

// Applying Brahmagupta's formula to
// get maximum area of quadrilateral
return Math.Sqrt((semiperimeter - a) *
                 (semiperimeter - b) *
                 (semiperimeter - c) *
                 (semiperimeter - d));
}

// Driver code
public static void Main ()
{
    double a = 1, b = 2, c = 1, d = 2;

    Console.WriteLine(maxArea(a, b, c, d));
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find maximum area of a
// quadrilateral

function maxArea( $a, $b, $c, $d)
{

    // Calculating the semi-perimeter
    // of the given quadrilateral
    $semiperimeter = ($a + $b + $c + $d) / 2;

    // Applying Brahmagupta's formula to
    // get maximum area of quadrilateral
    return sqrt(($semiperimeter - $a) *
                ($semiperimeter - $b) *
                ($semiperimeter - $c) *
                ($semiperimeter - $d));
}

// Driver code
$a = 1; $b = 2; $c = 1; $d = 2;
echo(maxArea($a, $b, $c, $d));

// This code is contributed by vt_m.
```

?>

Output:

2.00

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/maximum-area-quadrilateral/>

Chapter 113

Maximum distinct lines passing through a single point

Maximum distinct lines passing through a single point - GeeksforGeeks

Given N lines represented by two points (x_1, y_1) and (x_2, y_2) . The task is to find maximum number of lines which can pass through a single point, without superimposing (or covering) any other line. We can move any line but not rotate it.

Examples:

```
Input : Line 1 : x1 = 1, y1 = 1, x2 = 2, y2 = 2
        Line 2 : x2 = 2, y1 = 2, x2 = 4, y2 = 10
Output : 2
There are two lines. These two lines are not
parallel, so both of them will pass through
a single point.
```

```
Input : Line 1 : x1 = 1, y1 = 5, x2 = 1, y2 = 10
        Line 2 : x2 = 5, y1 = 1, x2 = 10, y2 = 1
Output : 2
```

- Represent lines as pair (m, c) where line can be given as $y = mx + c$, called line slope form. We can now see that we can change the c for any line, but cannot modify m .
- Lines having same value of m parallel, given that $(c1 \neq c2)$. Also no two parallel lines can pass through same point without superimposing to each other.
- So, our problem reduces to finding different values of slopes from given set of lines.

$$\left\{ \frac{y_2 - y_1}{x_2 - x_1} \right\}$$

We can calculate slope of a line as $\left\{ \frac{y_2 - y_1}{x_2 - x_1} \right\}$, add them to a set and count the number of distinct values of slope in set. But we have to handle vertical lines separately.

So, if $x_1 = x_2$ then, slope = INT_MAX.

$$\left\{ \frac{y_2 - y_1}{x_2 - x_1} \right\}$$

Otherwise, slope = $\left\{ \frac{y_2 - y_1}{x_2 - x_1} \right\}$.

Below is the implementation of the approach.

```
// C++ program to find maximum number of lines
// which can pass through a single point
#include <bits/stdc++.h>
using namespace std;

// function to find maximum lines which passes
// through a single point
int maxLines(int n, int x1[], int y1[], int x2[], int y2[])
{
    unordered_set<double> s;

    double slope;
    for (int i = 0; i < n; ++i) {
        if (x1[i] == x2[i])
            slope = INT_MAX;
        else
            slope = (y2[i] - y1[i]) * 1.0 / (x2[i] - x1[i]) * 1.0;

        s.insert(slope);
    }

    return s.size();
}

// Driver program
int main()
{
    int n = 2, x1[] = { 1, 2 }, y1[] = { 1, 2 },
        x2[] = { 2, 4 }, y2[] = { 2, 10 };
    cout << maxLines(n, x1, y1, x2, y2);
    return 0;
}
// This code is written by
// Sanjit_Prasad
```

Output:

2

Time Complexity: $\mathcal{O}(N^2)$

Source

<https://www.geeksforgeeks.org/maximum-distinct-lines-passing-through-a-single-point/>

Chapter 114

Maximum height when coins are arranged in a triangle

Maximum height when coins are arranged in a triangle - GeeksforGeeks

We have N coins which need to arrange in form of a triangle, i.e. first row will have 1 coin, second row will have 2 coins and so on, we need to tell maximum height which we can achieve by using these N coins.

Examples:

```
Input : N = 7
Output : 3
Maximum height will be 3, putting 1, 2 and
then 3 coins. It is not possible to use 1
coin left.
```

```
Input : N = 12
Output : 4
Maximum height will be 4, putting 1, 2, 3 and
4 coins, it is not possible to make height as 5,
because that will require 15 coins.
```

This problem can be solved by finding a relation between height of the triangle and number of coins. Let maximum height is H, then total sum of coin should be less than N,

```
Sum of coins for height H <= N
H*(H + 1)/2 <= N
H*H + H - 2*N <= 0
Now by Quadratic formula
```

(ignoring negative root)

Maximum H can be $(-1 + \sqrt{1 + 8N}) / 2$

Now we just need to find the square root of $(1 + 8N)$ for which we can use Babylonian method of finding square root

Below code is implemented on above stated concept,

CPP

```
// C++ program to find maximum height of arranged
// coin triangle
#include <bits/stdc++.h>
using namespace std;

/* Returns the square root of n. Note that the function */
float squareRoot(float n)
{
    /* We are using n itself as initial approximation
       This can definitely be improved */
    float x = n;
    float y = 1;

    float e = 0.000001; /* e decides the accuracy level*/
    while (x - y > e)
    {
        x = (x + y) / 2;
        y = n/x;
    }
    return x;
}

// Method to find maximum height of arrangement of coins
int findMaximumHeight(int N)
{
    // calculating portion inside the square root
    int n = 1 + 8*N;
    int maxH = (-1 + squareRoot(n)) / 2;
    return maxH;
}

// Driver code to test above method
int main()
{
    int N = 12;
    cout << findMaximumHeight(N) << endl;
```

```
    return 0;
}
```

Java

```
// Java program to find maximum height
// of arranged coin triangle
class GFG
{

    /* Returns the square root of n.
    Note that the function */
    static float squareRoot(float n)
    {

        /* We are using n itself as
        initial approximation. This
        can definitely be improved */
        float x = n;
        float y = 1;

        // e decides the accuracy level
        float e = 0.000001f;
        while (x - y > e)
        {
            x = (x + y) / 2;
            y = n / x;
        }

        return x;
    }

    // Method to find maximum height
    // of arrangement of coins
    static int findMaximumHeight(int N)
    {

        // calculating portion inside
        // the square root
        int n = 1 + 8*N;
        int maxH = (int)(-1 + squareRoot(n)) / 2;

        return maxH;
    }

    // Driver code
    public static void main (String[] args)
    {
```

```
int N = 12;

System.out.print(findMaximumHeight(N));
}

}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 program to find
# maximum height of arranged
# coin triangle

# Returns the square root of n.
# Note that the function
def squareRoot(n):

    # We are using n itself as
    # initial approximation
    # This can definitely be improved
    x = n
    y = 1

    e = 0.000001 # e decides the accuracy level
    while (x - y > e):
        x = (x + y) / 2
        y = n/x

    return x

# Method to find maximum height
# of arrangement of coins
def findMaximumHeight(N):

    # calculating portion inside the square root
    n = 1 + 8*N
    maxH = (-1 + squareRoot(n)) / 2
    return int(maxH)

# Driver code to test above method
N = 12
print(findMaximumHeight(N))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
// C# program to find maximum height
// of arranged coin triangle
using System;

class GFG
{
    /* Returns the square root of n.
    Note that the function */
    static float squareRoot(float n)
    {
        /* We are using n itself as
        initial approximation. This
        can definitely be improved */
        float x = n;
        float y = 1;

        // e decides the accuracy level
        float e = 0.000001f;
        while (x - y > e)
        {
            x = (x + y) / 2;
            y = n / x;
        }
        return x;
    }

    static int findMaximumHeight(int N)
    {

        // calculating portion inside
        // the square root
        int n = 1 + 8*N;
        int maxH = (int)(-1 + squareRoot(n)) / 2;

        return maxH;
    }

    /* program to test above function */
    public static void Main()
    {
        int N = 12;
        Console.Write(findMaximumHeight(N));
    }
}

// This code is contributed by _omg
```

PHP

```
<?php
// PHP program to find maximum height
// of arranged coin triangle

/* Returns the square root of n. Note
that the function */
function squareRoot( $n)
{
    /* We are using n itself as initial
approximation This can definitely
be improved */
    $x = $n;
    $y = 1;

    /* e decides the accuracy level*/
    $e = 0.000001;
    while ($x - $y > $e)
    {
        $x = ($x + $y) / 2;
        $y = $n/$x;
    }
    return $x;
}

// Method to find maximum height of
// arrangement of coins
function findMaximumHeight( $N)
{
    // calculating portion inside
    // the square root
    $n = 1 + 8 * $N;
    $maxH = (-1 + squareRoot($n)) / 2;
    return floor($maxH);
}

// Driver code to test above method
$N = 12;
echo findMaximumHeight($N) ;

// This code is contributed by anuj_67.
?>
```

Output:

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/maximum-height-coins-arranged-triangle/>

Chapter 115

Maximum integral co-ordinates with non-integer distances

Maximum integral co-ordinates with non-integer distances - GeeksforGeeks

Given a maximum limit of x – coordinate and y – coordinate, we want to calculate a set of coordinates such that the distance between any two points is a non-integer number. The coordinates (i, j) chosen should be of range $0 \leq i \leq x$ and $0 \leq j \leq y$. Also, we have to maximize the set. Examples:

```
Input : 4 4
Output : 0 4
         1 3
         2 2
         3 1
         4 0
```

Explanation : Distance between any two points mentioned in output is not integer.

Firstly, we want to create a set, that means our set cannot contain any other point with same x's or y's which are used before. Well, the reason behind it is that such points which either have same x-coordinate or y-coordinate would cancel that coordinate, resulting an integral distance between them.

Example, consider points (1, 4) and (1, 5), the x-coordinate would cancel and thus, we will get an integral distance.

Secondly, we can observe that, we have only $x+1$ distinct i-coordinates and $y+1$ distinct j-coordinates. Thus, the size of the set cannot exceed $\min(x, y)+1$.

Third observation is that we know that the diagonal elements are $|i-j| * \sqrt{2}$ distance apart, thus, we take evaluate along the diagonal element of i-coordinate and calculate the j-coordinate by formula $\min(i, j)-i$.

```
// C++ program to find maximum integral points
// such that distances between any two is not
// integer.
#include <bits/stdc++.h>
using namespace std;

// Making set of coordinates such that
// any two points are non-integral distance apart
void printSet(int x, int y)
{
    // used to avoid duplicates in result
    set<pair<int, int> > arr;

    for (int i = 0; i <= min(x, y); i++) {

        pair<int, int> pq;
        pq = make_pair(i, min(x, y) - i);
        arr.insert(pq);
    }

    for (auto it = arr.begin(); it != arr.end(); it++)
        cout << (*it).first << " " << (*it).second << endl;
}

// Driver function
int main()
{
    int x = 4, y = 4;
    printSet(x, y);
    return 0;
}
```

Output:

```
0 4
1 3
2 2
3 1
4 0
```

Source

<https://www.geeksforgeeks.org/maximum-integral-co-ordinates-non-integer-distances/>

Chapter 116

Maximum number of 2×2 squares that can be fit inside a right isosceles triangle

Maximum number of 2×2 squares that can be fit inside a right isosceles triangle - GeeksforGeeks

What is the maximum number of squares of size 2×2 units that can be fit in a right angled isosceles triangle of a given base (in units).

A side of the square must be parallel to the base of the triangle.

Examples:

```
Input : 8  
Output : 6  
Please refer below diagram for explanation.
```

```
Input : 7  
Output : 3
```

Since the triangle is isosceles, the given base would also be equal to the height. Now in the diagonal part, we would always need an extra length of 2 units in both height and base of the triangle to accommodate a triangle. (The CF and AM segment of the triangle in the image. The part that does not contribute to any square). In the remaining length of base, we can construct $\text{length} / 2$ squares. Since each square is of 2 units, same would be the case of height, there is no need to calculate that again.

So, for each level of given length we can construct $(\text{length}-2)/2$ squares. This gives us a base of $(\text{length}-2)$ above it. Continuing this process to get the no of squares for all available $\text{length}-2$ height, we can calculate the squares.

```
while length > 2
    answer += (length - 2 )/2
    length = length - 2
```

For more effective way, we can use the formula of sum of AP $n * (n + 1) / 2$, where $n = \text{length} - 2$

C++

```
// C++ program to count number of 2 x 2
// squares in a right isosceles triangle
#include<bits/stdc++.h>
using namespace std;

int number0fSquares(int base)
{
    // removing the extra part we would
    // always need
    base = (base - 2);

    // Since each square has base of
    // length of 2
    base = base / 2;

    return base * (base + 1)/2;
}

// Driver code
int main()
{
    int base = 8;
    cout << number0fSquares(base);
    return 0;
}
```

Java

```
// Java program to count number of 2 x 2
// squares in a right isosceles triangle

class Squares
{
    public static  int number0fSquares(int base)
    {
        // removing the extra part
        // we would always need
```

```
base = (base - 2);

// Since each square has
// base of length of 2
base = base / 2;

return base * (base + 1)/2;
}

// Driver code
public static void main(String args[])
{
    int base = 8;
    System.out.println(numberOfSquares(base));
}
}

// This code is contributed by Anshika Goyal.
```

Python3

```
# Python3 program to count number
# of  $2 \times 2$  squares in a right
# isosceles triangle
def numberOfSquares(base):

    # removing the extra part we would
    # always need
    base = (base - 2)

    # Since each square has base of
    # length of 2
    base = base / 2

    return base * (base + 1) / 2

# Driver code
base = 8

print(numberOfSquares(base))

# This code is contributed by Anant Agarwal.
```

C#

```
// C# program to count number of  $2 \times 2$ 
```

```
// squares in a right isosceles triangle
using System;

class GFG {

    public static int numberOfSquares(int _base)
    {

        // removing the extra part
        // we would always need
        _base = (_base - 2);

        // Since each square has
        // base of length of 2
        _base = _base / 2;

        return _base * (_base + 1)/2;
    }

    // Driver code
    public static void Main()
    {

        int _base = 8;
        Console.WriteLine(numberOfSquares(_base));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP program to count number of  $2 \times 2$ 
// squares in a right isosceles triangle

function numberOfSquares( $base)
{

    // removing the extra
    // part we would
    // always need
    $base = ($base - 2);

    // Since each square
    // has base of
    // length of 2
    $base = $base / 2;
```

```
        return $base * ($base + 1)/2;
    }

// Driver code
$base = 8;
echo numberOfSquares($base);

// This code is contributed by anuj_67.
?>
```

Output:

6

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/maximum-number-2x2-squares-can-fit-inside-right-isosceles-triangle/>

Chapter 117

Maximum number of pieces in N cuts

Maximum number of pieces in N cuts - GeeksforGeeks

Given a square piece and a total number of cuts available n, Find out the maximum number of rectangular or square pieces of equal size that can be obtained with n cuts. The allowed cuts are horizontal and vertical cut.

Note: Stacking and folding is not allowed.

Examples:

Input : n = 1

Output : 2

Explanation :

Input : n = 2

Output : 4

Explanation :

Input : n = 3

Output : 6

Explanation :

Given is n which is the number of allowed cuts. As it is required to maximize number of pieces after n cuts, So number of horizontal cuts will be equal to number of vertical cuts. This can be prove using differentiation. So number of horizontal cut will be $n/2$. and vertical cuts will be $n-n/2$.

So number of pieces = (horizontal cut + 1) * (vertical cut + 1).

Program:

C++

```
// C++ program to find maximum no of pieces
// by given number of cuts
#include <bits/stdc++.h>
using namespace std;

// Function for finding maximum pieces
// with n cuts.
int findMaximumPieces(int n)
{
    // to maximize number of pieces
    // x is the horizontal cuts
    int x = n / 2;

    // Now (x) is the horizontal cuts
    // and (n-x) is vertical cuts, then
    // maximum number of pieces = (x+1)*(n-x+1)
    return ((x + 1) * (n - x + 1));
}

// Driver code
int main()
{
    // Taking the maximum number of cuts allowed as 3
    int n = 3;

    // Finding and printing the max number of pieces
    cout << "Max number of pieces for n = " << n
        << " is " << findMaximumPieces(3);

    return 0;
}
```

Java

```
// Java program to find maximum
// no of pieces by given number
// of cuts
import java.util.*;

class GFG
{
    // Function for finding maximum
```

```
// pieces with n cuts.  
public static int findMaximumPieces(int n)  
{  
    // to maximize number of pieces  
    // x is the horizontal cuts  
    int x = n / 2;  
  
    // Now (x) is the horizontal cuts  
    // and (n-x) is vertical cuts, then  
    // maximum number of pieces = (x+1)*(n-x+1)  
    return ((x + 1) * (n - x + 1));  
}  
  
// Driver code  
public static void main (String[] args)  
{  
    // Taking the maximum number  
    // of cuts allowed as 3  
    int n = 3;  
  
    // Finding and printing the  
    // max number of pieces  
    System.out.print("Max number of pieces for n = " +  
        n + " is " + findMaximumPieces(3));  
}  
}  
  
// This code is contributed by Kirti_Mangal
```

Output:

```
Max number of pieces for n = 3 is 6
```

Improved By : [Kirti_Mangal](#)

Source

<https://www.geeksforgeeks.org/maximum-number-of-pieces-in-n-cuts/>

Chapter 118

Maximum number of squares that can fit in a right angle isosceles triangle

Maximum number of squares that can fit in a right angle isosceles triangle - GeeksforGeeks

You are given an isosceles (a triangle with at-least two equal sides) right angle triangle with base b , we need to find the maximum number of squares of side m , which can be fitted into given triangle.

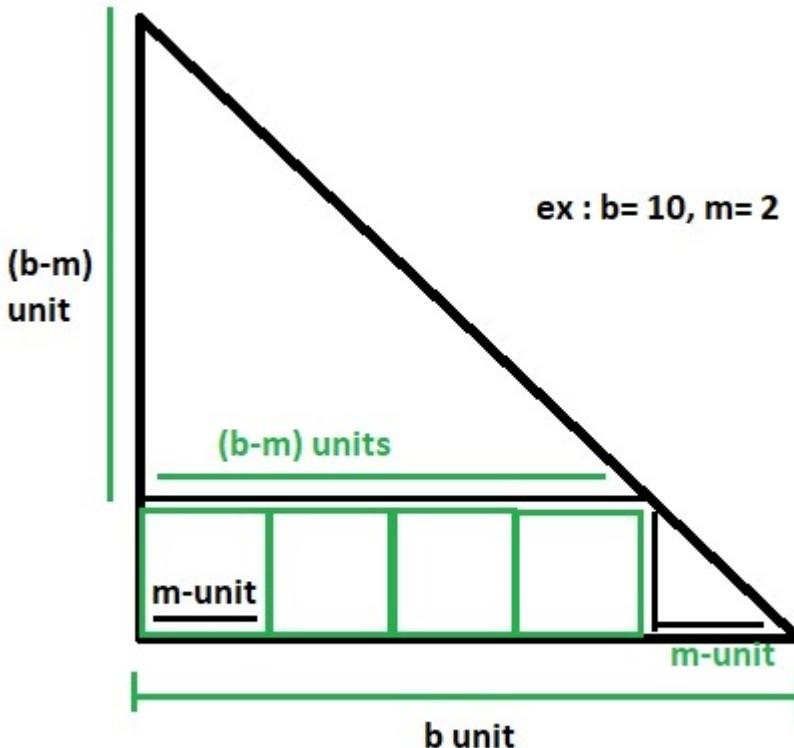
Examples:

Input : $b = 6$, $m = 2$
Output : 3

Input : $b = 4$, $m = 1$
Output : 6

Let's consider a right angle triangle XYZ, where YZ is the base of triangle. Suppose length of the base is b . If we consider the position of first square with the vertex Y, we will have $(b / m - 1)$ squares in the base, and we will be left with another isosceles right angle triangle having base length $(b - m)$.

Illustration :



Explanation of Maximum square inside triangle

Let $f(b, m)$ = Number of squares which can be fitted in triangle having base length b .

then $f(b, m) = (b / m - 1) + f(b - m, m)$

We can calculate $f(b)$ using above recursion, and with use of memoization. Later we can answer each query in $O(1)$ time. We can do it for even and odd numbers separately with the base case if $(b < 2 * m)$ $f(b, m) = 0$.

The given recursion can be solved as :

$$f(b, m) = b / m - 1 + f(b - m, m) = b / m - 1 + (b - m) / m - 1 + f(b - 2m, m)$$

$$f(b, m) = b / m - 1 + b / m - 2 + f(b - 3m, m) + \dots + f(b - (b / m)m, m)$$

$$f(b) = b / m - 1 + b / m - 2 + b / m - 3 + \dots + 1 + 0$$

With conditions, if $(b < 2 * m)$ $f(b, m) = 0$

$f(b)$ = sum of first $(b / m - 1)$ natural numbers

$$= (b / m - 1) * (b / m) / 2$$

This formula can be used to reduce the time complexity upto $O(1)$.

C++

```
// CPP program for finding maximum squares
// that can fit in right angle isosceles
// triangle
#include<bits/stdc++.h>
```

```
using namespace std;

// function for finding max squares
int maxSquare(int b, int m)
{
    // return in O(1) with derived
    // formula
    return (b / m - 1) * (b / m) / 2;
}

// driver program
int main()
{
    int b = 10, m = 2;
    cout << maxSquare (b,m);
    return 0;
}
```

Java

```
// Java program for finding maximum squares
// that can fit in right angle isosceles
// triangle
public class GFG
{
    // function for finding max squares
    static int maxSquare(int b, int m)
    {
        // return in O(1) with derived
        // formula
        return (b / m - 1) * (b / m) / 2;
    }

    // driver program
    public static void main(String args[])
    {
        int b = 10, m = 2;
        System.out.println(maxSquare (b,m));
    }
}

// This code is contribute by Sumit Ghosh
```

Python3

```
# Python3 program for
# finding maximum squares
```

```
# that can fit in
# right angle isosceles
# triangle

# function for finding max squares
def maxSquare(b, m):

    # return in O(1) with derived
    # formula
    return (b / m - 1) * (b / m) / 2

# driver program
b = 10
m = 2
print(int(maxSquare (b,m)))

# This code is contributed by
# Smitha Dinesh Semwal
```

C#

```
// C# program for finding maximum squares
// that can fit in right angle isosceles
// triangle
using System;

public class GFG
{
    // function for finding max squares
    static int maxSquare(int b, int m)
    {
        // return in O(1) with derived
        // formula
        return (b / m - 1) * (b / m) / 2;
    }

    // driver program
    public static void Main()
    {
        int b = 10, m = 2;
        Console.WriteLine(maxSquare (b, m));
    }
}

// This code is contribute by vt_m
```

PHP

```
<?php
// PHP program for finding
// maximum squares that can
// fit in right angle isosceles
// triangle

// function for finding
// max squares
function maxSquare($b, $m)
{

    // return in O(1) with
    // derived formula
    return ($b / $m - 1) *
           ($b / $m) / 2;
}

// Driver Code
$b = 10; $m = 2;
echo maxSquare($b,$m);
// This code is contribute by vt_m
?>
```

Output:

10

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/maximum-number-of-squares-that-can-be-fit-in-a-right-angle-isosceles-triangle/>

Chapter 119

Maximum points of intersection n circles

Maximum points of intersection n circles - GeeksforGeeks

Given a number n, we need to find the maximum number of times n circles intersect.

Examples:

Input : n = 2
Output : 2

Input : n = 3
Output : 6

As we can see in above diagram, for each pair of circles, there can be maximum two intersection points. Therefore if we have n circles then there can be nC_2 pairs of circles in which each pair will have two intersections. So by this we can conclude that by looking at all possible pairs of circles the mathematical formula can be made for the maximum number of intersection by n circles is given by $2 * {}^nC_2$.

$$2 * {}^nC_2 = 2 * n * (n - 1)/2 = n * (n-1)$$

C++

```
// CPP program to find maximum number of
// intersections of n circles
#include <bits/stdc++.h>
using namespace std;

// Returns maximum number of intersections
```

```
int intersection(int n)
{
    return n * (n - 1);
}

int main()
{
    cout << intersection(3) << endl;
    return 0;
}
// This code is contributed by
// Manish Kumar Rai.
```

Java

```
// Java program to find maximum umber of
// intersections of n circles
import java.io.*;

public class GFG {

    // for the calculation of 2*(nC2)
    static int intersection(int n)
    {
        return n * (n - 1);
    }

    public static void main(String[] args) throws IOException
    {
        System.out.println(intersection(3));
    }
}
// This code is contributed by
// Manish Kumar Rai
```

Python3

```
# python program to find maximum umber of
# intersections of n circles
# Returns maximum number of intersections
def intersection(n):

    return n * (n - 1);

# Drive code
print(intersection(3))

# This code is contributed by Sam007
```

C#

```
// C# program to find maximum number of
// intersections of n circles
using System;
class GFG {

    // for the calculation of 2*(nC2)
    static int intersection(int n)
    {
        return n * (n - 1);
    }

    // Driver Code
    public static void Main()
    {
        Console.WriteLine(intersection(3));
    }
}
```

// This code is contributed by Sam007

php

```
<?php
// php program to find maximum number of
// intersections of n circles

// Returns maximum number of intersections
function intersection($n)
{
    return $n * ($n - 1);
}

// Drive code

echo intersection(3);

// This code is contributed by Sam007
?>
```

Output:

6

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/maximum-points-intersection-n-circles/>

Chapter 120

Maximum possible intersection by moving centers of line segments

Maximum possible intersection by moving centers of line segments - GeeksforGeeks

Given three points on the X-axis which denotes the center of three line segments. The length of the line segment is also given as L. The task is to move the center of the given line segments by a distance K to maximize the length of intersection between the three lines.

Examples:

Input: c1 = 1, c2 = 2, c3 = 3, L = 1, K = 0

Output: 0

Since, no center can be moved, there is no intersection among three segments $\{(0.5, 1.5), (1.5, 2.5), (2.5, 3.5)\}$.

Input: c1 = 1, c2 = 2, c3 = 3, L = 1, K = 1

Output: 1

Center's 1 and 3 can be shifted 1 unit ahead and forward respectively to overlap with center 2

The line segments will be as follows:

1. Line-Segment 1: (Center1-L/2, Center1+L/2)
2. Line-Segment 2: (Center2-L/2, Center2+L/2)
3. Line-Segment 3: (Center3-L/2, Center3+L/2)

Approach: Initially sort the centers, since the middle segment will never move as left and right segment can always reach near to its center to increase the intersection length. There will be three cases which are to be dealt with:

- **Case 1:** When the distance between centers of the right and left is more than or equal to $2*K+L$ in such scenario intersection will always be zero. No overlapping is possible since even after shifting both centers with K distance still, they will be away at a distance of L or more.
- **Case 2:** When the distance between centers of the right and left is more than or equal to $2*K$ in such scenario there exists an intersection which is equal to $(2 * k - (center[2] - center[0] - length))$ which can be calculated using simple maths.
- **Case 3:** When the distance between centers of the right and left is less than $2*K$ in such case both centers can coincide with the middle center. Hence, the intersection is L .

Below is the implementation of the above approach:

C++

```
#include <bits/stdc++.h>
using namespace std;

// Function to print the maximum intersection
int max_intersection(int* center, int length, int k)
{
    sort(center, center + 3);

    // Case 1
    if (center[2] - center[0] >= 2 * k + length) {
        return 0;
    }

    // Case 2
    else if (center[2] - center[0] >= 2 * k) {
        return (2 * k - (center[2] - center[0] - length));
    }

    // Case 3
    else
        return length;
}

// Driver Code
int main()
{
    int center[3] = { 1, 2, 3 };
    int L = 1;
    int K = 1;
    cout << max_intersection(center, L, K);
}
```

Java

```
// Java implementation
// of above approach
import java.util.*;

class GFG
{

    // Function to print the
    // maximum intersection
    static int max_intersection(int center[],
                                int length, int k)
    {
        Arrays.sort(center);

        // Case 1
        if (center[2] - center[0] >= 2 * k + length)
        {
            return 0;
        }

        // Case 2
        else if (center[2] - center[0] >= 2 * k)
        {
            return (2 * k - (center[2] -
                center[0] - length));
        }

        // Case 3
        else
            return length;
    }

    // Driver Code
    public static void main(String args[])
    {
        int center[] = { 1, 2, 3 };
        int L = 1;
        int K = 1;
        System.out.println( max_intersection(center, L, K));
    }
}

// This code is contributed
// by Arnab Kundu

C#
// C# implementation
```

```
// of above approach
using System;

class GFG
{

    // Function to print the
    // maximum intersection
    static int max_intersection(int []center,
                                int length, int k)
    {
        Array.Sort(center);

        // Case 1
        if (center[2] - center[0] >= 2 * k + length)
        {
            return 0;
        }

        // Case 2
        else if (center[2] -
                  center[0] >= 2 * k)
        {
            return (2 * k - (center[2] -
                               center[0] - length));
        }

        // Case 3
        else
            return length;
    }

    // Driver Code
    public static void Main()
    {
        int []center = { 1, 2, 3 };
        int L = 1;
        int K = 1;
        Console.WriteLine(max_intersection(center, L, K));
    }
}

// This code is contributed
// by Subhadeep Gupta
```

PHP

```
= 2 * $k + $length)
{
```

```
return 0;
}

// Case 2
else if ($center[2] -
$center[0] >= 2 * $k)
{
    return (2 * $k - ($center[2] -
$center[0] - $length));
}

// Case 3
else
    return $length;
}

// Driver Code
$center = array(1, 2, 3);
$L = 1;
$K = 1;
echo max_intersection($center, $L, $K);

// This code is contributed
// by mits
?>
```

Output:

1

Improved By : [andrew1234](#), [tufan_gupta2000](#), Mithun Kumar

Source

<https://www.geeksforgeeks.org/maximum-possible-intersection-by-moving-centers-of-line-segments/>

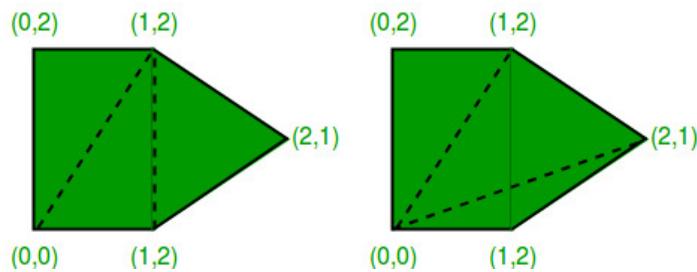
Chapter 121

Minimum Cost Polygon Triangulation

Minimum Cost Polygon Triangulation - GeeksforGeeks

A triangulation of a convex polygon is formed by drawing diagonals between non-adjacent vertices (corners) such that the diagonals never intersect. The problem is to find the cost of triangulation with the minimum cost. The cost of a triangulation is sum of the weights of its component triangles. Weight of each triangle is its perimeter (sum of lengths of all sides)

See following example taken from [this source](#).



Two triangulations of the same convex pentagon. The triangulation on the left has a cost of $8 + 2\sqrt{2} + 2\sqrt{5}$ (approximately 15.30), the one on the right has a cost of $4 + 2\sqrt{2} + 4\sqrt{5}$ (approximately 15.77).

This problem has recursive substructure. The idea is to divide the polygon into three parts: a single triangle, the sub-polygon to the left, and the sub-polygon to the right. We try all possible divisions like this and find the one that minimizes the cost of the triangle plus the cost of the triangulation of the two sub-polygons.

```
Let Minimum Cost of triangulation of vertices from i to j be minCost(i, j)
If j <= i + 2 Then
```

```

minCost(i, j) = 0
Else
    minCost(i, j) = Min { minCost(i, k) + minCost(k, j) + cost(i, k, j) }
        Here k varies from 'i+1' to 'j-1'

```

Cost of a triangle formed by edges (i, j), (j, k) and (k, i) is

$$\text{cost}(i, j, k) = \text{dist}(i, j) + \text{dist}(j, k) + \text{dist}(k, i)$$

Following is C++ implementation of above naive recursive formula.

```

// Recursive implementation for minimum cost convex polygon triangulation
#include <iostream>
#include <cmath>
#define MAX 1000000.0
using namespace std;

// Structure of a point in 2D plane
struct Point
{
    int x, y;
};

// Utility function to find minimum of two double values
double min(double x, double y)
{
    return (x <= y)? x : y;
}

// A utility function to find distance between two points in a plane
double dist(Point p1, Point p2)
{
    return sqrt((p1.x - p2.x)*(p1.x - p2.x) +
                (p1.y - p2.y)*(p1.y - p2.y));
}

// A utility function to find cost of a triangle. The cost is considered
// as perimeter (sum of lengths of all edges) of the triangle
double cost(Point points[], int i, int j, int k)
{
    Point p1 = points[i], p2 = points[j], p3 = points[k];
    return dist(p1, p2) + dist(p2, p3) + dist(p3, p1);
}

// A recursive function to find minimum cost of polygon triangulation
// The polygon is represented by points[i..j].
double mTC(Point points[], int i, int j)
{
    // There must be at least three points between i and j
}

```

```

// (including i and j)
if (j < i+2)
    return 0;

// Initialize result as infinite
double res = MAX;

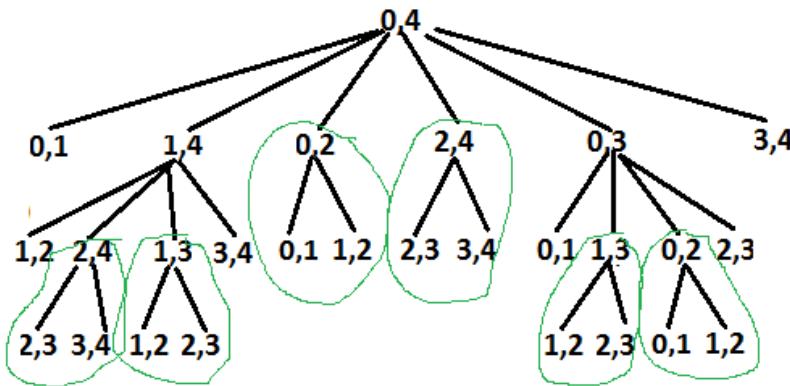
// Find minimum triangulation by considering all
for (int k=i+1; k<j; k++)
    res = min(res, (mTC(points, i, k) + mTC(points, k, j) +
                    cost(points, i, k, j)));
return res;
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 0}, {1, 0}, {2, 1}, {1, 2}, {0, 2}};
    int n = sizeof(points)/sizeof(points[0]);
    cout << mTC(points, 0, n-1);
    return 0;
}
    
```

Output:

15.3006

The above problem is similar to [Matrix Chain Multiplication](#). The following is recursion tree for $mTC(\text{points}[], 0, 4)$.



Recursion Tree for recursive implementation. Overlapping subproblems are encircled.

It can be easily seen in the above recursion tree that the problem has many overlapping subproblems. Since the problem has both properties: [Optimal Substructure](#) and [Overlapping Subproblems](#), it can be efficiently solved using dynamic programming.

Following is C++ implementation of dynamic programming solution.

```

// A Dynamic Programming based program to find minimum cost of convex
// polygon triangulation
#include <iostream>
#include <cmath>
#define MAX 1000000.0
using namespace std;

// Structure of a point in 2D plane
struct Point
{
    int x, y;
};

// Utility function to find minimum of two double values
double min(double x, double y)
{
    return (x <= y)? x : y;
}

// A utility function to find distance between two points in a plane
double dist(Point p1, Point p2)
{
    return sqrt((p1.x - p2.x)*(p1.x - p2.x) +
                (p1.y - p2.y)*(p1.y - p2.y));
}

// A utility function to find cost of a triangle. The cost is considered
// as perimeter (sum of lengths of all edges) of the triangle
double cost(Point points[], int i, int j, int k)
{
    Point p1 = points[i], p2 = points[j], p3 = points[k];
    return dist(p1, p2) + dist(p2, p3) + dist(p3, p1);
}

// A Dynamic programming based function to find minimum cost for convex
// polygon triangulation.
double mTCDP(Point points[], int n)
{
    // There must be at least 3 points to form a triangle
    if (n < 3)
        return 0;

    // table to store results of subproblems. table[i][j] stores cost of

```

```

// triangulation of points from i to j. The entry table[0] [n-1] stores
// the final result.
double table[n][n];

// Fill table using above recursive formula. Note that the table
// is filled in diagonal fashion i.e., from diagonal elements to
// table[0] [n-1] which is the result.
for (int gap = 0; gap < n; gap++)
{
    for (int i = 0, j = gap; j < n; i++, j++)
    {
        if (j < i+2)
            table[i][j] = 0.0;
        else
        {
            table[i][j] = MAX;
            for (int k = i+1; k < j; k++)
            {
                double val = table[i][k] + table[k][j] + cost(points,i,j,k);
                if (table[i][j] > val)
                    table[i][j] = val;
            }
        }
    }
    return table[0][n-1];
}

// Driver program to test above functions
int main()
{
    Point points[] = {{0, 0}, {1, 0}, {2, 1}, {1, 2}, {0, 2}};
    int n = sizeof(points)/sizeof(points[0]);
    cout << mTCDP(points, n);
    return 0;
}

```

Output:

15.3006

Time complexity of the above dynamic programming solution is $O(n^3)$.

Please note that the above implementations assume that the points of convex polygon are given in order (either clockwise or anticlockwise)

Exercise:

Extend the above solution to print triangulation also. For the above example, the optimal triangulation is 0 3 4, 0 1 3, and 1 2 3.

Sources:

<http://www.cs.utexas.edu/users/djimenez/utsa/cs3343/lecture12.html>

<http://www.cs.utoronto.ca/~heap/Courses/270F02/A4/chains/node2.html>

Source

<https://www.geeksforgeeks.org/minimum-cost-polygon-triangulation/>

Chapter 122

Minimum Perimeter of n blocks

Minimum Perimeter of n blocks - GeeksforGeeks

We are given n blocks of size 1 x 1, we need to find the minimum perimeter of the grid made by these blocks.

Examples :

```
Input : n = 4
Output : 8
Minimum possible perimeter with 4 blocks
is 8. See below explanation.
```

```
Input : n = 11
Output : 14
The square grid of above examples would be as
```

Let us take an example to see a pattern. Let us say that we have 4 blocks, following are different possibilities

```
+----+----+
|    |    |    |    Perimeter = 10
+----+----+
+----+----+
|    |    |
+---+
```

```
+---+---+
|   |   |
+---+---+
|   |
+---+
```

```
+----+
|   |   |
+----+
|   |
+----+
```

If we do some examples using pen and paper, we can notice that the perimeter becomes minimum when the shape formed is closest to a square. The reason for this is, we want maximum sides of blocks to face inside the shape so that perimeter of the shape becomes minimum.

If the Number of blocks is a perfect square then the perimeter would simply be $4 * \text{sqrt}(n)$. But, if the Number of blocks is not a perfect square root then we calculate number of rows and columns closest to square root. After arranging the blocks in a rectangular we still have blocks left then we will simply add 2 to the perimeter because only 2 extra side would be left.

The implementation of the above idea is given below.

C++

```
// CPP program to find minimum
// perimeter using n blocks.
#include <bits/stdc++.h>
using namespace std;

int minPerimeter(int n)
{
    int l = sqrt(n);
    int sq = l * l;

    // if n is a perfect square
    if (sq == n)
        return l * 4;
    else
    {
        // Number of rows
        long long int row = n / l;

        // perimeter of the
        // rectangular grid
        long long int perimeter
```

```
= 2 * (l + row);

// if there are blocks left
if (n % l != 0)
    perimeter += 2;
return perimeter;
}

// Driver code
int main()
{
    int n = 10;
    cout << minPerimeter(n);
    return 0;
}
```

Java

```
// JAVA Code to find minimum
// perimeter using n blocks
import java.util.*;

class GFG
{
    public static long minPerimeter(int n)
    {
        int l = (int) Math.sqrt(n);
        int sq = l * l;

        // if n is a perfect square
        if (sq == n)
            return l * 4;
        else
        {
            // Number of rows
            long row = n / l;

            // perimeter of the
            // rectangular grid
            long perimeter
                = 2 * (l + row);

            // if there are blocks left
            if (n % l != 0)
                perimeter += 2;
            return perimeter;
        }
    }
}
```

```
}

// Driver code
public static void main(String[] args)
{
    int n = 10;
    System.out.println(minPerimeter(n));
}
}

// This code is contributed by Arnav Kr. Mandal
```

Python3

```
# Python3 program to find minimum
# perimeter using n blocks.
import math

def minPerimeter(n):
    l = math.sqrt(n)
    sq = l * l

    # if n is a perfect square
    if (sq == n):
        return l * 4
    else :
        # Number of rows
        row = n / l

        # perimeter of the
        # rectangular grid
        perimeter = 2 * (l + row)

        # if there are blocks left
        if (n % l != 0):
            perimeter += 2
        return perimeter

# Driver code
n = 10
print(int(minPerimeter(n)))

# This code is contributed by
# Prasad Kshirsagar
```

C#

```
// C# Code to find minimum
```

```
// perimeter using n blocks
using System;

class GFG
{
    public static long minPerimeter(int n)
    {
        int l = (int) Math.Sqrt(n);
        int sq = l * l;

        // if n is a perfect square
        if (sq == n)
            return l * 4;
        else
        {
            // Number of rows
            long row = n / l;

            // perimeter of the
            // rectangular grid
            long perimeter
                = 2 * (l + row);

            // if there are blocks left
            if (n % l != 0)
                perimeter += 2;
            return perimeter;
        }
    }

    // Driver code
    public static void Main()
    {
        int n = 10;
        Console.Write(minPerimeter(n));
    }
}

// This code is contributed by nitin mittal
```

PHP

```
<?php
// PHP program to find minimum
// perimeter using n blocks.

function minPerimeter($n)
{
```

```
$l = floor(sqrt($n));
$sq = $l * $l;

// if n is a perfect square
if ($sq == $n)
    return $l * 4;
else
{
    // Number of rows
    $row = floor($n / $l);

    // perimeter of the
    // rectangular grid
    $perimeter = 2 * ($l + $row);

    // if there are blocks left
    if ($n % $l != 0)
        $perimeter += 2;
    return $perimeter;
}

// Driver code
$n = 10;
echo minPerimeter($n);

// This code is contributed
// by nitin mittal.
?>
```

Output :

14

References :

<http://mathforum.org/library/drmath/view/61595.html>
intermath.coe.uga.edu/tweb/gcsu-geo-spr06/aheath/aheath_rectperimeter.doc

Improved By : [nitin mittal](#), [Prasad_Kshirsagar](#)

Source

<https://www.geeksforgeeks.org/minimum-perimeter-n-blocks/>

Chapter 123

Minimum area of a Polygon with three points given

Minimum area of a Polygon with three points given - GeeksforGeeks

Given three points of a regular polygon($n > 3$), find the minimum area of a regular polygon (all sides same) possible with the points given .

Examples:

```
Input : 0.00 0.00
        1.00 1.00
        0.00 1.00
Output : 1.00
By taking point (1.00, 0.00) square is
formed of side 1.0 so area = 1.00 .
```

One thing to note in question before we proceed is that the number of sides must be at least 4 (note $n > 3$ condition)..

Here, we have to find the minimum area possible for a regular polygon, so to calculate minimum possible area, we need calculate required value of n . As the side length is not given, so we first calculate **circumradius of the triangle** formed by the points. It is given by the formula

$$R = abc / 4A$$

where a , b , c are the sides of the triangle formed and A is the area of the triangle. Here, the area of triangle can be calculated by **Heron's Formula** .

After calculating circumradius of the triangle, we calculate the area of the polygon by the formula

$$A = nX (\sin(360/n) xr^2 /2)$$

Here r represents the circumradius of n-gon (regular polygon of n sides) .

But, first we have to calculate value of n . To calculate n we first have to calculate all the angles of triangle by the **cosine formula**

$$\cos A = (b^2 + c^2 - a^2) / 2bc$$

$$\cos B = (a^2 + c^2 - b^2) / 2ac$$

$$\cos C = (a^2 + b^2 - c^2) / 2ab$$

Then, n is given by

$$n = \pi / \text{GCD}(A, B, C)$$

where A, B and C are the angles of the triangle . After calculating n we substitute this value to the formula for calculating area of polygon .

Below is C++ implementation of the given approach

```
// CPP program to find minimum area of polygon of
// number of sides more than three with given three points.
#include <bits/stdc++.h>
using namespace std;

// assigning pi value to variable
const double pi = 3.14159265359;

// calculating gcd value of two double values .
double gcd(double x, double y)
{
    return fabs(y) < 1e-4 ? x : gcd(y, fmod(x, y));
}

// Calculating minimum area of polygon through this function .
double min_area_of_polygon(double Ax, double Ay, double Bx,
                           double By, double Cx, double Cy)
{
    double a, b, c, Radius, Angle_A, Angle_B, Angle_C,
           semiperimeter, n, area;

    // calculating the length of the sides of the triangle
    // formed from given points a, b, c represents the
    // length of different sides of triangle .
    a = sqrt((Bx - Cx) * (Bx - Cx) + (By - Cy) * (By - Cy));
    b = sqrt((Ax - Cx) * (Ax - Cx) + (Ay - Cy) * (Ay - Cy));
    c = sqrt((Ax - Bx) * (Ax - Bx) + (Ay - By) * (Ay - By));

    // here we have calculated the semiperimeter of a triangle .
    semiperimeter = (a + b + c) / 2;

    // Now from the semiperimeter area of triangle is derived
    // through the heron's formula .
    double area_triangle = sqrt(semiperimeter * (semiperimeter - a)
                                * (semiperimeter - b))

```

```
* (semiperimeter - c));  
  
// thus circumradius of the triangle is derived from the  
// sides and area of the triangle calculated .  
Radius = (a * b * c) / (4 * area_triangle);  
  
// Now each angle of the triangle is derived from the sides  
// of the triangle .  
Angle_A = acos((b * b + c * c - a * a) / (2 * b * c));  
Angle_B = acos((a * a + c * c - b * b) / (2 * a * c));  
Angle_C = acos((b * b + a * a - c * c) / (2 * b * a));  
  
// Now n is calculated such that area is minimum for  
// the regular n-gon .  
n = pi / gcd(gcd(Angle_A, Angle_B), Angle_C);  
  
// calculating area of regular n-gon through the circumradius  
// of the triangle .  
area = (n * Radius * Radius * sin((2 * pi) / n)) / 2;  
  
return area;  
}  
  
int main()  
{  
    // three points are given as input .  
    double Ax, Ay, Bx, By, Cx, Cy;  
    Ax = 0.00;  
    Ay = 0.00;  
    Bx = 1.00;  
    By = 1.00;  
    Cx = 0.00;  
    Cy = 1.00;  
  
    printf("%.2f", min_area_of_polygon(Ax, Ay, Bx, By, Cx, Cy));  
    return 0;  
}
```

Output:

1.00

Source

<https://www.geeksforgeeks.org/minimum-area-polygon-three-points-given/>

Chapter 124

Minimum block jumps to reach destination

Minimum block jumps to reach destination - GeeksforGeeks

Given N lines and one starting point and destination point in 2-dimensional space. These N lines divide the space into some blocks. We need to print the minimum number of jumps to reach destination point from starting point. We can jump from one block to other block only if they share a side.

Examples:

```
Input : Lines = [x = 0, y = 0, x + y - 2 = 0]
        Start point = [1, 1],
        Dest point  = [-2, -1]
Output : 2
We need to jump 2 times (B4 -> B3 then B3 -> B5
or B4 -> B6 then B6 -> B5) to reach destination
point from starting point shown in below diagram.
Each block i is given an id Bi in the diagram.
```

We can solve this problem using a property of lines and points which is stated as, If we put two points in line equation then they will have same sign i.e. positive-positive or negative-negative of evaluated value if both points lies on the same side of line, in case of different sign i.e. positive-negative they will lie on different side of line.

Now we can use above property to solve this problem, For each line, we will check if start and destination point lie on the same side or not. If they lie to the different side of a line then that line must be jumped to come closer. As in above diagram start point and the destination point are on the same side of $x + y - 2 = 0$ line, so that line need not to be jumped, rest two lines need to be jumped because both points lies on opposite side.

Finally, we will check the sign of evaluation of points with respect to each line and we will

increase our jump count whenever we found opposite signs. Total time complexity of this problem will be linear.

```
// C++ program to find minimum jumps to reach
// a given destination from a given source
#include <bits/stdc++.h>
using namespace std;

// To represent point in 2D space
struct point
{
    int x, y;
    point(int x, int y) : x(x), y(y)
    {}
};

// To represent line of (ax + by + c)format
struct line
{
    int a, b, c;
    line(int a, int b, int c) : a(a), b(b), c(c)
    {}
    line()
    {}
};

// Returns 1 if evaluation is greater > 0,
// else returns -1
int evalPointOnLine(point p, line curLine)
{
    int eval = curLine.a* p.x +
              curLine.b * p.y +
              curLine.c;
    if (eval > 0)
        return 1;
    return -1;
}

// Returns minimum jumps to reach
// dest point from start point
int minJumpToReachDestination(point start,
                               point dest, line lines[], int N)
{
    int jumps = 0;
    for (int i = 0; i < N; i++)
    {
        // get sign of evaluation from point
        // co-ordinate and line equation
```

```
int signStart = evalPointOnLine(start, lines[i]);
int signDest = evalPointOnLine(dest, lines[i]);

// if both evaluation are of opposite sign,
// increase jump by 1
if (signStart * signDest < 0)
    jumps++;
}

return jumps;
}

// Driver code to test above methods
int main()
{
    point start(1, 1);
    point dest(-2, -1);

    line lines[3];
    lines[0] = line(1, 0, 0);
    lines[1] = line(0, 1, 0);
    lines[2] = line(1, 1, -2);

    cout << minJumpToReachDestination(start, dest, lines, 3);

    return 0;
}
```

Output:

2

Source

<https://www.geeksforgeeks.org/minimum-block-jumps-reach-destination/>

Chapter 125

Minimum distance to travel to cover all intervals

Minimum distance to travel to cover all intervals - GeeksforGeeks

Given many intervals as ranges and our position. We need to find minimum distance to travel to reach such a point which covers all the intervals at once.

Examples:

```
Input : Intervals = [(0, 7), (2, 14), (4, 6)]
        Position = 3
```

```
Output : 1
```

We can reach position 4 by traveling distance 1, at which all intervals will be covered. So answer will be 1

```
Input : Intervals = [(1, 2), (2, 3), (3, 4)]
        Position = 2
```

```
Output : -1
```

It is not possible to cover all intervals at once at any point

```
Input : Intervals = [(1, 2), (2, 3), (1, 4)]
        Position = 2
```

```
Output : 0
```

All Intervals are covered at current position only so no need travel and answer will be 0

All above examples are shown in below diagram.

We can solve this problem by concentrating only on endpoints. Since the requirement is to cover all intervals by reaching a point, all intervals must share a point for answer to exist. Even the interval with leftmost end point must overlap with the interval right most start point.

First, we find right most start point and left most end point from all intervals. Then we can compare our position with these points to get the result which is explained below :

1. If this right most start point is to the right of left most end point then it is not possible to cover all intervals simultaneously. (as in example 2)
2. If our position is in mid between to right most start and left most end then there is no need to travel and all intervals will be covered by current position only (as in example 3)
3. If our position is left to both points then we need to travel up to the rightmost start point and if our position is right to both points then we need to travel up to leftmost end point.

Refer above diagram to understand these cases. As in the first example, right most start is 4 and left most end is 6, so we need to reach 4 from current position 3 to cover all intervals. Please see below code for better understanding.

```
// C++ program to find minimum distance to
// travel to cover all intervals
#include <bits/stdc++.h>
using namespace std;

// structure to store an interval
struct Interval
{
    int start, end;
    Interval(int start, int end) : start(start),
                                    end(end)
    {}
};

// Method returns minimum distance to travel
// to cover all intervals
int minDistanceToCoverIntervals(Interval intervals[],
                                int N, int x)
{
    int rightMostStart = INT_MIN;
    int leftMostEnd = INT_MAX;

    // looping over all intervals to get right most
    // start and left most end
    for (int i = 0; i < N; i++)
    {
        if (rightMostStart < intervals[i].start)
```

```
rightMostStart = intervals[i].start;

if (leftMostEnd > intervals[i].end)
    leftMostEnd = intervals[i].end;
}

int res;

/* if rightmost start > leftmost end then all
   intervals are not aligned and it is not
   possible to cover all of them */
if (rightMostStart > leftMostEnd)
    res = -1;

// if x is in between rightmoststart and
// leftmostend then no need to travel any distance
else if (rightMostStart <= x && x <= leftMostEnd)
    res = 0;

// choose minimum according to current position x
else
    res = (x < rightMostStart) ? (rightMostStart - x) :
        (x - leftMostEnd);

return res;
}

// Driver code to test above methods
int main()
{
    int x = 3;
    Interval intervals[] = {{0, 7}, {2, 14}, {4, 6}};
    int N = sizeof(intervals) / sizeof(intervals[0]);

    int res = minDistanceToCoverIntervals(intervals, N, x);
    if (res == -1)
        cout << "Not Possible to cover all intervals\n";
    else
        cout << res << endl;
}
```

Output:

Source

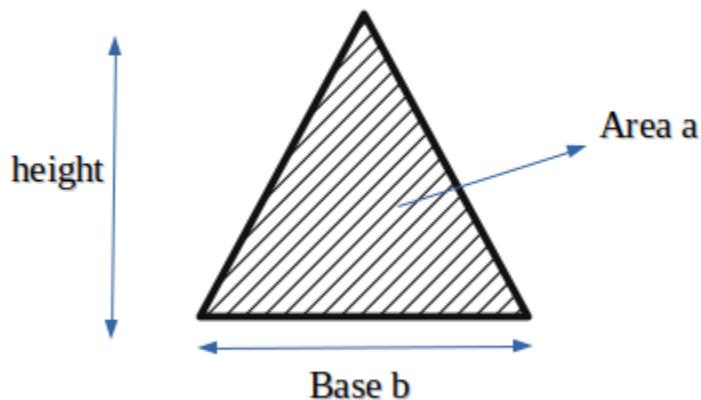
<https://www.geeksforgeeks.org/minimum-distance-travel-cover-intervals/>

Chapter 126

Minimum height of a triangle with given base and area

Minimum height of a triangle with given base and area - GeeksforGeeks

Given two integers a and b , find the smallest possible height such that a triangle of atleast area " a " and base " b " can be formed.



Examples :

Input : $a = 2, b = 2$

Output : Minimum height of triangle is 2

Explanation:

Input : a = 8, b = 4

Output : Minimum height of triangle is 4

Minimum height of Triangle with base “b” and area “a” can be evaluated by having the knowledge of the relationship between the three.

The relation between area, base and height is:

$$\text{area} = (1/2) * \text{base} * \text{height}$$

So height can be calculated as :

$$\text{height} = (2 * \text{area}) / \text{base}$$

Minimum height is the ceil of the height obtained using above formula.

C++

```
#include <bits/stdc++.h>
using namespace std;

// function to calculate minimum height of
// triangle
int minHeight(int base, int area){
    return ceil((float)(2*area)/base);
}

int main() {
    int base = 4, area = 8;
    cout << "Minimum height is "
        << minHeight(base, area) << endl;
    return 0;
}
```

Java

```
// Java code Minimum height of a
// triangle with given base and area

class GFG {

    // function to calculate minimum height of
    // triangle
```

```
static double minHeight(double base, double area)
{
    double d = (2 * area) / base;
    return Math.ceil(d);
}

// Driver code
public static void main (String[] args)
{
    double base = 4, area = 8;
    System.out.println("Minimum height is "+
                        minHeight(base, area));
}
// This code is contributed by Anant Agarwal.
```

Python

```
# Python Program to find minimum height of triangle
import math

def minHeight(area,base):
    return math.ceil((2*area)/base)

# Driver code
area = 8
base = 4
height = minHeight(area, base)
print("Minimum height is %d" % (height))
```

C#

```
// C# program to find minimum height of
// a triangle with given base and area
using System;

public class GFG {

    // function to calculate minimum
    // height of triangle
    static int minHeight(int b_ase, int area)
    {
        return (int)Math.Round((float)(2 * area) / b_ase);
    }

    // Driver function
    static public void Main()
```

```
{  
    int b_ase = 4, area = 8;  
    Console.WriteLine("Minimum height is "  
                      + minHeight(b_ase, area));  
}  
}  
  
// This code is contributed by vt_m.
```

PHP

```
<?php  
  
// function to calculate minimum  
// height of triangle  
function minHeight($base, $area)  
{  
    return ceil((2 * $area) / $base);  
}  
  
// Driver Code  
$base = 4;$area = 8;  
echo "Minimum height is ",  
     minHeight($base, $area);  
  
// This code is contributed by anuj_67.  
?>
```

Output :

Minimum height is 4

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/minimum-height-triangle-given-base-area/>

Chapter 127

Minimum length of the shortest path of a triangle

Minimum length of the shortest path of a triangle - GeeksforGeeks

Given N points on the plane, $(X_1, Y_1), (X_2, Y_2), (X_3, Y_3), \dots, (X_N, Y_N)$. The task is to calculate the minimum length of the shorter side of the triangle. and the path or points to place an isosceles triangle with any two sides of the triangle on the coordinate axis (X axis and Y axis) to cover all points.

Note: A point is covered if it lies inside the triangle or on the side of the triangle.

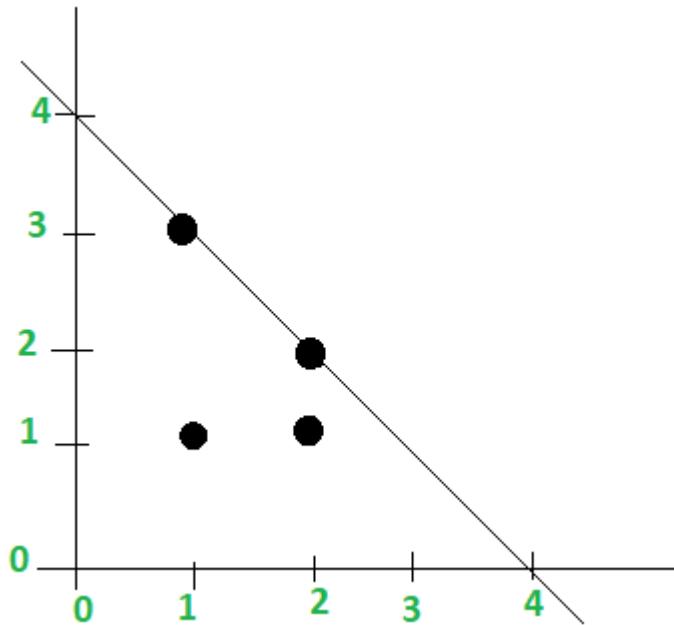
Examples:

Input: $(1, 3), (1, 1), (2, 1), (2, 2)$

Output: Length -> 4 , Path -> $(1, 4)$ and $(4, 1)$

Input: $(1, 2), (1, 1), (2, 1)$

Output: Length -> 3 , Path -> $(1, 3)$ and $(3, 1)$



In the first example, the minimum length of the shortest path is equal to the maximum sum of the points, which is 1+3 or 2+2. So the path which will cover all the points is (1, 4) and (4, 1) on the coordinate axis.

Below is the step by step algorithm to solve this problem:

1. Initialize 'N' points on a plane.
2. Traverse through each point and find the sum of each point and store it in a variable 'answer'.
3. Replace the next maximum sum of the points with the previous sum.
4. Then you will get the path on a coordinate axis (1, answer) and (answer, 1) which will cover all the points of isosceles triangle.

Below is the implementation of above algorithm:

```
// C++ program to illustrate
// the above problem
#include <bits/stdc++.h>
using namespace std;
#define ll long long

// function to get the minimum length of
// the shorter side of the triangle
void shortestLength(int n, int x[], int y[])
{
    int answer = 0;
```

```
// traversing through each points on the plane
int i = 0;
while (n--) {
    // if sum of a points is greater than the
    // previous one, the maximum gets replaced
    if (x[i] + y[i] > answer)
        answer = x[i] + y[i];

    i++;
}

// print the length
cout << "Length -> " << answer << endl;
cout << "Path -> "
    << "( 1, " << answer << " )"
    << "and ( " << answer << ", 1 )";
}

// Driver code
int main()
{
    // initialize the number of points
    int n = 4;

    // points on the plane
    int x[n] = { 1, 4, 2, 1 };
    int y[n] = { 4, 1, 1, 2 };

    shortestLength(n, x, y);

    return 0;
}
```

Output:

```
Length -> 5
Path -> ( 1, 5 )and ( 5, 1 )
```

Source

<https://www.geeksforgeeks.org/minimum-length-of-the-shortest-path-of-a-triangle/>

Chapter 128

Minimum lines to cover all points

Minimum lines to cover all points - GeeksforGeeks

Given N points in 2-dimensional space, we need to print the count of the minimum number of lines which traverse through all these N points and which go through a specific (x_0, y_0) point also.

Examples:

If given points are $(-1, 3), (4, 3), (2, 1), (-1, -2), (3, -3)$ and (x_0, y_0) point is $(1, 0)$ i.e. every line must go through this point.
Then we have to draw at least two lines to cover all these points going through (x_0, y_0) as shown in below diagram.

We can solve this problem by considering the slope of all points with (x_0, y_0) . If two distinct points have the same slope with (x_0, y_0) then they can be covered with same line only so we can track slope of each point and whenever we get a new slope we will increase our line count by one.

In below code slope is stored as a pair of integer to get rid of the precision problem and a set is used to keep track of occurred slopes.

Please see below code for better understanding.

```
// C++ program to get minimum lines to cover
// all the points
#include <bits/stdc++.h>
using namespace std;
```

```
// Utility method to get gcd of a and b
int gcd(int a, int b)
{
    if (b == 0)
        return a;
    return gcd(b, a % b);
}

// method returns reduced form of dy/dx as a pair
pair<int, int> getReducedForm(int dy, int dx)
{
    int g = gcd(abs(dy), abs(dx));

    // get sign of result
    bool sign = (dy < 0) ^ (dx < 0);

    if (sign)
        return make_pair(-abs(dy) / g, abs(dx) / g);
    else
        return make_pair(abs(dy) / g, abs(dx) / g);
}

/* method returns minimum number of lines to
   cover all points where all lines goes
   through (x0, y0) */
int minLinesToCoverPoints(int points[][][2], int N,
                           int x0, int y0)
{
    // set to store slope as a pair
    set< pair<int, int> > st;
    pair<int, int> temp;
    int minLines = 0;

    // loop over all points once
    for (int i = 0; i < N; i++)
    {
        // get x and y co-ordinate of current point
        int curX = points[i][0];
        int curY = points[i][1];

        temp = getReducedForm(curY - y0, curX - x0);

        // if this slope is not there in set,
        // increase ans by 1 and insert in set
        if (st.find(temp) == st.end())
        {
            st.insert(temp);
            minLines++;
        }
    }
}
```

```
        }
    }

    return minLines;
}

// Driver code to test above methods
int main()
{
    int x0, y0;
    x0 = 1;
    y0 = 0;

    int points[][] = {
        {-1, 3},
        {4, 3},
        {2, 1},
        {-1, -2},
        {3, -3}
    };

    int N = sizeof(points) / sizeof(points[0]);
    cout << minLinesToCoverPoints(points, N, x0, y0);
    return 0;
}
```

Output:

2

Source

<https://www.geeksforgeeks.org/minimum-lines-cover-points/>

Chapter 129

Minimum number of points to be removed to get remaining points on one side of axis

Minimum number of points to be removed to get remaining points on one side of axis - GeeksforGeeks

We are given **n** points in a Cartesian plane. Our task is to find the minimum number of points that should be removed in order to get the remaining points on one side of any axis.

Examples :

```
Input : 4
       1 1
       2 2
      -1 -1
      -2 2
Output : 1
Explanation :
If we remove (-1, -1) then all the remaining
points are above x-axis. Thus the answer is 1.
```

```
Input : 3
       1 10
       2 3
       4 11
Output : 0
Explanation :
All points are already above X-axis. Hence the
answer is 0.
```

Approach :

This problem is a simple example of constructive brute force algorithm on Geometry. The solution can be approached simply by finding the number of points on all sides of the X-axis and Y-axis. The minimum of this will be the answer.

```
// CPP program to find minimum points to be moved
// so that all points are on same side.
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

// Structure to store the coordinates of a point.
struct Point
{
    int x, y;
};

// Function to find the minimum number of points
int findmin(Point p[], int n)
{
    int a = 0, b = 0, c = 0, d = 0;
    for (int i = 0; i < n; i++)
    {
        // Number of points on the left of Y-axis.
        if (p[i].x <= 0)
            a++;
        // Number of points on the right of Y-axis.
        else if (p[i].x >= 0)
            b++;
        // Number of points above X-axis.
        if (p[i].y >= 0)
            c++;
        // Number of points below X-axis.
        else if (p[i].y <= 0)
            d++;
    }
    return min({a, b, c, d});
}

// Driver Function
int main()
{
    Point p[] = { {1, 1}, {2, 2}, {-1, -1}, {-2, 2} };
    int n = sizeof(p)/sizeof(p[0]);
```

```
cout << findmin(p, n);  
return 0;  
}
```

Output :

1

Source

<https://www.geeksforgeeks.org/minimum-number-points-removed-get-remaining-points-one-side-axis/>

Chapter 130

Minimum number of square tiles required to fill the rectangular floor

Minimum number of square tiles required to fill the rectangular floor - GeeksforGeeks

Given a rectangular floor of ($M \times N$) meters is to be paved with square tiles of ($s \times s$). The task is to find the minimum number of tiles required to pave the rectangular floor.

Constraints:

1. It's allowed to cover the surface larger than the floor, but the floor has to be covered.
2. It's not allowed to break the tiles.
3. The sides of tiles should be parallel to the sides of the floor.

Examples:

Input: 2 1 2

Output: 1

length of floor = 2

breadth of floor = 1

length of side of tile = 2

No of tiles required for paving is 2.

Input: 222 332 5

Output: 3015

Approach:

It is given that edges of each tile must be parallel to edges of the tiles allows us to analyze X and Y axes separately, that is, how many segments of length 's' are needed to cover a segment of length 'M' and 'N' — and take product of these two quantities.

`ceil(M/s) * ceil(N/s)`

, where $\text{ceil}(x)$ is the least integer which is above or equal to x . Using integers only, it is usually written as

`((M + s - 1) / s)*((N + s - 1) / s)`

Below is the implementation of above approach:

C++

```
// C++ implementation of above approach
#include <bits/stdc++.h>
using namespace std;

// Function to find the number of tiles
int solve(int M, int N, int s)
{
    // if breadth is divisible by side of square
    if (N % s == 0) {

        // tiles required is N/s
        N = N / s;
    }
    else {

        // one more tile required
        N = (N / s) + 1;
    }

    // if length is divisible by side of square
    if (M % s == 0) {

        // tiles required is M/s
        M = M / s;
    }
    else {

        // one more tile required
        M = (M / s) + 1;
    }

    return M * N;
}

// Driver Code
int main()
{
```

```
// input length and breadth of
// rectangle and side of square
int N = 12, M = 13, s = 4;

cout << solve(M, N, s);

return 0;
}
```

Java

```
// Java implementation
// of above approach
import java.util.*;
import java.lang.*;
import java.io.*;

class GFG
{

// Function to find the
// number of tiles
static int solve(int M, int N, int s)
{
    // if breadth is divisible
    // by side of square
    if (N % s == 0)
    {

        // tiles required is N/s
        N = N / s;
    }
    else
    {

        // one more tile required
        N = (N / s) + 1;
    }

    // if length is divisible
    // by side of square
    if (M % s == 0)
    {

        // tiles required is M/s
        M = M / s;
    }
    else
```

```
{  
    // one more tile required  
    M = (M / s) + 1;  
}  
  
return M * N;  
}  
  
// Driver Code  
public static void main(String args[])  
{  
    // input length and breadth of  
    // rectangle and side of square  
    int N = 12, M = 13, s = 4;  
  
    System.out.println(solve(M, N, s));  
}  
}  
  
// This code is contributed  
// by ChitraNayal
```

Python 3

```
# Python 3 implementation of  
# above approach  
  
# Function to find the number  
# of tiles  
def solve(M, N, s) :  
  
    # if breadth is divisible  
    # by side of square  
    if (N % s == 0) :  
  
        # tiles required is N/s  
        N = N // s  
  
    else :  
  
        # one more tile required  
        N = (N // s) + 1  
  
    # if length is divisible by  
    # side of square  
    if (M % s == 0) :  
        N = N * M
```

```
# tiles required is M/s
M = M // s

else :

# one more tile required
M = (M // s) + 1

return M * N

# Driver Code
if __name__ == "__main__":
    # input length and breadth of
    # rectangle and side of square
    N, M, s = 12, 13, 4

    print(solve(M, N, s))

# This code is contributed by ANKITRAI1
```

C#

```
// C# implementation of above approach
using System;

class GFG
{

// Function to find the
// number of tiles
static int solve(int M, int N, int s)
{
    // if breadth is divisible
    // by side of square
    if (N % s == 0)
    {

        // tiles required is N/s
        N = N / s;
    }
    else
    {

        // one more tile required
        N = (N / s) + 1;
    }
}
```

```
// if length is divisible
// by side of square
if (M % s == 0)
{
    // tiles required is M/s
    M = M / s;
}
else
{
    // one more tile required
    M = (M / s) + 1;
}

return M * N;
}

// Driver Code
static void Main()
{
    // input length and breadth of
    // rectangle and side of square
    int N = 12, M = 13, s = 4;

    Console.WriteLine(solve(M, N, s));
}
}

// This code is contributed
// by mits
```

PHP

Output:

12

Using ceil function:

C++

```
// C++ implementation of above approach
#include <bits/stdc++.h>
using namespace std;
```

```
// Function to find the number of tiles
int solve(double M, double N, double s)
{
    // no of tiles
    int ans = ((int)(ceil(M / s)) * (int)(ceil(N / s)));

    return ans;
}

// Driver Code
int main()
{
    // input length and breadth of
    // rectangle and side of square
    double N = 12, M = 13, s = 4;

    cout << solve(M, N, s);

    return 0;
}
```

Java

```
// Java implementation of above approach
class GFG
{
// Function to find the number of tiles
static int solve(double M,
                 double N, double s)
{
    // no of tiles
    int ans = ((int)(Math.ceil(M / s)) *
               (int)(Math.ceil(N / s)));

    return ans;
}

// Driver Code
public static void main(String[] args)
{
    // input length and breadth of
    // rectangle and side of square
    double N = 12, M = 13, s = 4;

    System.out.println(solve(M, N, s));
}
}
```

```
// This Code is contributed by mits
```

Output:

12

Improved By : [tufan_gupta2000](#), Mithun Kumar, ANKITRAI1

Source

<https://www.geeksforgeeks.org/minimum-number-of-square-tiles-required-to-fill-the-rectangular-floor/>

Chapter 131

Minimum revolutions to move center of a circle to a target

Minimum revolutions to move center of a circle to a target - GeeksforGeeks

Given a circle of radius r and center in point (x_1, y_1) and given a point (x_2, y_2) . The task is move center of circle from given center (x_1, y_1) to target (x_2, y_2) using minimum number of steps. In one step, we can put a pin on the border of the circle at any point, then rotate the circle around that pin by any angle and finally remove the pin.

Examples :

```
Input : r = 2
        x1 = 0, y1 = 0
        x2 = 0, y2 = 4
Output :1
```

```
Input : r = 1
        x1 = 1, y1 = 1,
        x2 = 4, y2 = 4
Output : 3
```

Let's consider a straight line between the two centers. Clearly to move the center with maximum distance we need to rotate it around the line and with 180 degrees. So the maximum distance we can move the center each time is $2 * r$. Let's continue moving the center with $2 * r$ distance each time until the two circles intersect. Now obviously we can make the center moves into its final position by rotating it around one of the intersection points of the two circles until it reaches the final destination.

Every time we make the circle moves $2 * r$ times except the last move it'll be $< 2 * r$. Let the initial distance between the two points be d . Solution to the problem will be $\text{ceil}(d/2*r)$.

C++

```
// C++ program to find minimum number of
// revolutions to reach a target center
#include<bits/stdc++.h>
using namespace std;

// Minimum revolutions to move center from
// (x1, y1) to (x2, y2)
int minRevolutions(double r, int x1, int y1,
                    int x2, int y2)
{
    double d = sqrt((x1 - x2)*(x1 - x2) +
                     (y1 - y2)*(y1 - y2));
    return ceil(d/(2*r));
}

// Driver code
int main()
{
    int r = 2, x1 = 0, y1 = 0, x2 = 0, y2 = 4;
    cout << minRevolutions(r, x1, y1, x2, y2);
    return 0;
}
```

Java

```
// Java program to find minimum number of
// revolutions to reach a target center
class GFG {

    // Minimum revolutions to move center
    // from (x1, y1) to (x2, y2)
    static double minRevolutions(double r,
                                 int x1, int y1, int x2, int y2)
    {

        double d = Math.sqrt((x1 - x2)
                             * (x1 - x2) + (y1 - y2)
                             * (y1 - y2));

        return Math.ceil(d / (2 * r));
    }

    // Driver Program to test above function
    public static void main(String arg[])
    {

        int r = 2, x1 = 0, y1 = 0;
        int x2 = 0, y2 = 4;
```

```
        System.out.print((int)minRevolutions(r,
                                              x1, y1, x2, y2));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python program to find
# minimum number of
# revolutions to reach
# a target center
import math

# Minimum revolutions to move center from
# (x1, y1) to (x2, y2)
def minRevolutions(r,x1,y1,x2,y2):

    d = math.sqrt((x1 -x2)*(x1 - x2) +
                  (y1 - y2)*(y1 - y2))
    return math.ceil(d//(2*r))

# Driver code

r = 2
x1 = 0
y1 = 0
x2 = 0
y2 = 4

print(minRevolutions(r, x1, y1, x2, y2))

# This code is contributed
# by Anant Agarwal.
```

C#

```
// C# program to find minimum number of
// revolutions to reach a target center
using System;

class GFG {

    // Minimum revolutions to move center
    // from (x1, y1) to (x2, y2)
    static double minRevolutions(double r,
```

```
        int x1, int y1, int x2, int y2)
    {

        double d = Math.Sqrt((x1 - x2)
            * (x1 - x2) + (y1 - y2)
            * (y1 - y2));

        return Math.Ceiling(d / (2 * r));
    }

    // Driver Program to test above function
    public static void Main()
    {
        int r = 2, x1 = 0, y1 = 0;
        int x2 = 0, y2 = 4;

        Console.WriteLine((int)minRevolutions(r,
            x1, y1, x2, y2));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// PHP program to find minimum
// number of revolutions to reach
// a target center

// Minimum revolutions to move
// center from (x1, y1) to (x2, y2)
function minRevolutions($r, $x1, $y1,
    $x2, $y2)
{
    $d = sqrt(($x1 - $x2) * ($x1 - $x2) +
        ($y1 - $y2) * ($y1 - $y2));
    return ceil($d / (2 * $r));
}

// Driver code
$r = 2; $x1 = 0; $y1 = 0; $x2 = 0; $y2 = 4;
echo minRevolutions($r, $x1, $y1, $x2, $y2);

// This code is contributed by nitin mittal.
?>
```

Output :

1

Time Complexity : O(1)

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/minimum-revolutions-move-center-circle-target/>

Chapter 132

Minimum squares to cover a rectangle

Minimum squares to cover a rectangle - GeeksforGeeks

Given a rectangle with length l and breadth b , we need to find the minimum number of squares that can cover the surface of the rectangle, given that each square has a side of length a . It is allowed to cover the surface larger than the rectangle, but the rectangle has to be covered. It is not allowed to break the square.

Examples:

```
Input : 1 2 3
Output :1
We have a 3x3 square and we need
to make a rectangles of size 1x2.
So we need only square to cover the
rectangle.
```

```
Input : 11 23 14
Output :2
```

The only way to actually fill the rectangle optimally is to arrange each square such that it is parallel to the sides of the rectangle. So we just need to find the number of squares to fully cover the length and breadth of the rectangle.

The length of the rectangle is l , and if the side length of the square is a divides l , then there must be l/a squares to cover the full length of l . If l isn't divisible by a , we need to add 1 to l/a , to round it down. For this we can use the **ceil function**, as ceil(x) returns the least integer which is above or equal to x .

We can do the same with the rectangle width, and take the number of squares across the width to be **ceil(b/a)**.

So, total number of squares=**ceil(m/a) * ceil(n/a)**.

```
// C++ program to find the minimum number
// of squares to cover the surface of the
// rectangle with given dimensions
#include <bits/stdc++.h>
using namespace std;
int squares(int l, int b, int a)
{
    // function to count
    // the number of squares that can
    // cover the surface of the rectangle
    return ceil(l / (double)a) * ceil(b / (double)a);
}

// Driver code
int main()
{
    int l = 11, b = 23, a = 14;
    cout << squares(l, b, a) << endl;
    return 0;
}
```

Output:

1

Source

<https://www.geeksforgeeks.org/minimum-squares-to-cover-a-rectangle/>

Chapter 133

Mirror of a point through a 3 D plane

Mirror of a point through a 3 D plane - GeeksforGeeks

Given a point(x, y, z) in 3-D and coefficients of the equation of plane, the task is to find the mirror image of that point through the given plane.

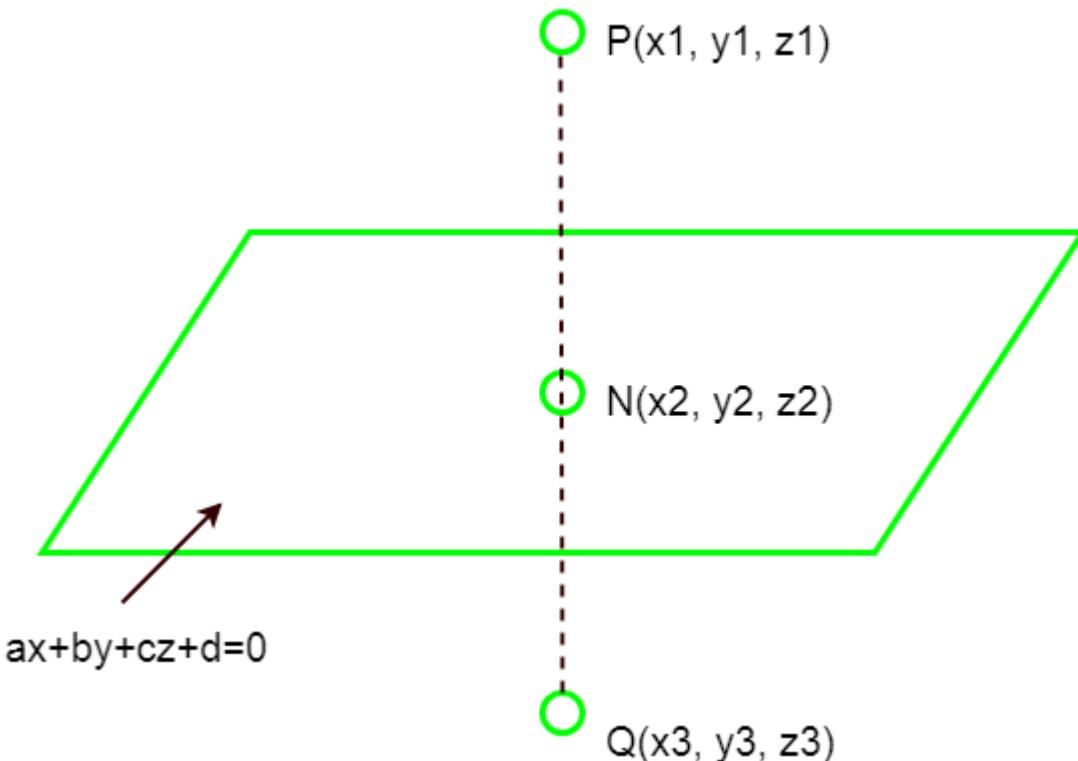
Examples:

Input: $a = 1, b = -2, c = 0, d = 0, x = -1, y = 3, z = 4$

Output: $x3 = 1.799999999999998, y3 = -2.599999999999996, z3 = 4.0$

Input: $a = 2, b = -1, c = 1, d = 3, x = 1, y = 3, z = 4$

Output: $x3 = -3.0, y3 = 5.0, z3 = 2.0$



Approach: Equation of plane is as $ax + by + cz + d = 0$. Therefore, direction ratios of the normal to the plane are (a, b, c) . Let N be the foot of perpendicular from given point to the given plane so, line PN has directed ratios (a, b, c) and it passes through $P(x_1, y_1, z_1)$.

The equation of line PN will be as:-

$$(x - x_1) / a = (y - y_1) / b = (z - z_1) / c = k$$

Hence any point on line PN can be written as:-

$$\begin{aligned} x &= a*k + x_1 \\ y &= b*k + y_1 \\ z &= c*k + z_1 \end{aligned}$$

since N lies in both line and plane so will satisfy $(ax + by + cz + d = 0)$.

$$\begin{aligned} \Rightarrow a * (a * k + x_1) + b * (b * k + y_1) + c * (c * k + z_1) + d &= 0. \\ \Rightarrow a * a * k + a * x_1 + b * b * k + b * y_1 + c * c * k + c * z_1 + d &= 0. \\ \Rightarrow (a * a + b * b + c * c)k &= -a * x_1 - b * y_1 - c * z_1 - d. \\ \Rightarrow k &= (-a * x_1 - b * y_1 - c * z_1 - d) / (a * a + b * b + c * c). \end{aligned}$$

Now, the coordinates of Point N in terms of k will be:-

```
x2 = a * k + x1  
y2 = b * k + y1  
z2 = c * k + z1
```

Since, Point N(x2, y2, z2) is midpoint of point P(x1, y1, z1) and point Q(x3, y3, z3), coordinates of Point Q are:-

```
=> x3 = 2 * x2 - x1  
=> y3 = 2 * y2 - y1  
=> z3 = 2 * z2 - z1
```

C

```
// C program to find  
// Mirror of a point  
// through a 3 D plane  
#include<stdio.h>  
  
// Function to mirror image  
void mirror_point(float a, float b,  
                  float c, float d,  
                  float x1, float y1,  
                  float z1)  
{  
    float k = (-a * x1 - b *  
               y1 - c * z1 - d) /  
            (float)(a * a + b * b + c * c);  
    float x2 = a * k + x1;  
    float y2 = b * k + y1;  
    float z2 = c * k + z1;  
    float x3 = 2 * x2 - x1;  
    float y3 = 2 * y2 - y1;  
    float z3 = 2 * z2 - z1;  
  
    printf("x3 = %.1f ", x3);  
    printf("y3 = %.1f ", y3);  
    printf("z3 = %.1f ", z3);  
}  
  
// Driver Code  
int main()  
{  
    float a = 1;  
    float b = -2;  
    float c = 0;  
    float d = 0;  
    float x1 = -1;
```

```
float y1 = 3;
float z1 = 4;

// function call
mirror_point(a, b, c, d,
              x1, y1, z1);
}

// This code is contributed
// by Amber_Saxena.
```

Java

```
// Java program to find
// Mirror of a point
// through a 3 D plane
import java.io.*;

class GFG
{

    // Function to mirror image
    static void mirror_point(int a, int b,
                            int c, int d,
                            int x1, int y1,
                            int z1)
    {
        float k = (-a * x1 - b * y1 - c * z1 - d) /
                  (float)(a * a + b * b + c * c);
        float x2 = a * k + x1;
        float y2 = b * k + y1;
        float z2 = c * k + z1;
        float x3 = 2 * x2 - x1;
        float y3 = 2 * y2 - y1;
        float z3 = 2 * z2 - z1;

        System.out.print("x3 = " + x3 + " ");
        System.out.print("y3 = " + y3 + " ");
        System.out.print("z3 = " + z3 + " ");
    }

    // Driver Code
    public static void main(String[] args)
    {
        int a = 1;
        int b = -2;
        int c = 0;
        int d = 0;
```

```
int x1 = -1;
int y1 = 3;
int z1 = 4;

// function call
mirror_point(a, b, c, d,
              x1, y1, z1) ;
}

}

// This code is contributed
// by inder_verma
```

Python

```
# Function to mirror image
def mirror_point(a, b, c, d, x1, y1, z1):

    k =(-a * x1-b * y1-c * z1-d)/float((a * a + b * b + c * c))
    x2 = a * k + x1
    y2 = b * k + y1
    z2 = c * k + z1
    x3 = 2 * x2-x1
    y3 = 2 * y2-y1
    z3 = 2 * z2-z1
    print "x3 =", x3,
    print "y3 =", y3,
    print "z3 =", z3,

# Driver Code
a = 1
b = -2
c = 0
d = 0
x1 = -1
y1 = 3
z1 = 4

# function call
mirror_point(a, b, c, d, x1, y1, z1)
```

PHP

```
<?php
// PHP program to find Mirror of
// a point through a 3 D plane
```

```
// Function to mirror image
function mirror_point($a, $b, $c, $d,
                     $x1, $y1, $z1)
{
    $k = (-$a * $x1 - $b *
           $y1 - $c * $z1 - $d) /
        ($a * $a + $b *
         $b + $c * $c);
    $x2 = $a * $k + $x1;
    $y2 = $b * $k + $y1;
    $z2 = $c * $k + $z1;
    $x3 = 2 * $x2 - $x1;
    $y3 = 2 * $y2 - $y1;
    $z3 = 2 * $z2 - $z1;
    echo sprintf("x3 = %.1f ", $x3);
    echo sprintf("y3 = %.1f ", $y3);
    echo sprintf("z3 = %.1f ", $z3);
}

// Driver Code
$a = 1;
$b = -2;
$c = 0;
$d = 0;
$x1 = -1;
$y1 = 3;
$z1 = 4;
// function call
mirror_point($a, $b, $c, $d,
              $x1, $y1, $z1);

// This code is contributed
// by Amber_Saxena.
?>
```

Output:

x3 = 1.8 y3 = -2.6 z3 = 4.0

Improved By : [inderDuMCA](#), [Amber_Saxena](#)

Source

<https://www.geeksforgeeks.org/mirror-of-a-point-through-a-3-d-plane/>

Chapter 134

Neighbors of a point on a circle using Bresenham's algorithm

Neighbors of a point on a circle using Bresenham's algorithm - GeeksforGeeks

Given a center of a circle and its radius. our task is to find the neighbors of any point on the discrete circle.

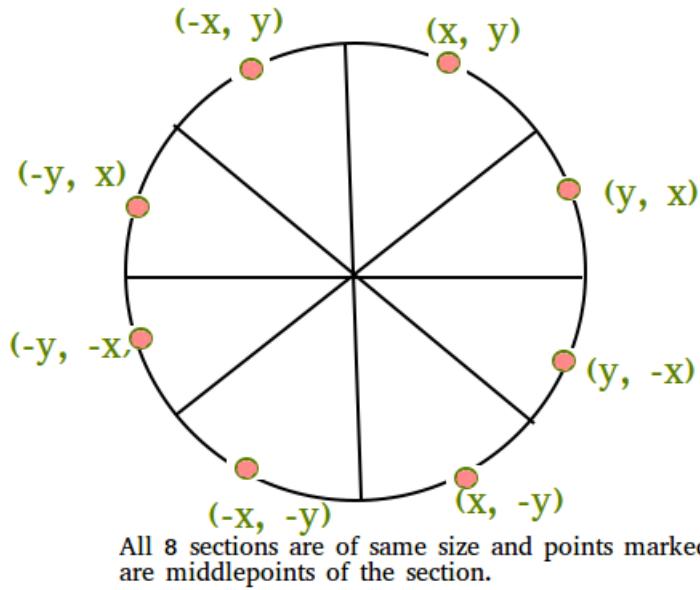
Examples:

```
Input : Center = (0, 0),
        Radius = 3
        Point for determining neighbors = (2, 2)
Output : Neighbors of given point are : (1, 3), (3, 1)

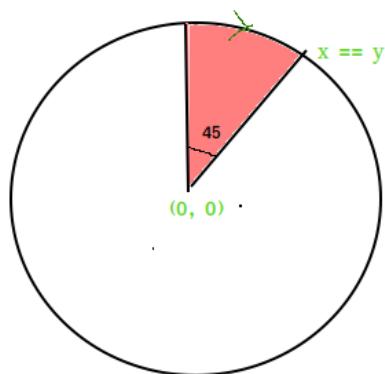
Input : Center = (2, 2)
        Radius 2
        Point of determining neighbors = (0, 2)
Output : Neighbors of given point are : (1, 4), (3, 4)
```

The neighbours of any point on the discrete circle are those points which are having one less or one more x coordinate from its original x coordinate.

We use [bresenham's circle generation algorithm](#) to extract out integer points required to draw a circle on computer screen of pixels.



Circles have the property of being highly symmetrical which is needed when it comes to drawing them on the computer screen of pixels. Bresenham's circle algorithm calculates the locations of the pixels in the first 45 degrees and remaining pixels on the periphery of a circle which is centered at origin are computed by using 8-way symmetry property of the circle.



Derivation: Consider an infinitesimally small continuous arc of a circle as shown in the figure below, Let's suppose that we want to move along a clockwise circular arc with center at the origin, with radius r and our motion lies in the first octant lies in first octant, so

our limits are from $(0, r)$ to $(r/\sqrt{2}, r/\sqrt{2})$ where $x = y$. as we know that, In this particular octant, the decrease in the y coordinate is less than increase in the x coordinate or you can say that the movement along x axis is more than the movement along y axis so, the x coordinate always increases in first octant. now, we want to know whether the y will change with x or not. In order to know the variation of y with x , bresenham's introduced a variable named as decision parameter which will update their value as the loop runs.

Now, we need to understand that how we will choose the next pixel, In the figure, $f(N)$ and $f(S)$ are errors involved in calculating the distances from origin to pixel N and S respectively and whichever is comes out to be less we'll choose that pixel. decision parameter is defined as $d = f(N) + f(S)$, if $d \leq 0$ then N will be the next pixel otherwise S will be the next pixel. we will keep doing this procedure until $x < y$ condition holds and by taking all symmetric points we will end up with all the integer points required to show a circle on computer screen of pixels.

This article is not predominantly focused on bresenham's algorithm therefore, I'll skip the derivation of decision parameter but if you want to understand of derivation of decision parameter then go to the reference links.

Note: neighbors of a point can be of any number and y -coordinate of the neighbors should have same sign as of its input pixel and for those pixels which have y coordinate zero we will print out all the neighbors irrespective of its signs.

A discrete geometric object consists of a finite set of integer points. This set is usually very sparse. Therefore, an array representation is not at all space efficient to store. we will use hash map having data value is a linked list of y -coordinate and key value as x - coordinate. we can easily access them using the key value and its also a space efficient.

Below is C++ stl program for determining Neighbors of a given point.

```
// C++ program to find neighbors of a given point on circle
#include <bits/stdc++.h>
using namespace std;

// map to store all the pixels of circle
map<int, list<int> > mymap;
map<int, list<int> >::iterator it;

// This program will print all the stored pixels.
void showallpoints(map<int, list<int> >& mymap)
{
    // To print out all the stored pixels,
    // we will traverse the map using iterator
    for (it = mymap.begin(); it != mymap.end(); it++) {

        // List contains all the y-coordinate.
```

```
list<int> temp = it->second;

for (auto p = temp.begin(); p != temp.end(); p++) {
    cout << "(" << it->first << ", " << *p << ")\n";
}
}

// This function will stored the pixels.
void putpixelone(int m, int n, map<int, list<int> >& mymap)
{
    // check if the given pixel is present already in the
    // map then discard that pixel and return the function.
    map<int, list<int> >::iterator it;

    // if x-coordinate of the pixel is present in the map then
    // it will give iterator pointing to list of those pixels
    // which are having same x-coordinate as the input pixel
    if (mymap.find(m) != mymap.end()) {

        it = mymap.find(m);
        list<int> temp = it->second;
        list<int>::iterator p;

        // Checking for y coordinate
        for (p = temp.begin(); p != temp.end(); p++)
            if (*p == n)
                return;

        // if map doesn't contain pixels having same y-
        // coordinate then pixel are different and store
        // the pixel
        mymap[m].push_back(n);
    } else

        // Neither x nor y coordinate are same.
        // put the pixel into the map
        mymap[m].push_back(n);

    return;
}

// generate all the pixels using 8 way-symmetry of circle
void putpixelall(int p, int q, int x1, int y1)
{
    putpixelone(p + x1, q + y1, mymap);
    putpixelone(q + x1, p + y1, mymap);
    putpixelone(q + x1, -p + y1, mymap);
```

```
putpixelone(p + x1, -q + y1, mymap);
putpixelone(-p + x1, -q + y1, mymap);
putpixelone(-q + x1, -p + y1, mymap);
putpixelone(-q + x1, p + y1, mymap);
putpixelone(-p + x1, q + y1, mymap);
return;
}

// Bresenham's cirle algorithm
void circle(int centerx, int centery, int r)
{
    // initial coordinate will be (0, radius) and we
    // will move counter-clockwise from this coordinate
    int x = 0;
    int y = r;

    // decision parameter for initial coordinate
    float decision_para = 3 - 2 * (r);
    putpixelall(x, y, centerx, centery);

    while (x < y) {

        // x will always increase by 1 unit
        x = x + 1;
        if (decision_para <= 0) {

            // if decision parameter is negative then N
            // will be next pixel N(x+1, y)
            decision_para = decision_para + 4 * x + 6;
        } else {

            // if decision parameter is positive then N
            // will be next pixel S(x+1, y-1)
            y = y - 1;
            decision_para = decision_para + 4 * (x - y) + 10;
        }

        // Function call to generate all the pixels by symmetry
        putpixelall(x, y, centerx, centery);
    }
    return;
}
// this program will find the neighbors of a given point
void neighbours(map<int, list<int> >& mymap, int given_pointx,
                int given_pointy)
{
    for (it = mymap.begin(); it != mymap.end(); ++it) {
        if (it->first == given_pointx + 1 ||

```

```
it->first == given_pointx - 1) {  
list<int> temp1 = it->second;  
list<int>::iterator itr1;  
for (itr1 = temp1.begin(); itr1 != temp1.end(); ++itr1) {  
  
    // Checking for same-sign.  
    if (given_pointy >= 0 && *itr1 >= 0)  
        cout << "(" << it->first << ", " << *itr1 << ")\\n";  
    else if (given_pointy <= 0 && *itr1 <= 0)  
        cout << "(" << it->first << ", " << *itr1 << ")\\n";  
    else  
        continue;  
}  
}  
}  
  
// Driver code  
int main()  
{  
    int center_x = 0, center_y = 0;  
    float r = 3.0;  
    circle(center_x, center_y, r);  
    showallpoints(mymap);  
    int nx = 3, ny = 0;  
    neighbours(mymap, nx, ny);  
    cout << endl;  
    return 0;  
}
```

Output:

```
(-3, 0), (-3, -1), (-3, 1), (-2, -2), (-2, 2), (-1, -3), (-1, 3), (0, 3)  
(0, -3), (1, 3), (1, -3), (2, 2), (2, -2), (3, 0), (3, 1), (3, -1)  
Neighbours of given point are : (2, 2), (2, -2)
```

References :

<https://www.slideshare.net/tahersb/bresenham-circle>

Source

<https://www.geeksforgeeks.org/neighbors-point-circle-using-bresenhams-algorithm/>

Chapter 135

Non-crossing lines to connect points in a circle

Non-crossing lines to connect points in a circle - GeeksforGeeks

Consider a circle with **n** points on circumference of it where **n is even**. Count number of ways we can connect these points such that no two connecting lines cross each other and every point is connected with exactly one other point. Any point can be connected with any other point.

Consider a circle with 4 points.

```
    1  
2      3  
    4
```

In above diagram, there are two non-crossing ways to connect $\{\{1, 2\}, \{3, 4\}\}$ and $\{\{1, 3\}, \{2, 4\}\}$.

Note that $\{\{2, 3\}, \{1, 4\}\}$ is invalid as it would cause a cross

Examples :

Input : n = 2
Output : 1

Input : n = 4
Output : 2

Input : n = 6

Output : 5

Input : n = 3
Output : Invalid
n must be even.

We need to draw $n/2$ lines to connect n points. When we draw a line, we divide the points in two sets that need to be connected. Each set needs to be connected within itself. Below is recurrence relation for the same.

Let $m = n/2$

```
// For each line we draw, we divide points
// into two sets such that one set is going
// to be connected with  $i$  lines and other
// with  $m-i-1$  lines.
Count(m) = &Sum; Count(i) * Count(m-i-1)
           where  $0 \leq i < m$ 
Count(0) = 1

Total number of ways with  $n$  points
= Count(m) = Count(n/2)
```

If we take a closer look at above recurrence, it is actually recurrence of [Catalan Numbers](#). So the task reduces to finding $n/2$ 'th Catalan number.

Below is implementation based on above idea.

C++

```
// C++ program to count number of ways to connect  $n$  (where  $n$ 
// is even) points on a circle such that no two connecting
// lines cross each other and every point is connected with
// one other point.
#include<iostream>
using namespace std;

// A dynamic programming based function to find nth
// Catalan number
unsigned long int catalanDP(unsigned int n)
{
    // Table to store results of subproblems
    unsigned long int catalan[n+1];

    // Initialize first two values in table
    catalan[0] = catalan[1] = 1;
```

```
// Fill entries in catalan[] using recursive formula
for (int i=2; i<=n; i++)
{
    catalan[i] = 0;
    for (int j=0; j<i; j++)
        catalan[i] += catalan[j] * catalan[i-j-1];
}

// Return last entry
return catalan[n];
}

// Returns count of ways to connect n points on a circle
// such that no two connecting lines cross each other and
// every point is connected with one other point.
unsigned long int countWays(unsigned long int n)
{
    // Throw error if n is odd
    if (n & 1)
    {
        cout << "Invalid";
        return 0;
    }

    // Else return n/2'th Catalan number
    return catalanDP(n/2);
}

// Driver program to test above function
int main()
{
    cout << countWays(6) << " ";
    return 0;
}
```

Java

```
// Java program to count number
// of ways to connect n (where
// n is even) points on a circle
// such that no two connecting
// lines cross each other and
// every point is connected with
// one other point.
import java.io.*;

class GFG
{
```

```
// A dynamic programming
// based function to find
// nth Catalan number
static int catalanDP(int n)
{
    // Table to store
    // results of subproblems
    int []catalan = new int [n + 1];

    // Initialize first
    // two values in table
    catalan[0] = catalan[1] = 1;

    // Fill entries in catalan[]
    // using recursive formula
    for (int i = 2; i <= n; i++)
    {
        catalan[i] = 0;
        for (int j = 0; j < i; j++)
            catalan[i] += catalan[j] *
                          catalan[i - j - 1];
    }

    // Return last entry
    return catalan[n];
}

// Returns count of ways to
// connect n points on a circle
// such that no two connecting
// lines cross each other and
// every point is connected
// with one other point.
static int countWays(int n)
{
    // Throw error if n is odd
    if (n < 1)
    {
        System.out.println("Invalid");
        return 0;
    }

    // Else return n/2'th
    // Catalan number
    return catalanDP(n / 2);
}
```

```
// Driver Code
public static void main (String[] args)
{
    System.out.println(countWays(6) + " ");
}
```

// This code is contributed
// by akt_mit

C#

```
// C# program to count number
// of ways to connect n (where
// n is even) points on a circle
// such that no two connecting
// lines cross each other and
// every point is connected with
// one other point.
using System;

class GFG
{

    // A dynamic programming
    // based function to find
    // nth Catalan number
    static int catalanDP(int n)
    {
        // Table to store
        // results of subproblems
        int []catalan = new int [n + 1];

        // Initialize first
        // two values in table
        catalan[0] = catalan[1] = 1;

        // Fill entries in catalan[]
        // using recursive formula
        for (int i = 2; i <= n; i++)
        {
            catalan[i] = 0;
            for (int j = 0; j < i; j++)
                catalan[i] += catalan[j] *
                    catalan[i - j - 1];
        }

        // Return last entry
    }
}
```

```
        return catalan[n];
    }

// Returns count of ways to
// connect n points on a circle
// such that no two connecting
// lines cross each other and
// every point is connected
// with one other point.
static int countWays(int n)
{
    // Throw error if n is odd
    if (n < 1)
    {
        Console.WriteLine("Invalid");
        return 0;
    }

    // Else return n/2'th
    // Catalan number
    return catalanDP(n / 2);
}

// Driver Code
static public void Main ()
{
    Console.WriteLine(countWays(6) + " ");
}
}

// This code is contributed
// by M_kit
```

PHP

```
<?php
// PHP program to count number of
// ways to connect n (where n is
// even) points on a circle such
// that no two connecting lines
// cross each other and every
// point is connected with one
// other point.

// A dynamic programming based
// function to find nth Catalan number
function catalanDP($n)
{
```

```
// Table to store results
// of subproblems Initialize
// first two values in table
$catalan[0] = $catalan[1] = 1;

// Fill entries in catalan[]
// using recursive formula
for ($i = 2; $i <= $n; $i++)
{
    $catalan[$i] = 0;
    for ($j = 0; $j < $i; $j++)
        $catalan[$i] += $catalan[$j] *
                        $catalan[$i - $j - 1];
}

// Return last entry
return $catalan[$n];
}

// Returns count of ways to connect
// n points on a circle such that
// no two connecting lines cross
// each other and every point is
// connected with one other point.
function countWays($n)
{
// Throw error if n is odd
if ($n & 1)
{
    echo "Invalid";
    return 0;
}

// Else return n/2'th
// Catalan number
return catalanDP($n / 2);
}

// Driver Code
echo countWays(6) , " ";

// This code is contributed by aj_36
?>
```

Output :

Time Complexity : $O(n^2)$

Auxiliary Space : $O(n)$

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/non-crossing-lines-connect-points-circle/>

Chapter 136

Number of Integral Points between Two Points

Number of Integral Points between Two Points - GeeksforGeeks

Given two points **p** (x_1, y_1) and **q** (x_2, y_2), calculate the number of integral points lying on the line joining them.

Example : If points are (0, 2) and (4, 0), then the number of integral points lying on it is only one and that is (2, 1).

Similarly, if points are (1, 9) and (8, 16), the integral points lying on it are 6 and they are (2, 10), (3, 11), (4, 12), (5, 13), (6, 14) and (7, 15).

Simple Approach

Start from any of the given points, reach the other end point by using loops. For every point inside the loop, check if it lies on the line that joins given two points. If yes, then increment the count by 1. Time Complexity for this approach will be $O(\min(x_2-x_1, y_2-y_1))$.

Optimal Approach

1. If the edge formed by joining p and q is parallel to the X-axis, then the number of integral points between the vertices is :
$$\text{abs}(p.y - q.y) - 1$$
2. Similarly if edge is parallel to the Y-axis, then the number of integral points in between is :
$$\text{abs}(p.x - q.x) - 1$$

3. Else, we can find the integral points between the vertices using below formula:
 $\text{GCD}(\text{abs}(p.x - q.x), \text{abs}(p.y - q.y)) - 1$

How does the GCD formula work?

The idea is to find the equation of the line in simplest form, i.e., in equation $ax + by + c = 0$, coefficients a , b and c become co-prime. We can do this by calculating the GCD (greatest common divisor) of a , b and c and convert a , b and c in the simplest form. Then, the answer will be (difference of y coordinates) divided by $(a) - 1$. This is because after calculating $ax + by + c = 0$, for different y values, x will be number of y values which are exactly divisible by a .

Below is C++ implementation of above idea.

```
// C++ code to find the number of integral points
// lying on the line joining the two given points
#include <iostream>
#include <cmath>
using namespace std;

// Class to represent an Integral point on XY plane.
class Point
{
public:
    int x, y;
    Point(int a=0, int b=0):x(a),y(b) {}
};

// Utility function to find GCD of two numbers
// GCD of a and b
int gcd(int a, int b)
{
    if (b == 0)
        return a;
    return gcd(b, a%b);
}

// Finds the no. of Integral points between
// two given points.
int getCount(Point p, Point q)
{
    // If line joining p and q is parallel to
    // x axis, then count is difference of y
    // values
    if (p.x==q.x)
        return abs(p.y - q.y) - 1;
```

```
// If line joining p and q is parallel to
// y axis, then count is difference of x
// values
if (p.y == q.y)
    return abs(p.x-q.x) - 1;

return gcd(abs(p.x-q.x), abs(p.y-q.y))-1;
}

// Driver program to test above
int main()
{
    Point p(1, 9);
    Point q(8, 16);

    cout << "The number of integral points between "
        << "(" << p.x << ", " << p.y << ") and (" 
        << q.x << ", " << q.y << ") is "
        << getCount(p, q);

    return 0;
}
```

Output:

The number of integral points between (1, 9) and (8, 16) is 6

Reference :

<https://www.geeksforgeeks.org/count-integral-points-inside-a-triangle/>

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [nikhil52](#)

Source

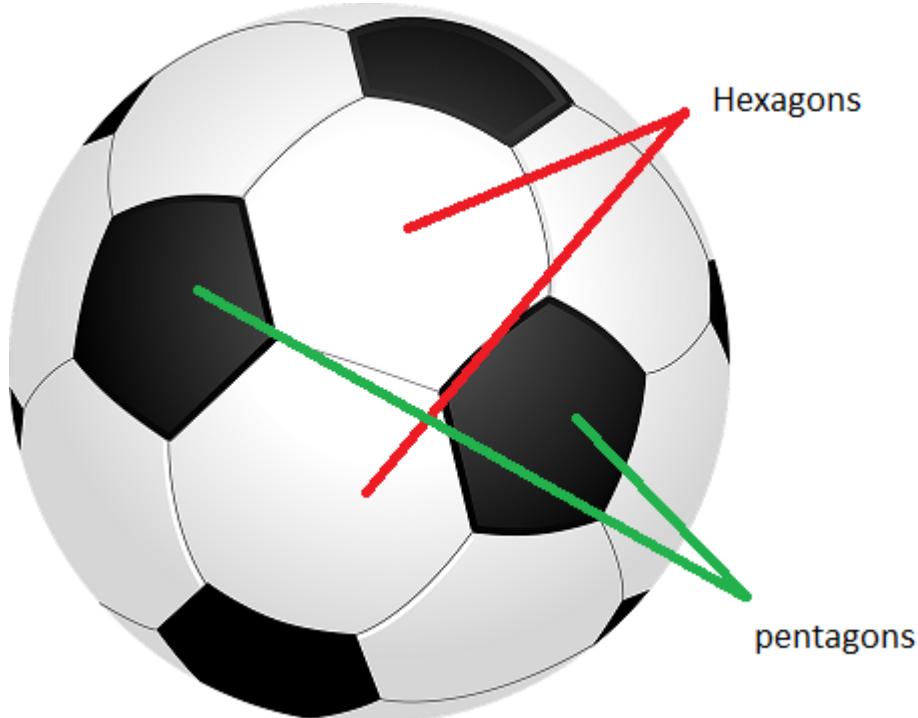
<https://www.geeksforgeeks.org/number-integral-points-two-points/>

Chapter 137

Number of Pentagons and Hexagons on a Football

Number of Pentagons and Hexagons on a Football - GeeksforGeeks

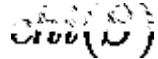
Given a standard football, regular hexagons and pentagons are drawn on it as shown in the picture. Find out the number of hexagons and pentagons.



We can apply [Euler Characteristics](#) to find out number of Hexagons and Pentagons on a

standard Football.

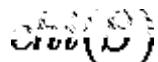
According to Euler Characteristics: For every surface S there exists an integer



such that whenever a graph G with V vertices and E edges is embedded in S so that there are F faces(regions divided by the graph), we have:

$$V - E + F = 2$$

For a sphere(shape of football)



$= 2$.

Hence, the equation becomes $V - E + F = 2$.

Now, let number of pentagons P and number of hexagons H.

Number of vertices will be:

6 vertices for each of the hexagons, i.e. $6*H$.

5 vertices for each of the pentagons, i.e. $5*P$.

But we have counted each vertex thrice, once for each adjacent polygon, follow the picture



Hence, number of vertices, $V = (6*H + 5*P)/3$.

Number of edges will be:

6 edges for each of the hexagons, i.e. $6*H$.

5 edges for each of the pentagons, i.e. $5*P$.

But we have counted each edge twice, once for each adjacent polygon, follow the picture



Hence, number of edges, $E = (6*H + 5*P)/2$.

Number of faces will be:

There are H hexagons and P pentagons, each forming a face. Hence, total number of faces, $F = (H + P)$.

So, we can write:

after solving this equation we will get $P = 12$. So, there are 12 Pentagons.

Now number of Hexagons:

We can see that each pentagon is surrounded by 5 Hexagons. So there should be $5*P$ hexagons, but we have counted each hexagon thrice for each of its 3 adjacent pentagons. Hence, number of hexagons = $5*P/3 = 5*12/3 = 20$.

Hence, there are 20 Hexagons and 12 Pentagons in a standard football.

Source

<https://www.geeksforgeeks.org/number-pentagons-hexagons-football/>

Chapter 138

Number of Triangles that can be formed given a set of lines in Euclidean Plane

Number of Triangles that can be formed given a set of lines in Euclidean Plane - Geeks-forGeeks

Given a set $L = \{l_1, l_2, \dots, l_n\}$ of 'n' distinct lines on the Euclidean Plane. The i^{th} line is given by an equation in the form $a_i x + b_i y = c_i$. Find the number of triangles that can be formed using the lines in the set L . Note that no two pairs of lines intersect at the same point.

Note: This problem doesn't mention that the lines can't be parallel which makes the problem difficult to solve.

Examples:

```
Input: a[] = {1, 2, 3, 4}
       b[] = {2, 4, 5, 5}
       c[] = {5, 7, 8, 6}
```

Output: 2

The number of triangles that can be formed are: 2

```
Input: a[] = {1, 2, 3, 2, 4, 1, 2, 3, 4, 5}
       b[] = {2, 4, 6, 3, 6, 5, 10, 15, 20, 25}
       c[] = {3, 5, 11, 10, 9, 17, 13, 11, 7, 3}
```

Output: 30

The number of triangles that can be formed are: 30

The naive algorithm can be described as:

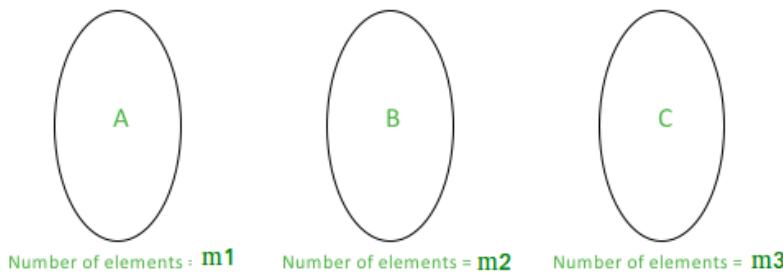
1. Pick up 3 arbitrary lines from the set L.
2. Now check if a triangle can be formed using the selected 3 lines. This can be done easily by checking that none of them is pairwise parallel.
3. Increment the counter if the triangle can be formed.

Time Complexity: There are nC_3 triplets of lines. For each triplet, we have to do 3 comparisons to check that any 2 lines are not parallel which means the check can be done in $O(1)$ time. This makes the naive algorithm $O(n^3)$.

This can also be achieved in $O(n \log n)$. The logic behind the efficient algorithm is described below.

We divide the set L in various subsets. The formation of subsets is based on slopes i.e. all the lines in a particular subset have the same slope i.e. they are parallel to each other.

Let us consider three sets (say A, B and C). For a particular set (say A) the lines belonging to this are all parallel to each other. If we have A, B and C, we can pick one line from each set to get a triangle because none of these lines will be parallel. By making the subsets we have ensured that no two lines which are parallel are being picked together.



Now if we have only these **3 subsets**,

$$\begin{aligned} \text{Number of triangles} &= (\text{Number of ways to pick a line from A}) * \\ &\quad (\text{Number of ways to pick a line from B}) * \\ &\quad (\text{Number of ways to pick a line from C}) \\ &= m1*m2*m3 \end{aligned}$$

Here $m1$ is count of elements with first slope (in Set A)

Here $m2$ is count of elements with first slope (in Set B)

Here $m3$ is count of elements with first slope (in Set C)

Similarly, if we have **4 subsets**, we can extend this logic to get,

$$\text{Number of triangles} = m1*m2*m3 + m1*m2*m4 + m1*m3*m4 + m2*m3*m4$$

For number of subsets greater than 3, If we have ' k ' subsets, our task is to find the sum of number of elements of the subset taken 3 at a time. This can be done by maintaining a count array. We make a count array where count_i denotes the count of the i^{th} subset of parallel lines.

We one by one compute following values.

```
sum1 = m1 + m2 + m3 .....
sum2 = m1*m2 + m1*m3 + ... + m2*m3 + m2*m4 + ...
sum3 = m1*m2*m3 + m1*m2*m4 + ..... m2*m3*m4 + .....
sum3 gives our final answer
```

```
// C++ program to find the number of
// triangles that can be formed
// using a set of lines in Euclidean
// Plane
#include <bits/stdc++.h>
using namespace std;

#define EPSILON numeric_limits<double>::epsilon()

// double variables can't be checked precisely
// using '==' this function returns true if
// the double variables are equal
bool compareDoubles(double A, double B)
{
    double diff = A-B;
    return (diff<EPSILON) && (-diff<EPSILON);
}

// This function returns the number of triangles
// for a given set of lines
int numberOfTriangles(int a[], int b[], int c[], int n)
{
    //slope array stores the slope of lines
    double slope[n];
    for (int i=0; i<n; i++)
        slope[i] = (a[i]*1.0)/b[i];

    // slope array is sorted so that all lines
    // with same slope come together
    sort(slope, slope+n);

    // After sorting slopes, count different
    // slopes. k is index in count[].
    int count[n], k = 0;
    int this_count = 1; // Count of current slope
    for (int i=1; i<n; i++)
    {
        if (compareDoubles(slope[i], slope[i-1]))
            this_count++;
        else
        {
```

```
        count[k++] = this_count;
        this_count = 1;
    }
}
count[k++] = this_count;

// calculating sum1 (Sum of all slopes)
// sum1 = m1 + m2 + ...
int sum1 = 0;
for (int i=0; i<k; i++)
    sum1 += count[i];

// calculating sum2. sum2 = m1*m2 + m2*m3 + ...
int sum2 = 0;
int temp[n]; // Needed for sum3
for (int i=0; i<k; i++)
{
    temp[i] = count[i]*(sum1-count[i]);
    sum2 += temp[i];
}
sum2 /= 2;

// calculating sum3 which gives the final answer
// m1 * m2 * m3 + m2 * m3 * m4 + ...
int sum3 = 0;
for (int i=0; i<k; i++)
    sum3 += count[i]*(sum2-temp[i]);
sum3 /= 3;

return sum3;
}

// Driver code
int main()
{
    // lines are stored as arrays of a, b
    // and c for 'ax+by=c'
    int a[] = {1, 2, 3, 4};
    int b[] = {2, 4, 5, 5};
    int c[] = {5, 7, 8, 6};

    // n is the number of lines
    int n = sizeof(a)/sizeof(a[0]);

    cout << "The number of triangles that"
        " can be formed are: "
        << number0fTringles(a, b, c, n);
```

```
    return 0;  
}
```

Output:

The number of triangles that can be formed are: 2

Time Complexity: All the loops in the code are $O(n)$. The time complexity in this implementation is thus driven by the sort function used to sort the slope array. This makes the algorithm $O(n\log n)$.

Source

<https://www.geeksforgeeks.org/number-triangles-can-formed-given-set-lines-euclidean-plane/>

Chapter 139

Number of horizontal or vertical line segments to connect 3 points

Number of horizontal or vertical line segments to connect 3 points - GeeksforGeeks

Given three points on the x-y coordinate plane. You need to find the no. of line segments formed by making a polyline passing through these points. (Line segment can be vertically or horizontally aligned to the coordinate axis)

Examples :

```
Input : A = {-1, -1}, B = {-1, 3}, C = {4, 3}
Output : 2
Explanation:
There are two segments in this polyline.
Input : A = {1, 1}, B = {2, 3} C = {3, 2}
Output : 3
```

The result is one if all points are on x axis or y axis. The result is 2 if points can form L shape. L shape is formed if any of the three points can be used as a joining point. Otherwise answer is 3.

C++

```
// CPP program to find number of horizontal (or vertical
// line segments needed to connect three points.
#include <iostream>
using namespace std;
```

```
// Function to check if the third point forms a
// rectangle with other two points at corners
bool isBetween(int a, int b, int c)
{
    return min(a, b) <= c && c <= max(a, b);
}

// Returns true if point k can be used as a joining
// point to connect using two line segments
bool canJoin(int x[], int y[], int i, int j, int k)
{
    // Check for the valid polyline with two segments
    return (x[k] == x[i] || x[k] == x[j]) &&
           isBetween(y[i], y[j], y[k]) ||
           (y[k] == y[i] || y[k] == y[j]) &&
           isBetween(x[i], x[j], x[k]);
}

int countLineSegments(int x[], int y[])
{
    // Check whether the X-coordinates or
    // Y-coordinates are same.
    if ((x[0] == x[1] && x[1] == x[2]) ||
        (y[0] == y[1] && y[1] == y[2]))
        return 1;

    // Iterate over all pairs to check for two
    // line segments
    else if (canJoin(x, y, 0, 1, 2) ||
             canJoin(x, y, 0, 2, 1) ||
             canJoin(x, y, 1, 2, 0))
        return 2;

    // Otherwise answer is three.
    else
        return 3;
}

// Driver code
int main()
{
    int x[3], y[3];
    x[0] = -1; y[0] = -1;
    x[1] = -1; y[1] = 3;
    x[2] = 4; y[2] = 3;
    cout << countLineSegments(x, y);
    return 0;
}
```

}

Java

```
// Java program to find number of horizontal
// (or vertical line segments needed to
// connect three points.
import java.io.*;

class GFG {

    // Function to check if the third
    // point forms a rectangle with
    // other two points at corners
    static boolean isBetween(int a, int b, int c)
    {
        return (Math.min(a, b) <= c &&
                c <= Math.max(a, b));
    }

    // Returns true if point k can be
    // used as a joining point to connect
    // using two line segments
    static boolean canJoin(int x[], int y[],
                           int i, int j, int k)
    {
        // Check for the valid polyline
        // with two segments
        return (x[k] == x[i] || x[k] == x[j]) &&
               isBetween(y[i], y[j], y[k]) ||
               (y[k] == y[i] || y[k] == y[j]) &&
               isBetween(x[i], x[j], x[k]);
    }

    static int countLineSegments(int x[], int y[])
    {
        // Check whether the X-coordinates or
        // Y-coordinates are same.
        if ((x[0] == x[1] && x[1] == x[2]) ||
            (y[0] == y[1] && y[1] == y[2]))
            return 1;

        // Iterate over all pairs to check for two
        // line segments
        else if (canJoin(x, y, 0, 1, 2) ||
                 canJoin(x, y, 0, 2, 1) ||
                 canJoin(x, y, 1, 2, 0))
            return 2;
    }
}
```

```
// Otherwise answer is three.  
else  
    return 3;  
}  
  
// Driver code  
public static void main (String[] args) {  
  
    int x[]={new int[3], y[]={new int[3];  
  
        x[0] = -1; y[0] = -1;  
        x[1] = -1; y[1] = 3;  
        x[2] = 4; y[2] = 3;  
  
        System.out.println(countLineSegments(x, y));  
    }  
  
}  
  
// This code is contributed by vt_m
```

Python3

```
# Python program to find number  
# of horizontal (or vertical  
# line segments needed to  
# connect three points.  
  
import math  
  
# Function to check if the  
# third point forms a  
# rectangle with other  
# two points at corners  
def isBetween(a, b, c) :  
  
    return min(a, b) <= c and c <= max(a, b)  
  
# Returns true if point k  
# can be used as a joining  
# point to connect using  
# two line segments  
def canJoin( x, y, i, j, k) :  
  
    # Check for the valid polyline
```

```
# with two segments
return (x[k] == x[i] or x[k] == x[j]) and isBetween(y[i], y[j], y[k]) or (y[k] == y[i] or y[k] == y[j])

def countLineSegments( x, y):

    # Check whether the X-coordinates or
    # Y-coordinates are same.
    if ((x[0] == x[1] and x[1] == x[2]) or
        (y[0] == y[1] and y[1] == y[2])):
        return 1

    # Iterate over all pairs to check for two
    # line segments
    elif (canJoin(x, y, 0, 1, 2) or
          canJoin(x, y, 0, 2, 1) or
          canJoin(x, y, 1, 2, 0)):
        return 2

    # Otherwise answer is three.
    else:
        return 3

#driver code
x= [-1,-1, 4]
y= [-1, 3, 3]

print(countLineSegments(x, y))

# This code is contributed by Gitanjali.
```

C#

```
// C# program to find number of horizontal
// (or vertical) line segments needed to
// connect three points.
using System;

class GFG {

    // Function to check if the third
    // point forms a rectangle with
    // other two points at corners
    static bool isBetween(int a, int b, int c)
    {
        return (Math.Min(a, b) <= c &&
                c <= Math.Max(a, b));
    }
}
```

```
// Returns true if point k can be
// used as a joining point to connect
// using two line segments
static bool canJoin(int[] x, int[] y,
                     int i, int j, int k)
{
    // Check for the valid polyline
    // with two segments
    return (x[k] == x[i] || x[k] == x[j])
        && isBetween(y[i], y[j], y[k])
        || (y[k] == y[i] || y[k] == y[j])
        && isBetween(x[i], x[j], x[k]);
}

static int countLineSegments(int[] x, int[] y)
{
    // Check whether the X-coordinates or
    // Y-coordinates are same.
    if ((x[0] == x[1] && x[1] == x[2]) ||
        (y[0] == y[1] && y[1] == y[2]))
        return 1;

    // Iterate over all pairs to check for two
    // line segments
    else if (canJoin(x, y, 0, 1, 2)
            || canJoin(x, y, 0, 2, 1)
            || canJoin(x, y, 1, 2, 0))
        return 2;

    // Otherwise answer is three.
    else
        return 3;
}

// Driver code
public static void Main()
{
    int[] x = new int[3];
    int[] y = new int[3];

    x[0] = -1;
    y[0] = -1;
    x[1] = -1;
    y[1] = 3;
    x[2] = 4;
```

```
y[2] = 3;  
  
Console.WriteLine(countLineSegments(x, y));  
}  
}  
  
// This code is contributed by vt_m.
```

PHP

```
<?php  
// PHP program to find number  
// of horizontal (or vertical  
// line segments needed to  
// connect three points.  
  
// Function to check if the  
// third point forms a  
// rectangle with other  
// two points at corners  
function isBetween( $a, $b, $c)  
{  
    return min($a, $b) <= $c and  
           $c <= max($a, $b);  
}  
  
// Returns true if point k  
// can be used as a joining  
// point to connect using  
// two line segments  
function canJoin($x, $y, $i, $j, $k)  
{  
    // Check for the valid  
    // polyline with two segments  
    return ($x[$k] == $x[$i] or  
           $x[$k] == $x[$j]) and  
           isBetween($y[$i], $y[$j], $y[$k]) or  
           ($y[$k] == $y[$i] or  
            $y[$k] == $y[$j]) and  
           isBetween($x[$i], $x[$j], $x[$k]);  
}  
  
function countLineSegments( $x, $y)  
{  
    // Check whether the X-coordinates  
    // or Y-coordinates are same.  
    if (($x[0] == $x[1] and $x[1] == $x[2]) or
```

```
($y[0] == $y[1] and $y[1] == $y[2]))
return 1;

// Iterate over all pairs to
// check for two line segments
else if (canJoin($x, $y, 0, 1, 2) or
         canJoin($x, $y, 0, 2, 1) ||
         canJoin($x, $y, 1, 2, 0))
return 2;

// Otherwise answer is three.
else
    return 3;
}

// Driver code
$x = array();
$y = array();
$x[0] = -1; $y[0] = -1;
$x[1] = -1; $y[1] = 3;
$x[2] = 4; $y[2] = 3;
echo countLineSegments($x, $y);

// This code is contributed by anuj_67.
?>
```

Output :

2

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-no-segments-polyline/>

Chapter 140

Number of jump required of given length to reach a point of form $(d, 0)$ from origin in 2D plane

Number of jump required of given length to reach a point of form $(d, 0)$ from origin in 2D plane - GeeksforGeeks

Given three positive integers **a**, **b** and **d**. You are currently at origin $(0, 0)$ on infinite 2D coordinate plane. You are allowed to jump on any point in the 2D plane at euclidean distance either equal to **a** or **b** from your current position. The task is to find the minimum number of jump required to reach $(d, 0)$ from $(0, 0)$.

Examples:

```
Input : a = 2, b = 3, d = 1
Output : 2
First jump of length a = 2, (0, 0) -> (1/2, √15/2)
Second jump of length a = 2, (1/2, √15/2) -> (1, 0)
Thus, only two jump are required to reach
(1, 0) from (0, 0).
```

```
Input : a = 3, b = 4, d = 11
Output : 3
(0, 0) -> (4, 0) using length b = 4
(4, 0) -> (8, 0) using length b = 4
(8, 0) -> (11, 0) using length a = 3
```

First, observe we don't need to find the intermediate points, we only want the minimum number of jumps required.

So, from (0, 0) we will move in direction of (d, 0) i.e vertically with the jump of length equal to **max(a, b)** until the Euclidean distance between the current position coordinate and (d, 0) either became less than **max(a, b)** or equal to 0. This will increase the minimum number of jumps required by 1 at each jump. So this value can be found by **floor(d/max(a, b))**.

Now, for the rest of the distance, if (d, 0) is **a** Euclidean distance away, we will make one jump of length **a** and reach the (d, 0). So, the minimum number of jumps required will be increased by 1 in this case.

Now, let's solve the case if rest of the distance is not equal to **a**. Let the rest of distance left be **x**. Also, observe **x** will be greater than 0 and less than **max(a, b)**, therefore, $0 < x < 2*\max(a, b)$. We can reach to (d, 0) in 2 steps.

How ?

Lets try to build a triangle ABC such that A is our current position, B is target position i.e (d, 0) and C will be the point such that AC = BC = **max(a, b)**. And this is possible to triangle because sum of two side AC + BC is greater than third side AB. So, one jump is from A to C and another jump from C to B.

Below is the implementation of this approach:

C++

```
#include <bits/stdc++.h>
using namespace std;

// Return the minimum jump of length either a or b
// required to reach (d, 0) from (0, 0).
int minJumps(int a, int b, int d)
{
    // Assigning maximum of a and b to b
    // and assigning minimum of a and b to a.
    int temp = a;
    a = min(a, b);
    b = max(temp, b);

    // if d is greater than or equal to b.
    if (d >= b)
        return (d + b - 1) / b;

    // if d is 0
    if (d == 0)
        return 0;

    // if d is equal to a.
    if (d == a)
        return 1;

    // else make triangle, and only 2
    // steps required.
```

```
    return 2;
}

int main()
{
    int a = 3, b = 4, d = 11;
    cout << minJumps(a, b, d) << endl;
    return 0;
}
```

Java

```
// Java code to find the minimum number
// of jump required to reach
// (d, 0) from (0, 0).
import java.io.*;

class GFG {

    // Return the minimum jump of length either a or b
    // required to reach (d, 0) from (0, 0).
    static int minJumps(int a, int b, int d)
    {
        // Assigning maximum of a and b to b
        // and assigning minimum of a and b to a.
        int temp = a;
        a = Math.min(a, b);
        b = Math.max(temp, b);

        // if d is greater than or equal to b.
        if (d >= b)
            return (d + b - 1) / b;

        // if d is 0
        if (d == 0)
            return 0;

        // if d is equal to a.
        if (d == a)
            return 1;

        // else make triangle, and only 2
        // steps required.
        return 2;
    }

    // Driver code
    public static void main(String[] args)
```

```
{  
    int a = 3, b = 4, d = 11;  
    System.out.println(minJumps(a, b, d));  
}  
}  
  
// This code is contributed by vt_m
```

Python3

```
# Python3 code to find the minimum  
# number of jump required to reach  
# (d, 0) from (0, 0)  
  
def minJumps(a, b, d):  
  
    temp = a  
    a = min(a, b)  
    b = max(temp, b)  
  
    if (d >= b):  
        return (d + b - 1) / b  
  
    # if d is 0  
    if (d == 0):  
        return 0  
  
    # if d is equal to a.  
    if (d == a):  
        return 1  
  
    # else make triangle, and  
    # only 2 steps required.  
    return 2  
  
# Driver Code  
a, b, d = 3, 4, 11  
print (int(minJumps(a, b, d)))  
  
# This code is contributed by _omg
```

C#

```
// C# code to find the minimum number  
// of jump required to reach  
// (d, 0) from (0, 0).  
using System;
```

```
class GFG {

    // Return the minimum jump of length either a or b
    // required to reach (d, 0) from (0, 0).
    static int minJumps(int a, int b, int d)
    {
        // Assigning maximum of a and b to b
        // and assigning minimum of a and b to a.
        int temp = a;
        a = Math.Min(a, b);
        b = Math.Max(temp, b);

        // if d is greater than or equal to b.
        if (d >= b)
            return (d + b - 1) / b;

        // if d is 0
        if (d == 0)
            return 0;

        // if d is equal to a.
        if (d == a)
            return 1;

        // else make triangle, and only 2
        // steps required.
        return 2;
    }

    // Driver code
    public static void Main()
    {
        int a = 3, b = 4, d = 11;
        Console.WriteLine(minJumps(a, b, d));
    }
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP code to find the minimum
// number of jump required to
// reach (d, 0) from (0, 0).

// Return the minimum jump
```

```
// of length either a or b
// required to reach (d, 0)
// from (0, 0).
function minJumps( $a, $b, $d)
{

    // Assigning maximum of
    // a and b to b and
    // assigning minimum of
    // a and b to a.
    $temp = $a;
    $a = min($a, $b);
    $b = max($temp, $b);

    // if d is greater than
    // or equal to b.
    if ($d >= $b)
        return ($d + $b - 1) / $b;

    // if d is 0
    if ($d == 0)
        return 0;

    // if d is equal to a.
    if ($d == $a)
        return 1;

    // else make triangle,
    // and only 2
    // steps required.
    return 2;
}

// Driver Code
$a = 3;
$b = 4;
$d = 11;
echo floor(minJumps($a, $b, $d));

// This code is contributed by anuj_67.
?>
```

Output

3

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/number-jump-required-given-length-reach-point-form-d-0-origin-2d-plane/>

Chapter 141

Number of ordered points pair satisfying line equation

Number of ordered points pair satisfying line equation - GeeksforGeeks

Given an array of n integers, slope of a line i.e., m and the intercept of the line i.e c, Count the number of ordered pairs(i, j) of points where $i \neq j$, such that point (A_i, A_j) satisfies the line formed with given slope and intercept.

Note : The equation of the line is $y = mx + c$, where m is the slope of the line and c is the intercept.

Examples :

Input : $m = 1, c = 1, arr[] = [1, 2, 3, 4, 2]$

Output : 5 ordered points

Explanation : The equation of the line with given slope and intercept is : $y = x + 1$. The Number of pairs (i, j), for which (arr_i, arr_j) satisfies the above equation of the line are : (1, 2), (1, 5), (2, 3), (3, 4), (5, 3).

Input : $m = 2, c = 1, arr[] = [1, 2, 3, 4, 2, 5]$

Output : 3 ordered points

Method 1 (Brute Force):

Generate all possible pairs (i, j) and check if a particular ordered pair (i, j) is such that, (arr_i, arr_j) satisfies the given equation of the line $y = mx + c$, and $i \neq j$. If the point is valid(a point is valid if the above condition is satisfied), increment the counter which stores the total number of valid points.

Source

<https://www.geeksforgeeks.org/number-ordered-points-pair-satisfying-line-equation/>

C++

```
// CPP code to count the number of ordered
// pairs satisfying Line Equation
#include <bits/stdc++.h>

using namespace std;

/* Checks if (i, j) is valid, a point (i, j)
   is valid if point (arr[i], arr[j])
   satisfies the equation y = mx + c And
   i is not equal to j*/
bool isValid(int arr[], int i, int j,
             int m, int c)
{
    // check if i equals to j
    if (i == j)
        return false;

    // Equation LHS = y, and RHS = mx + c
    int lhs = arr[j];
    int rhs = m * arr[i] + c;

    return (lhs == rhs);
}

/* Returns the number of ordered pairs
   (i, j) for which point (arr[i], arr[j])
   satisfies the equation of the line
   y = mx + c */
int findOrderedPoints(int arr[], int n,
                      int m, int c)
{
    int counter = 0;

    // for every possible (i, j) check
    // if (a[i], a[j]) satisfies the
    // equation y = mx + c
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            // (firstIndex, secondIndex)
            // is same as (i, j)
            int firstIndex = i, secondIndex = j;

            // check if (firstIndex,
```

```
// secondIndex) is a valid point
if (isValid(arr, firstIndex, secondIndex, m, c))
    counter++;
}
}
return counter;
}

// Driver Code
int main()
{
    int arr[] = { 1, 2, 3, 4, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);

    // equation of line is y = mx + c
    int m = 1, c = 1;
    cout << findOrderedPoints(arr, n, m, c);
    return 0;
}
```

Java

```
// Java code to find number of ordered
// points satisfying line equation
import java.io.*;

public class GFG {

    // Checks if (i, j) is valid,
    // a point (i, j) is valid if
    // point (arr[i], arr[j])
    // satisfies the equation
    // y = mx + c And
    // i is not equal to j
    static boolean isValid(int []arr, int i,
                           int j, int m, int c)
    {

        // check if i equals to j
        if (i == j)
            return false;

        // Equation LHS = y,
        // and RHS = mx + c
        int lhs = arr[j];
        int rhs = m * arr[i] + c;
```

```
        return (lhs == rhs);
    }

/* Returns the number of ordered pairs
(i, j) for which point (arr[i], arr[j])
satisfies the equation of the line
y = mx + c */
static int findOrderedPoints(int []arr,
                             int n, int m, int c)
{
    int counter = 0;

    // for every possible (i, j) check
    // if (a[i], a[j]) satisfies the
    // equation y = mx + c
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {

            // (firstIndex, secondIndex)
            // is same as (i, j)
            int firstIndex = i,
                secondIndex = j;

            // check if (firstIndex,
            // secondIndex) is a
            // valid point
            if (isValid(arr, firstIndex,
                        secondIndex, m, c))
                counter++;
        }
    }
    return counter;
}

// Driver Code
public static void main(String args[])
{
    int []arr = { 1, 2, 3, 4, 2 };
    int n = arr.length;

    // equation of line is y = mx + c
    int m = 1, c = 1;
    System.out.print(
        findOrderedPoints(arr, n, m, c));
}
```

```
}
```

```
// This code is contributed by
// Manish Shaw (manishshaw1)
```

Python3

```
# Python code to count the number of ordered
# pairs satisfying Line Equation

# Checks if (i, j) is valid, a point (i, j)
# is valid if point (arr[i], arr[j])
# satisfies the equation y = mx + c And
# i is not equal to j
def isValid(arr, i, j, m, c) :

    # check if i equals to j
    if (i == j) :
        return False

    # Equation LHS = y, and RHS = mx + c
    lhs = arr[j];
    rhs = m * arr[i] + c

    return (lhs == rhs)

# Returns the number of ordered pairs
# (i, j) for which point (arr[i], arr[j])
# satisfies the equation of the line
# y = mx + c */
def findOrderedPoints(arr, n, m, c) :

    counter = 0

    # for every possible (i, j) check
    # if (a[i], a[j]) satisfies the
    # equation y = mx + c
    for i in range(0, n) :
        for j in range(0, n) :
            # (firstIndex, secondIndex)
            # is same as (i, j)
            firstIndex = i
            secondIndex = j

            # check if (firstIndex,
            # secondIndex) is a valid point
            if (isValid(arr, firstIndex,
```

```
        secondIndex, m, c)) :  
    counter = counter + 1  
  
return counter  
  
# Driver Code  
arr = [ 1, 2, 3, 4, 2 ]  
n = len(arr)  
  
# equation of line is y = mx + c  
m = 1  
c = 1  
print (findOrderedPoints(arr, n, m, c))  
  
# This code is contributed by Manish Shaw  
# (manishshaw1)
```

C#

```
// C# code to find number of ordered  
// points satisfying line equation  
using System;  
class GFG {  
  
    // Checks if (i, j) is valid,  
    // a point (i, j) is valid if  
    // point (arr[i], arr[j])  
    // satisfies the equation  
    // y = mx + c And  
    // i is not equal to j  
    static bool isValid(int []arr, int i,  
                        int j, int m, int c)  
    {  
  
        // check if i equals to j  
        if (i == j)  
            return false;  
  
        // Equation LHS = y,  
        // and RHS = mx + c  
        int lhs = arr[j];  
        int rhs = m * arr[i] + c;  
  
        return (lhs == rhs);  
    }  
  
    /* Returns the number of ordered pairs
```

```
(i, j) for which point (arr[i], arr[j])
satisfies the equation of the line
y = mx + c */
static int findOrderedPoints(int []arr, int n,
                             int m, int c)
{
    int counter = 0;

    // for every possible (i, j) check
    // if (a[i], a[j]) satisfies the
    // equation y = mx + c
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {

            // (firstIndex, secondIndex)
            // is same as (i, j)
            int firstIndex = i, secondIndex = j;

            // check if (firstIndex,
            // secondIndex) is a valid point
            if (isValid(arr, firstIndex, secondIndex, m, c))
                counter++;
        }
    }
    return counter;
}

// Driver Code
public static void Main()
{
    int []arr = { 1, 2, 3, 4, 2 };
    int n = arr.Length;

    // equation of line is y = mx + c
    int m = 1, c = 1;
    Console.WriteLine(findOrderedPoints(arr, n, m, c));
}
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

PHP

```
<?php
```

```
// PHP code to count the
// number of ordered pairs
// satisfying Line Equation

/* Checks if (i, j) is valid,
a point (i, j) is valid if
point (arr[i], arr[j]) satisfies
the equation y = mx + c And i
is not equal to j*/
function isValid($arr, $i,
                 $j, $m, $c)
{
    // check if i equals to j
    if ($i == $j)
        return false;

    // Equation LHS = y, and
    // RHS = mx + c
    $lhs = $arr[$j];
    $rhs = $m * $arr[$i] + $c;

    return ($lhs == $rhs);
}

/* Returns the number of
ordered pairs (i, j) for
which point (arr[i], arr[j])
satisfies the equation of
the line y = mx + c */
function findOrderedPoints($arr, $n,
                           $m, $c)
{
    $counter = 0;

    // for every possible (i, j)
    // check if (a[i], a[j])
    // satisfies the equation
    // y = mx + c
    for ($i = 0; $i < $n; $i++)
    {
        for ($j = 0; $j < $n; $j++)
        {
            // (firstIndex, secondIndex)
            // is same as (i, j)
            $firstIndex = $i; $secondIndex = $j;

            // check if (firstIndex,
```

```

        // secondIndex) is a valid point
        if (isValid($arr, $firstIndex,
                    $secondIndex, $m, $c))
            $counter++;
    }
}
return $counter;
}

// Driver Code
$arr = array( 1, 2, 3, 4, 2 );
$n = count($arr);

// equation of line
// is y = mx + c
$m = 1; $c = 1;
echo (findOrderedPoints($arr, $n, $m, $c));

// This code is contributed by
// Manish Shaw (manishshaw1)
?>

```

Output :

5

Time Complexity : $O(n^2)$

Method 2 (Efficient) :

Given a x coordinate of a point, for each x there is a unique value of y and the value of y is nothing but $m * x + c$. So, for each possible x coordinate of the array arr, calculate how many times the unique value of y which satisfies the equation of the line occurs in that array. Store count of all the integers of array, arr in a map. Now, for each value, arr_i , add to the answer, the number of occurrences of $m * arr_i + c$. For a given i, $m * arr_i + c$ occurs x times in the array, then, add x to our counter for total valid points, but need to check that if $arr[i] = m * arr[i] + c$ then, it is obvious that since this occurs x times in the array then one occurrence is at the i^{th} index and rest $(x - 1)$ occurrences are the valid y coordinates so add $(x - 1)$ to our points counter.

C++

```

// CPP code to find number of ordered
// points satisfying line equation
#include <bits/stdc++.h>
using namespace std;

/* Returns the number of ordered pairs

```

```
(i, j) for which point (arr[i], arr[j])
satisfies the equation of the line
y = mx + c */
int findOrderedPoints(int arr[], int n,
                      int m, int c)
{
    int counter = 0;

    // map stores the frequency
    // of arr[i] for all i
    unordered_map<int, int> frequency;

    for (int i = 0; i < n; i++)
        frequency[arr[i]]++;

    for (int i = 0; i < n; i++)
    {
        int xCoordinate = arr[i];
        int yCoordinate = (m * arr[i] + c);

        // if for a[i](xCoordinate),
        // a yCoordinate exists in the map
        // add the frequency of yCoordinate
        // to the counter

        if (frequency.find(yCoordinate) !=
            frequency.end())
            counter += frequency[yCoordinate];

        // check if xCoordinate = yCoordinate,
        // if this is true then since we only
        // want (i, j) such that i != j, decrement
        // the counter by one to avoid points
        // of type (arr[i], arr[i])
        if (xCoordinate == yCoordinate)
            counter--;
    }
    return counter;
}

// Driver Code
int main()
{
    int arr[] = { 1, 2, 3, 4, 2 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int m = 1, c = 1;
    cout << findOrderedPoints(arr, n, m, c);
    return 0;
}
```

}

Output:

5

Time Complexity : $O(n)$

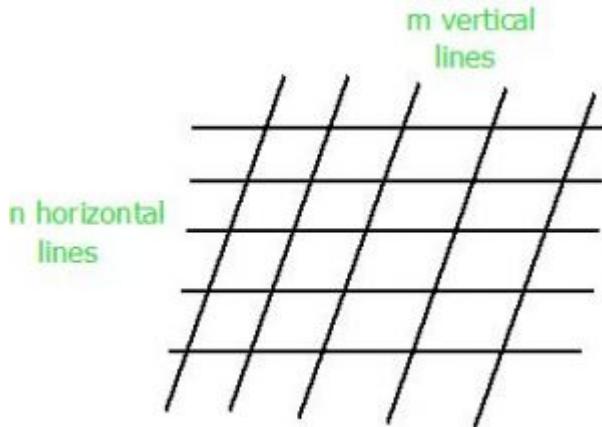
Improved By : [manishshaw1](#)

Chapter 142

Number of parallelograms when n horizontal parallel lines intersect m vertical parallel lines

Number of parallelograms when n horizontal parallel lines intersect m vertical parallel lines
- GeeksforGeeks

Given two positive integers **n** and **m**. The task is to count number of parallelogram that can be formed of any size when n horizontal parallel lines intersect with m vertical parallel lines.



Examples:

```
Input : n = 3, m = 2
Output : 3
2 parallelograms of size 1x1 and 1 parallelogram
of size 2x1.
```

Input : n = 5, m = 5
Output : 100

The idea is to use **Combination**, which state, number of ways to choose k items from given n items is given by nC_r .

To form a parallelogram, we need two horizontal parallel lines and two vertical parallel lines. So, number of ways to choose two horizontal parallel lines are nC_2 and number of ways to choose two vertical parallel lines are mC_2 . So, total number of possible parallelogram will be ${}^nC_2 \times {}^mC_2$.

Below is C++ implementation of this approach:

C++

```
// CPP Program to find number of parallelogram when
// n horizontal parallel lines intersect m vertical
// parallel lines.
#include<bits/stdc++.h>
#define MAX 10
using namespace std;

// Find value of Binomial Coefficient
int binomialCoeff(int C[][][MAX], int n, int k)
{
    // Calculate value of Binomial Coefficient
    // in bottom up manner
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= min(i, k); j++)
        {
            // Base Cases
            if (j == 0 || j == i)
                C[i][j] = 1;

            // Calculate value using previously
            // stored values
            else
                C[i][j] = C[i-1][j-1] + C[i-1][j];
        }
    }
}

// Return number of parallelogram when n horizontal
// parallel lines intersect m vertical parallel lines.
int countParallelogram(int n, int m)
{
    int C[MAX][MAX] = { 0 };
    binomialCoeff(C, max(n, m), 2);
```

```
    return C[n][2] * C[m][2];
}

// Driver Program
int main()
{
    int n = 5, m = 5;
    cout << countParallelogram(n, m) << endl;
    return 0;
}
```

Java

```
// Java Program to find number of parallelogram when
// n horizontal parallel lines intersect m vertical
// parallel lines.
class GFG
{
    static final int MAX = 10;

    // Find value of Binomial Coefficient
    static void binomialCoeff(int C[][], int n, int k)
    {
        // Calculate value of Binomial Coefficient
        // in bottom up manner
        for (int i = 0; i <= n; i++)
        {
            for (int j = 0; j <= Math.min(i, k); j++)
            {
                // Base Cases
                if (j == 0 || j == i)
                    C[i][j] = 1;

                // Calculate value using previously
                // stored values
                else
                    C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
            }
        }
    }

    // Return number of parallelogram when n horizontal
    // parallel lines intersect m vertical parallel lines.
    static int countParallelogram(int n, int m)
    {
        int C[][] = new int[MAX][MAX];

        binomialCoeff(C, Math.max(n, m), 2);

        return C[n][2] * C[m][2];
    }
}
```

```
        return C[n][2] * C[m][2];
    }

    // Driver code
    public static void main(String arg[])
    {
        int n = 5, m = 5;
        System.out.println(countParallelogram(n, m));
    }
}

// This code is contributed By Anant Agarwal.
```

C#

```
// C# Program to find number of parallelogram when
// n horizontal parallel lines intersect m vertical
// parallel lines.
using System;

class GFG
{
    static int MAX = 10;

    // Find value of Binomial Coefficient
    static void binomialCoeff(int[,] C, int n, int k)
    {
        // Calculate value of Binomial Coefficient
        // in bottom up manner
        for (int i = 0; i <= n; i++)
        {
            for (int j = 0; j <= Math.Min(i, k); j++)
            {
                // Base Cases
                if (j == 0 || j == i)
                    C[i, j] = 1;

                // Calculate value using previously
                // stored values
                else
                    C[i, j] = C[i - 1, j - 1] + C[i - 1, j];
            }
        }
    }

    // Return number of parallelogram when n horizontal
    // parallel lines intersect m vertical parallel lines.
}
```

```
static int countParallelogram(int n, int m)
{
    int [,]C = new int[MAX, MAX];

    binomialCoeff(C, Math.Max(n, m), 2);

    return C[n, 2] * C[m, 2];
}

// Driver code
public static void Main()
{
    int n = 5, m = 5;
    Console.WriteLine(countParallelogram(n, m));
}
}

// This code is contributed By vt_m.
```

Output:

100

Source

<https://www.geeksforgeeks.org/number-of-parallelograms-when-n-horizontal-parallel-lines-intersect-m-vertical-par>

Chapter 143

Number of possible pairs of Hypotenuse and Area to form right angled triangle

Number of possible pairs of Hypotenuse and Area to form right angled triangle - Geeks-forGeeks

Given two arrays H and S. The array H[] contains the length of the hypotenuse and the array S[] contains Area of a right-angled triangle. The task is to find all possible pairs of (H, S) such that we can construct a right-angled triangle with hypotenuse H and area S.

Examples:

```
Input : H[] = {1, 6, 4} ; S[] = {23, 3, 42, 14}
Output : 2
Possible pairs are {6, 3} {4, 3}
```

```
Input : H[] = {1, 6, 4, 3} ; S[] = {23, 3, 42, 5}
Output : 3
Possible pairs are {6, 3} {6, 5} {4, 3}
```

Say,

a = Base of Right Angled Triangle

b = Height of the Right Angled Triangle

Therefore,

```
area S = (a*b)/2
or, 4*S*S=a*a*b*b
```

Also,

$$a^2 + b^2 = H^2$$

Therefore,

$$4*S^2 = a^2(H^2 - a^2)$$

Solving this quadratic equation in a^2 and putting discriminant $>= 0$ (condition for a to exist). We will get,

$$H^2 \geq 4*S$$

For a right-angled triangle to exist with hypotenuse H and area S .

Naive Approach: The naive approach is to find all possible pairs of (H, S) and check if they satisfy the condition, $H^2 \geq 4*S$. Count the number of pairs which satisfies this condition and print the count.

Below is the implementation of the naive approach:

C++

```
#include <iostream>
using namespace std;

// Function to check the condition
bool check(int H, int S)
{
    // Condition for triangle to exist
    return H * H >= 4 * S;
}

// Function to find all pairs
int findPairs(int H[], int n, int S[], int m)
{
    int count = 0;

    // Checking all possible pairs
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (check(H[i], S[j]))
                count++;
    }
}
```

```
        }
    }

    return count;
}

// Driver code
int main()
{
    int H[] = { 1, 6, 4 };
    int n = sizeof(H)/sizeof(H[0]);

    int S[] = { 23, 3, 42, 14 };
    int m = sizeof(S)/sizeof(S[0]);

    cout<<findPairs(H, n, S, m);

    return 0;
}
```

Java

```
class GFG
{

    // Function to check the condition
    static boolean check(int H, int S)
    {
        // Condition for triangle to exist
        return H * H >= 4 * S;
    }

    // Function to find all pairs
    static int findPairs(int H[], int n,
                         int S[], int m)
    {
        int count = 0;

        // Checkinag all possible pairs
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                if (check(H[i], S[j]))
                    count++;
            }
        }
    }
}
```

```
        return count;
    }

// Driver code
public static void main(String args[])
{
    int H[] = { 1, 6, 4 };
    int n = H.length;

    int S[] = { 23, 3, 42, 14 };
    int m = S.length;

    System.out.println(findPairs(H, n, S, m));
}
}

// This code is contributed
// by ankita_saini
```

Python3

```
# Python 3 implementation
# of above approach

# Function to check the condition
def check(H, S) :

    # Condition for triangle to exist
    return H * H >= 4 * S

# Function to find all pairs
def findPairs(H, n, S, m):

    count = 0

    # Checking all possible pairs
    for i in range(n) :
        for j in range(m) :
            if check(H[i], S[j]) :
                count += 1

    return count

# Driver Code
if __name__ == "__main__":
    H = [ 1, 6, 4]
    n = len(H)
```

```
S = [ 23, 3, 42, 14]
m = len(S)

# function calling
print(findPairs(H, n, S,m))

# This code is contributed by ANKITRAI1
```

Output:

2

Efficient Approach: An efficient approach is to sort both the arrays available in increasing order. Then, for every possible length of the hypotenuse, apply Binary search to find the maximum area which satisfies the necessary condition.

Say, after the Binary search maximum possible area is available at index 4 in the array S[]. Then we can form 4 such possible pairs since all area less than that at index 4 will also be satisfying the condition.

Below is the implementation of the efficient approach:

C++

```
#include <bits/stdc++.h>
using namespace std;

// Function to check the condition
bool check(int H, int S)
{
    // Condition for triangle to exist
    return H * H >= 4 * S;
}

// Function to find all pairs
int findPairs(int H[], int n, int S[], int m)
{
    int count = 0;

    // Sort both the arrays
    sort(H, H + n);
    sort(S, S + m);

    // To keep track of last possible Area
    int index = -1;
```

```
for (int i = 0; i < n; i++) {
    // Apply Binary Search for
    // each Hypotenuse Length
    int start = 0;
    int end = m - 1;

    while (start <= end) {
        int mid = start + (end - start) / 2;
        if (check(H[i], S[mid])) {
            index = mid;
            start = mid + 1;
        }
        else {
            end = mid - 1;
        }
    }

    // Check if we get any
    // possible Area or Not
    if (index != -1) {
        // All area less than area[index]
        // satisfy property
        count += index + 1;
    }
}

return count;
}

// Driver code
int main()
{
    int H[] = { 1, 6, 4 };
    int n = sizeof(H)/sizeof(H[0]);

    int S[] = { 23, 3, 42, 14 };
    int m = sizeof(S)/sizeof(S[0]);

    cout<<findPairs(H, n, S, m);

    return 0;
}
```

Output:

Improved By : [ankita_saini](#), [ANKITRAI1](#)

Source

<https://www.geeksforgeeks.org/number-of-possible-pairs-of-hypotenuse-and-area-to-form-right-angled-triangle/>

Chapter 144

Number of quadrilaterals possible from the given points

Number of quadrilaterals possible from the given points - GeeksforGeeks

Given four points (x, y) in the cartesian coordinate. Find the possible no of quadrilaterals than can be formed by joining all the four points.

Examples:

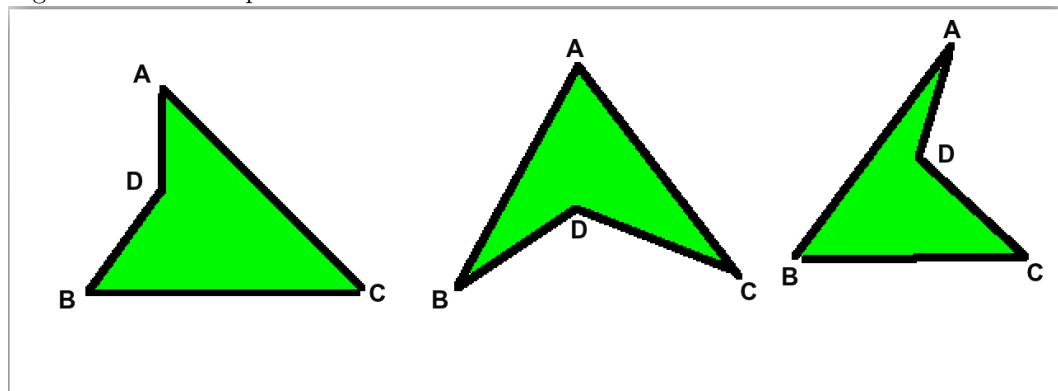
Input: $A=(0, 9)$, $B=(-1, 0)$, $C=(5, -1)$, $D=(5, 9)$

Output: Only one quadrilateral is possible (ABCD) in any orientation

Input: $A=(0, 9)$, $B=(-1, 0)$, $C=(5, -1)$, $D=(0, 3)$

Output: 3 quadrilaterals are possible (ABCD), (ADBC), (ABDC)

Figure for 2nd example:



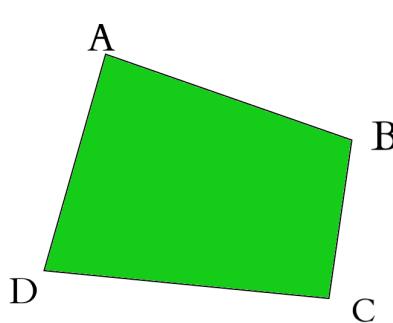
Approach:

1. We need to check whether any of the given points are same. **If yes, no of quadrilateral = 0**
2. Then we need to check whether any of the 3 points of the given 4 points are collinear or not. **If yes, no of quadrilateral=0.** Check [Program to check if three points are collinear](#) link to check collinearity of 3 points.
3. (a) if its a **convex quadrilateral** then there is only **one possible quadrilateral**.
 (b) if its a **concave quadrilateral** then there are **3 possible quadrilaterals**.

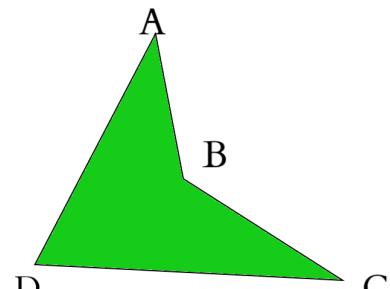
This can be determined by [How to check if two given line segments intersect?](#) of diagonals.
 In case of a **convex quadrilateral**, the diagonals will intersect whereas in case of a concave quadrilateral the diagonals won't intersect.

since we don't know the orientation of the points, we can't specifically determine the diagonals so all the distinct line segments(no common points in the two line segments) of the quadrilateral and determine whether they intersect or not.

Refer to the figure to understand how to determine the type of quadrilateral :



Convex quadrilateral



Concave quadrilateral

Convex quadrilateral:

line AB and line DC do not intersect
 line AD and line BC do not intersect
 line AC and line BD intersect
 so total no of intersection= 1

Concave quadrilateral

line AB and line DC do not intersect
 line AD and line BC do not intersect
 line AC and line BD intersect
 so total no of intersection= 0

if no of intersection = 1, its a convex quadrilateral so no of possibility= 1
if no of intersection = 0, its a concave quadrilateral so no of possibilities = 3

C++

```
#include <iostream>
using namespace std;

struct Point // points
{
    int x;
    int y;
};

// determines the orientation of points
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);

    if (val == 0)
        return 0;
    return (val > 0) ? 1 : 2;
}

// check whether the distinct line segments intersect
bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);

    if (o1 != o2 && o3 != o4)
        return true;

    return false;
}

// check if points overlap(similar)
bool similar(Point p1, Point p2)
{
    // it is same, we are returning false because
    // quadrilateral is not possible in this case
    if (p1.x == p2.x && p1.y == p2.y)
        return false;

    // it is not same, So there is a
```

```
// possibility of a quadrilateral
return true;
}

// check for collinearity
bool collinear(Point p1, Point p2, Point p3)
{
    int x1 = p1.x, y1 = p1.y;
    int x2 = p2.x, y2 = p2.y;
    int x3 = p3.x, y3 = p3.y;

    // it is collinear, we are returning false
    // because quadrilateral is not possible in this case
    if ((y3 - y2) * (x2 - x1) == (y2 - y1) * (x3 - x2))
        return false;

    // it is not collinear, So there
    // is a possibility of a quadrilateral
    else
        return true;
}

int no_of_quads(Point p1, Point p2, Point p3, Point p4)
{
    // ** Checking for cases where no quadrilateral = 0 **

    // check if any of the points are same
    bool same = true;
    same = same & similar(p1, p2);
    same = same & similar(p1, p3);
    same = same & similar(p1, p4);
    same = same & similar(p2, p3);
    same = same & similar(p2, p4);
    same = same & similar(p3, p4);

    // similar points exist
    if (same == false)
        return 0;

    // check for collinearity
    bool coll = true;
    coll = coll & collinear(p1, p2, p3);
    coll = coll & collinear(p1, p2, p4);
    coll = coll & collinear(p1, p3, p4);
    coll = coll & collinear(p2, p3, p4);

    // points are collinear
    if (coll == false)
```

```
    return 0;

/** Checking for cases where no of quadrilaterals= 1 or 3 **

int check = 0;

if (doIntersect(p1, p2, p3, p4))
    check = 1;
if (doIntersect(p1, p3, p2, p4))
    check = 1;
if (doIntersect(p1, p2, p4, p3))
    check = 1;

if (check == 0)
    return 3;
return 1;
}

// Driver code
int main()
{
    struct Point p1, p2, p3, p4;
    // A =(0, 9), B = (-1, 0), C = (5, -1), D=(5, 9)
    p1.x = 0, p1.y = 9;
    p2.x = -1, p2.y = 0;
    p3.x = 5, p3.y = -1;
    p4.x = 5, p4.y = 9;
    cout << no_of_quads(p1, p2, p3, p4) << endl;

    // A=(0, 9), B=(-1, 0), C=(5, -1), D=(0, 3)
    p1.x = 0, p1.y = 9;
    p2.x = -1, p2.y = 0;
    p3.x = 5, p3.y = -1;
    p4.x = 0, p4.y = 3;
    cout << no_of_quads(p1, p2, p3, p4) << endl;

    // A=(0, 9), B=(0, 10), C=(0, 11), D=(0, 12)
    p1.x = 0, p1.y = 9;
    p2.x = 0, p2.y = 10;
    p3.x = 0, p3.y = 11;
    p4.x = 0, p4.y = 12;
    cout << no_of_quads(p1, p2, p3, p4) << endl;

    // A=(0, 9), B=(0, 9), C=(5, -1), D=(0, 3)
    p1.x = 0, p1.y = 9;
    p2.x = 0, p2.y = 9;
    p3.x = 5, p3.y = -1;
    p4.x = 0, p4.y = 3;
```

```
    cout << no_of_quads(p1, p2, p3, p4) << endl;  
  
    return 0;  
}
```

Java

```
class GFG  
{  
    static class Point // points  
    {  
        int x;  
        int y;  
    }  
  
    // determines the orientation of points  
    static int orientation(Point p, Point q,  
                           Point r)  
    {  
        int val = (q.y - p.y) * (r.x - q.x) -  
                  (q.x - p.x) * (r.y - q.y);  
  
        if (val == 0)  
            return 0;  
        return (val > 0) ? 1 : 2;  
    }  
  
    // check whether the distinct  
    // line segments intersect  
    static boolean doIntersect(Point p1, Point q1,  
                               Point p2, Point q2)  
    {  
        int o1 = orientation(p1, q1, p2);  
        int o2 = orientation(p1, q1, q2);  
        int o3 = orientation(p2, q2, p1);  
        int o4 = orientation(p2, q2, q1);  
  
        if (o1 != o2 && o3 != o4)  
            return true;  
  
        return false;  
    }  
  
    // check if points overlap(similar)  
    static boolean similar(Point p1, Point p2)  
    {  
  
        // it is same, we are returning
```

```
// false because quadrilateral is
// not possible in this case
if (p1.x == p2.x && p1.y == p2.y)
    return false;

// it is not same, So there is a
// possibility of a quadrilateral
return true;
}

// check for collinearity
static boolean collinear(Point p1, Point p2,
                        Point p3)
{
    int x1 = p1.x, y1 = p1.y;
    int x2 = p2.x, y2 = p2.y;
    int x3 = p3.x, y3 = p3.y;

    // it is collinear, we are returning
    // false because quadrilateral is not
    // possible in this case
    if ((y3 - y2) *
        (x2 - x1) == (y2 - y1) *
        (x3 - x2))
        return false;

    // it is not collinear, So there
    // is a possibility of a quadrilateral
    else
        return true;
}

static int no_of_quads(Point p1, Point p2,
                      Point p3, Point p4)
{
    // Checking for cases where
    // no quadrilateral = 0

    // check if any of the
    // points are same
    boolean same = true;
    same = same & similar(p1, p2);
    same = same & similar(p1, p3);
    same = same & similar(p1, p4);
    same = same & similar(p2, p3);
    same = same & similar(p2, p4);
    same = same & similar(p3, p4);
```

```
// similar points exist
if (same == false)
    return 0;

// check for collinearity
boolean coll = true;
coll = coll & collinear(p1, p2, p3);
coll = coll & collinear(p1, p2, p4);
coll = coll & collinear(p1, p3, p4);
coll = coll & collinear(p2, p3, p4);

// points are collinear
if (coll == false)
    return 0;

// Checking for cases where
// no of quadrilaterals= 1 or 3

int check = 0;

if (doIntersect(p1, p2, p3, p4))
    check = 1;
if (doIntersect(p1, p3, p2, p4))
    check = 1;
if (doIntersect(p1, p2, p4, p3))
    check = 1;

if (check == 0)
    return 3;
return 1;
}

// Driver code
public static void main(String args[])
{
    Point p1, p2, p3, p4;
    p1 = new Point();
    p2 = new Point();
    p3 = new Point();
    p4 = new Point();

    // A =(0, 9), B = (-1, 0),
    // C = (5, -1), D=(5, 9)
    p1.x = 0; p1.y = 9;
    p2.x = -1; p2.y = 0;
    p3.x = 5; p3.y = -1;
    p4.x = 5; p4.y = 9;
    System.out.println(no_of_quads(p1, p2, p3, p4));
}
```

```
// A=(0, 9), B=(-1, 0),
// C=(5, -1), D=(0, 3)
p1.x = 0; p1.y = 9;
p2.x = -1; p2.y = 0;
p3.x = 5; p3.y = -1;
p4.x = 0; p4.y = 3;
System.out.println(no_of_quads(p1, p2, p3, p4));

// A=(0, 9), B=(0, 10),
// C=(0, 11), D=(0, 12)
p1.x = 0; p1.y = 9;
p2.x = 0; p2.y = 10;
p3.x = 0; p3.y = 11;
p4.x = 0; p4.y = 12;
System.out.println(no_of_quads(p1, p2, p3, p4));

// A=(0, 9), B=(0, 9),
// C=(5, -1), D=(0, 3)
p1.x = 0; p1.y = 9;
p2.x = 0; p2.y = 9;
p3.x = 5; p3.y = -1;
p4.x = 0; p4.y = 3;
System.out.println(no_of_quads(p1, p2, p3, p4));
}

}

// This code is contributed
// by Arnab Kundu
```

Output:

```
1
3
0
0
```

Improved By : [andrew1234](#)

Source

<https://www.geeksforgeeks.org/number-of-quadrilaterals-possible-from-the-given-points/>

Chapter 145

Number of rectangles in N*M grid

Number of rectangles in N*M grid - GeeksforGeeks

We are given a N*M grid, print the number of rectangles in it.

Examples:

Input : N = 2, M = 2

Output : 9

There are 4 rectangles of size 1 x 1.

There are 2 rectangles of size 1 x 2

There are 2 rectangles of size 2 x 1

There is one rectangle of size 1 x 1.

Input : N = 5, M = 4

Output : 150

Input : N = 4, M = 3

Output: 60

We have discussed [counting number of squares in a n x m grid](#),

Let us derive a formula for number of rectangles.

If the grid is 1×1 , there is 1 rectangle.

If the grid is 2×1 , there will be $2 + 1 = 3$ rectangles

If the grid is 3×1 , there will be $3 + 2 + 1 = 6$ rectangles.

we can say that for $N \times 1$ there will be $N + (N-1) + (N-2) \dots + 1 = (N)(N+1)/2$ rectangles

If we add one more column to $N \times 1$, firstly we will have as many rectangles in the 2nd column as the first,

and then we have that same number of $2 \times M$ rectangles.

$$\text{So } N \times 2 = 3(N)(N+1)/2$$

After deducing this we can say

$$\text{For } N^*M \text{ we'll have } (M)(M+1)/2 (N)(N+1)/2 = M(M+1)(N)(N+1)/4$$

So the formula for total rectangles will be $M(M+1)(N)(N+1)/4$

C++

```
// C++ program to count number of rectangles
// in a n x m grid
#include <bits/stdc++.h>
using namespace std;

int rectCount(int n, int m)
{
    return (m * n * (n + 1) * (m + 1)) / 4;
}

/* driver code */
int main()
{
    int n = 5, m = 4;
    cout << rectCount(n, m);
    return 0;
}
```

Java

```
// JAVA Code to count number of
// rectangles in N*M grid
import java.util.*;

class GFG {

    public static long rectCount(int n, int m)
    {
        return (m * n * (n + 1) * (m + 1)) / 4;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int n = 5, m = 4;
        System.out.println(rectCount(n, m));
    }
}
```

```
// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
# Python3 program to count number
# of rectangles in a n x m grid

def rectCount(n, m):

    return (m * n * (n + 1) * (m + 1)) // 4

# Driver code
n, m = 5, 4
print(rectCount(n, m))

# This code is contributed by Anant Agarwal.
```

C#

```
// C# Code to count number of
// rectangles in N*M grid
using System;

class GFG {

    public static long rectCount(int n, int m)
    {
        return (m * n * (n + 1) * (m + 1)) / 4;
    }

    // Driver program
    public static void Main()
    {
        int n = 5, m = 4;
        Console.WriteLine(rectCount(n, m));
    }
}

// This code is contributed by Anant Agarwal.
```

PHP

```
<?php
// PHP program to count
// number of rectangles
```

```
// in a n x m grid

function rectCount($n, $m)
{
    return ($m * $n *
            ($n + 1) *
            ($m + 1)) / 4;
}

// Driver Code
$n = 5;
$m = 4;
echo rectCount($n, $m);

// This code is contributed
// by ajit
?>
```

Output:

150

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/number-rectangles-nm-grid/>

Chapter 146

Number of rectangles in a circle of radius R

Number of rectangles in a circle of radius R - GeeksforGeeks

Given a circular sheet of radius R and the task is to find the total number of rectangles with integral length and width that can be cut from the circular sheet, one at a time.

Examples:

Input: R = 2

Output: 8

8 rectangles can be cut from a circular sheet of radius 2.

These are: 1×1, 1×2, 2×1, 2×2, 1×3, 3×1, 2×3, 3×2.

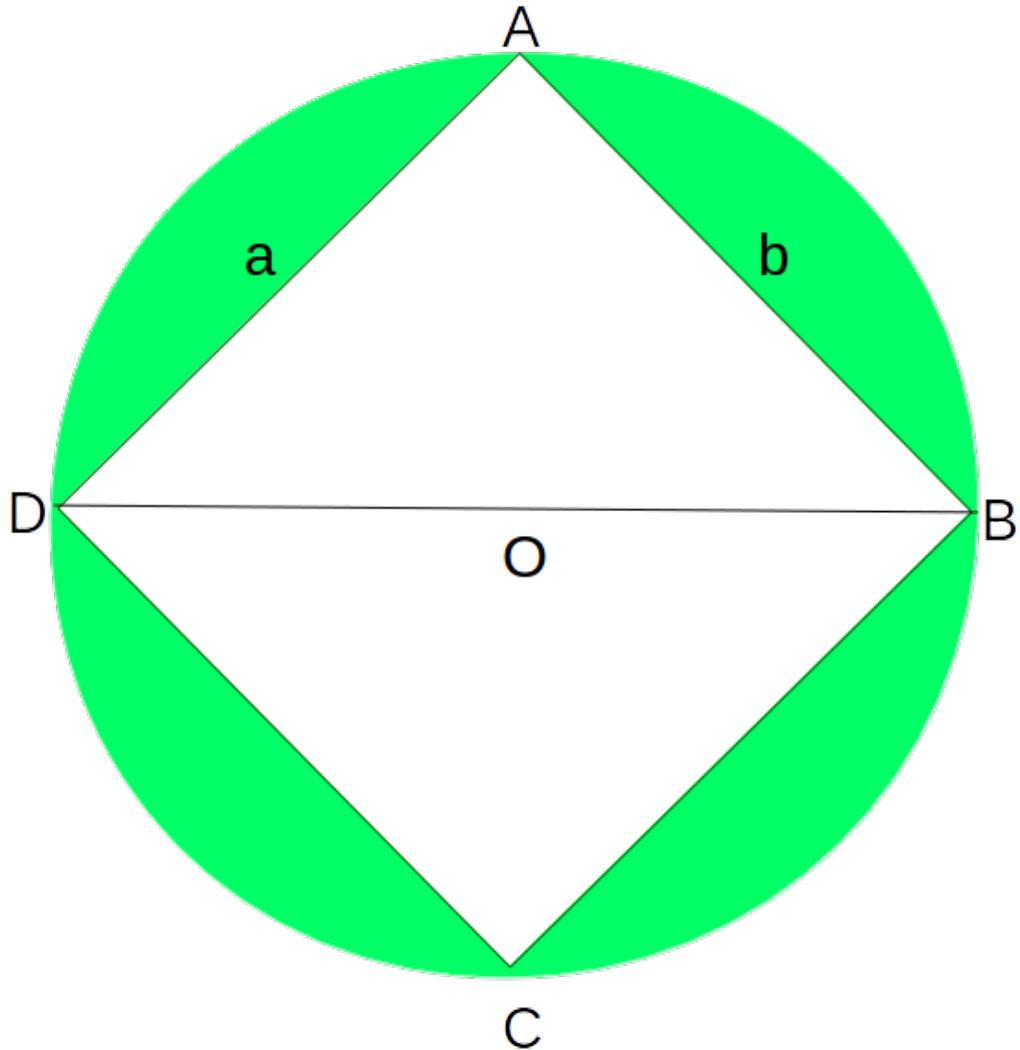
Input: R = 1

Output: 1

Only one rectangle with dimensions 1 X 1 is possible.

Approach

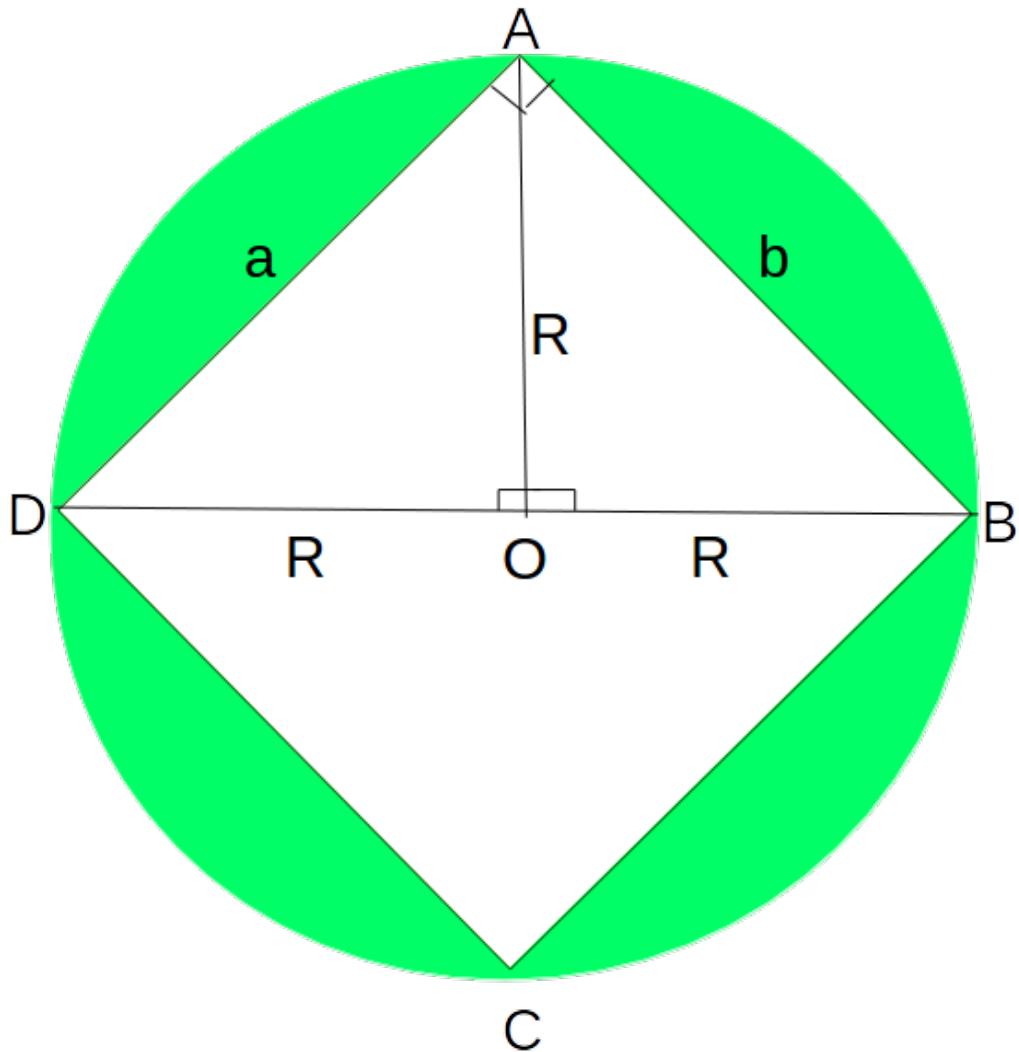
Consider the following diagram,



It's easy to see, that ABCD is the largest rectangle that can be formed in the given circle with radius R and centre O, having dimensions $a \times b$

Drop a perpendicular AO such that, $\angle AOD = \angle AOB = 90^\circ$

Consider the following diagram for further analysis,



Consider triangles AOD and AOB ,

In these triangles,

$AO = AO$ (Common Side)

$\angle AOD = \angle AOB = 90^\circ$

$OD = OB = R$

Thus, by SAS congruence $\triangle AOD \cong \triangle AOB$

$AD = AB$ by CPCTC (i.e Corresponding Parts on Congruent Triangles)

or, $a = b$

\Rightarrow The rectangle $ABCD$ is a square

The diameter BD is the maximum diagonal the rectangle can have to be able to be cut from the Circular Sheet.

Thus, all the combinations of a and b can be checked to form all possible rectangles, and if the diagonal of any such rectangle is less than or equal to the length of the diagonal of the largest rectangle formed (i.e $2 * R$, where R is the Radius of the circle as explained above)

Now, the maximum length of a and b will always be strictly less than the diameter of the circle so all possible values of a and b will lie in the closed interval $[1, (2 * R - 1)]$.

Below is the implementation of the above approach:

C++

```
// C++ program to find the number of rectangles
// that can be cut from a circle of Radius R
#include <bits/stdc++.h>
using namespace std;

// Function to return the total possible
// rectangles that can be cut from the circle
int countRectangles(int radius)
{

    int rectangles = 0;

    // Diameter = 2 * Radius
    int diameter = 2 * radius;

    // Square of diameter which is the square
    // of the maximum length diagonal
    int diameterSquare = diameter * diameter;

    // generate all combinations of a and b
    // in the range (1, (2 * Radius - 1))(Both inclusive)
    for (int a = 1; a < 2 * radius; a++) {
        for (int b = 1; b < 2 * radius; b++) {

            // Calculate the Diagnal length of
            // this rectange
            int diagonalLengthSquare = (a * a + b * b);

            // If this rectangle's Diagonal Length
            // is less than the Diameter, it is a
            // valid rectangle, thus increment counter
            if (diagonalLengthSquare <= diameterSquare) {
                rectangles++;
            }
        }
    }
}
```

```
    }

    return rectangles;
}

// Driver Code
int main()
{

    // Radius of the circle
    int radius = 2;

    int totalRectangles;
    totalRectangles = countRectangles(radius);
    cout << totalRectangles << " rectangles can be"
        << "cut from a circle of Radius " << radius;
    return 0;
}
```

Java

```
// Java program to find the
// number of rectangles that
// can be cut from a circle
// of Radius R
import java.io.*;

class GFG
{

    // Function to return
    // the total possible
    // rectangles that can
    // be cut from the circle
    static int countRectangles(int radius)
    {
        int rectangles = 0;

        // Diameter = 2 * Radius
        int diameter = 2 * radius;

        // Square of diameter
        // which is the square
        // of the maximum length
        // diagonal
        int diameterSquare = diameter *
            diameter;
```

```
// generate all combinations
// of a and b in the range
// (1, (2 * Radius - 1))
// (Both inclusive)
for (int a = 1;
     a < 2 * radius; a++)
{
    for (int b = 1;
         b < 2 * radius; b++)
    {

        // Calculate the
        // Diagnal length of
        // this rectange
        int diagonalLengthSquare = (a * a +
                                     b * b);

        // If this rectangle's Diagonal
        // Length is less than the Diameter,
        // it is a valid rectangle, thus
        // increment counter
        if (diagonalLengthSquare <= diameterSquare)
        {
            rectangles++;
        }
    }
}

return rectangles;
}

// Driver Code
public static void main (String[] args)
{

    // Radius of the circle
    int radius = 2;

    int totalRectangles;
    totalRectangles = countRectangles(radius);
    System.out.println(totalRectangles +
                       " rectangles can be " +
                       "cut from a circle of" +
                       " Radius " + radius);
}
}

// This code is contributed
```

```
// by anuj_67.
```

Python3

```
# Python3 program to find
# the number of rectangles
# that can be cut from a
# circle of Radius R Function
# to return the total possible
# rectangles that can be cut
# from the circle
def countRectangles(radius):

    rectangles = 0

    # Diameter = 2 * Radius
    diameter = 2 * radius

    # Square of diameter which
    # is the square of the
    # maximum length diagonal
    diameterSquare = diameter * diameter

    # generate all combinations
    # of a and b in the range
    # (1, (2 * Radius - 1))(Both inclusive)
    for a in range(1, 2 * radius):
        for b in range(1, 2 * radius):

            # Calculate the Diagnal
            # length of this rectangle
            diagonalLengthSquare = (a * a +
                                     b * b)

            # If this rectangle's Diagonal
            # Length is less than the
            # Diameter, it is a valid
            # rectangle, thus increment counter
            if (diagonalLengthSquare <= diameterSquare) :
                rectangles += 1

    return rectangles

# Driver Code

# Radius of the circle
radius = 2
totalRectangles = countRectangles(radius)
```

```
print(totalRectangles , "rectangles can be" ,
      "cut from a circle of Radius" , radius)
```

```
# This code is contributed by Smita
```

C#

```
// C# program to find the
// number of rectangles that
// can be cut from a circle
// of Radius R
using System;

class GFG
{

    // Function to return
    // the total possible
    // rectangles that can
    // be cut from the circle
    static int countRectangles(int radius)
    {
        int rectangles = 0;

        // Diameter = 2 * Radius
        int diameter = 2 * radius;

        // Square of diameter
        // which is the square
        // of the maximum length
        // diagonal
        int diameterSquare = diameter *
                            diameter;

        // generate all combinations
        // of a and b in the range
        // (1, (2 * Radius - 1))
        // (Both inclusive)
        for (int a = 1;
             a < 2 * radius; a++)
        {
            for (int b = 1;
                 b < 2 * radius; b++)
            {

                // Calculate the
                // Diagnal length of
                // this rectange
```

```
int diagonalLengthSquare = (a * a +
                             b * b);

// If this rectangle's
// Diagonal Length is
// less than the Diameter,
// it is a valid rectangle,
// thus increment counter
if (diagonalLengthSquare <=
    diameterSquare)
{
    rectangles++;
}
}

return rectangles;
}

// Driver Code
public static void Main ()
{
    // Radius of the circle
    int radius = 2;

    int totalRectangles;
    totalRectangles = countRectangles(radius);
    Console.WriteLine(totalRectangles +
                      " rectangles can be " +
                      "cut from a circle of" +
                      " Radius " + radius);
}

// This code is contributed
// by anuj_67.
```

PHP

```
<?php
// PHP program to find the
// number of rectangles that
// can be cut from a circle
// of Radius R

// Function to return the
// total possible rectangles
```

```
// that can be cut from the circle
function countRectangles($radius)
{
    $rectangles = 0;

    // Diameter = 2 * $Radius
    $diameter = 2 * $radius;

    // Square of diameter which
    // is the square of the
    // maximum length diagonal
    $diameterSquare = $diameter *
                      $diameter;

    // generate all combinations
    // of a and b in the range
    // (1, (2 * Radius - 1))(Both inclusive)
    for ($a = 1;
         $a < 2 * $radius; $a++)
    {
        for ($b = 1;
             $b < 2 * $radius; $b++)
        {

            // Calculate the Diagnal
            // length of this rectangle
            $diagnallLengthSquare = ($a * $a +
                                      $b * $b);

            // If this rectangle's Diagonal
            // Length is less than the
            // Diameter, it is a valid
            // rectangle, thus increment counter
            if ($diagnallLengthSquare <= $diameterSquare)
            {
                $rectangles++;
            }
        }
    }

    return $rectangles;
}

// Driver Code

// Radius of the circle
$radius = 2;
```

```
$totalRectangles;  
$totalRectangles = countRectangles($radius);  
echo $totalRectangles , " rectangles can be " ,  
    "cut from a circle of Radius " , $radius;  
  
// This code is contributed  
// by anuj_67.  
?>
```

Output:

8 rectangles can be cut from a circle of Radius 2

Time Complexity: $O(R^2)$, where R is the Radius of the Circle

Improved By : [vt_m](#), Smitha Dinesh Semwal

Source

<https://www.geeksforgeeks.org/number-of-rectangles-in-a-circle-of-radius-r/>

Chapter 147

Number of squares of maximum area in a rectangle

Number of squares of maximum area in a rectangle - GeeksforGeeks

Given a rectangle of sides m and n . Cut the rectangle into smaller identical pieces such that each piece is a square having maximum possible side length with no leftover part of the rectangle. Print number of such squares formed.

Examples:

Input: 9 6
Output: 6
Rectangle can be cut into squares of size 3.

Input: 4 2
Output: 2
Rectangle can be cut into squares of size 2.

Approach: The task is to cut the rectangle in squares with the side of length s without pieces of the rectangle left over, so s must divide both m and n . Also, the side of the square should be maximum possible, therefore, s should be the greatest common divisor of m and n .

so, $s = \text{gcd}(m, n)$.

To find the number of squares the rectangle is cut into, the task to be done is to divide the area of a rectangle with an area of the square of size s .

C++

```
// C++ code for calculating the
// number of squares
```

```
#include <bits/stdc++.h>
using namespace std;

// Function to find number of squares
int NumberOfSquares(int x, int y)
{
    // Here in built c++ gcd function is used
    int s = __gcd(x, y);

    int ans = (x * y) / (s * s);

    return ans;
}

// Driver code
int main()
{
    int m = 385, n = 60;

    // Call the function NumberOfSquares
    cout << NumberOfSquares(m, n);

    return 0;
}
```

Java

```
// Java code for calculating
// the number of squares
import java.io.*;

class GFG
{
    // Recursive function to
    // return gcd of a and b
    static int __gcd(int a, int b)
    {
        // Everything divides 0
        if (a == 0 || b == 0)
            return 0;

        // base case
        if (a == b)
            return a;

        // a is greater
        if (a > b)
            return __gcd(a - b, b);
    }
}
```

```
        return __gcd(a, b - a);
    }

// Function to find
// number of squares
static int NumberOfSquares(int x,
                           int y)
{
    // Here in built c++
    // gcd function is used
    int s = __gcd(x, y);

    int ans = (x * y) / (s * s);

    return ans;
}

// Driver Code
public static void main (String[] args)
{
    int m = 385, n = 60;

    // Call the function
    // NumberOfSquares
    System.out.println(NumberOfSquares(m, n));
}
}

// This code is contributed by anuj_67.
```

Python3

```
# Python3 code for calculating
# the number of squares

# Recursive function to
# return gcd of a and b
def __gcd(a, b):

    # Everything divides 0
    if (a == 0 or b == 0):
        return 0;

    # base case
    if (a == b):
        return a;
```

```
# a is greater
if (a > b):
    return __gcd(a - b, b);
return __gcd(a, b - a);

# Function to find
# number of squares
def NumberOfSquares(x, y):

    # Here in built PHP
    # gcd function is used
    s = __gcd(x, y);

    ans = (x * y) / (s * s);

    return int(ans);

# Driver Code
m = 385;
n = 60;

# Call the function
# NumberOfSquares
print(NumberOfSquares(m, n));

# This code is contributed
# by mit
```

C#

```
// C# code for calculating
// the number of squares
using System;

class GFG
{

    // Recursive function to
    // return gcd of a and b
    static int __gcd(int a, int b)
    {
        // Everything divides 0
        if (a == 0 || b == 0)
            return 0;

        // base case
        if (a == b)
            return a;
```

```
// a is greater
if (a > b)
    return __gcd(a - b, b);
return __gcd(a, b - a);
}

// Function to find
// number of squares
static int NumberOfSquares(int x,
                           int y)
{
    // Here in built c++
    // gcd function is used
    int s = __gcd(x, y);

    int ans = (x * y) /
              (s * s);

    return ans;
}

// Driver Code
static public void Main ()
{
    int m = 385, n = 60;

    // Call the function
    // NumberOfSquares
    Console.WriteLine(NumberOfSquares(m, n));
}
}

// This code is contributed by ajit
```

PHP

```
<?php
// PHP code for calculating
// the number of squares

// Recursive function to
// return gcd of a and b
function __gcd($a, $b)
{
    // Everything divides 0
    if ($a == 0 || $b == 0)
```

```
return 0;

// base case
if ($a == $b)
    return $a;

// a is greater
if ($a > $b)
    return __gcd($a - $b, $b);
return __gcd($a, $b - $a);
}

// Function to find
// number of squares
function NumberOfSquares($x, $y)
{
    // Here in built PHP
    // gcd function is used
    $s = __gcd($x, $y);

    $ans = ($x * $y) /
        ($s * $s);

    return $ans;
}

// Driver Code
$m = 385;
$n = 60;

// Call the function
// NumberOfSquares
echo (NumberOfSquares($m, $n));

// This code is contributed
// by akt_mit
?>
```

Output:

924

Improved By : [vt_m](#), [jit_t](#), Mithun Kumar

Source

<https://www.geeksforgeeks.org/number-of-squares-of-maximum-area-in-a-rectangle/>

Chapter 148

Number of triangles in a plane if no more than two points are collinear

Number of triangles in a plane if no more than two points are collinear - GeeksforGeeks

Given n points in a plane and no more than two points are collinear, the task is to count the number of triangles in a given plane.

Examples:

Input : n = 3
Output : 1

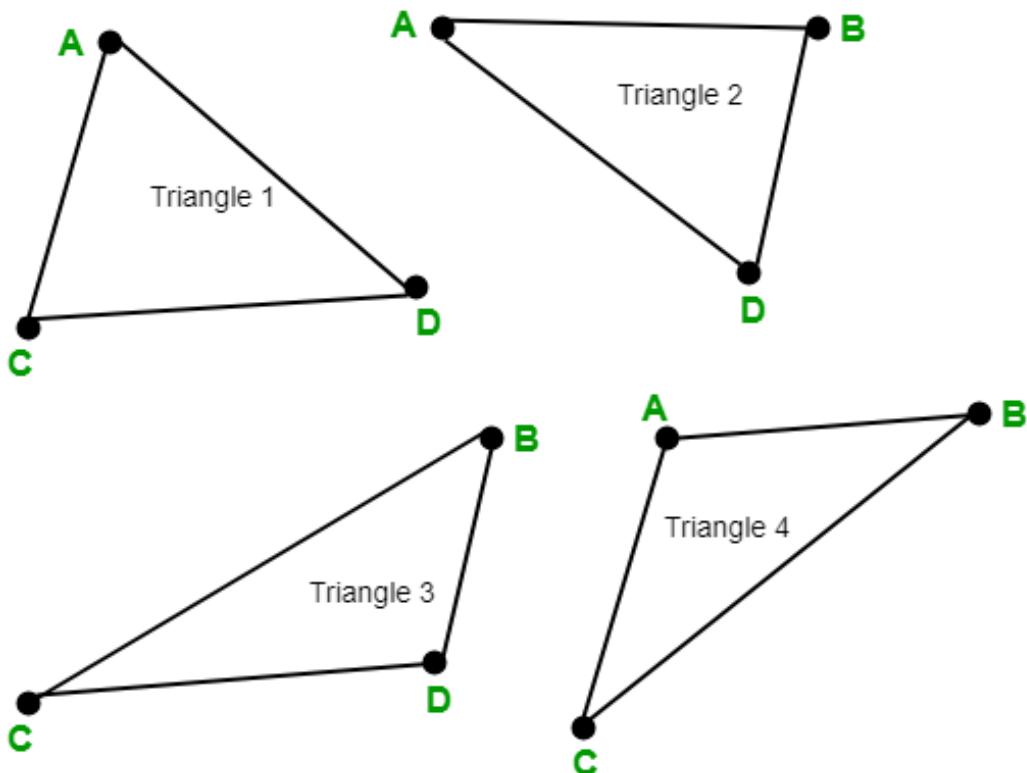
Input : n = 4
Output : 4

Let A, B, C and D are four points in a plane

A ● ● B

C ● ● D

Triangle formed by points A, B, C and D are given below:



Let there are n points in a plane and no three or more points are collinear then number of triangles in the given plane is given by

C++

```
// C++ program to find the number of
```

```
// triangles in a plane if no more
// then two points are collinear.
#include <bits/stdc++.h>
using namespace std;

// Function to find number of triangles
// in a plane.
int countNumberOfTriangles(int n)
{

    // Formula to find number of triangles
    //  $nC3 = n * (n - 1) * (n - 2) / 6$ 
    return n * (n - 1) * (n - 2) / 6;
}

// Driver code
int main()
{
    int n = 4;
    cout << countNumberOfTriangles(n);
    return 0;
}
```

Java

```
// Java program to find the number of
// triangles in a plane if no more
// then two points are collinear.
import java.io.*;

class GFG {

    // Function to find number of triangles
    // in a plane.
    static int countNumberOfTriangles(int n)
    {

        // Formula to find number of triangle
        //  $nC3 = n * (n - 1) * (n - 2) / 6$ 
        return n * (n - 1) * (n - 2) / 6;
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 4;

        System.out.println(
```

```
        countNumberOfTriangles(n));  
    }  
}
```

Python3

```
# Python3 program to find  
# the number of triangles  
# in a plane if no more  
# then two points are collinear.  
  
# Function to find number  
# of triangles in a plane.  
def countNumberOfTriangles(n) :  
  
    # Formula to find  
    # number of triangles  
    # nC3 = n * (n - 1) *  
    # (n - 2) / 6  
    return (n * (n - 1) *  
           (n - 2) // 6)  
  
# Driver Code  
if __name__ == '__main__' :  
  
    n = 4  
    print(countNumberOfTriangles(n))  
  
# This code is contributed  
# by ajit
```

C#

```
// C# program to find the  
// number of triangles in  
// a plane if no more than  
// two points are collinear.  
using System;  
  
class GFG  
{  
  
    // Function to find number  
    // of triangles in a plane.  
    static int countNumberOfTriangles(int n)  
    {
```

```
// Formula to find number
// of triangle
// nC3 = n * (n - 1) *
//           (n - 2) / 6
return n * (n - 1) *
           (n - 2) / 6;
}

// Driver code
public static void Main()
{
    int n = 4;

    Console.WriteLine(
        countNumberOfTriangles(n));
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP program to find the
// number of triangles in a
// plane if no more than
// two points are collinear.

// Function to find number
// of triangles in a plane.
function countNumberOfTriangles($n)
{
    // Formula to find number
    // of triangles nC3 = n *
    // (n - 1) * (n - 2) / 6
    return $n * ($n - 1) *
           ($n - 2) / 6;
}

// Driver code
$n = 4;
echo countNumberOfTriangles($n);

// This code is contributed
// by anuj_67.
?>
```

Output:

4

Improved By : [vt_m, jit_t](#)

Source

<https://www.geeksforgeeks.org/number-of-triangles-in-a-plane-if-no-more-than-two-points-are-collinear/>

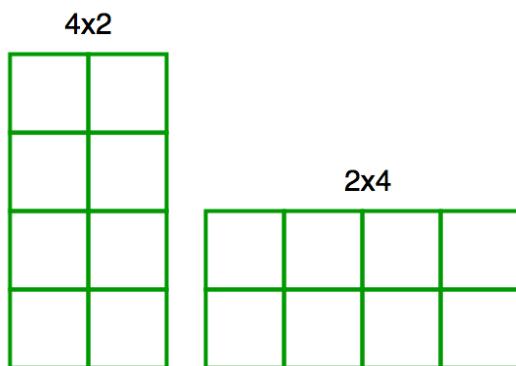
Chapter 149

Number of unique rectangles formed using N unit squares

Number of unique rectangles formed using N unit squares - GeeksforGeeks

You are given N unit squares (squares with side length 1 unit), and you are asked to make rectangles using these squares. You have to count the number of rotationally unique rectangles than you can make. What does rotationally unique mean? Well, two rectangles are rotationally unique if one can't be rotated to become equivalent to the other one.

Example – The 4×2 rectangle can be rotated 90 degrees clockwise to make it the exact same as the 2×4 rectangle and so these are not rotationally unique.



Examples :

```
Input : N = 4
Output : 5
We can make following five rectangles
1 x 1, 1 x 2, 2 x 2, 1 x 3 and 1 x 4
```

Input : N = 5

Output : 6

Input : 6

Output : 8

So how do we solve this problem?

Every rectangle is uniquely determined by its length and its height.

A rectangle of length = l and height = h then $l * h \leq n$ is considered equivalent to a rectangle with length = h and height = l provided l is not equal to h. If we can have some sort of "ordering" in these pairs then we can avoid counting (l, h) and (h, l) as different rectangles. One way to define such an ordering is:

Assume that length \leq height and count for all such pairs such that length*height $\leq n$.

We have, length \leq height

or, length*length \leq length*height

or, length*length $\leq n$

or, length $\leq \sqrt{n}$

C++

```
// C++ program to count rotationally equivalent
// rectangles with n unit squares
#include<bits/stdc++.h>
using namespace std;

int countRect(int n)
{
    int ans = 0;
    for (int length = 1; length <= sqrt(n); ++length)
        for (int height = length; height*length <= n; ++height)
            // height >= length is maintained
            ans++;
    return ans;
}

// Driver code
int main() {
    int n = 5;
    printf("%d", countRect(n));
    return 0;
}
```

Java

```
// Java program to count rotationally equivalent
```

```
// rectangles with n unit squares
class GFG {

    static int countRect(int n)
    {
        int ans = 0;

        for (int length = 1; length <= Math.sqrt(n);
              ++length)
            for (int height = length; height*length <= n;
                  ++height)
                // height >= length is maintained
                ans++;

        return ans;
    }

    //driver code
    public static void main (String[] args)
    {
        int n = 5;

        System.out.print(countRect(n));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 program to count rotationally
# equivalent rectangles with n unit squares
import math

def countRect(n):

    ans = 0
    for length in range(1, int(math.sqrt(n)) + 1):
        height = length
        while(height * length <= n):

            # height >= length is maintained
            ans += 1
            height += 1

    return ans

# Driver code
```

```
n = 5
print(countRect(n))

# This code is contributed by Anant Agarwal.
```

C#

```
// C# program to count rotationally
// equivalent rectangles with n unit
// squares
using System;

class GFG {

    static int countRect(int n)
    {
        int ans = 0;
        for (int length = 1;
             length <= Math.Sqrt(n); ++length)

            for (int height = length;
                 height*length <= n; ++height)
                ans++;

        return ans;
    }

    //driver code
    public static void Main()
    {

        int n = 5;

        Console.Write(countRect(n));
    }
}

//This code is contributed by Anant Agarwal.
```

PHP

```
<?php
// PHP program to count
// rotationally equivalent
// rectangles with n unit squares
function countRect($n)
{
```

```
$ans = 0;
for ($length = 1;
      $length <= sqrt($n); $length++)
for ($height = $length;
      $height * $length <= $n; $height++)
      // height >= length is maintained
      $ans++;
return $ans;
}

// Driver code
$n = 5;
echo countRect($n);

// This code is contributed by @ajit
?>
```

Output :

6

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/number-unique-rectangles-formed-using-n-unit-squares/>

Chapter 150

Optimum location of point to minimize total distance

Optimum location of point to minimize total distance - GeeksforGeeks

Given a set of points as and a line as $ax+by+c = 0$. We need to find a point on given line for which sum of distances from given set of points is minimum.

Example:

In above figure optimum location of point of $x - y - 3 = 0$ line is $(2, -1)$, whose total distance with other points is 20.77, which is minimum obtainable total distance.

If we take one point on given line at infinite distance then total distance cost will be infinite, now when we move this point on line towards given points the total distance cost starts decreasing and after some time, it again starts increasing which reached to infinite on the other infinite end of line so distance cost curve looks like a U-curve and we have to find the bottom value of this U-curve.

As U-curve is not monotonically increasing or decreasing we can't use binary search for finding bottom most point, here we will use ternary search for finding bottom most point, ternary search skips one third of search space at each iteration, you can read more about ternary search [here](#).

So solution proceeds as follows, we start with low and high initialized as some smallest and largest values respectively, then we start iteration, in each iteration we calculate two mids, mid1 and mid2, which represent 1/3rd and 2/3rd position in search space, we calculate total distance of all points with mid1 and mid2 and update low or high by comparing these distance cost, this iteration continues until low and high become approximately equal.

```
// C/C++ program to find optimum location and total cost
```

```
#include <bits/stdc++.h>
using namespace std;
#define sq(x) ((x)*(x))
#define EPS 1e-6
#define N 5

// structure defining a point
struct point
{
    int x, y;
    point() {}
    point(int x, int y) : x(x), y(y) {}
};

// structure defining a line of ax + by + c = 0 form
struct line
{
    int a, b, c;
    line(int a, int b, int c) : a(a), b(b), c(c) {}
};

// method to get distance of point (x, y) from point p
double dist(double x, double y, point p)
{
    return sqrt(sq(x - p.x) + sq(y - p.y));
}

/* Utility method to compute total distance all points
when choose point on given line has x-coordinate
value as X */
double compute(point p[], int n, line l, double X)
{
    double res = 0;

    // calculating Y of choosen point by line equation
    double Y = -1 * (l.c + l.a*X) / l.b;
    for (int i = 0; i < n; i++)
        res += dist(X, Y, p[i]);

    return res;
}

// Utility method to find minimum total distance
double findOptimumCostUtil(point p[], int n, line l)
{
    double low = -1e6;
    double high = 1e6;
```

```
// loop untill difference between low and high
// become less than EPS
while ((high - low) > EPS)
{
    // mid1 and mid2 are representative x co-ordinates
    // of search space
    double mid1 = low + (high - low) / 3;
    double mid2 = high - (high - low) / 3;

    //
    double dist1 = compute(p, n, l, mid1);
    double dist2 = compute(p, n, l, mid2);

    // if mid2 point gives more total distance,
    // skip third part
    if (dist1 < dist2)
        high = mid2;

    // if mid1 point gives more total distance,
    // skip first part
    else
        low = mid1;
}

// compute optimum distance cost by sending average
// of low and high as X
return compute(p, n, l, (low + high) / 2);
}

// method to find optimum cost
double findOptimumCost(int points[N][2], line l)
{
    point p[N];

    // converting 2D array input to point array
    for (int i = 0; i < N; i++)
        p[i] = point(points[i][0], points[i][1]);

    return findOptimumCostUtil(p, N, l);
}

// Driver code to test above method
int main()
{
    line l(1, -1, -3);
    int points[N][2] = {{-3, -2}, {-1, 0}, {-1, 2},
                        {1, 2}, {3, 4}};
    cout << findOptimumCost(points, l) << endl;
```

```
    return 0;  
}
```

Output:

20.7652

Source

<https://www.geeksforgeeks.org/optimum-location-point-minimize-total-distance/>

Chapter 151

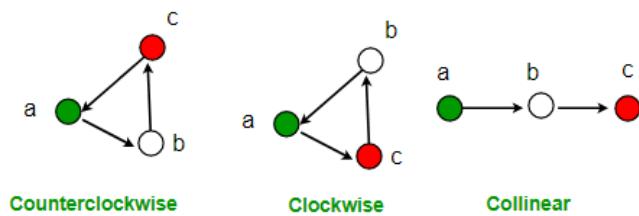
Orientation of 3 ordered points

Orientation of 3 ordered points - GeeksforGeeks

Orientation of an ordered triplet of points in the plane can be

- counterclockwise
- clockwise
- collinear

The following diagram shows different possible orientations of (a,b,c)



If orientation of (p_1, p_2, p_3) is collinear, then orientation of (p_3, p_2, p_1) is also collinear.
If orientation of (p_1, p_2, p_3) is clockwise, then orientation of (p_3, p_2, p_1) is counterclockwise
and vice versa is also true.

Given three points p_1 , p_2 and p_3 , find orientation of (p_1, p_2, p_3) .
Example:

Input: $p_1 = \{0, 0\}$, $p_2 = \{4, 4\}$, $p_3 = \{1, 2\}$
Output: CounterClockWise

Input: $p_1 = \{0, 0\}$, $p_2 = \{4, 4\}$, $p_3 = \{1, 1\}$
Output: Colinear

How to compute Orientation?

The idea is to use slope.

Slope of line segment (p₁, p₂): = (y₂ - y₁)/(x₂ - x₁)
Slope of line segment (p₂, p₃): = (y₃ - y₂)/(x₃ - x₂)

If < , the orientation is counterclockwise (left turn)
If = , the orientation is collinear
If > , the orientation is clockwise (right turn)

Using above values of and , we can conclude that,
the orientation depends on sign of below expression:

$$(y_2 - y_1)(x_3 - x_2) - (y_3 - y_2)(x_2 - x_1)$$

Above expression is negative when < , i.e., counterclockwise
Above expression is 0 when = , i.e., collinear
Above expression is positive when > , i.e., clockwise

Below is the implementation of above idea.

C++

```
// A C++ program to find orientation of three points
#include <iostream>
using namespace std;

struct Point
{
    int x, y;
};

// To find orientation of ordered triplet (p1, p2, p3).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p1, Point p2, Point p3)
{
    // See 10th slides from following link for derivation
    // of the formula
    int val = (p2.y - p1.y) * (p3.x - p2.x) -
              (p2.x - p1.x) * (p3.y - p2.y);

    if (val == 0) return 0; // colinear
}
```

```
        return (val > 0)? 1: 2; // clock or counterclock wise
    }

// Driver program to test above functions
int main()
{
    Point p1 = {0, 0}, p2 = {4, 4}, p3 = {1, 2};
    int o = orientation(p1, p2, p3);
    if (o==0)          cout << "Linear";
    else if (o == 1)   cout << "Clockwise";
    else              cout << "CounterClockwise";
    return 0;
}
```

Java

```
// JAVA Code to find Orientation of 3
// ordered points
class Point
{
    int x, y;
    Point(int x,int y){
        this.x=x;
        this.y=y;
    }
}

class GFG {

    // To find orientation of ordered triplet
    // (p1, p2, p3). The function returns
    // following values
    // 0 --> p, q and r are colinear
    // 1 --> Clockwise
    // 2 --> Counterclockwise
    public static int orientation(Point p1, Point p2,
                                  Point p3)
    {
        // See 10th slides from following link
        // for derivation of the formula
        int val = (p2.y - p1.y) * (p3.x - p2.x) -
                  (p2.x - p1.x) * (p3.y - p2.y);

        if (val == 0) return 0; // colinear

        // clock or counterclock wise
        return (val > 0)? 1: 2;
    }
}
```

```
}

/* Driver program to test above function */
public static void main(String[] args)
{
    Point p1 = new Point(0, 0);
    Point p2 = new Point(4, 4);
    Point p3 = new Point(1, 2);

    int o = orientation(p1, p2, p3);

    if (o==0)
        System.out.print("Linear");
    else if (o == 1)
        System.out.print("Clockwise");
    else
        System.out.print("CounterClockwise");

}
}

//This code is contributed by Arnav Kr. Mandal.
```

Output:

CounterClockwise

The concept of orientation is used in below articles:

[Find Simple Closed Path for a given set of points](#)

[How to check if two given line segments intersect?](#)

[Convex Hull | Set 1 \(Jarvis's Algorithm or Wrapping\)](#)

[Convex Hull | Set 2 \(Graham Scan\)](#)

Source:

<http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf>

This article is contributed by **Rajeev Agrawal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Source

<https://www.geeksforgeeks.org/orientation-3-ordered-points/>

Chapter 152

Paper Cut into Minimum Number of Squares

Paper Cut into Minimum Number of Squares - GeeksforGeeks

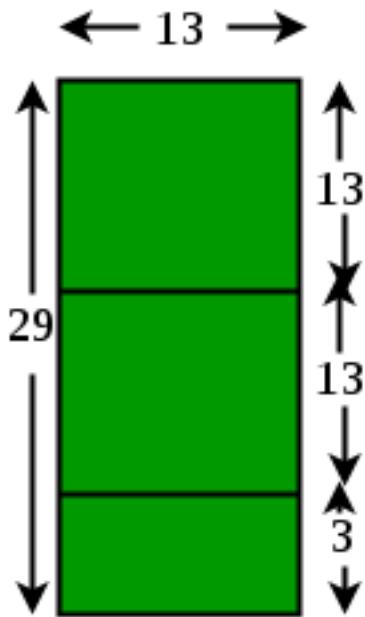
Given a paper of size A x B. Task is to cut the paper into squares of any size. Find the minimum number of squares that can be cut from the paper.

Examples:

```
Input  : 13 x 29
Output : 9
Explanation :
2 (squares of size 13x13) +
4 (squares of size 3x3) +
3 (squares of size 1x1)=9
```

```
Input  : 4 x 5
Output : 5
Explanation :
1 (squares of size 4x4) +
4 (squares of size 1x1)
```

We know that if we want to cut minimum number of squares from the paper then we would have to cut largest square possible from the paper first and largest square will have same side as smaller side of the paper. For example if paper have the size 13 x 29, then maximum square will be of side 13. so we can cut 2 square of size 13 x 13 ($29/13 = 2$). Now remaining paper will have size 3 x 13. Similarly we can cut remaining paper by using 4 squares of size 3 x 3 and 3 squares of 1 x 1. So minimum 9 squares can be cut from the Paper of size 13 x 29.



Below is the implementation of above approach.

C++

```
// C++ program to find minimum number of squares
// to cut a paper.
#include<bits/stdc++.h>
using namespace std;

// Returns min number of squares needed
int minimumSquare(int a, int b)
{
    long long result = 0, rem = 0;

    // swap if a is small size side .
    if (a < b)
        swap(a, b);

    // Iterate until small size side is
    // greater than 0
    while (b > 0)
    {
        // Update result
        result += a/b;

        long long rem = a % b;
        a = b;
    }
}
```

```
        b = rem;
    }

    return result;
}

// Driver code
int main()
{
    int n = 13, m = 29;
    cout << minimumSquare(n, m);
    return 0;
}
```

Java

```
// Java program to find minimum
// number of squares to cut a paper.
class GFG{

// To swap two numbers
static void swap(int a,int b)
{
    int temp = a;
    a = b;
    b = temp;
}

// Returns min number of squares needed
static int minimumSquare(int a, int b)
{
    int result = 0, rem = 0;

    // swap if a is small size side .
    if (a < b)
        swap(a, b);

    // Iterate until small size side is
    // greater then 0
    while (b > 0)
    {
        // Update result
        result += a/b;
        rem = a % b;
        a = b;
        b = rem;
    }
}
```

```
        return result;
    }

// Driver code
public static void main(String[] args)
{
    int n = 13, m = 29;
    System.out.println(minimumSquare(n, m));
}
}

//This code is contributed by Smitha Dinesh Semwal.
```

Python3

```
# Python 3 program to find minimum
# number of squares to cut a paper.

# Returns min number of squares needed
def minimumSquare(a, b):

    result = 0
    rem = 0

    # swap if a is small size side .
    if (a < b):
        a, b = b, a

    # Iterate until small size side is
    # greater then 0
    while (b > 0):

        # Update result
        result += int(a / b)

        rem = int(a % b)
        a = b
        b = rem

    return result

# Driver code
n = 13
m = 29

print(minimumSquare(n, m))

# This code is contributed by
```

Smitha Dinesh Semwal

Output:

9

Note that the above Greedy solution doesn't always produce optimal result. For example if input is 36 x 30, the above algorithm would produce output 6, but we can cut the paper in 5 squares

- 1) Three squares of size 12 x 12
- 2) Two squares of size 18 x 18.

Thanks to Sergey V. Pereslavytsev for pointing out the above case.

Source

<https://www.geeksforgeeks.org/paper-cut-minimum-number-squares/>

Chapter 153

Path in a Rectangle with Circles

Path in a Rectangle with Circles - GeeksforGeeks

There is a $m \times n$ rectangular matrix whose top-left(start) location is $(1, 1)$ and bottom-right(end) location is $(m \times n)$. There are k circles each with radius r . Find if there is any path from start to end without touching any circle.

The input contains values of m , n , k , r and two array of integers X and Y , each of length k . $(X[i], Y[i])$ is the centre of i th circle.

Source : [Directi Interview](#)

```
Input1 : m = 5, n = 5, k = 2, r = 1,
         X = {1, 3}, Y = {3, 3}
Output1 : Possible
```

Here is a path from start to end point.

```
Input2 : m = 5, n = 5, k = 2, r = 1,
         X = {1, 1}, Y = {2, 3}.
Output2 : Not Possible
```

Approach : Check if the centre of a cell (i, j) of the rectangle comes within any of the circles then do not traverse through that cell and mark that as ‘blocked’. Mark rest of the cells initially as ‘unvisited’. Then use [BFS](#) to find out shortest path of each cell from starting position. If the end cell is visited then we will return “Possible” otherwise “Not Possible”.

Algorithm :

1. Take an array of size $m \times n$. Initialize all the cells to 0.
2. For each cell of the rectangle check whether it comes within any circle or not (by calculating the distance of that cell from each circle). If it comes within any circle then change the value of that cell to -1(‘blocked’).

3. Now, apply BFS from the starting cell and if a cell can be reached then change the value of that cell to 1.
4. If the value of the ending cell is 1, then return ‘Possible’, otherwise return ‘Not Possible’.

C++

```

// C++ program to find out path in
// a rectangle containing circles.
#include <iostream>
#include <math.h>
#include <vector>

using namespace std;

// Function to find out if there is
// any possible path or not.
bool isPossible(int m, int n, int k, int r,
                vector<int> X, vector<int> Y)
{
    // Take an array of m*n size and
    // initialize each element to 0.
    int rect[m][n] = {0};

    // Now using Pythagorean theorem find if a
    // cell touches or within any circle or not.
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            for (int p = 0; p < k; p++) {
                if (sqrt((pow((X[p] - 1 - i), 2) +
                           pow((Y[p] - 1 - j), 2))) <= r)
                {
                    rect[i][j] = -1;
                }
            }
        }
    }

    // If the starting cell comes within
    // any circle return false.
    if (rect[0][0] == -1)
        return false;

    // Now use BFS to find if there
    // is any possible path or not.

    // Initialize the queue which holds
    // the discovered cells whose neighbors

```

```

// are not discovered yet.
vector<vector<int>> qu;

rect[0][0] = 1;
qu.push_back({0, 0});

// Discover cells until queue is not empty
while (!qu.empty()) {
    vector<int> arr = qu.front();
    qu.erase(qu.begin());
    int elex = arr[0];
    int eley = arr[1];

    // Discover the eight adjacent nodes.
    // check top-left cell
    if ((elex > 0) && (eley > 0) &&
        (rect[elex - 1][eley - 1] == 0))
    {
        rect[elex - 1][eley - 1] = 1;
        vector<int> v = {elex - 1, eley - 1};
        qu.push_back(v);
    }

    // check top cell
    if ((elex > 0) &&
        (rect[elex - 1][eley] == 0))
    {
        rect[elex - 1][eley] = 1;
        vector<int> v = {elex - 1, eley};
        qu.push_back(v);
    }

    // check top-right cell
    if ((elex > 0) && (eley < n - 1) &&
        (rect[elex - 1][eley + 1] == 0))
    {
        rect[elex - 1][eley + 1] = 1;
        vector<int> v = {elex - 1, eley + 1};
        qu.push_back(v);
    }

    // check left cell
    if ((eley > 0) &&
        (rect[elex][eley - 1] == 0))
    {
        rect[elex][eley - 1] = 1;
        vector<int> v = {elex, eley - 1};
        qu.push_back(v);
    }
}

```

```

    }

    // check right cell
    if ((eley > n - 1) &&
        (rect[elex][eley + 1] == 0))
    {
        rect[elex][eley + 1] = 1;
        vector<int> v = {elex, eley + 1};
        qu.push_back(v);
    }

    // check bottom-left cell
    if ((elex < m - 1) && (eley > 0) &&
        (rect[elex + 1][eley - 1] == 0))
    {
        rect[elex + 1][eley - 1] = 1;
        vector<int> v = {elex + 1, eley - 1};
        qu.push_back(v);
    }

    // check bottom cell
    if ((elex < m - 1) &&
        (rect[elex + 1][eley] == 0))
    {
        rect[elex + 1][eley] = 1;
        vector<int> v = {elex + 1, eley};
        qu.push_back(v);
    }

    // check bottom-right cell
    if ((elex < m - 1) && (eley < n - 1) &&
        (rect[elex + 1][eley + 1] == 0))
    {
        rect[elex + 1][eley + 1] = 1;
        vector<int> v = {elex + 1, eley + 1};
        qu.push_back(v);
    }

    // Now if the end cell (i.e. bottom right cell)
    // is 1(reachable) then we will send true.
    return (rect[m - 1][n - 1] == 1);
}

// Driver Program
int main() {

    // Test case 1
}

```

```
int m1 = 5, n1 = 5, k1 = 2, r1 = 1;
vector<int> X1 = {1, 3};
vector<int> Y1 = {3, 3};
if (isPossible(m1, n1, k1, r1, X1, Y1))
    cout << "Possible" << endl;
else
    cout << "Not Possible" << endl;

// Test case 2
int m2 = 5, n2 = 5, k2 = 2, r2 = 1;
vector<int> X2 = {1, 1};
vector<int> Y2 = {2, 3};
if (isPossible(m2, n2, k2, r2, X2, Y2))
    cout << "Possible" << endl;
else
    cout << "Not Possible" << endl;

return 0;
}
```

Output:

```
Possible
Not Possible
```

Time Complexity : It takes $O(m*n*k)$ time to compute whether a cell is within or not in any circle. And it takes $O(V+E)$ time in BFS. Here, number of edges in $m*n$ grid is $m*(n-1)+n*(m-1)$ and vertices $m*n$. So it takes $O(m*n)$ time in DFS. Hence, the time complexity is $O(m*n*k)$. The complexity can be improved if we iterate through each circles and mark -1 the cells which are coming within it.

Source

<https://www.geeksforgeeks.org/path-rectangle-containing-circles/>

Chapter 154

Pentagonal Pyramidal Number

Pentagonal Pyramidal Number - GeeksforGeeks

Given a number n, find the **nth** pentagonal pyramidal number.

A **Pentagonal Pyramidal Number** belongs to the figurate number class. It is the number of objects in a pyramid with a pentagonal base. The nth pentagonal pyramidal number is equal to sum of first n [pentagonal numbers](#).

Examples:

Input : n = 3
Output : 18

Input : n = 7
Output : 196

Method 1: (Naive Approach) :

This approach is simple. It says to add all the pentagonal numbers up to n (by running loop) to get nth Pentagonal pyramidal number.

Below is the implementation of this approach:

C++

```
// CPP Program to get nth Pentagonal
// pyramidal number.
#include <bits/stdc++.h>
using namespace std;

// function to get nth Pentagonal
// pyramidal number.
int pentagon_pyramidal(int n)
{
```

```
int sum = 0;

// Running loop from 1 to n
for (int i = 1; i <= n; i++) {

    // get nth pentagonal number
    int p = (3 * i * i - i) / 2;

    // add to sum
    sum = sum + p;
}
return sum;
}

// Driver Program
int main()
{
    int n = 4;
    cout << pentagon_pyramidal(n) << endl;
    return 0;
}
```

Java

```
// Java Program to get nth
// Pentagonal pyramidal number.
import java.io.*;

class GFG
{

    // function to get nth
    // Pentagonal pyramidal number.
    static int pentagon_pyramidal(int n)
    {
        int sum = 0;

        // Running loop from 1 to n
        for (int i = 1; i <= n; i++)
        {

            // get nth pentagonal number
            int p = (3 * i * i - i) / 2;

            // add to sum
            sum = sum + p;
        }
        return sum;
    }
}
```

```
}

// Driver Code
public static void main (String[] args)
{
    int n = 4;
    System.out.println(pentagon_pyramidal(n));
}
}

// This code is contributed by anuj_67.
```

Python3

```
# Python3 Program to get nth Pentagonal
# pyramidal number.

# function to get nth Pentagonal
# pyramidal number.
def pentagon_pyramidal(n):
    sum = 0

    # Running loop from 1 to n
    for i in range(1, n + 1):

        # get nth pentagonal number
        p = ( 3 * i * i - i ) / 2

        # add to sum
        sum = sum + p

    return sum

# Driver Program
n = 4
print(int(pentagon_pyramidal(n)))
```

C#

```
// C# Program to get nth
// Pentagonal pyramidal number.
using System;

class GFG
{
```

```
// function to get nth
// Pentagonal pyramidal number.
static int pentagon_pyramidal(int n)
{
    int sum = 0;

    // Running loop from 1 to n
    for (int i = 1; i <= n; i++)
    {

        // get nth pentagonal number
        int p = (3 * i *
                  i - i) / 2;

        // add to sum
        sum = sum + p;
    }
    return sum;
}

// Driver Code
static public void Main ()
{
    int n = 4;
    Console.WriteLine(pentagon_pyramidal(n));
}
}

// This code is contributed by ajit.
```

PHP

```
<?php
// PHP Program to get nth
// Pentagonal pyramidal number.

// function to get nth
// Pentagonal pyramidal number.
function pentagon_pyramidal($n)
{
    $sum = 0;

    // Running loop from 1 to n
    for ($i = 1; $i <= $n; $i++)
    {

        // get nth pentagonal number
        $p = (3 * $i *
```

```
$i - $i) / 2;

// add to sum
$sum = $sum + $p;
}
return $sum;
}

// Driver Code
$n = 4;
echo pentagon_pyramidal($n);

// This code is contributed by m_kit
?>
```

Output :

40

Time Complexity : O(n)

Method 2: (Efficient Approach) :

In this approach, we use formula to get nth Pentagonal pyramidal number in O(1) time.

nth Pentagonal pyramidal number = $n^2 (n + 1) / 2$

Below is the implementation of this approach:

C++

```
// CPP Program to get nth Pentagonal
// pyramidal number.
#include <bits/stdc++.h>
using namespace std;

// function to get nth Pentagonal
// pyramidal number.
int pentagon_pyramidal(int n)
{
    return n * n * (n + 1) / 2;
}

// Driver Program
int main()
{
    int n = 4;
    cout << pentagon_pyramidal(n) << endl;
```

```
    return 0;
}
```

Java

```
// Java Program to get nth
// Pentagonal pyramidal number.
import java.io.*;

class GFG
{

// function to get nth
// Pentagonal pyramidal number.
static int pentagon_pyramidal(int n)
{
    return n * n *
           (n + 1) / 2;
}

// Driver Code
public static void main (String[] args)
{
    int n = 4;
    System.out.println(pentagon_pyramidal(n));
}
}

// This code is contributed by ajit
```

Python3

```
# Python3 Program to get nth Pentagonal
# pyramidal number.

# function to get nth Pentagonal
# pyramidal number.
def pentagon_pyramidal(n):
    return n * n * (n + 1) / 2

# Driver Program
n = 4
print(int(pentagon_pyramidal(n)))
```

C#

```
// C# Program to get nth
// Pentagonal pyramidal number.
using System;

class GFG
{

    // function to get nth
    // Pentagonal pyramidal number.
    static int pentagon_pyramidal(int n)
    {
        return n * n *
               (n + 1) / 2;
    }

    // Driver Code
    static public void Main ()
    {
        int n = 4;
        Console.WriteLine(
            pentagon_pyramidal(n));
    }
}

// This code is contributed
// by ajit
```

PHP

```
<?php
// PHP Program to get
// nth Pentagonal
// pyramidal number.

// function to get
// nth Pentagonal
// pyramidal number.

function pentagon_pyramidal($n)
{
    return $n * $n *
           ($n + 1) / 2;
}

// Driver Code
$n = 4;
echo pentagon_pyramidal($n);
```

```
// This code is contributed  
// by akt_mit  
?>
```

Output :

40

Time Complexity : O(1)

Improved By : [vt_m, jit_t](#)

Source

<https://www.geeksforgeeks.org/pentagonal-pyramidal-number/>

Chapter 155

Pentatope number

Pentatope number - GeeksforGeeks

Given a number n, the task is to write a program to find Nth pentatope number. A pentatope number is class of [figurative number](#) and can be represented as regular and discrete geometric patterns. In [pascal triangle's](#) the number in the fifth cell of any row starting with 5-th term row 1 4 6 4 1 from right to left or left to right is **Pentatope number**.

Examples :

Input : 5
Output :70

Input :8
Output :1001

First few Pentatope numbers are:

1, 5, 15, 35, 70, 126, 210, 330, 495, 715, 1001.....

Formula for Nth Pentatope number:

C++

```
// C++ Program to find the
// nth Pentatope Number
#include <bits/stdc++.h>
using namespace std;

// Function that returns nth
// pentatope number
int pentatopeNum(int n)
```

```
{  
    return (n * (n + 1) * (n + 2) * (n + 3)) / 24;  
}  
  
// Drivers Code  
int main()  
{  
    // For 5th Pentatope Number  
    int n = 5;  
    cout << pentatopeNum(n) << endl;  
  
    // For 11th Pentatope Number  
    n = 11;  
    cout << pentatopeNum(n) << endl;  
  
    return 0;  
}
```

Java

```
// Java Program to find the  
// nth Pentatope Number  
import java.io.*;  
  
class GFG  
{  
  
    // Function that returns nth  
    // pentatope number  
    static int pentatopeNum(int n)  
{  
        return (n * (n + 1) *  
                (n + 2) * (n + 3)) / 24;  
    }  
    // Driver Code  
    public static void main (String[] args)  
    {  
        // For 5th Pentatope Number  
        int n = 5;  
        System.out.println(pentatopeNum(n));  
  
        // For 11th Pentatope Number  
        n = 11;  
        System.out.println(pentatopeNum(n));  
  
    }  
}
```

```
// This code is contributed by m_kit
```

Python3

```
# Python3 program to find
# nth Pentatope number

# Function to calculate
# Pentatope number
def pentatopeNum(n):

    # Formula to calculate nth
    # Pentatope number
    return (n * (n + 1) *
           (n + 2) *
           (n + 3)) // 24

# Driver Code
n = 5
print(pentatopeNum(n))
n = 11
print(pentatopeNum(n))

# This code is contributed
# by ajit.
```

C#

```
// C# Program to find the
// nth Pentatope Number
using System;

class GFG
{

    // Function that returns
    // nth pentatope number
    static int pentatopeNum(int n)
    {
        return (n * (n + 1) *
               (n + 2) * (n + 3)) / 24;
    }

    // Driver Code
    static public void Main(String []args)
    {
```

```
// For 5th PentaTope Number
int n = 5;

Console.WriteLine(pentatopeNum(n));

// For 11th PentaTope Number
n = 11;

Console.WriteLine(pentatopeNum(n));
}
}

// This code is contributed by Arnab Kundu
```

PHP

```
<?php
// PHP Program to find the
// nth Pentatope Number

// Function that returns
// nth pentatope number
function pentatopeNum($n)
{
    return ($n * ($n + 1) *
            ($n + 2) *
            ($n + 3)) / 24;
}

// Driver Code

// For 5th PentaTope Number
$n = 5;
echo pentatopeNum($n), "\n";

// For 11th PentaTope Number
$n = 11;
echo pentatopeNum($n), "\n" ;

// This code is contributed
// by aj_36
?>
```

Output :

70
1001

Reference:https://en.wikipedia.org/wiki/Pentatope_number

Improved By : [jit_t](#), [andrew1234](#)

Source

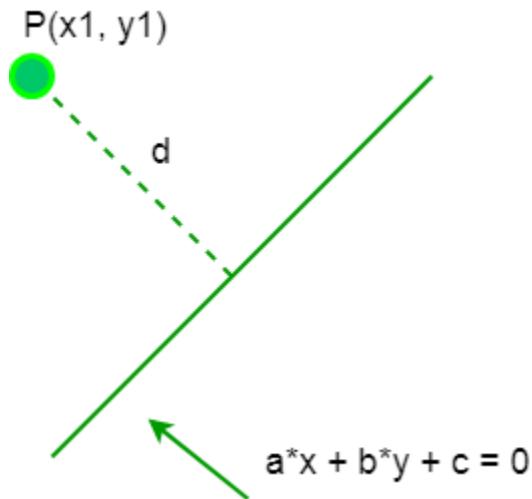
<https://www.geeksforgeeks.org/pentatope-number-2/>

Chapter 156

Perpendicular distance between a point and a Line in 2 D

Perpendicular distance between a point and a Line in 2 D - GeeksforGeeks

Given a point (x_1, y_1) and a line $(ax + by + c = 0)$. The task is to find the perpendicular distance between the given point and the line.



Examples :

Input: $x_1 = 5, y_1 = 6, a = -2, b = 3, c = 4$

Output: 3.32820117735

Input: $x_1 = -1, y_1 = 3, a = 4, b = -3, c = -5$

Output: 3.6

Approach: The distance (i.e shortest distance) from a given point to a line is the perpendicular distance from that point to the given line. The equation of a line in the plane is

given by the equation $ax + by + c = 0$, where a, b and c are real constants. the co-ordinate of the point is (x_1, y_1)

The formula for distance between a point and a line in 2-D is given by:

```
Distance = (| a*x1 + b*y1 + c |) / (sqrt( a*a + b*b))
```

Below is the implementation of the above formulae:

Program 1:

C

```
// C program to find the distance
// between a given point and a
// given line in 2 D.
#include<stdio.h>
#include<math.h>

// Function to find distance
void shortest_distance(float x1, float y1,
                      float a, float b,
                      float c)
{
    float d = fabs((a * x1 + b * y1 + c)) /
              (sqrt(a * a + b * b));
    printf("Perpendicular distance is %f\n", d);
    return;
}

// Driver Code
int main()
{
    float x1 = 5;
    float y1 = 6;
    float a = -2;
    float b = 3;
    float c = 4;
    shortest_distance(x1, y1, a, b, c);
    return 0;
}

// This code is contributed
// by Amber_Saxena.
```

Java

```
// Java program to find
// the distance between
// a given point and a
// given line in 2 D.
import java.io.*;

class GFG
{

    // Function to find distance
    static void shortest_distance(float x1, float y1,
                                  float a, float b,
                                  float c)
    {
        double d = Math.abs(((a * x1 + b * y1 + c)) /
                           (Math.sqrt(a * a + b * b)));
        System.out.println("Perpendicular " +
                           "distance is " + d);
        return;
    }

    // Driver code
    public static void main (String[] args)
    {
        float x1 = 5;
        float y1 = 6;
        float a = -2;
        float b = 3;
        float c = 4;
        shortest_distance(x1, y1, a, b, c);
    }
}

// This code is contributed
// by Mahadev.
```

Python

```
# Python program to find the distance between
# a given point and a given line in 2 D.

import math

# Function to find distance
def shortest_distance(x1, y1, a, b, c):

    d = abs((a * x1 + b * y1 + c)) / (math.sqrt(a * a + b * b))
    print("Perpendicular distance is"),d
```

```
# Driver Code
x1 = 5
y1 = 6
a = -2
b = 3
c = 4
shortest_distance(x1, y1, a, b, c)
```

Output:

```
Perpendicular distance is 3.32820117735
```

Program 2:

C

```
// C program to find the distance
// between a given point and a
// given line in 2 D.
#include<stdio.h>
#include<math.h>

// Function to find distance
void shortest_distance(float x1, float y1,
                      float a, float b,
                      float c)
{
    float d = fabs((a * x1 + b * y1 + c)) /
              (sqrt(a * a + b * b));
    printf("Perpendicular distance is %f\n", d);
    return;
}

// Driver Code
int main()
{
    float x1 = -1;
    float y1 = 3;
    float a = 4;
    float b = -3;
    float c = - 5;
    shortest_distance(x1, y1, a, b, c);
    return 0;
}
```

```
// This code is contributed  
// by Amber_Saxena.
```

Python

```
# Python program to find the distance between  
# a given point and a given line in 2 D.  
  
import math  
  
# Function to find distance  
def shortest_distance(x1, y1, a, b, c):  
  
    d = abs((a * x1 + b * y1 + c)) / (math.sqrt(a * a + b * b))  
    print("Perpendicular distance is"),d  
  
# Driver Code  
x1 = -1  
y1 = 3  
a = 4  
b = -3  
c = - 5  
shortest_distance(x1, y1, a, b, c)
```

Output:

Perpendicular distance is 3.6

Improved By : [Amber_Saxena](#), [Mahadev99](#)

Source

<https://www.geeksforgeeks.org/perpendicular-distance-between-a-point-and-a-line-in-2-d/>

Chapter 157

Pizza cut problem (Or Circle Division by Lines)

Pizza cut problem (Or Circle Division by Lines) - GeeksforGeeks

Given number of cuts, find the maximum number of possible pieces.

Examples:

Input : 2
Output : 4

Input : 3
Output : 7

This problem is nothing but [The Lazy Caterer's Problem](#) and has below formula.

Maximum number of pieces = $1 + n*(n+1)/2$

Refer [this](#) for proof.

C++

```
// C++ program to find maximum no of pieces
// by given number of cuts
#include<bits/stdc++.h>
using namespace std;

// Function for finding maximum pieces
// with n cuts.
int findMaximumPieces(int n)
{
```

```
    return 1 + n*(n+1)/2;
}

// Driver code
int main()
{
    cout << findMaximumPieces(3);
    return 0;
}
```

Java

```
// Java program to find maximum no of
// pieces by given number of cuts
class GFG {

    // Function for finding maximum pieces
    // with n cuts.
    static int findMaximumPieces(int n)
    {
        return 1 + n * (n + 1) / 2;
    }

    // Driver Program to test above function
    public static void main(String arg[])
    {

        System.out.print(findMaximumPieces(3));
    }
}

// This code is contributed by Anant Agarwal.
```

C#

```
// C# program to find maximum no of
// pieces by given number of cuts
using System;

class GFG {

    // Function for finding maximum pieces
    // with n cuts.
    static int findMaximumPieces(int n)
    {
        return 1 + n * (n + 1) / 2;
    }
}
```

```
// Driver Program to test above function
public static void Main()
{
    Console.WriteLine(findMaximumPieces(3));
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// PHP program to find maximum
// no. of pieces by given
// number of cuts

// Function for finding maximum
// pieces with n cuts.
function findMaximumPieces($n)
{
    return 1 + $n * ($n + 1) / 2;
}

// Driver code
echo findMaximumPieces(3);

// This code is contributed by nitin mittal.
?>
```

Output:

7

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/pizza-cut-problem-circle-division-lines/>

Chapter 158

Polygon Clipping | Sutherland–Hodgman Algorithm

Polygon Clipping | Sutherland–Hodgman Algorithm - GeeksforGeeks

A convex polygon and a convex clipping area are given. The task is to clip polygon edges using the Sutherland–Hodgman Algorithm. Input is in the form of vertices of the polygon in **clockwise order**.

Examples:

```
Input : Polygon : (100,150), (200,250), (300,200)
        Clipping Area : (150,150), (150,200), (200,200),
                          (200,150) i.e. a Square
Output : (150, 162) (150, 200) (200, 200) (200, 174)
```

Example 2

```
Input : Polygon : (100,150), (200,250), (300,200)
        Clipping Area : (100,300), (300,300), (200,100)
Output : (242, 185) (166, 166) (150, 200) (200, 250) (260, 220)
```

Overview of the algorithm:

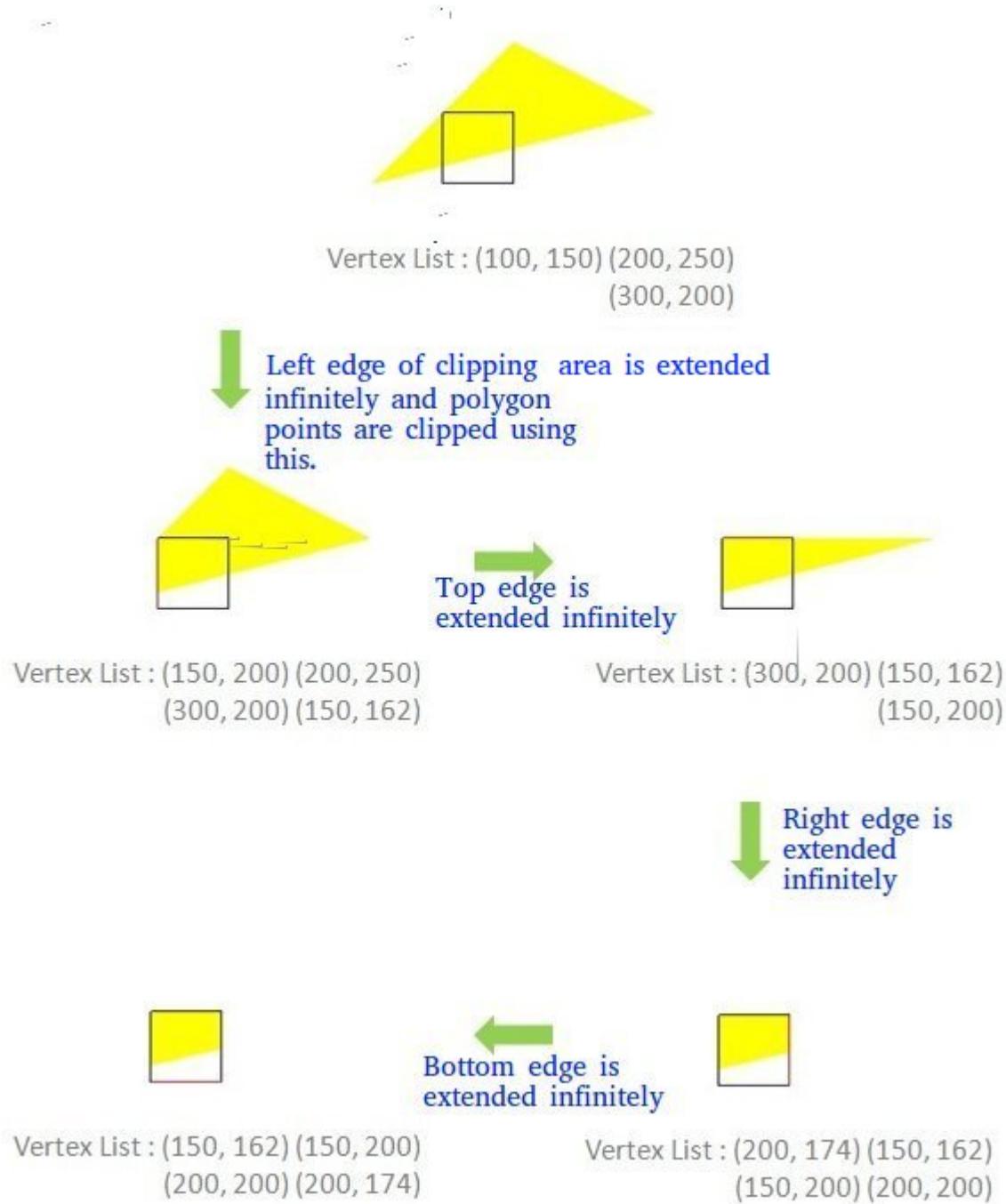
Consider each edge e of clipping Area and do following:
a) Clip given polygon against e .

How to clip against an edge of clipping area?

The edge (of clipping area) is extended infinitely to create a boundary and all the vertices are clipped using this boundary. The new list of vertices generated is passed to the next edge of the clip polygon in clockwise fashion until all the edges have been used.

There are four possible cases for any given edge of given polygon against current clipping edge e.

1. **Both vertices are inside :** Only the second vertex is added to the output list
2. **First vertex is outside while second one is inside :** Both the point of intersection of the edge with the clip boundary and the second vertex are added to the output list
3. **First vertex is inside while second one is outside :** Only the point of intersection of the edge with the clip boundary is added to the output list
4. **Both vertices are outside :** No vertices are added to the output list



There are two sub-problems that need to be discussed before implementing the algorithm:-

To decide if a point is inside or outside the clipper polygon

If the vertices of the clipper polygon are given in clockwise order then all the points lying on the **right side** of the clipper edges are inside that polygon. This can be calculated using

: Given that the line starts from (x_1, y_1) and ends at (x_2, y_2)

$$P = (x_2 - x_1)(y - y_1) - (y_2 - y_1)(x - x_1)$$

if $P < 0$, the point is on the right side of the line

$P = 0$, the point is on the line

$P > 0$, the point is on the left side of the line

To find the point of intersection of an edge with the clip boundary

If two points of each line(1,2 & 3,4) are known, then their point of intersection can be calculated using the formula :-

$$(P_x, P_y) = \left(\frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_1 - x_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}, \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \right)$$

```
// C++ program for implementing Sutherland-Hodgman
// algorithm for polygon clipping
#include<iostream>
using namespace std;

const int MAX_POINTS = 20;

// Returns x-value of point of intersectipn of two
// lines
int x_intersect(int x1, int y1, int x2, int y2,
                int x3, int y3, int x4, int y4)
{
    int num = (x1*y2 - y1*x2) * (x3-x4) -
              (x1-x2) * (x3*y4 - y3*x4);
    int den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);
    return num/den;
}

// Returns y-value of point of intersectipn of
// two lines
int y_intersect(int x1, int y1, int x2, int y2,
                int x3, int y3, int x4, int y4)
{
    int num = (x1*y2 - y1*x2) * (y3-y4) -
              (y1-y2) * (x3*y4 - y_3*x4);
    int den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);
    return num/den;
}
```

```

// This function clips all the edges w.r.t one clip
// edge of clipping area
void clip(int poly_points[][] [2], int &poly_size,
          int x1, int y1, int x2, int y2)
{
    int new_points[MAX_POINTS] [2], new_poly_size = 0;

    // (ix,iy),(kx,ky) are the co-ordinate values of
    // the points
    for (int i = 0; i < poly_size; i++)
    {
        // i and k form a line in polygon
        int k = (i+1) % poly_size;
        int ix = poly_points[i][0], iy = poly_points[i][1];
        int kx = poly_points[k][0], ky = poly_points[k][1];

        // Calculating position of first point
        // w.r.t. clipper line
        int i_pos = (x2-x1) * (iy-y1) - (y2-y1) * (ix-x1);

        // Calculating position of second point
        // w.r.t. clipper line
        int k_pos = (x2-x1) * (ky-y1) - (y2-y1) * (kx-x1);

        // Case 1 : When both points are inside
        if (i_pos < 0 && k_pos < 0)
        {
            //Only second point is added
            new_points[new_poly_size][0] = kx;
            new_points[new_poly_size][1] = ky;
            new_poly_size++;
        }

        // Case 2: When only first point is outside
        else if (i_pos >= 0 && k_pos < 0)
        {
            // Point of intersection with edge
            // and the second point is added
            new_points[new_poly_size][0] = x_intersect(x1,
                y1, x2, y2, ix, iy, kx, ky);
            new_points[new_poly_size][1] = y_intersect(x1,
                y1, x2, y2, ix, iy, kx, ky);
            new_poly_size++;

            new_points[new_poly_size][0] = kx;
            new_points[new_poly_size][1] = ky;
            new_poly_size++;
        }
    }
}

```

```

    }

    // Case 3: When only second point is outside
    else if (i_pos < 0 && k_pos >= 0)
    {
        //Only point of intersection with edge is added
        new_points[new_poly_size][0] = x_intersect(x1,
                                                    y1, x2, y2, ix, iy, kx, ky);
        new_points[new_poly_size][1] = y_intersect(x1,
                                                    y1, x2, y2, ix, iy, kx, ky);
        new_poly_size++;
    }

    // Case 4: When both points are outside
    else
    {
        //No points are added
    }
}

// Copying new points into original array
// and changing the no. of vertices
poly_size = new_poly_size;
for (int i = 0; i < poly_size; i++)
{
    poly_points[i][0] = new_points[i][0];
    poly_points[i][1] = new_points[i][1];
}
}

// Implements Sutherland-Hodgman algorithm
void suthHodgClip(int poly_points[][] [2], int poly_size,
                  int clipper_points[][] [2], int clipper_size)
{
    //i and k are two consecutive indexes
    for (int i=0; i<clipper_size; i++)
    {
        int k = (i+1) % clipper_size;

        // We pass the current array of vertices, it's size
        // and the end points of the selected clipper line
        clip(poly_points, poly_size, clipper_points[i][0],
              clipper_points[i][1], clipper_points[k][0],
              clipper_points[k][1]);
    }

    // Printing vertices of clipped polygon
    for (int i=0; i < poly_size; i++)

```

```
        cout << '(' << poly_points[i][0] <<
            ", " << poly_points[i][1] << ") ";
    }

//Driver code
int main()
{
    // Defining polygon vertices in clockwise order
    int poly_size = 3;
    int poly_points[20][2] = {{100,150}, {200,250},
                                {300,200}};

    // Defining clipper polygon vertices in clockwise order
    // 1st Example with square clipper
    int clipper_size = 4;
    int clipper_points[] [2] = {{150,150}, {150,200},
                                {200,200}, {200,150}};

    // 2nd Example with triangle clipper
    /*int clipper_size = 3;
    int clipper_points[] [2] = {{100,300}, {300,300},
                                {200,100}};*/

    //Calling the clipping function
    suthHodgClip(poly_points, poly_size, clipper_points,
                  clipper_size);

    return 0;
}
```

Output:

(150, 162) (150, 200) (200, 200) (200, 174)

Related Articles:

- [Line Clipping | Set 1 \(Cohen–Sutherland Algorithm\)](#)
- [Point Clipping Algorithm in Computer Graphics](#)

Source

<https://www.geeksforgeeks.org/polygon-clipping-sutherland-hodgman-algorithm-please-change-bmp-images-jpeg-png/>

Chapter 159

Probability that the pieces of a broken stick form a n sided polygon

Probability that the pieces of a broken stick form a n sided polygon - GeeksforGeeks

We have a stick of length L. The stick got broken at $(n-1)$ randomly chosen points (lengths of parts can be non-integer or floating point numbers also) so we get n parts. We need to find the probability that these n pieces can form a n sided polygon.

Examples:

```
Input : L = 5 n = 3
Output : 0.25
We need to cut rope of length 5
into three parts.
```

First we need to find the condition when n lengths can form a n sided polygon. Let us consider a triangle, we know for a triangle the length of the largest side must be smaller than the sum of the lengths of other sides. Similarly for a n sided polygon the the length of the largest side must be less than the sum of the other $(n-1)$ sides. Why? Suppose we break the stick into two equal halves. We further break one of the halves into $(n-1)$ parts. We can never place them such that they make a closed polygon. (Actually the best we can do is to make 2 parallel lines). So we just need to find the probability that no part has the length greater than or equal to $L/2$.

Now we need to work on the probability. There are many ways to calculate the required probability we will use a geometric approach. Consider a circle of perimeter L. We place

n points on the perimeter. The probability that they lie on the same semicircle is $\frac{1}{2^{n-1}}$. Please refer [this](#) link for more information, let us denote it by $P(E)$.

This probability is actually same as breaking the stick such that at least one part is of length $L/2$. But we want just the complement of this event hence our answer is

$$\frac{2^n - 1}{2^n}$$

C++

```
// CPP program to find probability that
// a rope of length L when broken into
// n parts form a polygon.
#include<iostream>

using namespace std;

double printProbability(unsigned L, unsigned n)
{
    unsigned p = (1 << (n-1));
    return 1.0 - ((double)n) / ((double)p);
}

int main()
{
    unsigned n = 3, L = 5;
    cout << printProbability(L, n);
    return 0;
}
```

PHP

Output:

0.25

Improved By : SujanDutta, Abby_akku

Source

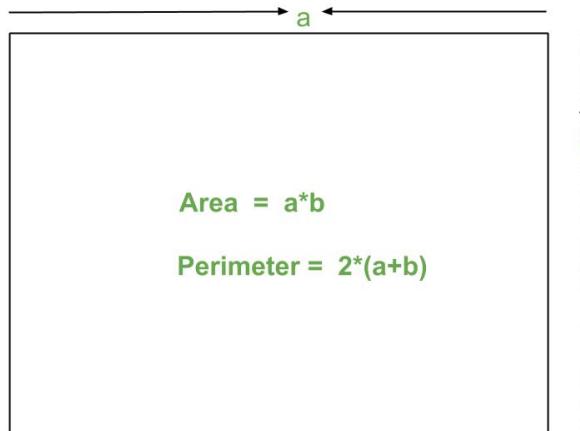
<https://www.geeksforgeeks.org/probability-that-the-pieces-of-a-broken-stick-form-a-n-sided-polygon/>

Chapter 160

Program for Area And Perimeter Of Rectangle

Program for Area And Perimeter Of Rectangle - GeeksforGeeks

A rectangle is a flat figure in a plane. It has four sides and four equal angles of 90 degree each. In rectangle all four sides are not of equal length like square, sides opposite to each other have equal length. Both diagonals of rectangle have equal length.



Examples:

Input : 4 5
Output : Area = 20

```
Perimeter = 18
```

```
Input : 2 3
Output : Area = 6
          Perimeter = 10
```

Formulae :

```
Area of rectangle : a*b
Perimeter of rectangle: 2*(a + b)
```

C++

```
// CPP program to find area
// and perimeter of rectangle
#include<bits/stdc++.h>
using namespace std;

// Utility function
int areaRectangle(int a, int b)
{
    int area = a * b;
    return area;
}

int perimeterRectangle(int a, int b)
{
    int perimeter = 2*(a + b);
    return perimeter;
}

// Driver program
int main()
{
    int a = 5;
    int b = 6;
    cout << "Area = " << areaRectangle(a, b) << endl;
    cout << "Perimeter = " << perimeterRectangle(a, b);
    return 0;
}
```

python

```
# Python3 code to find area
# and perimeter of rectangle
```

```
# Utility function
def areaRectangle(a, b):
    return (a * b)

def perimeterRectangle(a, b):
    return (2 * (a + b))

# Driver function
a = 5;
b = 6;
print ("Area = ", areaRectangle(a, b))
print ("Perimeter = ", perimeterRectangle(a, b))

# This code is contributed by 'saloni1297'.
```

Java

```
// Java program to find area
// and perimeter of rectangle
import java.io.*;

class Geometry {

    // Utility function
    static int areaRectangle(int a, int b)
    {
        int area = a * b;
        return area;
    }

    static int perimeterRectangle(int a, int b)
    {
        int perimeter = 2*(a + b);
        return perimeter;
    }

    // Driver Function
    public static void main (String[] args) {

        int a = 5;
        int b = 6;
        System.out.println("Area = "+ areaRectangle(a, b));
        System.out.println("Perimeter = "+ perimeterRectangle(a, b));

    }
}
```

```
// This code is contributed by Chinmoy Lenka
```

C#

```
// C# program to find area
// and perimeter of rectangle
using System;

class GFG {

    // Utility function
    static int areaRectangle(int a, int b)
    {
        int area = a * b;
        return area;
    }

    static int perimeterRectangle(int a, int b)
    {
        int perimeter = 2 * (a + b);
        return perimeter;
    }

    // Driver Function
    public static void Main()
    {

        int a = 5;
        int b = 6;

        Console.WriteLine("Area = "
                        + areaRectangle(a, b));
        Console.WriteLine("Perimeter = "
                        + perimeterRectangle(a, b));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find area
// and perimeter of rectangle

// Utility function
function areaRectangle( $a, $b)
```

```
{  
    $area = $a * $b;  
    return $area;  
}  
  
function perimeterRectangle( $a, $b)  
{  
    $perimeter = 2 * ($a + $b);  
    return $perimeter;  
}  
  
// Driver program  
$a = 5;  
$b = 6;  
echo("Area = ");  
echo(areaRectangle($a, $b));  
echo("\n");  
echo( "Perimeter = ");  
echo(perimeterRectangle($a, $b));  
  
// This code is contributed by vt_m.  
?>
```

Output :

```
Area = 30  
Perimeter = 22
```

Improved By : [vt_m](#)

Source

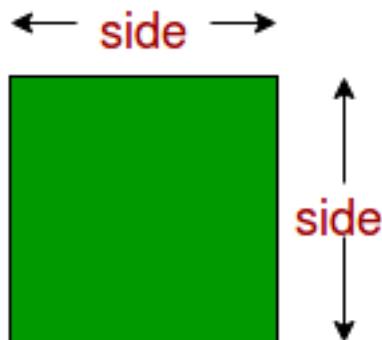
<https://www.geeksforgeeks.org/program-area-perimeter-rectangle/>

Chapter 161

Program for Area Of Square

Program for Area Of Square - GeeksforGeeks

A square is a flat shape, in one plane, defined by four points at the four corners. A square has four sides all of equal length, and four corners, all right angles (90 degree angles). A square is a kind of rectangle.



$$\text{Area} = \text{side}^2$$

Examples :

Input : 4
Output : 16

Input : 8
Output : 64

Formula

$$\text{Area} = \text{side} * \text{side}$$

C++

```
// CPP program to find
// the area of the square
#include <iostream>
using namespace std;

int areaSquare(int side)
{
    int area = side * side;
    return area;
}

// Driver Code
int main()
{
    int side = 4;
    cout << areaSquare(side);
    return 0;
}
```

Java

```
// Java program to find
// the area of the square
import java.util.*;

class GFG
{
    static int areaSquare(int side)
    {
        int area = side * side;
        return area;
    }

    // Driver Code
    public static void main(String[] args)
    {
        int side = 5;
        System.out.println(areaSquare(4));
    }
}
```

Python3

```
# Python3 code to find
# the area of the square
```

```
def areaSquare( side ):
    area = side * side
    return area

# Driver Code
side = 4
print(areaSquare(side))

# This code is contributed
# by "Sharad_Bhardwaj".
```

C#

```
// C# program to find
// the area of the square
using System;

class GFG
{
    static int areaSquare(int side)
    {
        int area = side * side;
        return area;
    }

    // Driver code
    public static void Main()
    {
        int side = 4;

        Console.WriteLine(areaSquare(side));
    }
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP program to find
// the area of the square

function areaSquare($side)
{
    $area = $side * $side;
    return $area;
```

```
}
```

```
// Driver Code
$side = 4;
echo(areaSquare($side));
```

```
// This code is contributed by Ajit.
?>
```

Output :

16

Improved By : [Rajinikanth Maila, jit_t](#)

Source

<https://www.geeksforgeeks.org/program-area-square/>

Chapter 162

Program for Circumference of a Parallelogram

Program for Circumference of a Parallelogram - GeeksforGeeks

Given the sides of Parallelogram then calculate the circumference.

Examples :

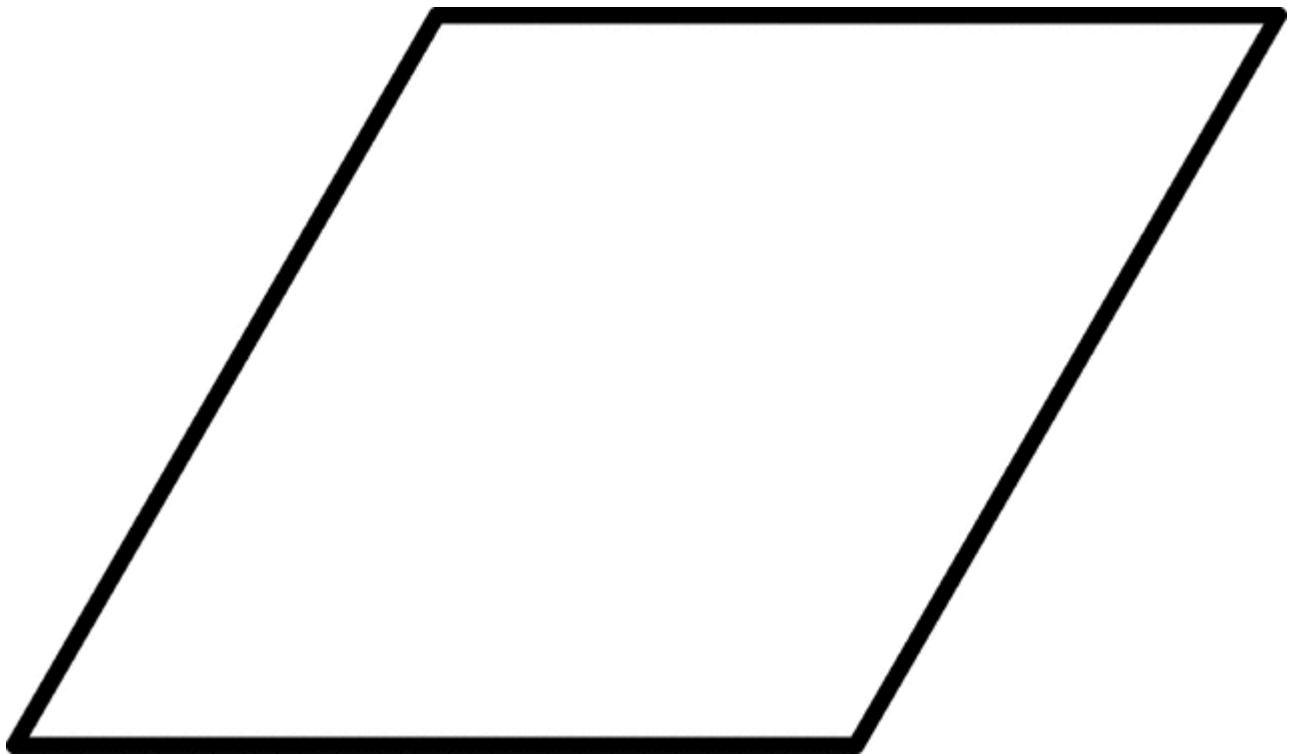
Input: a = 10, b = 8

Output: 36.00

Input: a = 25.12, b = 20.4

Output: 91.04

The opposite sides of a parallelogram are of equal length and parallel. The angles are pairwise equal but not necessarily 90 degree. The circumference of a Parallelogram can be computed as sum of two adjacent sides each multiplied by 2.



Formula used to calculate circumference of a Parallelogram:
 $(2*a)+(2*b)$

C++

```
// C Program to calculate the
// Circumference of a Parallelogram.
#include <stdio.h>

float circumferenceparallelogram(float a, float b)
{
    return ((2 * a) + (2 * b));
}

int main()
{
    float a = 10, b = 8;
    printf("Circumference of a given Parallelogram is : %.2f",
           circumferenceparallelogram(a, b));
    return 0;
}
```

Java

```
// Java Program To Calculate
// Circumference of a Parallelogram
import java.io.*;

class GFG
{
    static float circumferenceparallelogram(float a, float b)
    {
        return ((2 * a) + (2 * b));
    }

    // Driver code
    public static void main(String arg[])
    {
        float a = 10, b = 8;
        System.out.print("Circumference of a given Parallelogram is :");
        System.out.println(circumferenceparallelogram(a, b));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 Program to calculate the
# Circumference of a Parallelogram

# Utility Function
def circumferenceparallelogram(a,b):
    return ((2 * a) + (2 * b))

# Driver Function
a = 10
b = 8

print("Circumference of a given Parallelogram is :",
      round(circumferenceparallelogram(a, b),4))

# This code is contributed
# by Azkia Anam.
```

C#

```
// C# Program To Calculate
// Circumference of a Parallelogram
using System;
```

```
class GFG
{
    static float circumferenceparallelogram(float a,
                                              float b)
    {
        return ((2 * a) + (2 * b));
    }

    // Driver code
    public static void Main()
    {
        float a = 10, b = 8;
        Console.Write("Circumference of a" +
                      "given Parallelogram is :");

        Console.WriteLine(circumferenceparallelogram(a, b));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP Program to calculate
// the Circumference of a
// Parallelogram.

function circumferenceparallelogram( $a, $b)
{
    return ((2 * $a) + (2 * $b));
}

$a = 10; $b = 8;

echo "Circumference of a ".
      "given Parallelogram is :",
      circumferenceparallelogram($a, $b);

// This code is contributed by anuj_67.
?>
```

Output :

```
Circumference of a given parallelogram is : 36.00
```

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/program-circumference-parallelogram/>

Chapter 163

Program for Point of Intersection of Two Lines

Program for Point of Intersection of Two Lines - GeeksforGeeks

Given points A and B corresponding to line AB and points P and Q corresponding to line PQ, find the point of intersection of these lines. The points are given in 2D Plane with their X and Y Coordinates.

Examples:

```
Input : A = (1, 1), B = (4, 4)
        C = (1, 8), D = (2, 4)
Output : The intersection of the given lines
         AB and CD is: (2.4, 2.4)
```

```
Input : A = (0, 1), B = (0, 4)
        C = (1, 8), D = (1, 4)
Output : The given lines AB and CD are parallel.
```

First of all, let us assume that we have two points (x_1, y_1) and (x_2, y_2) . Now, we find the equation of line formed by these points.

Let the given lines be :

1. $a_1x + b_1y = c_1$
2. $a_2x + b_2y = c_2$

We have to now solve these 2 equations to find the point of intersection. To solve, we multiply 1. by b_2 and 2 by b_1
This gives us,

$$\begin{aligned} a_1 b_2 x + b_1 b_2 y &= c_1 b_2 \\ a_2 b_1 x + b_2 b_1 y &= c_2 b_1 \end{aligned}$$

Subtracting these we get,
 $(a_1 b_2 - a_2 b_1) x = c_1 b_2 - c_2 b_1$

This gives us the value of x. Similarly, we can find the value of y. (x, y) gives us the point of intersection.

Note: This gives the point of intersection of two lines, but if we are given line segments instead of lines, we have to also recheck that the point so computed actually lies on both the line segments.

If the line segment is specified by points (x_1, y_1) and (x_2, y_2) , then to check if (x, y) is on the segment we have to just check that

- $\min(x_1, x_2) \leq x \leq \max(x_1, x_2)$
- $\min(y_1, y_2) \leq y \leq \max(y_1, y_2)$

The pseudo code for the above implementation:

```
determinant = a1 b2 - a2 b1
if (determinant == 0)
{
    // Lines are parallel
}
else
{
    x = (c1b2 - c2b1)/determinant
    y = (a1c2 - a2c1)/determinant
}
```

These can be derived by first getting the slope directly and then finding the intercept of the line.

C++

```
// C++ Implementation. To find the point of
// intersection of two lines
#include <bits/stdc++.h>
using namespace std;

// This pair is used to store the X and Y
// coordinates of a point respectively
#define pdd pair<double, double>

// Function used to display X and Y coordinates
// of a point
```

```
void displayPoint(pdd P)
{
    cout << "(" << P.first << ", " << P.second
        << ")" << endl;
}

pdd lineLineIntersection(pdd A, pdd B, pdd C, pdd D)
{
    // Line AB represented as a1x + b1y = c1
    double a1 = B.second - A.second;
    double b1 = A.first - B.first;
    double c1 = a1*(A.first) + b1*(A.second);

    // Line CD represented as a2x + b2y = c2
    double a2 = D.second - C.second;
    double b2 = C.first - D.first;
    double c2 = a2*(C.first)+ b2*(C.second);

    double determinant = a1*b2 - a2*b1;

    if (determinant == 0)
    {
        // The lines are parallel. This is simplified
        // by returning a pair of FLT_MAX
        return make_pair(FLT_MAX, FLT_MAX);
    }
    else
    {
        double x = (b2*c1 - b1*c2)/determinant;
        double y = (a1*c2 - a2*c1)/determinant;
        return make_pair(x, y);
    }
}

// Driver code
int main()
{
    pdd A = make_pair(1, 1);
    pdd B = make_pair(4, 4);
    pdd C = make_pair(1, 8);
    pdd D = make_pair(2, 4);

    pdd intersection = lineLineIntersection(A, B, C, D);

    if (intersection.first == FLT_MAX &&
        intersection.second==FLT_MAX)
    {
        cout << "The given lines AB and CD are parallel.\n";
    }
}
```

```
}

else
{
    // NOTE: Further check can be applied in case
    // of line segments. Here, we have considered AB
    // and CD as lines
    cout << "The intersection of the given lines AB "
        "and CD is: ";
    displayPoint(intersection);
}

return 0;
}
```

Java

```
// Java Implementation. To find the point of
// intersection of two lines

// Class used to store the X and Y
// coordinates of a point respectively
class Point
{
    double x,y;

    public Point(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    // Method used to display X and Y coordinates
    // of a point
    static void displayPoint(Point p)
    {
        System.out.println("(" + p.x + ", " + p.y + ")");
    }
}

class Test
{
    static Point lineLineIntersection(Point A, Point B, Point C, Point D)
    {
        // Line AB represented as a1x + b1y = c1
        double a1 = B.y - A.y;
        double b1 = A.x - B.x;
        double c1 = a1*(A.x) + b1*(A.y);
```

```
// Line CD represented as a2x + b2y = c2
double a2 = D.y - C.y;
double b2 = C.x - D.x;
double c2 = a2*(C.x)+ b2*(C.y);

double determinant = a1*b2 - a2*b1;

if (determinant == 0)
{
    // The lines are parallel. This is simplified
    // by returning a pair of FLT_MAX
    return new Point(Double.MAX_VALUE, Double.MAX_VALUE);
}
else
{
    double x = (b2*c1 - b1*c2)/determinant;
    double y = (a1*c2 - a2*c1)/determinant;
    return new Point(x, y);
}

// Driver method
public static void main(String args[])
{
    Point A = new Point(1, 1);
    Point B = new Point(4, 4);
    Point C = new Point(1, 8);
    Point D = new Point(2, 4);

    Point intersection = lineLineIntersection(A, B, C, D);

    if (intersection.x == Double.MAX_VALUE &&
        intersection.y == Double.MAX_VALUE)
    {
        System.out.println("The given lines AB and CD are parallel.");
    }
    else
    {
        // NOTE: Further check can be applied in case
        // of line segments. Here, we have considered AB
        // and CD as lines
        System.out.print("The intersection of the given lines AB" +
                        "and CD is: ");
        Point.displayPoint(intersection);
    }
}
```

}

Output:

The intersection of the given lines AB and
CD is: (2.4, 2.4)

Source

<https://www.geeksforgeeks.org/program-for-point-of-intersection-of-two-lines/>

Chapter 164

Program for Surface Area of Octahedron

Program for Surface Area of Octahedron - GeeksforGeeks

Given the sides of Octahedron then calculate the surface area.

Examples:

Input : 7
Output : 169.741

Input : 9
Output : 280.59

An regular octahedron has eight faces, which are all in the shape of equilateral triangles. The area of an octahedron is 2 multiplied by the length of an edge squared multiplied by the square root of three.

Formula:
Surface area= $2 * (\sqrt{3}) * (\text{side} * \text{side})$

C++

```
// CPP Program to calculate
// surface area of Octahedron
#include <bits/stdc++.h>
using namespace std;

// utility Function
double surface_area_octahedron(double side)
```

```
{  
    return (2*(sqrt(3))*(side*side));  
}  
  
// Driver Function  
int main()  
{  
    double side = 7;  
    cout << "Surface area of octahedron ="  
        << surface_area_octahedron(side)  
        << endl;  
}
```

Java

```
// Java Program to calculate  
// surface area of Octahedron.  
  
import java.io.*;  
import java.util.*;  
  
class GFG {  
  
    // utility Function  
    static double surface_area_octahedron(double side)  
{  
        return (2*(Math.sqrt(3))*(side*side));  
    }  
    public static void main (String[] args) {  
        double side = 7;  
        System.out.println("Surface area of octahedron ="  
            + surface_area_octahedron(side));  
    }  
}  
  
// This code is contributed by Gitanjali.
```

Python3

```
# Python Program to calculate  
# surface area of Octahedron.  
import math  
  
# utility Function  
def surface_area_octahedron( side):
```

```
return (2*(math.sqrt(3))*(side*side))

# driver code
side = 7
print("Surface area of octahedron =" ,
      surface_area_octahedron(side))

# This code is contributed by Gitanjali.
```

C#

```
// C# program to calculate
// surface area of Octahedron.
using System;

class GFG {

    // utility Function
    static double surface_area_octahedron(double side)
    {
        return (2 * (Math.Sqrt(3)) * (side * side));
    }

    // Driver code
    public static void Main()
    {
        double side = 7;
        Console.WriteLine("Surface area of octahedron =" +
                          + surface_area_octahedron(side));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP Program to calculate
// surface area of Octahedron

// utility Function
function surface_area_octahedron($side)
{
    return (2 * (sqrt(3)) *
           ($side * $side));
}
```

```
// Driver Code
$side = 7;
echo("Surface area of octahedron =");
echo( surface_area_octahedron($side));

// This code is contributed by vt_m.
?>
```

Output:

```
Surface area of octahedron =169.741
```

Improved By : [vt_m](#)

Source

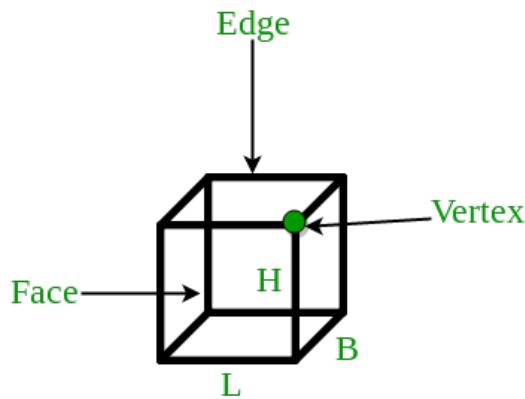
<https://www.geeksforgeeks.org/program-for-surface-area-of-octahedron/>

Chapter 165

Program for Volume and Surface Area of Cube

Program for Volume and Surface Area of Cube - GeeksforGeeks

Cube is a 3-dimensional box-like figure represented in the 3-dimensional plane. Cube has 6 squared-shape equal faces. Each face meet another face at 90 degree each. Three sides of cube meet at same vertex.



$$\text{Length}(L) = \text{Breadth}(B) = \text{Height}(H)$$

Examples:

```
Input : Side of a cube = 2
Output : Area = 8
         Total surface area = 24
```

```
Input : Side of a cube = 3
Output : Area = 27
         Total surface area = 54
```

Volume: a^3

Total Surface area: $6a^2$

C++

```
// CPP program to find area
// and total surface area of cube
#include <bits/stdc++.h>
using namespace std;

// utility function
double areaCube(double a)
{
    return (a * a * a);
}

double surfaceCube(double a)
{
    return (6 * a * a);
}

// driver function
int main()
{
    double a = 5;
    cout << "Area = " << areaCube(a) << endl;
    cout << "Total surface area = " << surfaceCube(a);
    return 0;
}
```

Java

```
// Java program to find area
// and total surface area of cube

class GFG
{
    // utility function
    static double areaCube(double a)
    {
        return (a * a * a);
    }

    static double surfaceCube(double a)
    {
        return (6 * a * a);
    }
}
```

```
// Driver code
public static void main (String[] args)
{
    double a = 5;
    System.out.println("Area = "+areaCube(a));
    System.out.println("Total surface area = "
                       +surfaceCube(a));
}
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 code to find area
# and total surface area of cube

# utility function
def areaCube( a ):
    return (a * a * a)

def surfaceCube( a ):
    return (6 * a * a)

# driver function
a = 5
print("Area =", areaCube(a))
print("Total surface area =", surfaceCube(a))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# program to find area
// and total surface area of cube
using System;

class GFG {

    // utility function
    static double areaCube(double a)
    {
        return (a * a * a);
    }

    static double surfaceCube(double a)
```

```
{  
    return (6 * a * a);  
}  
  
// Driver code  
public static void Main()  
{  
    double a = 5;  
  
    Console.WriteLine("Area = " + areaCube(a));  
    Console.WriteLine("Total surface area = "  
                     + surfaceCube(a));  
}  
}  
  
// This code is contributed by vt_m.
```

PHP

```
<?php  
// PHP program to find area  
// and total surface area of cube  
  
// utility function  
function areaCube($a)  
{  
    return ($a * $a * $a);  
}  
  
function surfaceCube( $a)  
{  
    return (6 * $a * $a);  
}  
  
// driver function  
  
$a = 5;  
echo ("Area = ");  
echo(areaCube($a));  
echo("\n");  
echo("Total surface area = ");  
echo(surfaceCube($a));  
  
// This code is contributed by vt_m.  
?>
```

Output:

```
Area = 125
Total surface area = 150
```

Improved By : [vt_m](#)

Source

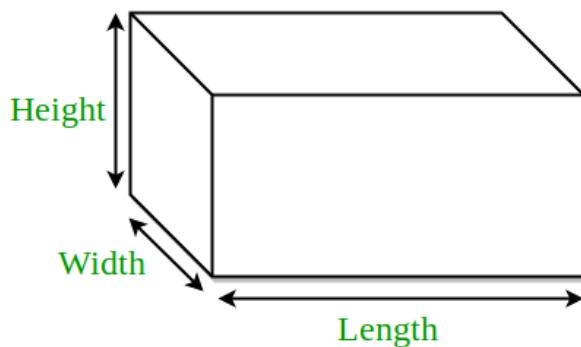
<https://www.geeksforgeeks.org/program-volume-surface-area-cube/>

Chapter 166

Program for Volume and Surface Area of Cuboid

Program for Volume and Surface Area of Cuboid - GeeksforGeeks

Cuboid is a 3-dimensional box-like figure represented in the 3-dimensional plane. Cuboid has 6 rectangle-shaped faces. Each face meets another face at 90 degree each. Three sides of cuboid meet at same vertex. Since it is made up of 6 rectangle faces, it has length, width and height of different dimension.



Examples :

```
Input : 2 3 4
Output : Area = 24
          Total Surface Area = 52
```

```
Input : 5 6 12
Output : Area = 360
          Total Surface Area = 324
```

Formulae :

```
Area = l*w*h  
Total Surface Area = 2*l*w + 2*w*h + 2*l*h  
where l, h, w are length, height and width of  
cuboid respectively.
```

C++

```
// CPP program to find volume and  
// total surface area of cuboid  
#include <bits/stdc++.h>  
using namespace std;  
  
// utility function  
double areaCuboid(double l, double h, double w)  
{  
    return (l * h * w);  
}  
  
double surfaceAreaCuboid(double l, double h, double w)  
{  
    return (2 * l * w + 2 * w * h + 2 * l * h);  
}  
  
// driver function  
int main()  
{  
    double l = 1;  
    double h = 5;  
    double w = 7;  
    cout << "Area = " << areaCuboid(l, h, w) << endl;  
    cout << "Total Surface Area = "  
        << surfaceAreaCuboid(l, h, w);  
    return 0;  
}
```

Java

```
// Java program to find volume and  
// total surface area of cuboid  
  
class GFG  
{  
    // utility function  
    static double areaCuboid(double l, double h,
```

```
        double w)
{
    return (l * h * w);
}

static double surfaceAreaCuboid(double l, double h,
                                double w)
{
    return (2 * l * w + 2 * w * h + 2 * l * h);
}

// Driver code
public static void main (String[] args)
{
    double l = 1;
    double h = 5;
    double w = 7;
    System.out.println("Area = " + areaCuboid(l, h, w));
    System.out.println("Total Surface Area = "
                       + surfaceAreaCuboid(l, h, w));
}
}

// This code is contributed By Anant Agarwal.
```

Python3

```
# Python3 code to find volume and
# total surface area of cuboid

# utility function
def volumeCuboid( l , h , w ):
    return (l * h * w)

def surfaceAreaCuboid( l , h , w ):
    return (2 * l * w + 2 * w * h + 2 * l * h)

# driver function
l = 1
h = 5
w = 7
print("Volume =", volumeCuboid(l, h, w))
print("Total Surface Area =", surfaceAreaCuboid(l, h, w))

#This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# program to find volume and
```

```
// total surface area of cuboid
using System;

class GFG {

    // utility function
    static double areaCuboid(double l, double h,
                             double w)
    {
        return (l * h * w);
    }

    static double surfaceAreaCuboid(double l, double h,
                                    double w)
    {
        return (2 * l * w + 2 * w * h + 2 * l * h);
    }

    // Driver code
    public static void Main()
    {
        double l = 1;
        double h = 5;
        double w = 7;

        Console.WriteLine("Area = " + areaCuboid(l, h, w));
        Console.WriteLine("Total Surface Area = "
                         + surfaceAreaCuboid(l, h, w));
    }
}

// This code is contributed By vt_m.
```

PHP

```
<?php
// PHP program to find volume and
// total surface area of cuboid

// utility function
function areaCuboid($l, $h, $w)
{
    return ($l * $h * $w);
}

function surfaceAreaCuboid($l, $h, $w)
{
    return (2 * $l * $w + 2 * $w *
```

```
$h + 2 * $l * $h);  
}  
  
// Driver Code  
$l = 1;  
$h = 5;  
$w = 7;  
echo "Area = " ,  
     areaCuboid($l, $h, $w) , "\n";  
echo "Total Surface Area = ",  
     surfaceAreaCuboid($l, $h, $w);  
  
// This code is contributed by ajit  
?>
```

Output :

```
Area = 35  
Total Surface Area = 94
```

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/program-for-volume-and-surface-area-of-cuboid/>

Chapter 167

Program for Volume and Surface area of Frustum of Cone

Program for Volume and Surface area of Frustum of Cone - GeeksforGeeks

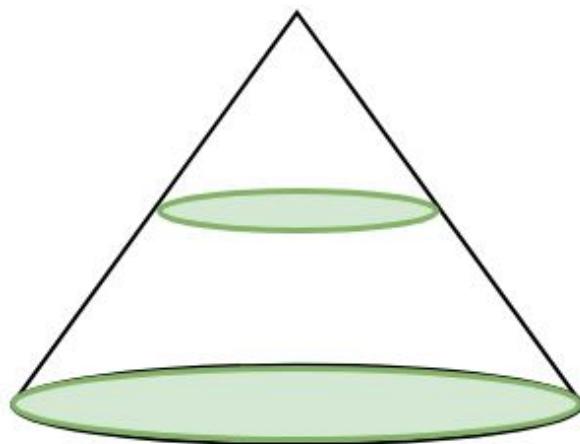
Given slant height, height and radius of a frustum of a cone, we have to calculate the volume and surface area of the frustum of a cone.

Frustum of cone

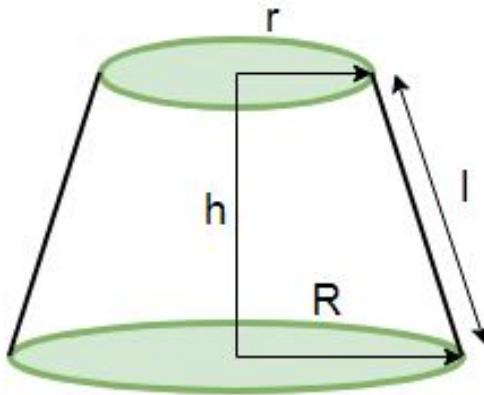
In geometry, a frustum is the portion of a solid (normally a cone or pyramid) that lies between one or two parallel planes cutting it.

If we cut a right circular cone by a plane parallel to its base, the portion of the solid between this plane and the base is known as the frustum of a cone.

Given below is a right circular cone.



The right circular cone after being cut by a plane parallel to its base results in a frustum as follows:



which has a circular base at bottom of radius R
circular upper portion with radius r
height h
and slant height l

- **Volume of frustum of cone:**

$$\text{Volume } (V) = \frac{1}{3} * \pi * h(r^2 + R^2 + r*R)$$

where

r = radius of smaller circle

R = radius of bigger circle (or radius of base of the cone)

h = height of the frustum

Curved Surface Area of frustum of cone:

$$\text{Curved Surface Area (CSA)} = \pi * l(R + r)$$

where

r = radius of smaller circle

R = radius of bigger circle

l = slant height of the frustum

TotalSurface Area of frustum of cone:

Total Surface Area (TSA) = $\pi * l(R + r) + \pi(R^2 + r^2)$

where

r = radius of smaller circle

R = radius of bigger circle

l = slant height of frustum

Examples:

Input : Radius of smaller circle = 3
Radius of bigger circle = 8
Height of frustum = 12
Slant height of frustum = 13
Output :
Volume Of Frustum of Cone : 1218.937
Curved Surface Area Of Frustum of Cone : 449.24738
Total Surface Area Of Frustum of Cone : 678.58344

Input : Radius of smaller circle = 7
Radius of bigger circle = 10
Height of frustum = 4
Slant height of frustum = 5
Output :
Volume Of Frustum of Cone : 917.34436
Curved Surface Area Of Frustum of Cone : 267.03516
Total Surface Area Of Frustum of Cone : 735.1321

C++

```
// CPP program to calculate Volume and
// Surface area of frustum of cone
#include <iostream>
using namespace std;

float pi = 3.14159;

// Function to calculate Volume of frustum of cone
float volume(float r, float R, float h)
{
    return (float(1) / float(3)) * pi * h *
           (r * r + R * R + r * R);
}

// Function to calculate Curved Surface area of
```

```
// frustum of cone
float curved_surface_area(float r, float R, float l)
{
    return pi * l * (R + r);
}

// Function to calculate Total Surface area of
// frustum of cone
float total_surface_area(float r, float R, float l,
                        float h)
{
    return pi * l * (R + r) + pi * (r * r + R * R);
}

// Driver function
int main()
{
    float small_radius = 3;
    float big_radius = 8;
    float slant_height = 13;
    float height = 12;

    // Printing value of volume and surface area
    cout << "Volume Of Frustum of Cone : "
        << volume(small_radius, big_radius, height)
        << endl;

    cout << "Curved Surface Area Of Frustum of Cone : "
        << curved_surface_area(small_radius, big_radius,
                               slant_height) << endl;

    cout << "Total Surface Area Of Frustum of Cone : "
        << total_surface_area(small_radius, big_radius,
                              slant_height, height);
    return 0;
}
```

Java

```
// Java program to calculate Volume and Surface area
// of frustum of cone

public class demo {

    static float pi = 3.14159f;

    // Function to calculate Volume of frustum of cone
    public static float volume(float r, float R, float h)
```

```
{  
    return (float)1 / 3 * pi * h * (r * r + R * R +  
                                    r * R);  
}  
  
// Function to calculate Curved Surface area of  
// frustum of cone  
public static float curved_surface_area(float r,  
                                         float R, float l)  
{  
    return pi * l * (R + r);  
}  
  
// Function to calculate Total Surface area of  
// frustum of cone  
public static float total_surface_area(float r,  
                                       float R, float l, float h)  
{  
    return pi * l * (R + r) + pi * (r * r + R * R);  
}  
  
// Driver function  
public static void main(String args[])  
{  
    float small_radius = 3;  
    float big_radius = 8;  
    float slant_height = 13;  
    float height = 12;  
  
    // Printing value of volume and surface area  
    System.out.print("Volume Of Frustum of Cone : ");  
    System.out.println(volume(small_radius,  
                             big_radius, height));  
  
    System.out.print("Curved Surface Area Of" +  
                     " Frustum of Cone : ");  
    System.out.println(curved_surface_area(small_radius,  
                                           big_radius, slant_height));  
    System.out.print("Total Surface Area Of" +  
                     " Frustum of Cone : ");  
  
    System.out.println(total_surface_area(small_radius,  
                                         big_radius, slant_height, height));  
}
```

Python3

```
# Python3 code to calculate
# Volume and Surface area of
# frustum of cone
import math

pi = math.pi

# Function to calculate Volume
# of frustum of cone
def volume( r , R , h ):
    return 1 /3 * pi * h * (r
                           * r + R * R + r * R)

# Function to calculate Curved
# Surface area of frustum of cone
def curved_surface_area( r , R , l ):
    return pi * l * (R + r)

# Function to calculate Total
# Surface area of frustum of cone
def total_surface_area( r , R , l , h ):
    return pi * l * (R + r) + pi * (r
                                    * r + R * R)

# Driver Code
small_radius = 3
big_radius = 8
slant_height = 13
height = 12

# Printing value of volume
# and surface area
print("Volume Of Frustum of Cone : "
      ,end='')
print(volume(small_radius, big_radius,
            height))

print("Curved Surface Area Of Frustum"+
      " of Cone : ",end='')
print(curved_surface_area(small_radius,
                          big_radius,slant_height))

print("Total Surface Area Of Frustum"+
      " of Cone : ",end='')
print(total_surface_area(small_radius,
                        big_radius,slant_height, height))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# program to calculate Volume and
// Surface area of frustum of cone
using System;

public class demo {

    static float pi = 3.14159f;

    // Function to calculate
    // Volume of frustum of cone
    public static float volume(float r, float R, float h)
    {
        return (float)1 / 3 * pi * h * (r * r + R *
                                         R + r * R);
    }

    // Function to calculate Curved
    // Surface area of frustum of cone
    public static float curved_surface_area(float r,
                                             float R, float l)
    {
        return pi * l * (R + r);
    }

    // Function to calculate Total
    // Surface area of frustum of cone
    public static float total_surface_area(float r, float R,
                                           float l, float h)
    {
        return pi * l * (R + r) + pi *
               (r * r + R * R);
    }

    // Driver function
    public static void Main()
    {
        float small_radius = 3;
        float big_radius = 8;
        float slant_height = 13;
        float height = 12;

        // Printing value of volume
        // and surface area
        Console.Write("Volume Of Frustum of Cone : ");

        Console.WriteLine(volume(small_radius,
```

```
        big_radius, height));  
  
Console.WriteLine("Curved Surface Area Of" +  
    " Frustum of Cone : ");  
  
Console.WriteLine(curved_surface_area(small_radius,  
        big_radius, slant_height));  
  
Console.WriteLine("Total Surface Area Of" +  
    " Frustum of Cone : ");  
  
Console.WriteLine(total_surface_area(small_radius,  
        big_radius, slant_height, height));  
}  
}  
  
// This article is contributed by vt_m
```

PHP

```
<?php  
// PHP program to calculate Volume and  
// Surface area of frustum of cone  
  
// Function to calculate  
// Volume of frustum of cone  
function volume($r, $R, $h)  
{  
    $pi = 3.14159;  
    return (1 / (3)) * $pi * $h *  
        ($r * $r + $R * $R + $r * $R);  
}  
  
// Function to calculate Curved  
// Surface area of frustum of cone  
function curved_surface_area($r, $R, $l)  
{  
    $pi = 3.14159;  
    return $pi * $l * ($R + $r);  
}  
  
// Function to calculate Total Surface  
// area of frustum of cone  
function total_surface_area( $r, $R, $l, $h)  
{  
    $pi = 3.14159;  
    return ($pi * $l * ($R + $r) +  
        $pi * ($r * $r + $R * $R));
```

```
}

// Driver Code
$small_radius = 3;
$big_radius = 8;
$slant_height = 13;
$height = 12;

// Printing value of volume
// and surface area
echo("Volume Of Frustum of Cone : ");
echo(volume($small_radius,
            $big_radius,
            $height));
echo("\n");

echo("Curved Surface Area Of Frustum of Cone : ");
echo (curved_surface_area($small_radius,
                           $big_radius ,
                           $slant_height));
echo("\n");

echo("Total Surface Area Of Frustum of Cone : ");
echo(total_surface_area($small_radius,
                        $big_radius,
                        $slant_height,
                        $height));

// This code is contributed by vt_m
?>
```

Output:

```
Volume Of Frustum of Cone : 1218.937
Curved Surface Area Of Frustum of Cone : 449.24738
Total Surface Area Of Frustum of Cone : 678.58344
```

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/program-for-volume-and-surface-area-of-frustum-of-cone/>

Chapter 168

Program for distance between two points on earth

Program for distance between two points on earth - GeeksforGeeks

Given latitude and longitude in degrees find the distance between two points on the earth.

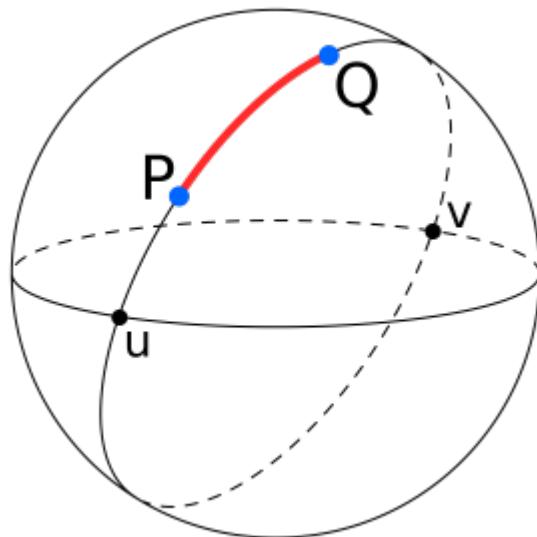


Image Source : [Wikipedia](#)

Examples:

```
Input : Latitude 1: 53.3205555555556
        Latitude 2: 53.31861111111111
        Longitude 1: -1.7297222222222221
```

```
Longitude 2: -1.6997222222222223
Output: Distance is: 2.0043678382716137 Kilometers
```

Problem can be solved using Haversine formula:

The great circle distance or the **orthodromic** distance is the shortest distance between two points on a sphere (or the surface of Earth). In order to use this method, we need to have the co-ordinates of point A and point B. The great circle method is chosen over other methods.

First, convert the latitude and longitude values from decimal degrees to radians. For this divide the values of longitude and latitude of both the points by 180/pi. The value of pi is 22/7. The value of 180/pi is approximately 57.29577951. If we want to calculate the distance between two places in miles, use the value 3, 963, which is the radius of Earth. If we want to calculate the distance between two places in kilometers, use the value 6, 378.8, which is the radius of Earth.

Find the value of the latitude in radians:

Value of Latitude in Radians, lat = Latitude / (180/pi) OR

Value of Latitude in Radians, lat = Latitude / 57.29577951

Find the value of longitude in radians:

Value of Longitude in Radians, long = Longitude / (180/pi) OR

Value of Longitude in Radians, long = Longitude / 57.29577951

Get the co-ordinates of point A in terms of latitude and longitude. Use the above conversion method to convert the values of latitude and longitude in radians. I will call it as lat1 and long1. Do the same for the co-ordinates of Point B and get lat2 and long2.

Now, to get the distance between point A and point B use the following formula:

$$\text{Distance, } d = 3963.0 * \arccos[(\sin(\text{lat1}) * \sin(\text{lat2})) + \cos(\text{lat1}) * \cos(\text{lat2}) * \cos(\text{long2} - \text{long1})]$$

The obtained distance, d, is in miles. If you want your value to be in units of kilometers, multiple d by 1.609344.

$$d \text{ in kilometers} = 1.609344 * d \text{ in miles}$$

Thus you can have the shortest distance between two places on Earth using the great circle distance approach.

C++

```
// C++ program to calculate Distance
// Between Two Points on Earth
#include <bits/stdc++.h>
```

```
using namespace std;

// Utility function for
// converting degrees to radians
long double toRadians(const long double &degree)
{
    // cmath library in C++
    // defines the constant
    // M_PI as the value of
    // pi accurate to 1e-30
    long double one_deg = (M_PI) / 180;
    return (one_deg * degree);
}

long double distance(long double lat1, long double long1,
                     long double lat2, long double long2)
{
    // Convert the latitudes
    // and longitudes
    // from degree to radians.
    lat1 = toRadians(lat1);
    long1 = toRadians(long1);
    lat2 = toRadians(lat2);
    long2 = toRadians(long2);

    // Haversine Formula
    long double dlong = long2 - long1;
    long double dlat = lat2 - lat1;

    long double ans = pow(sin(dlat / 2), 2) +
                    cos(lat1) * cos(lat2) *
                    pow(sin(dlong / 2), 2);

    ans = 2 * asin(sqrt(ans));

    // Radius of Earth in
    // Kilometers, R = 6371
    // Use R = 3956 for miles
    long double R = 6371;

    // Calculate the result
    ans = ans * R;

    return ans;
}

// Driver Code
int main()
```

```
{  
    long double lat1 = 53.32055555555556;  
    long double long1 = -1.729722222222221;  
    long double lat2 = 53.31861111111111;  
    long double long2 = -1.699722222222223;  
  
    // call the distance function  
    cout << setprecision(15) << fixed;  
    cout << distance(lat1, long1,  
                      lat2, long2) << " K.M";  
  
    return 0;  
}  
  
// This code is contributed  
// by Aayush Chaturvedi
```

Java

```
// Java program to calculate Distance Between  
// Two Points on Earth  
import java.util.*;  
import java.lang.*;  
  
class GFG {  
  
    public static double distance(double lat1,  
                                 double lat2, double lon1,  
                                 double lon2)  
    {  
  
        // The math module contains a function  
        // named toRadians which converts from  
        // degrees to radians.  
        lon1 = Math.toRadians(lon1);  
        lon2 = Math.toRadians(lon2);  
        lat1 = Math.toRadians(lat1);  
        lat2 = Math.toRadians(lat2);  
  
        // Haversine formula  
        double dlon = lon2 - lon1;  
        double dlat = lat2 - lat1;  
        double a = Math.pow(Math.sin(dlat / 2), 2)  
                  + Math.cos(lat1) * Math.cos(lat2)  
                  * Math.pow(Math.sin(dlon / 2), 2);  
  
        double c = 2 * Math.asin(Math.sqrt(a));  
    }  
}
```

```
// Radius of earth in kilometers. Use 3956
// for miles
double r = 6371;

// calculate the result
return(c * r);
}

// driver code
public static void main(String[] args)
{
    double lat1 = 53.3205555555556;
    double lat2 = 53.31861111111111;
    double lon1 = -1.7297222222222221;
    double lon2 = -1.6997222222222223;
    System.out.println(distance(lat1, lat2,
                                lon1, lon2) + " K.M");
}
}

// This code is contributed by Prasad Kshirsagar
```

Python3

```
# Python 3 program to calculate Distance Between Two Points on Earth
from math import radians, cos, sin, asin, sqrt
def distance(lat1, lat2, lon1, lon2):

    # The math module contains a function named
    # radians which converts from degrees to radians.
    lon1 = radians(lon1)
    lon2 = radians(lon2)
    lat1 = radians(lat1)
    lat2 = radians(lat2)

    # Haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2

    c = 2 * asin(sqrt(a))

    # Radius of earth in kilometers. Use 3956 for miles
    r = 6371

    # calculate the result
    return(c * r)
```

```
# driver code
lat1 = 53.32055555555556
lat2 = 53.31861111111111
lon1 = -1.7297222222222221
lon2 = -1.6997222222222223
print(distance(lat1, lat2, lon1, lon2), "K.M")
```

C#

```
// C# program to calculate
// Distance Between Two
// Points on Earth
using System;

class GFG
{
    static double toRadians(
        double angleIn10thofaDegree)
    {
        // Angle in 10th
        // of a degree
        return (angleIn10thofaDegree *
            Math.PI) / 180;
    }
    static double distance(double lat1,
        double lat2,
        double lon1,
        double lon2)
    {

        // The math module contains
        // a function named toRadians
        // which converts from degrees
        // to radians.
        lon1 = toRadians(lon1);
        lon2 = toRadians(lon2);
        lat1 = toRadians(lat1);
        lat2 = toRadians(lat2);

        // Haversine formula
        double dlon = lon2 - lon1;
        double dlat = lat2 - lat1;
        double a = Math.Pow(Math.Sin(dlat / 2), 2) +
            Math.Cos(lat1) * Math.Cos(lat2) *
            Math.Pow(Math.Sin(dlon / 2), 2);

        double c = 2 * Math.Asin(Math.Sqrt(a));
```

```
// Radius of earth in
// kilometers. Use 3956
// for miles
double r = 6371;

// calculate the result
return (c * r);
}

// Driver code
static void Main()
{
    double lat1 = 53.3205555555556;
    double lat2 = 53.31861111111111;
    double lon1 = -1.729722222222221;
    double lon2 = -1.699722222222223;
    Console.WriteLine(distance(lat1, lat2,
                               lon1, lon2) + " K.M");
}
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

PHP

```
<?php

function twopoints_on_earth($latitudeFrom, $longitudeFrom,
                           $latitudeTo,   $longitudeTo)
{
    $long1 = deg2rad($longitudeFrom);
    $long2 = deg2rad($longitudeTo);
    $lat1 = deg2rad($latitudeFrom);
    $lat2 = deg2rad($latitudeTo);

    //Haversine Formula
    $dlong = $long2 - $long1;
    $dlati = $lat2 - $lat1;

    $val = pow(sin($dlati/2),2)+cos($lat1)*cos($lat2)*pow(sin($dlong/2),2);

    $res = 2 * asin(sqrt($val));

    $radius = 3958.756;

    return ($res*$radius);
```

```
}

// latitude and longitude of Two Points
$latitudeFrom = 19.017656 ;
$longitudeFrom = 72.856178;
$latitudeTo = 40.7127;
$longitudeTo = -74.0059;

// Distance between Mumbai and New York
print_r(twopoints_on_earth( $latitudeFrom, $longitudeFrom,
                           $latitudeTo,   $longitudeTo).' .'.'miles');

// This code is contributed by akash1295
// https://auth.geeksforgeeks.org/user/akash1295/articles
?>
```

Output:

2.0043678382716137 K.M

Reference: [Wikipedia](#)

Improved By : [Prasad_Kshirsagar](#), [AayushChaturvedi](#), [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/program-distance-two-points-earth/>

Chapter 169

Program for volume of Pyramid

Program for volume of Pyramid - GeeksforGeeks

A pyramid is a 3-dimensional geometric shape formed by connecting all the corners of a polygon to a central apex.

There are many types of pyramids. Most often, they are named after the type of base they have. Let's look at some common types of pyramids below.

Volume of a square pyramid [base of Pyramid is square] = $(1/3) * (b^2) * h$

Volume of a triangular pyramid [base of Pyramid is triangle] = $(1/6) * a * b * h$

Volume of a pentagonal pyramid [base of Pyramid is pentagonal] = $(5/6) * a * b * h$

Volume of a hexagonal pyramid [base of Pyramid is hexagonal] = $a * b * h$

Below is code for calculating volume of Pyramids :

C++

```
// CPP program to find the volume.
#include <bits/stdc++.h>
using namespace std;

// Function to find the volume
// of triangular pyramid
float volumeTriangular(int a,
                        int b,
                        int h)
{
    float vol = (0.1666) * a *
                b * h;
    return vol;
}
```

```
// Function to find the
// volume of square pyramid
float volumeSquare(int b, int h)
{
    float vol = (0.33) * b *
                b * h;
    return vol;
}

// Function to find the volume
// of pentagonal pyramid
float volumePentagonal(int a,
                      int b,
                      int h)
{
    float vol = (0.83) * a * b * h;
    return vol;
}

// Function to find the volume
// of hexagonal pyramid
float volumeHexagonal(int a,
                      int b,
                      int h)
{
    float vol = a * b * h;
    return vol;
}

// Driver Code
int main()
{
    int b = 4, h = 9, a = 4;
    cout << "Volume of triangular"
        << " base pyramid is "
        << volumeTriangular(a, b, h)
        << endl;

    cout << "Volume of square "
        << " base pyramid is "
        << volumeSquare(b, h)
        << endl;

    cout << "Volume of pentagonal"
        << " base pyramid is "
        << volumePentagonal(a, b, h)
        << endl;
```

```
    cout << "Volume of Hexagonal"
        << " base pyramid is "
        << volumeHexagonal(a, b, h);
    return 0;
}
```

Java

```
// Java Program for volume
// of Pyramid.
import java.util.*;
import java.lang.*;

class GfG
{

    // Function to find the volume of
    // triangular pyramid
    public static float volumeTriangular(int a,
                                         int b,
                                         int h)
    {
        float vol = (float)(0.1666) * a * b * h;
        return vol;
    }

    // Function to find the volume
    // of square pyramid
    public static float volumeSquare(int b,
                                     int h)
    {
        float vol = (float)(0.33) * b * b * h;
        return vol;
    }

    // Function to find the volume of
    // pentagonal pyramid
    public static float volumePentagonal(int a,
                                         int b,
                                         int h)
    {
        float vol = (float)(0.83) * a * b * h;
        return vol;
    }

    // Function to find the volume of hexagonal
    // pyramid
```

```
public static float volumeHexagonal(int a,
                                    int b,
                                    int h)
{
    float vol = (float)a * b * h;
    return vol;
}

// Driver Code
public static void main(String argc[])
{
    int b = 4, h = 9, a = 4;
    System.out.println("Volume of triangular"+
                       " base pyramid is " +
                       volumeTriangular(a, b, h));

    System.out.println("Volume of square base" +
                       " pyramid is " +
                       volumeSquare(b, h));

    System.out.println("Volume of pentagonal"+
                       " base pyramid is " +
                       volumePentagonal(a, b, h));

    System.out.println("Volume of Hexagonal"+
                       " base pyramid is " +
                       volumeHexagonal(a, b, h));
}

}

// This code is contributed by Sagar Shukla
```

Python3

```
# Python3 program to Volume of Pyramid

# Function to calculate
# Volume of Triangular Pyramid
def volumeTriangular(a, b, h):
    return (0.1666) * a * b * h

# Function To calculate
# Volume of Square Pyramid
def volumeSquare(b, h):
    return (0.33) * b * b * h

# Function To calculate Volume
# of Pentagonal Pyramid
```

```
def volumePentagonal(a, b, h):
    return (0.83) * a * b * h

# Function To calculate Volume
# of Hexagonal Pyramid
def volumeHexagonal(a, b, h):
    return a * b * h

# Driver Code
b = float(4)
h = float(9)
a = float(4)
print("Volume of triangular base pyramid is ",
      volumeTriangular(a, b, h))
print("Volume of square base pyramid is ",
      volumeSquare(b, h))
print("Volume of pentagonal base pyramid is ",
      volumePentagonal(a, b, h))
print("Volume of Hexagonal base pyramid is ",
      volumeHexagonal(a, b, h))

# This code is contributed by rishabh_jain
```

C#

```
// C# Program for volume of Pyramid.
using System;

class GFG
{

    // Function to find the volume of
    // triangular pyramid
    public static float volumeTriangular(int a,
                                          int b,
                                          int h)
    {
        float vol = (float)(0.1666) * a * b * h;
        return vol;
    }

    // Function to find the volume
    // of square pyramid
    public static float volumeSquare(int b,
                                      int h)
    {
        float vol = (float)(0.33) * b * b * h;
```

```
        return vol;
    }

    // Function to find the volume
    // of pentagonal pyramid
    public static float volumePentagonal(int a,
                                         int b,
                                         int h)
    {
        float vol = (float)(0.83) * a * b * h;
        return vol;
    }

    // Function to find the volume
    // of hexagonal pyramid
    public static float volumeHexagonal(int a,
                                         int b,
                                         int h)
    {
        float vol = (float)a * b * h;
        return vol;
    }

    // Driver Code
    public static void Main()
    {
        int b = 4, h = 9, a = 4;
        Console.WriteLine("Volume of triangular"+
                          " base pyramid is " +
                          volumeTriangular(a, b, h));

        Console.WriteLine("Volume of square "+
                          "base pyramid is " +
                          volumeSquare(b, h));

        Console.WriteLine("Volume of pentagonal"+
                          " base pyramid is " +
                          volumePentagonal(a, b, h));

        Console.WriteLine("Volume of Hexagonal"+
                          " base pyramid is " +
                          volumeHexagonal(a, b, h));
    }
}

// This code is contributed by vt_m
```

```
<?php
// PHP program to find the volume.

// Function to find the volume
// of triangular pyramid
function volumeTriangular($a, $b, $h)
{
    $vol = (0.1666) * $a * $b * $h;
    return $vol;
}

// Function to find the
// volume of square pyramid
function volumeSquare($b, $h)
{
    $vol = (0.33) * $b * $b * $h;
    return $vol;
}

// Function to find the volume
// of pentagonal pyramid
function volumePentagonal($a, $b, $h)
{
    $vol = (0.83) * $a * $b * $h;
    return $vol;
}

// Function to find the volume
// of hexagonal pyramid
function volumeHexagonal($a, $b, $h)
{
    $vol = $a * $b * $h;
    return $vol;
}

// Driver Code
$b = 4; $h = 9; $a = 4;
echo ("Volume of triangular base pyramid is ");
echo( volumeTriangular($a, $b, $h));
echo("\n");
echo ("Volume of square base pyramid is ");
echo( volumeSquare($b, $h));
echo("\n");
echo ("Volume of pentagonal base pyramid is ");
echo( volumePentagonal($a, $b, $h));
echo("\n");
echo("Volume of Hexagonal base pyramid is ");
echo( volumeHexagonal($a, $b, $h));
```

```
// This code is contributed by vt_m  
?>
```

Output :

```
Volume of triangular base pyramid is 23.9904  
Volume of square base pyramid is 47.52  
Volume of pentagonal base pyramid is 119.52  
Volume of Hexagonal base pyramid is 144
```

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/program-for-volume-of-pyramid/>

Chapter 170

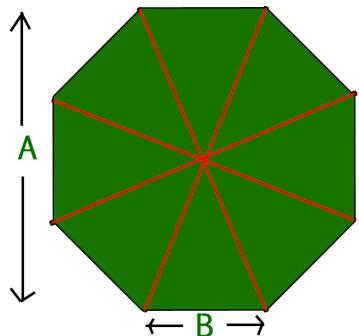
Program to calculate Area Of Octagon

Program to calculate Area Of Octagon - GeeksforGeeks

A regular octagon is a closed figure with sides of the same length and internal angles of the same size. It has eight lines of reflective symmetry and rotational symmetry of order 8. The internal angle at each vertex of a regular octagon is 135° . The central angle is 45° .

Properties :

Convex polygon, Equilateral polygon, Isogonal figure, Isotoxal figure, Cyclic.



Formula :

$$\text{Area} : 2 \times (\text{side length})^2 \times (1+\sqrt{2})$$

Examples :

```
Input : side = 3
Output : Area of Regular Octagon = 43.4558
```

```
Input : side = 4
Output : Area of Regular Octagon = 77.2548
```

C++

```
// CPP program to find area of octagon
#include <bits/stdc++.h>
using namespace std;

// Utility function
double areaOctagon(double side)
{
    return (float)(2 * (1 + sqrt(2)) *
                  side * side);
}

// Driver Code
int main()
{
    double side = 4;
    cout << "Area of Regular Octagon = "
        << areaOctagon(side) << endl;
    return 0;
}
```

Java

```
// Java Program to find
// area of Octagon.
import java.io.*;

class GFG
{
    // utility function
    static double areaOctagon(double side)
    {
        return (float)(2 * (1 + Math.sqrt(2))
                      * side * side);
    }

    // driver code
    public static void main(String arg[])
    {
```

```
        double side = 4;
        System.out.print("Area of Regular Octagon = "
                        + areaOctagon(side));
    }
}

// This code is contributed by Anant Agarwal.
```

Python3

```
# Python3 program to
# find area of octagon

import math

# Utility function
def areaOctagon(side):
    return (2 * (1 + (math.sqrt(2))) * side * side)

# Driver function
side = 4
print("Area of Regular Octagon =",
      round(areaOctagon(side), 4))

# This code is contributed
# by Azkia Anam.
```

C#

```
// C# Program to find
// area of Octagon.
using System;

class GFG
{
    // utility function
    static double areaOctagon(double side)
    {
        return (float)(2 * (1 + Math.Sqrt(2))
                      * side * side);
    }

    // Driver code
    public static void Main()
    {
        double side = 4;
        Console.WriteLine("Area of Regular Octagon = "
```

```
        + areaOctagon(side));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find area of octagon

// Utility function
function areaOctagon( $side)
{
    return (2 * (1 + sqrt(2)) *
            $side * $side);
}

// Driver Code

$side = 4;
echo("Area of Regular Octagon = ");
echo(areaOctagon($side));

// This code is contributed by vt_m.
?>
```

Output :

Area of Regular Octagon = 77.2548

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/program-calculate-area-octagon/>

Chapter 171

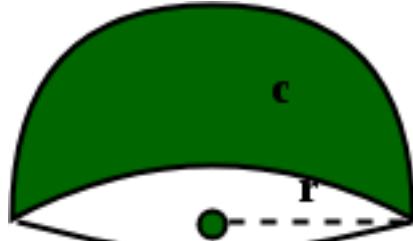
Program to calculate Volume and Surface area of Hemisphere

Program to calculate Volume and Surface area of Hemisphere - GeeksforGeeks

Calculate volume and surface area of a hemisphere.

Hemisphere :

In geometry, it is an exact half of a sphere. We can find many of the real life examples of the hemispheres such as our planet Earth can be divided into two hemisphere the southern & northern hemispheres.



Volume of Hemisphere : Volume of a Hemisphere is nothing but the amount of space occupied by the hemisphere. It is also defined as the quantity of three dimensional space enclosed by the boundary of the Hemisphere.

Surface area : The number of square units that will exactly cover the surface of a hemisphere.

$$\text{surface area} = 2\pi r^2$$
$$\text{volume} = \frac{2}{3}\pi r^3$$

Examples :

Input : Radius = 7
Output : Volume = 718.378

Surface Area = 307.876

Input : Radius = 11
Output : Volume = 2787.64
Surface Area = 760.265

C++

```
// CPP Program to calculate volume and
// and surface area of a Hemisphere.
#include <bits/stdc++.h>
using namespace std;
// Initializing value of pi
#define pi 3.141592653589793

// Function to calculate volume
void volume(float r)
{
    float volume = float(2 * pi * pow(r, 3)) / float(3);
    cout << "Volume = " << volume << endl;
}
// Function to calculate surface area
void surface_area(float r)
{
    float s_area = 2 * pi * pow(r, 2);
    cout << "Surface Area = " << s_area << endl;
}

// Driver program
int main()
{
    float r = 11;
    volume(r);
    surface_area(r);
    return 0;
}
```

Java

```
// Java Program to calculate volume and
// and surface area of a Hemisphere.
import java.util.*;
import java.lang.*;

public class GfG{

    // Initializing value of pi
```

```
private static final float pi = (float) 3.141592653589793;

// Function to calculate volume
public static void volume(float r)
{
    float volume = (float)(2 * pi * (float) Math.pow(r, 3))
                  / (float)(3);
    System.out.println("Volume = " + volume);
}
// Function to calculate surface area
public static void surface_area(float r)
{
    float s_area = (float)2 * pi * (float)Math.pow(r, 2);
    System.out.println("Surface Area = " + s_area);
}

// Driver function
public static void main(String argc[]){
    float r = 11;
    volume(r);
    surface_area(r);
}

/*
 * This code is contributed by Sagar Shukla */

```

Python

```
# Python code Program to calculate volume and
# and surface area of a Hemisphere.
import math

# Function to calculate volume
def volume(r):
    volume = 2 * math.pi * math.pow(r, 3) / 3
    print("Volume = ", '%.4f' %volume)

# Function to calculate surface area
def surface_area(r):
    s_area = 2 * math.pi * math.pow(r, 2)
    print("Surface Area = ", '%.4f' %s_area)

# Driver code
r = 11
volume(r)
surface_area(r)
```

C#

```
// C# Program to calculate volume and
// and surface area of a Hemisphere.
using System;

public class GfG{

    // Initializing value of pi
    private static float pi = (float) 3.141592653589793;

    // Function to calculate volume
    public static void volume(float r)
    {
        float volume = (float)(2 * pi * (float) Math.Pow(r, 3))
                      / (float)(3);
        Console.WriteLine("Volume = " + volume);
    }

    // Function to calculate surface area
    public static void surface_area(float r)
    {
        float s_area = (float)2 * pi * (float)Math.Pow(r, 2);
        Console.WriteLine("Surface Area = " + s_area);
    }

    // Driver function
    public static void Main()
    {
        float r = 11;
        volume(r);
        surface_area(r);
    }
}

/* This code is contributed by vt_m */
```

PHP

```
<?php
// PHP Program to calculate volume and
// and surface area of a Hemisphere.

// Initializing value of pi &
// Function to calculate volume
function volume($r)
{
    $pi = 3.141592653589793;
    $volume = (2 * $pi *
```

```
        pow($r, 3)) /(3);
echo("Volume = " );
echo($volume);
echo("\n");
}

// Function to calculate
// surface area
function surface_area($r)
{
    $pi = 3.141592653589793;
    $s_area = 2 * $pi * pow($r, 2);
    echo( "Surface Area = ");
    echo($s_area) ;
}

// Driver code
$r = 11;
volume($r);
surface_area($r);

// This code is contributed by vt_m
?>
```

Output :

```
Volume = 2787.64
Surface Area = 760.265
```

Improved By : [vt_m](#)

Source

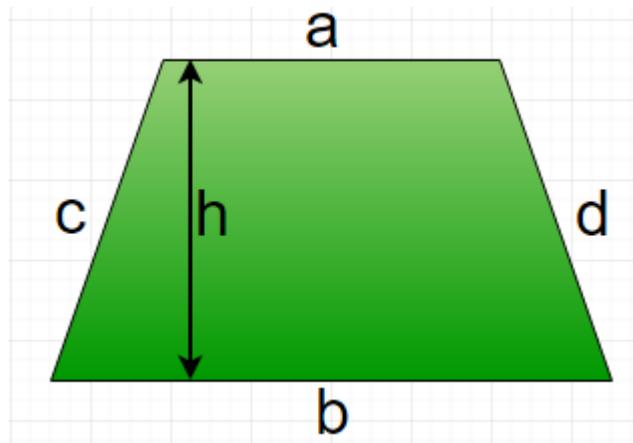
<https://www.geeksforgeeks.org/program-calculate-volume-surface-area-hemisphere/>

Chapter 172

Program to calculate area and perimeter of Trapezium

Program to calculate area and perimeter of Trapezium - GeeksforGeeks

A trapezium is a quadrilateral with at least one pair of parallel sides, other two sides may not be parallel. The parallel sides are called the bases of the trapezium and the other two sides are called it's legs. The perpendicular distance between parallel sides is called height of trapezium.



Formula :

Area of Trapezium : $0.5 * (a + b) * h$
Perimeter of Trapezium : $a + b + c + d$

Examples :

Input : a = 5, b = 6, c = 4, d = 3, h = 8

Output : Area of Trapezium : 44

Perimeter of Trapezium : 18

Input : a = 10, b = 15, c = 14, d = 11, h = 21

Output : Area of Trapezium: 262.5

Perimeter of Trapezium: 50

Below is the implementation of above formula :

C++

```
// CPP program to find area
// and perimeter of trapezium
#include <bits/stdc++.h>
using namespace std;

// Function to calculate Area of trapezium
float areaTrapezium(float a, float b, float h)
{
    return (1.0 / 2 * (a + b) * h);
}

// Function to calculate perimeter of trapezium
float perimeterTrapezium(float a, float b, float c,
                           float d)
{
    return (a + b + c + d);
}

// Driver function
int main()
{
    float a = 5, b = 15, c = 11, d = 4, h = 20;
    cout << "Area of Trapezium = " <<
           areaTrapezium(a, b, h) << endl;
    cout << "Perimeter of Trapezium = " <<
           perimeterTrapezium(a, b, c, d);
    return 0;
}
```

Java

```
// Java program to calculate area
// and perimeter of Trapezium
```

```
public class GFG {

    // Function to calculate area of Trapezium
    public static float areaTrapezium (float a,
                                      float b, float h)
    {
        return ((a + b) * h) / 2;
    }

    // Function to perimeter of Trapezium
    public static float perimeterTrapezium (float a,
                                           float b, float c, float d)
    {
        return (a + b + c + d);
    }

    // Driver function
    public static void main(String args[])
    {

        // a, b, c, d are four sides of Trapezium
        // and h is height between two parallel sides.
        float a = 5;
        float b = 15;
        float c = 11;
        float d = 4;
        float h = 20;

        // Printing value of area.
        System.out.print("Area Of Trapezium : ");
        System.out.println(areaTrapezium (a, b, h));

        // Printing value of Perimeter.
        System.out.print("Perimeter Of Trapezium : ");
        System.out.println(perimeterTrapezium (a, b, c, d));
    }
}

// This code is contributed by "akanshgupta"
```

Python3

```
# Python3 code to find area
# and perimeter of trapezium
```

```
# Function to calculate
# Area of trapezium
def areaTrapezium (a, b, h):
    return (1.0 / 2 * (a + b) * h)

# Function to calculate
# perimeter of trapezium
def perimeterTrapezium (a, b, c, d):
    return (a + b + c + d)

# Driver function
a = 5
b = 15
c = 11
d = 4
h = 20
print("Area of Trapezium =",
      areaTrapezium(a, b, h))

print("Perimeter of Trapezium =",
      perimeterTrapezium(a, b, c, d))

# This code is contributed by "Sharad_Bhardwaj"
```

C#

```
// C# program to calculate area
// and perimeter of Trapezium
using System;

class GFG {

    // Function to calculate area of Trapezium
    public static float areaTrapezium (float a,
                                      float b, float h)
    {
        return ((a + b) * h) / 2;
    }

    // Function to perimeter of Trapezium
    public static float perimeterTrapezium (float a,
                                            float b, float c, float d)
    {
        return (a + b + c + d);
    }
}
```

```
// Driver function
public static void Main()
{
    // a, b, c, d are four sides of Trapezium
    // and h is height between two parallel sides.
    float a = 5;
    float b = 15;
    float c = 11;
    float d = 4;
    float h = 20;

    // Printing value of area.
    Console.Write("Area Of Trapezium : ");
    Console.WriteLine(areaTrapezium (a, b, h));

    // Printing value of Perimeter.
    Console.Write("Perimeter Of Trapezium : ");
    Console.WriteLine(perimeterTrapezium (a, b, c, d));
}

// This code is contributed by "vt_m"
```

PHP

```
<?php
// PHP program to find area
// and perimeter of trapezium

// Function to calculate
// Area of trapezium
function areaTrapezium($a, $b, $h)
{
    return (1.0 / 2 * ($a + $b) * $h);
}

// Function to calculate
// perimeter of trapezium
function perimeterTrapezium($a, $b,
                            $c, $d)
{
    return ($a + $b + $c + $d);
}

// Driver Code
```

```
$a = 5; $b = 15;$c = 11;  
$d = 4; $h = 20;  
echo ("Area of Trapezium = ");  
echo(areaTrapezium($a, $b, $h));  
echo("\n");  
echo( "Perimeter of Trapezium = ");  
echo(perimeterTrapezium($a, $b, $c, $d));  
  
// This code is contributed by vt_m.  
?>
```

Output:

```
Area of Trapezium = 200  
Perimeter of Trapezium = 35
```

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/program-calculate-area-perimeter-trapezium/>

Chapter 173

Program to calculate area and perimeter of equilateral triangle

Program to calculate area and perimeter of equilateral triangle - GeeksforGeeks

An equilateral triangle is a triangle in which all three sides and angles are equal. All three internal angles of equilateral triangle measures 60 degree.

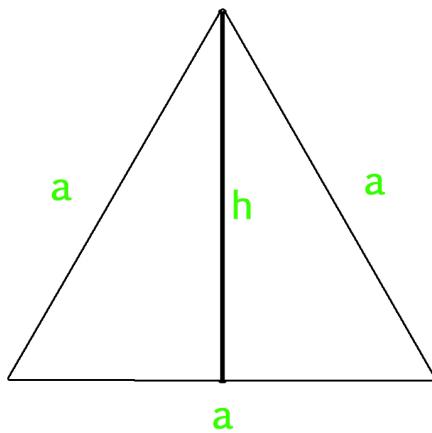
- If we know the length of each sides of equilateral triangle, then we can use below mentioned formula to calculate area of equilateral triangle.

$$\text{Area of Equilateral Triangle} = (\sqrt{3} / 4) \text{Side} * \text{Side}$$

- If we know the length of altitude of equilateral triangle along with the length of side, then we can use below mentioned formula to calculate it's area.

$$\text{Area of Equilateral Triangle} = (1/2) \times \text{Side} \times \text{Altitude}$$

Perimeter of Equilateral Triangle : $3 \times a$



Examples :

```
Input : side = 4
Output : Area of Equilateral Triangle: 6.9282
          Perimeter of Equilateral Triangle: 12
```

```
Input : side = 12
Output : Area of Equilateral Triangle: 62.3538
          Perimeter of Equilateral Triangle: 36
```

C++

```
// CPP program to find area
// and perimeter of equilateral triangle
#include <bits/stdc++.h>
using namespace std;

// Function to calculate Area
// of equilateral triangle
float area_equi_triangle(float side)
{
    return sqrt(3) / 4 * side * side;
}

// Function to calculate Perimeter
// of equilateral triangle
float peri_equi_triangle(float side)
{
    return 3 * side;
}

// Driver Code
```

```
int main()
{
    float side = 4;
    cout << "Area of Equilateral Triangle: "
        << area_equi_triangle(side) << endl;
    cout << "Perimeter of Equilateral Triangle: "
        << peri_equi_triangle(side);
    return 0;
}
```

Java

```
// Java Program to to find area and
// perimeter of equilateral triangle
import java.io.*;

class GFG
{
    // Function to calculate
    // Area of equilateral triangle
    static float area_equi_triangle(float side)
    {

        return (float)((Math.sqrt(3)) / 4) *
               side * side;
    }

    // Function to calculate
    // Perimeter of equilateral
    // triangle
    static float peri_equi_triangle(float side)
    {
        return 3 * side;
    }

    // Driver Code
    public static void main(String arg[])
    {
        float side = 4;
        System.out.print("Area of Equilateral Triangle:");
        System.out.println(area_equi_triangle(side));
        System.out.print("Perimeter of Equilateral Triangle:");
        System.out.println(peri_equi_triangle(side));
    }
}

// This code is contributed
// by Anant Agarwal.
```

Python

```
# Python3 program to calculate Area and
# Perimeter of equilateral Triangle

# Importing Math library for sqrt
from math import *

# Function to calculate Area
# of equilateral triangle
def area_equilateral( side ):
    area = (sqrt(3) / 4) * side * side
    print ("Area of Equilateral Triangle: % f"% area)

# Function to calculate Perimeter
# of equilateral triangle
def perimeter( side ):
    perimeter = 3 * side
    print ("Perimeter of Equilateral Triangle: % f"% perimeter)

# Driver code
side = 4
area_equilateral( side )
perimeter( side )
```

C#

```
// C# Program to to find area and
// perimeter of equilateral triangle
using System;

class GFG
{
    // Function to calculate
    // Area of equilateral triangle
    static float area_equi_triangle(float side)
    {

        return (float)((Math.Sqrt(3)) / 4) *
            side * side;
    }

    // Function to calculate
    // Perimeter of equilateral
    // triangle
    static float peri_equi_triangle(float side)
    {
```

```
        return 3 * side;
    }

    // Driver Code
    public static void Main()
    {
        float side = 4;
        Console.Write("Area of Equilateral Triangle:");
        Console.WriteLine(area_equi_triangle(side));
        Console.Write("Perimeter of Equilateral Triangle:");
        Console.WriteLine(peri_equi_triangle(side));
    }
}

// This code is contributed
// by vt_m.
```

PHP

```
<?php
// PHP program to find area
// and perimeter of equilateral triangle

// Function to calculate Area
// of equilateral triangle
function area_equi_triangle( $side)
{
    return sqrt(3) / 4 * $side * $side;
}

// Function to calculate Perimeter
// of equilateral triangle
function peri_equi_triangle( $side)
{
    return 3 * $side;
}

// Driver Code

$side = 4;
echo("Area of Equilateral Triangle: ");
echo(area_equi_triangle($side));
echo("\n");
echo("Perimeter of Equilateral Triangle: ");
echo( peri_equi_triangle($side));

// This code is contributed
```

```
// by vt_m.  
?>
```

Output :

```
Area of Equilateral Triangle: 6.9282  
Perimeter of Equilateral Triangle: 12
```

Improved By : [vt_m](#)

Source

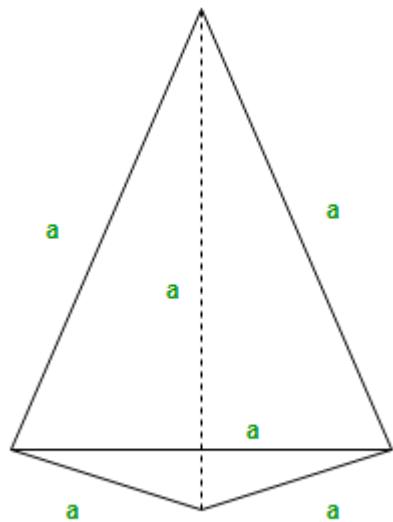
<https://www.geeksforgeeks.org/program-calculate-area-perimeter-equilateral-triangle/>

Chapter 174

Program to calculate area and volume of a Tetrahedron

Program to calculate area and volume of a Tetrahedron - GeeksforGeeks

A Tetrahedron is simply a pyramid with a triangular base. It is a solid object with four triangular faces, three on the sides or lateral faces, one on the bottom or the base and four vertices or corners. If the faces are all congruent equilateral triangles, then the tetrahedron is called regular.



The area of Tetrahedron can be found by using the formula :

```
Area = sqrt(3)*(side*side)
```

Examples :

Input : side = 3
Output : 15.5885

Input : side = 20
Output : 692.82

C

```
// C++ Program to Calculate
// area of tetrahedron
#include<iostream>
#include<math.h>
using namespace std;

//Utility Function
double area_of_tetrahedron(int side)
{
    return (sqrt(3)*(side*side));
}

//Driver Code
int main()
{
    int side=3;
    cout<< "Area of Tetrahedron ="
        << area_of_tetrahedron(side);
}

// This code is contributed by anant321.
```

Java

```
// Java Program to Calculate
// area of tetrahedron
import java.util.*;
import java.lang.*;

class GFG {

    // Utility Function
    public static double area_of_tetrahedron(int side)
```

```
{  
    return (Math.sqrt(3) * (side * side));  
  
}  
  
// Driver code  
public static void main(String[] args)  
{  
    int side = 3;  
    System.out.println("Area of Tetrahedron ="  
        + area_of_tetrahedron(side));  
}  
}  
// This code is contributed  
// by Prasad Kshirsagar
```

Python3

```
# Python3 Program to  
# Calculate area of  
# tetrahedron  
import math  
  
def area_of_tetrahedron(side):  
    return (math.sqrt(3) *  
        (side * side));  
  
# Driver Code  
side = 3;  
print("Area of Tetrahedron = ",  
    round(area_of_tetrahedron(side), 4));  
  
# This code is contributed by mits
```

C#

```
// C# Program to Calculate  
// area of tetrahedron  
using System;  
  
class GFG  
{  
// Utility Function  
public static double area_of_tetrahedron(int side)  
{  
    return (Math.Sqrt(3) *  
        (side * side));
```

```
}

// Driver code
static public void Main ()
{
    int side = 3;
    Console.WriteLine("Area of Tetrahedron = " +
                      area_of_tetrahedron(side));
}
}

// This code is contributed
// by akt_mit
```

PHP

```
<?php
// PHP Program to Calculate
// area of tetrahedron

function area_of_tetrahedron($side)
{
    return (sqrt(3) * ($side * $side));
}

// Driver Code
$side = 3;
echo "Area of Tetrahedron = ",
     area_of_tetrahedron($side);

// This code is contributed by aj_36.
?>
```

Output :

Area of Tetrahedron =15.5885

The volume of the tetrahedron can be found by using the following formula :

$$\text{Volume} = a^3 / (6\sqrt{2})$$

Examples :

Input : side = 3
Output : 3.18

Input : side = 20
Output : 942.81

C/C++

```
// CPP program to find the volume of a tetrahedron
#include <math.h>
#include <stdio.h>

// Function to calculate volume
double vol_tetra(int side)
{
    double volume = (pow(side, 3) / (6 * sqrt(2)));
    return volume;
}

// Driver Code
int main()
{
    int side = 3;
    double vol = vol_tetra(side);
    printf("%.2f", vol);
}
```

Java

```
// Java code to find the volume of a tetrahedron
import java.io.*;

class Tetrahedron {
    // Function to calculate volume
    static double vol_tetra(int side)
    {
        double volume = (Math.pow(side, 3) / (6 * Math.sqrt(2)));
        return volume;
    }

    // Driver Code
    public static void main(String[] args)
    {
        int side = 3;
        double vol = vol_tetra(side);
```

```
    vol = (double)Math.round(vol * 100) / 100;
    System.out.println(vol);
}
}
```

Python

```
# Python code to find the volume of a tetrahedron
import math
def vol_tetra(side):
    volume = (side ** 3 / (6 * math.sqrt(2)))
    return round(volume, 2)

# Driver Code
side = 3
vol = vol_tetra(side)
print(vol)
```

C#

```
// C# code to find the volume of a tetrahedron
using System;

class Tetrahedron {
    // Function to calculate volume
    static double vol_tetra(int side)
    {
        double volume = (Math.Pow(side, 3) / (6 * Math.Sqrt(2)));
        return volume;
    }

    // Driver Code
    public static void Main()
    {
        int side = 3;
        double vol = vol_tetra(side);
        vol = (double)Math.Round(vol * 100) / 100;
        Console.WriteLine(vol);
    }
}

// This code is contributed
// by vt_m.
```

PHP

```
<?php
```

```
// PHP program to find the
// volume of a tetrahedron

// Function to calculate volume
function vol_tetra($side)
{
    $volume = (pow($side, 3) /
               (6 * sqrt(2)));
    return $volume;
}

// Driver Code
$side = 3;
$vol = vol_tetra($side);
echo $vol;

// This code is contributed by ajit
?>
```

Output :

3.18

Improved By : [jit_t](#), [Prasad_Kshirsagar](#), [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/calculate-area-tetrahedron/>

Chapter 175

Program to calculate area of Circumcircle of an Equilateral Triangle

Program to calculate area of Circumcircle of an Equilateral Triangle - GeeksforGeeks

Given the length of sides of an equilateral triangle. We need to write a program to find the area of Circumcircle of the given equilateral triangle.

Examples:

```
Input : side = 6
Output : Area of circumscribed circle is: 37.69
```

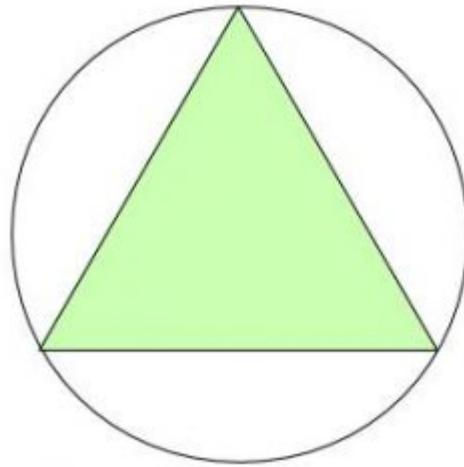
```
Input : side = 9
Output : Area of circumscribed circle is: 84.82
```

All three sides of equilateral triangle are of equal length and all three interior angles are 60 degrees.

Properties of a Circumcircle are as follows:

- The center of the circumcircle is the point where the medians of the equilateral triangle intersect.
- Circumscribed circle of an equilateral triangle is made through the three vertices of an equilateral triangle.
- The radius of a circumcircle of an equilateral triangle is equal to $(a / \sqrt{3})$, where 'a' is the length of the side of equilateral triangle.

Below image shows an equilateral triangle with circumcircle:



The formula used to calculate the area of circumscribed circle is:

$$(\ast a^2)/3$$

where a is the length of the side of the given equilateral triangle.

How this formulae works?

We know that [area of circle](#) = $\ast r^2$, where r is the radius of given circle.

We also know that radius of Circumcircle of an equilateral triangle = (side of the equilateral triangle) / $\sqrt{3}$.

Therefore, area = $\ast r^2 = \ast a^2/3$.

C

```
// C program to find the area of Cicumscribed
// circle of equilateral triangle
#include <stdio.h>
#define PI 3.14159265

// function to find area of
// circumscribed circle
float area_circumscribed(float a)
{
    return (a * a * (PI / 3));
}

// Driver code
int main()
```

```
{  
    float a = 6;  
    printf("Area of circumscribed circle is :%f",  
          area_circumscribed(a));  
    return 0;  
}
```

Java

```
// Java code to find the area of circumscribed  
// circle of equilateral triangle  
import java.lang.*;  
  
class GFG {  
  
    static double PI = 3.14159265;  
  
    // function to find the area of  
    // circumscribed circle  
    public static double area_circumscribed(double a)  
    {  
        return (a * a * (PI / 3));  
    }  
  
    // Driver code  
    public static void main(String[] args)  
    {  
        double a = 6.0;  
        System.out.println("Area of circumscribed circle is :"  
                           + area_circumscribed(a));  
    }  
}
```

Python3

```
# Python3 code to find the area of circumscribed  
# circle of equilateral triangle  
PI = 3.14159265  
  
# Function to find the area of  
# circumscribed circle  
def area_circumscribed(a):  
    return (a * a * (PI / 3))  
  
# Driver code  
a = 6.0  
print("Area of circumscribed circle is :%f"
```

```
%area_circumscribed(a))

# This code is contributed by Anant Agarwal.

C#

// C# code to find the area of
// circumscribed circle
// of equilateral triangle
using System;

class GFG {
    static double PI = 3.14159265;

    // function to find the area of
    // circumscribed circle
    public static double area_circumscribed(double a)
    {
        return (a * a * (PI / 3));
    }

    // Driver code
    public static void Main()
    {
        double a = 6.0;
        Console.WriteLine("Area of circumscribed circle is :" +
            area_circumscribed(a));
    }
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// PHP program to find the
// area of Circumscribed
// circle of equilateral triangle
$PI = 3.14159265;

// function to find area of
// circumscribed circle
function area_circumscribed($a)
{
    global $PI;
    return ($a * $a * ($PI / 3));
}
```

```
// Driver code
$a = 6;
echo("Area of circumscribed circle is :");
echo(area_circumscribed($a));

// This code is contributed by Ajit.
?>
```

Outuput: Area of circumscribed circle is :37.6991118

Improved By : [nitin mittal, jit_t](#)

Source

<https://www.geeksforgeeks.org/program-calculate-area-circumcircle-equilateral-triangle/>

Chapter 176

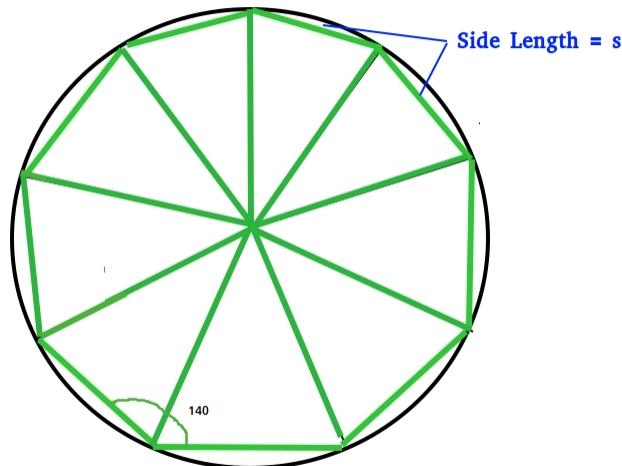
Program to calculate area of Enneagon

Program to calculate area of Enneagon - GeeksforGeeks

Enneagon is a polygon with 9 sides and 9 internal angles. Enneagon is also known as Nonagon. A regular nonagon has an internal angle of 140 degrees each. Sum of interior angles of the nonagon is 1260 degree.

The center of the circumcircle is also taken as the center of the regular Nonagon.

The line segment drawn perpendicular to a side of the Nonagon is called the Apothem and is represented by 'a'.



Area $\approx 6.1818 * s * s$
where s is side length.

Examples:

Input : 6
Output : Area of Regular Nonagon = 222.5448

Input : 8
Output : Area of Regular Nonagon = 395.6352

C++

```
// CPP program to find area of a Enneagon
#include <iomanip>
#include <iostream>
#include <math.h>
using namespace std;

// Function to calculate area of nonagon
double Nonagon_Area(double s) {
    return (6.1818 * s * s);
}

// driver function
int main() {
    double s = 6; // Length of a side
    cout << "Area of Regular Nonagon = " << std::setprecision(7)
        << Nonagon_Area(s);
    return 0;
}
```

Java

```
// Java program to find area of a Enneagon
class Nonagon {

    // Function for calculating the area of the nonagon
    public static double Nonagon_Area(double s) {
        return ((6.1818 * (s * s)));
    }

    // driver code
    public static void main(String[] args) {
        double s = 6; // Length of a side
    }
}
```

```
        System.out.print("Area of Regular Nonagon = " + Nonagon_Area(s));
    }
}
```

Python

```
# python program to find area of a Enneagon
length = 6
Nonagon_area = 6.1818 * (length ** 2)
print("Area of regular Nonagon is =", Nonagon_area)
```

C#

```
// C# program to find area of a Hexagon
using System;

class Nonagon {

    // Function for calculating
    // the area of the nonagon
    public static double Nonagon_Area(double s)
    {
        return ((6.1818 * (s * s)));
    }

    // driver code
    public static void Main()
    {
        // Length of a side
        double s = 6;
        Console.WriteLine("Area of Regular Nonagon = " +
                          Nonagon_Area(s));
    }
}

// This article is contributed by vt_m
```

PHP

```
<?php
// PHP program to find area of a Hexagon

// Function to calculate
// area of nonagon
function Nonagon_Area($s)
{
```

```
        return (6.1818 * $s * $s);  
    }  
  
// Driver Code  
  
// Length of a side  
$s = 6;  
  
echo "Area of Regular Nonagon = "  
     , Nonagon_Area($s);  
  
// This code is contributed by anuj_67.  
?>
```

Output:

Area of Regular Nonagon = 222.5448

Improved By : [vt_m](#)

Source

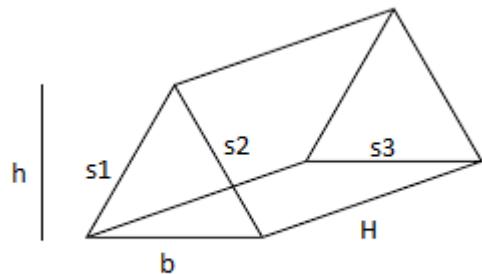
<https://www.geeksforgeeks.org/program-to-calculate-area-of-enneagon/>

Chapter 177

Program to calculate the Surface Area of a Triangular Prism

Program to calculate the Surface Area of a Triangular Prism - GeeksforGeeks

In mathematics, a triangular prism is a three-dimensional solid shape with two identical ends connected by equal parallel lines, and have 5 faces, 9 edges, and 6 vertices.



where “b” is the length of the base, “h” is the height of the triangle, “s₁, s₂, s₃” are the respective length of each side of the triangle, and H is the height of the prism (which is also the length of the rectangle).

Given the base, the height of the triangle, height of prism and the length of each side of triangle base and the task is to calculate the surface area of the triangular prism.

Examples:

Input: b = 3, h = 4, s₁ = 3, s₂ = 6, s₃ = 6, Ht = 8

Output: The area of triangular prism is 132.000000

Input: b = 2, h = 3, s1 = 4, s2 = 5, s3 = 6, Ht = 8

Output: The area of triangular prism is 126.000000

Formula for calculating the surface area:

As stated above, the prism contains two triangles of the area $(1/2)*(b)*(h)$ and three rectangles of the area **H*s1, H*s2 and H*s3**.

Now after adding all the terms we get the total surface area:

SA = b * h + (s1 + s2 + s3) * H

C

```
// C Program to calculate the
// Surface area of a triangular prism
#include <stdio.h>

// Function for calculating the area
void Calculate_area()
{
    // Initialization
    float b = 3, h = 4, s1 = 3, s2 = 6;
    float s3 = 6, Ht = 8, SA;

    // Formula for calculating the area
    SA = b * h + (s1 + s2 + s3) * Ht;

    // Displaying the output
    printf("The area of triangular prism is : %f", SA);
}
```

```
// Driver code
int main()
{
    // Function calling
    Calculate_area();

    return 0;
}
```

C++

```
// C++ Program to calculate the
// Surface area of a triangular prism
#include <bits/stdc++.h>
using namespace std;
```

```
// Function for calculating the area
void Calculate_area()
{
    // Initialization
    float b = 3, h = 4, s1 = 3, s2 = 6;
    float s3 = 6, Ht = 8, SA;

    // Formula for calculating the area
    SA = b * h + (s1 + s2 + s3) * Ht;

    // Displaying the area
    cout << "The area of triangular prism is : " << SA;
}

// Driver code
int main()
{
    // Function calling
    Calculate_area();

    return 0;
}
```

C#

```
// C# Program to calculate the
// Surface area of a triangular prism
using System;
public class Prism {

    static void Calculate_area()
    {
        // Initialization
        double b = 3, h = 4, s1 = 3, s2 = 6;
        double s3 = 6, Ht = 8, SA;

        // Formula for calculating the area
        SA = b * h + (s1 + s2 + s3) * Ht;

        // Displaying the area
        Console.WriteLine("The area of triangular prism is : " + SA);
    }
    static public void Main(String[] args)
    {
        Calculate_area();
    }
}
```

Java

```
// Java Program to calculate the
// Surface area of a triangular prism

import java.util.Scanner;
public class Prism {

    public static void Calculate_area()
    {
        // Initialization
        double b = 3, h = 4, s1 = 3, s2 = 6;
        double s3 = 6, Ht = 8, SA;

        // Formula for calculating the area
        SA = b * h + (s1 + s2 + s3) * Ht;

        // Displaying the area
        System.out.printf("The area of triangular prism is : %f", SA);
    }
    public static void main(String[] args)
    {
        Calculate_area();
    }
}
// This code is contributed by Nishant Tanwar
```

PHP

```
<?php
// PHP Program to calculate
// the Surface area of a
// triangular prism

// Function for calculating
// the area
function Calculate_area()
{
    // Initialization
    $b = 3; $h = 4;
    $s1 = 3; $s2 = 6;
    $s3 = 6; $Ht = 8; $SA;

    // Formula for calculating
    // the area
    $SA = $b * $h + ($s1 +
                      $s2 + $s3) * $Ht;
```

```
// Displaying the area
echo "The area of triangular".
      " prism is : " , $SA;
}

// Driver code

// Function calling
Calculate_area();

// This code is contributed by m_kit
?>
```

Output:

The area of triangular prism is : 132.00000

Improved By : [jit_t](#), Nishant Tanwar

Source

<https://www.geeksforgeeks.org/program-to-calculate-the-surface-area-of-a-triangular-prism/>

Chapter 178

Program to calculate volume of Ellipsoid

Program to calculate volume of Ellipsoid - GeeksforGeeks

Ellipsoid, closed surface of which all plane cross sections are either ellipses or circles. An ellipsoid is symmetrical about three mutually perpendicular axes that intersect at the center. It is a three-dimensional, closed geometric shape, all planar sections of which are ellipses or circles.

An ellipsoid has three independent axes, and is usually specified by the lengths a , b , c of the three semi-axes. If an ellipsoid is made by rotating an ellipse about one of its axes, then two axes of the ellipsoid are the same, and it is called an ellipsoid of revolution, or spheroid. If the lengths of all three of its axes are the same, it is a sphere.

Standard equation of Ellipsoid : $x^2 / a^2 + y^2 / b^2 + z^2 / c^2 = 1$ where a , b , c are positive real numbers.

Volume of Ellipsoid : $(4/3) * \pi * r1 * r2 * r3$

Below is code for calculating volume of ellipsoid :

C++

```
// CPP program to find the
// volume of Ellipsoid.
#include <bits/stdc++.h>
using namespace std;

// Function to find the volume
float volumeOfEllipsoid(float r1,
                        float r2,
```

```
        float r3)
{
    float pi = 3.14;
    return 1.33 * pi * r1 *
           r2 * r3;
}

// Driver Code
int main()
{
    float r1 = 2.3, r2 = 3.4, r3 = 5.7;
    cout << "volume of ellipsoid is : "
         << volumeOfEllipsoid(r1, r2, r3);
    return 0;
}
```

Java

```
// Java program to find the
// volume of Ellipsoid.
import java.util.*;
import java.lang.*;

class GfG
{

    // Function to find the volume
    public static float volumeOfEllipsoid(float r1,
                                           float r2,
                                           float r3)
    {
        float pi = (float)3.14;
        return (float) 1.33 * pi * r1 * r2 * r3;
    }

    // Driver Code
    public static void main(String args[])
    {
        float r1 = (float) 2.3,
              r2 = (float) 3.4,
              r3 = (float) 5.7;
        System.out.println("volume of ellipsoid is : "
                           + volumeOfEllipsoid(r1, r2, r3));
    }
}

// This code is contributed by Sagar Shukla
```

Python

```
''' Python3 program to Volume of ellipsoid'''
import math

# Function To calculate Volume
def volumeOfEllipsoid(r1, r2, r3):
    return 1.33 * math.pi * r1 * r2 * r3

# Driver Code
r1 = float(2.3)
r2 = float(3.4)
r3 = float(5.7)
print("Volume of ellipsoid is : ",
      volumeOfEllipsoid(r1, r2, r3))
```

C#

```
// C# program to find the
// volume of Ellipsoid.
using System;

class GfG
{

    // Function to find the volume
    public static float volumeOfEllipsoid(float r1,
                                           float r2,
                                           float r3)
    {
        float pi = (float)3.14;
        return (float) 1.33 * pi * r1 * r2 * r3;
    }

    // Driver Code
    public static void Main()
    {
        float r1 = (float)2.3,
              r2 =(float) 3.4,
              r3 = (float)5.7;
        Console.WriteLine("volume of ellipsoid is : " +
                          volumeOfEllipsoid(r1, r2, r3));
    }
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP program to find the
// volume of Ellipsoid.

// Function to find the volume
function volumeOfEllipsoid( $r1,
                            $r2,
                            $r3)
{
    $pi = 3.14;
    return 1.33 * $pi * $r1 *
           $r2 * $r3;
}

// Driver Code

$r1 = 2.3; $r2 = 3.4;
$r3 = 5.7;
echo ( "volume of ellipsoid is : ");
echo( volumeOfEllipsoid($r1, $r2, $r3));

// This code is contributed by vt_m .
?>
```

Output :

Volume of ellipsoid is : 186.15

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/program-calculate-volume-ellipsoid/>

Chapter 179

Program to calculate volume of Octahedron

Program to calculate volume of Octahedron - GeeksforGeeks

Given the side of the Octahedron then calculate the volume of Octahedron.

Examples:

Input : 3
Output : 12.7279

Input : 7
Output : 161.692

A regular Octahedron has eight faces,twelve edges and six vertices. It has eight triangles with edges of equal length and effectively two square pyramids meeting at their bases.

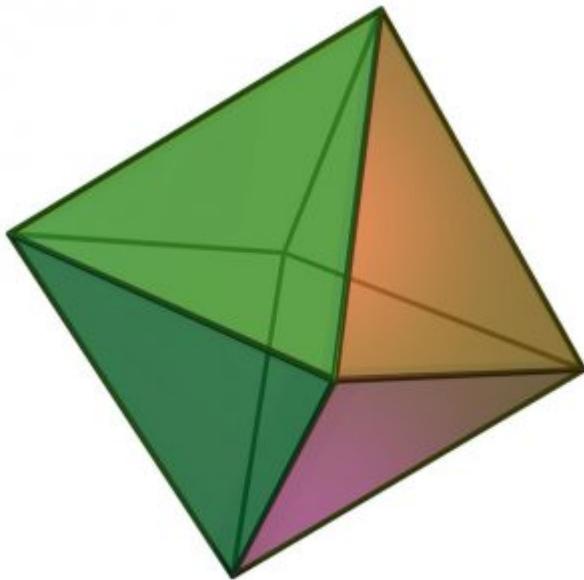


Image Source : [Wikimedia](#)

Properties of Octahedron:

Number of faces: 8

Number of edges: 12

Number of vertices: 6

Volume = $\sqrt{2}/3 \times a^3$ where a is the side of Octahedron

CPP

```
// CPP Program to calculate
// volume of Octahedron
#include <bits/stdc++.h>
using namespace std;

// utility Function
double vol_of_octahedron(double side)
{
    return ((side*side*side)*(sqrt(2)/3));
}

// Driver Function
int main()
{
    double side = 3;
    cout << "Volume of octahedron ="
        << vol_of_octahedron(side)
        << endl;
}
```

Java

```
// Java Program to calculate
// volume of Octahedron

import java.io.*;
class GFG
{
    public static void main (String[] args)
    {
        // Driver Function
        double side = 3;
        System.out.print("Volume of octahedron = ");
        System.out.println(vol_of_octahedron(side));

    }

    // utility Function
    static double vol_of_octahedron(double side)
    {
        return ((side*side*side)*(Math.sqrt(2)/3));
    }
}

// This code is contributed
// by Azkia Anam.
```

Python3

```
# Python3 Program to calculate
# volume of Octahedron

import math

# utility Function
def vol_of_octahedron(side):
    return ((side*side*side)*(math.sqrt(2)/3))

# Driver Function
side = 3
print("Volume of octahedron =",
      round(vol_of_octahedron(side),4))

# This code is contributed
# by Azkia Anam.
```

C#

```
// C# Program to calculate
// volume of Octahedron
using System;

class GFG
{
    public static void Main ()
    {
        // Driver Function
        double side = 3;
        Console.Write("Volume of octahedron = ");
        Console.WriteLine(vol_of_octahedron(side));

    }

    // utility Function
    static double vol_of_octahedron(double side)
    {
        return ((side*side*side)*(Math.Sqrt(2)/3));
    }
}

// This code is contributed
// by vt_m.
```

PHP

```
<?php
// PHP Program to calculate
// volume of Octahedron

// utility Function
function vol_of_octahedron( $side )
{
    return (($side * $side *
             $side) * (sqrt(2) / 3));
}

// Driver Function
$side = 3;
echo ("Volume of octahedron =");
echo(vol_of_octahedron($side));

// This code is contributed
// by vt_m.
?>
```

Output:

Volume of octahedron = 12.7279

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/program-calculate-volume-octahedron/>

Chapter 180

Program to check congruency of two triangles

Program to check congruency of two triangles - GeeksforGeeks

Given four arrays of 3 numbers each which represents sides and angles of two triangles. The task is to check if the two triangles are [Congruent](#) or not. Also print the theorem by which they are congruent.

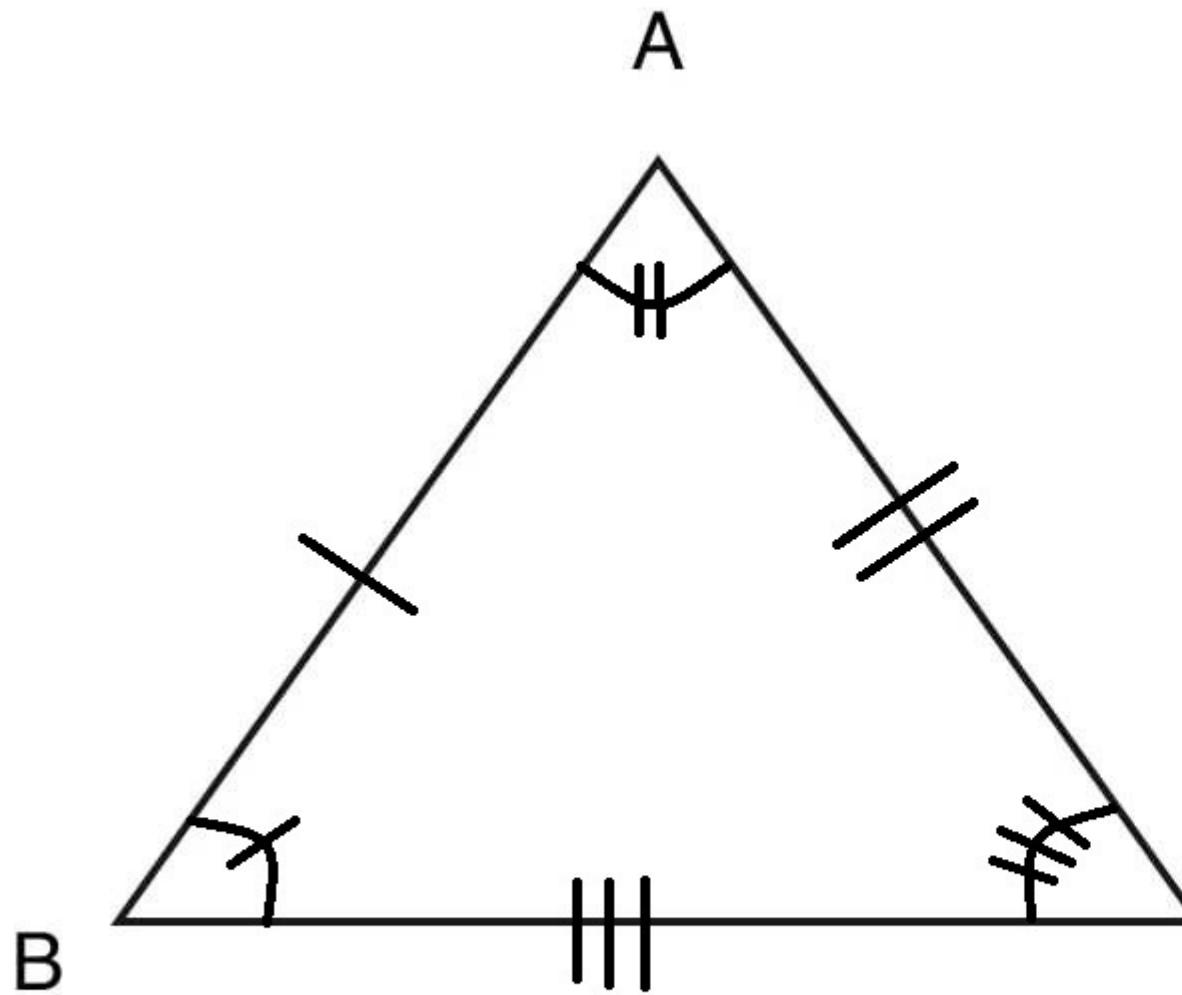
Note: All sides and angles given as input are for valid triangles.

Examples:

```
Input : side1 = [3, 4, 5]    angle1 = [90, 60, 30]
        side2 = [4, 3, 5]    angle2 = [60, 30, 90]
Output: Triangles are congruent by SSS SAS ASA AAS HL.
```

```
Input : side1 = [3, 5, 6]    angle1 = [80, 50, 50]
        side2 = [1, 1, 1]    angle2 = [60, 60, 60]
Output: Triangles are not congruent
```

Congruent triangles are two or more triangles that have all corresponding sides that are equal or a pair of sides and between angle are equal or a pair of angle and side between are equal or a pair of angle and other side are equal or hypotenuse and one side are equal.



The congruency of triangles can be proved by the following theorems:

1. **Side-Side-Side (SSS) Congruency criteria:** If all the sides of a triangle are equal

to the sides of another triangle then the triangles are said to be congruent by the property of *Side-Side-Side* (SSS).

In above triangle ABC and A'B'C' if, AB=A'B' and BC=B'C' and CA=C'A' then, triangles are congruent.

2. **Side-Angle-Side (SAS) Congruent criteria:** If two sides of the two triangles are equal and the angle between them is same in both triangle then the triangles are said to be congruent by the property of *Side-Angle-Side* (SAS). In above triangle ABC and A'B'C' if, AB=A'B' and BC=B'C' and $\angle A = \angle A'$ triangles are congruent.

3. **Angle-Side-Angle (ASA) Congruent criteria :**If two angles of the two triangles are equal and the length of side between them is same in both triangle then the triangles are said to be congruent by the property of *Angle-Side-Angle* (ASA).In above triangle ABC and A'B'C' if, $\angle A = \angle A'$ and $\angle B = \angle B'$ and $\angle C = \angle C'$ then, triangles are congruent.

4. **Angle-Angle-Side (AAS) Congruent criteria :**If two angles of the two triangles are equal and the length of other side is same in both triangle then the triangles are said to be congruent by the property of *Angle-Angle-Side* (AAS). In above triangle ABC and A'B'C' if, $\angle A = \angle A'$ and $\angle B = \angle B'$ and $\angle C = \angle C'$ and CA=C'A' then, triangles are congruent.

5. **Hypotenuse-Leg (HL) Congruent criteria :**

If the hypotenuse of the two triangles are equal and the length of any other one side is same in both triangle then the triangles are said to be congruent by the property of *Hypotenuse-Leg* (HL).

Below is the implementation of the above theorems.

```
# Python program to check
# similarity between two triangles.

# Function for SAS congruency
def cong_sas(s1, s2, a1, a2):

    s1 = [float(i) for i in s1]
    s2 = [float(i) for i in s2]
    a1 = [float(i) for i in a1]
    a2 = [float(i) for i in a2]

    s1.sort()
    s2.sort()
    a1.sort()
    a2.sort()

    # Check for SAS
```

```
# angle b / w two smallest sides is largest.
if s1[0] == s2[0] and s1[1] == s2[1]:
    # since we take angle b / w the sides.
    if a1[2] == a2[2]:
        return 1

if s1[1] == s2[1] and s1[2] == s2[2]:
    if a1[0] == a2[0]:
        return 1

if s1[2] == s2[2] and s1[0] == s2[0]:
    if a1[1] == a2[1]:
        return 1

return 0

# Function for ASA congruency
def cong_asa(s1, s2, a1, a2):

    s1 = [float(i) for i in s1]
    s2 = [float(i) for i in s2]
    a1 = [float(i) for i in a1]
    a2 = [float(i) for i in a2]

    s1.sort()
    s2.sort()
    a1.sort()
    a2.sort()

    # Check for ASA

    # side b / w two smallest angle is largest.
    if a1[0] == a2[0] and a1[1] == a2[1]:
        # since we take side b / w the angle.
        if s1[2] == s2[2]:
            return 1

    if a1[1] == a2[1] and a1[2] == a2[2]:
        if s1[0] == s2[0]:
            return 1

    if a1[2] == a2[2] and a1[0] == a2[0]:
        if s1[1] == s2[1]:
            return 1

return 0
```

```
# Function for AAS congruency
def cong_aas(s1, s2, a1, a2):

    s1 = [float(i) for i in s1]
    s2 = [float(i) for i in s2]
    a1 = [float(i) for i in a1]
    a2 = [float(i) for i in a2]

    s1.sort()
    s2.sort()
    a1.sort()
    a2.sort()

    # Check for AAS

    # side other two smallest angle is smallest or 2nd smallest.
    if a1[0] == a2[0] and a1[1] == a2[1]:

        # since we take side other than angles.
        if s1[0] == s2[0] or s1[1] == s2[1]:
            return 1

    if a1[1] == a2[1] and a1[2] == a2[2]:
        if s1[1] == s2[1] or s1[2] == s2[2]:
            return 1

    if a1[2] == a2[2] and a1[0] == a2[0]:
        if s1[0] == s2[0] or s1[2] == s2[2]:
            return 1

    return 0

# Function for HL congruency
def cong_hl(s1, s2):

    s1 = [float(i) for i in s1]
    s2 = [float(i) for i in s2]
    s1.sort()
    s2.sort()

    # Check for HL
    if s1[2] == s2[2]:
        if s1[1] == s2[1] or s1[0] == s2[0]:
            return 1

    return 0
```

```
# Function for SSS congruency
def cong_sss(s1, s2):

    s1 = [float(i) for i in s1]
    s2 = [float(i) for i in s2]
    s1.sort()
    s2.sort()

    # Check for SSS
    if(s1[0] == s2[0] and s1[1] == s2[1] and s1[2] == s2[2]):
        return 1

    return 0

# Driver Code
s1 = [3, 4, 5]
s2 = [4, 3, 5]

a1 = [90, 60, 30]
a2 = [60, 30, 90]

# function call for SSS congruency
sss = cong_sss(s1, s2)

# function call for SAS congruency
sas = cong_sas(s1, s2, a1, a2)

# function call for ASA congruency
asa = cong_asa(s1, s2, a1, a2)

# function call for AAS congruency
aas = cong_aas(s1, s2, a1, a2)

# function call for HL congruency
hl = cong_hl(s1, s2, )

# Check if triangles are congruent or not
if sss or sas or asa or aas or hl :
    print "Triangles are congruent by",
    if sss: print "SSS",
    if sas: print "SAS",
    if asa: print "ASA",
    if aas: print "AAS",
    if hl: print "HL",
else: print "Triangles are not congruent"
```

Output:

Triangles are congruent by SSS SAS ASA AAS HL

Source

<https://www.geeksforgeeks.org/program-to-check-congruency-of-two-triangles/>

Chapter 181

Program to check if tank will overflow, underflow or filled in given time

Program to check if tank will overflow, underflow or filled in given time - GeeksforGeeks

Given a tank with definite height and radius and the flow of water available to fill the tank. Determine whether the tank will overflow or not in a given amount of time.

Note: The flow of water will be available in volume per minute.

Examples:

```
--r=5--  
----- ^  
| | |  
| | |  
| | h = 10  
| | |  
----- ^  
  
rate_of_flow = 10  
  
Input : given_time = 10.0  
Output : Underflow  
  
Input : given_time = 100.0  
Output : Overflow
```

Approach:

Volume of a cylindrical tank is $(22/7) * \text{radius} * \text{radius} * \text{height}$. First, find out the amount

of time required to completely fill the tank then compare it with the given time. If the given time is greater than required time, it will result in an overflow condition. If the given time is less than the required time then it will result in an underflow condition otherwise the tank is filled.

Below is the implementation of above approach:

C++

```
// C++ program to check if Tank will
// overflow or not in given time
#include <bits/stdc++.h>
using namespace std;

// function to calculate the volume of tank
float volume(int radius, int height)
{
    return ((22 / 7) * radius * radius * height);
}

// function to print overflow / filled /
// underflow accordingly
void check_and_print(float required_time,
                      float given_time)
{
    if (required_time < given_time)
        cout << "Overflow";
    else if (required_time > given_time)
        cout << "Underflow";
    else
        cout << "Filled";
}

// driver function
int main()
{
    int radius = 5, // radius of the tank
        height = 10, // height of the tank
        rate_of_flow = 10; // rate of flow of water

    float given_time = 70.0; // time given

    // calculate the required time
    float required_time = volume(radius, height) /
                           rate_of_flow;

    // printing the result
    check_and_print(required_time, given_time);
```

```
    return 0;
}
```

Java

```
// Java program to check if Tank will
// overflow or not in given time

class Number
{
    // function to calculate the volume of tank
    public static float volume(int radius, int height)
    {
        return ((22 / 7) * radius * radius * height);
    }

    // function to print overflow / filled /
    // underflow accordingly
    public static void check_and_print(double required_time,
                                       double given_time)
    {
        if (required_time < given_time)
            System.out.print( "Overflow" );
        else if (required_time > given_time)
            System.out.print( "Underflow" );
        else
            System.out.print( "Filled" );
    }

    // driver code
    public static void main(String[] args)
    {
        int radius = 5, // radius of the tank
        height = 10, // height of the tank
        rate_of_flow = 10; // rate of flow of water

        double given_time = 70.0; // time given

        // calculate the required time
        double required_time = volume(radius, height) /
                               rate_of_flow;

        // printing the result
        check_and_print(required_time, given_time);
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
# Python3 code to check if Tank will
# overflow or not in given time

# function to calculate the volume of tank
def volume(radius, height):
    return ((22 / 7) * radius * radius * height)

# function to print overflow / filled /
# underflow accordingly
def check_and_print(required_time, given_time):

    if required_time < given_time:
        print("Overflow")
    elif required_time > given_time:
        print("Underflow")
    else:
        print("Filled")

# driver code
radius = 5 # radius of the tank
height = 10 # height of the tank
rate_of_flow = 10 # rate of flow of water

given_time = 70.0 # time given

# calculate the required time
required_time = volume(radius, height) /rate_of_flow

# printing the result
check_and_print(required_time, given_time)

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# program to check if Tank will
// overflow or not in given time
using System;

class Number {

    // function to calculate the volume of tank
    public static float volume(int radius, int height)
    {
        return ((22 / 7) * radius * radius * height);
```

```
}

// function to print overflow / filled /
// underflow accordingly
public static void check_and_print(double required_time,
                                    double given_time)
{
    if (required_time < given_time)
        Console.WriteLine("Overflow");
    else if (required_time > given_time)
        Console.WriteLine("Underflow");
    else
        Console.WriteLine("Filled");
}

// driver code
public static void Main()
{
    int radius = 5, // radius of the tank
        height = 10, // height of the tank
        rate_of_flow = 10; // rate of flow of water

    double given_time = 70.0; // time given

    // calculate the required time
    double required_time = volume(radius, height) / rate_of_flow;

    // printing the result
    check_and_print(required_time, given_time);
}
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP program to check if Tank will
// overflow or not in given time

// function to calculate
// the volume of tank
function volume($radius, $height)
{
    return ((22 / 7) * $radius *
            $radius * $height);
}
```

```
// function to print overflow
// / filled / underflow accordingly
function check_and_print($required_time,
                        $given_time)
{
    if ($required_time < $given_time)
        echo("Overflow");
    else if ($required_time > $given_time)
        echo("Underflow");
    else
        echo("Filled");
}

// Driver code

// radius of the tank
$radius = 5;

// height of the tank
$height = 10;

// rate of flow of water
$rate_of_flow = 10;

// time given
$given_time = 70.0;

// calculate the required time
$required_time = volume($radius, $height) /
                $rate_of_flow;

// printing the result
check_and_print($required_time, $given_time);

// This code is contributed by Ajit.
?>
```

Output:

Underflow

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/program-check-tank-will-overflow-underflow-filled-given-time/>

Chapter 182

Program to check if the points are parallel to X axis or Y axis

Program to check if the points are parallel to X axis or Y axis - GeeksforGeeks

Given n points, we need to check if these n points are parallel to X axis or Y axis or to No axis.

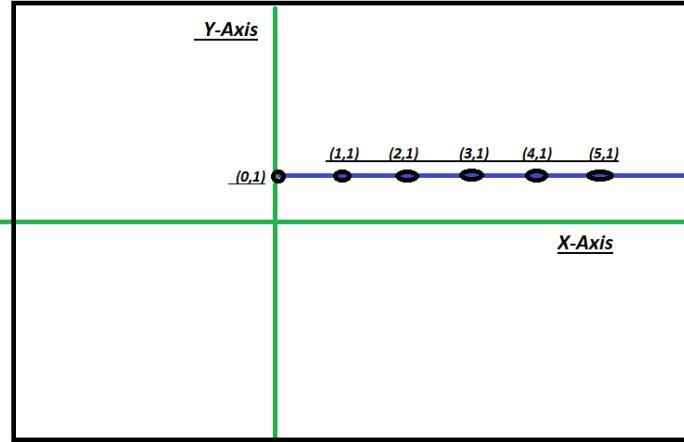
Examples:

```
Input : x[] = {0, 0, 0, 0, 0|
               y[] = {9, 2, 1, 3, 4}
Output : Parallel to X Axis
```

```
Input : x[] = {1, 2, 3|
               y[] = {9, 2, 1}
Output : Not Parallel to X or Y Axis
```

To find the points parallel to X or Y axis just check if the points are same for any axis or not. If all the points on X axis are same then the line is parallel to X axis. If all the points on Y axis are same so the line is parallel to Y axis otherwise it is not parallel to any axis.

Input the value N. And then Input the value of points in a



C++

```
// CPP program to check for parallel
// to X and Y Axis
#include <bits/stdc++.h>
using namespace std;

// To check for parallel line
void parallel(int n, int a[][])
{
    bool x = true, y = true;

    // checking for parallel to X and Y
    // axis condition
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < 2; j++) {
            if (a[i][0] != a[i + 1][0])
                x = false;
            if (a[i][1] != a[i + 1][1])
                y = false;
        }
    }

    // To display the output
    if (x)
```

```
        cout << "parallel to X Axis" << endl;
else if (y)
    cout << "parallel to Y Axis" << endl;
else
    cout << "Not parallel to X"
        << " and Y Axis" << endl;
}

// Driver's Code
int main()
{
    int a[] [2] = { { 1, 2 },
                    { 1, 4 },
                    { 1, 6 },
                    { 1, 0 } };

    int n = sizeof(a) / sizeof(a[0]);
    parallel(n, a);
    return 0;
}
```

Java

```
// JAVA program to illustrate..
// To check for parallel
// To X and Y Axis

import java.io.*;
import java.util.*;

class GFG {

    // To check for parallel line
    static void parallel(int a[][])
    {
        boolean x = true, y = true;

        // checking for parallel to X and Y
        // axis condition
        for (int i = 0; i < a.length - 1; i++) {
            for (int j = 0; j < 2; j++) {
                if (a[i][0] != a[i + 1][0])
                    x = false;
                if (a[i][1] != a[i + 1][1])
                    y = false;
            }
        }
    }
}
```

```
// To display the output
if (x)
    System.out.println("Parallel to X Axis");
else if (y)
    System.out.println("Parallel to Y Axis");
else
    System.out.println("Not parallel to X" +
                        " and Y axis");
}

public static void main(String[] args)
{
    int a[][] = { { 1, 2 },
                  { 1, 4 },
                  { 1, 6 },
                  { 1, 0 } };
    parallel(a);
}
}
```

PHP

```
<?php
// PHP program to check for parallel
// to X and Y Axis

// To check for parallel line
function parallel($n, $a)
{
    $x = true; $y = true;

    // checking for parallel
    // to X and Y axis condition
    for ($i = 0; $i < $n - 1; $i++)
    {
        for ($j = 0; $j < 2; $j++)
        {
            if ($a[$i][0] != $a[$i + 1][0])
                $x = false;
            if ($a[$i][1] != $a[$i + 1][1])
                $y = false;
        }
    }

    // To display the output
    if ($x)
        echo "parallel to X Axis" ;
    else if (y)
```

```
    echo "parallel to Y Axis" ;
else
    echo "Not parallel to X"
    , " and Y Axis";
}

// Driver's Code
$a = array(array(1, 2),
           array(1, 4),
           array(1, 6),
           array(1, 0));

$n = count($a);
parallel($n, $a);

//This code is contributed by anuj_67
?>
```

Output:

Parallel to X Axis

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/program-check-points-parallel-x-axis-y-axis/>

Chapter 183

Program to check if three points are collinear

Program to check if three points are collinear - GeeksforGeeks

Given three points, check whether they lie on a straight (collinear) or not

Examples :

Input : (1, 1), (1, 4), (1, 5)

Output : Yes

The points lie on a straight line

Input : (1, 5), (2, 5), (4, 6)

Output : No

The points do not lie on a straight line

First approach

Three points lie on the straight line if the area formed by the triangle of these three points is zero. So we will check if the area formed by the triangle is zero or not

Formula for area of triangle is :

$0.5 * [x_1 * (y_2 - y_3) + x_2 * (y_3 - y_1) + x_3 * (y_1 - y_2)]$

The formula is basically half of determinant value of following.

x₁ x₂ x₃

y₁ y₂ y₃

1 1 1

The above formula is derived from shoelace formula.

C

```
// C program to check if three
// points are collinear or not
// using area of triangle.
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

// function to check if point
// collinear or not
void collinear(int x1, int y1, int x2,
               int y2, int x3, int y3)
{
    // Calculation the area of
    // triangle. We have skipped
    // multiplication with 0.5
    // to avoid floating point
    // computations
    int a = x1 * (y2 - y3) +
           x2 * (y3 - y1) +
           x3 * (y1 - y2);

    if (a == 0)
        printf("Yes");
    else
        printf("No");
}

// Driver Code
int main()
{
    int x1 = 1, x2 = 1, x3 = 1,
        y1 = 1, y2 = 4, y3 = 5;
    collinear(x1, y1, x2, y2, x3, y3);
    return 0;
}
```

Java

```
// Java program to check if
// three points are collinear
// or not using area of triangle.
class GFG
{

    // function to check if
```

```
// point collinear or not
static void collinear(int x1, int y1, int x2,
                      int y2, int x3, int y3)
{
    /* Calculation the area of
     triangle. We have skipped
     multiplication with 0.5
     to avoid floating point
     computations */
    int a = x1 * (y2 - y3) +
           x2 * (y3 - y1) +
           x3 * (y1 - y2);

    if (a == 0)
        System.out.println("Yes");
    else
        System.out.println("No");
}

// Driver Code
public static void main(String args[])
{
    int x1 = 1, x2 = 1, x3 = 1,
        y1 = 1, y2 = 4, y3 = 5;

    collinear(x1, y1, x2, y2, x3, y3);
}

// This code is contributed by Sam007.
```

Python

```
# Python program to check
# if three points are collinear
# or not using area of triangle.

# function to check if
# point collinear or not
def collinear(x1, y1, x2, y2, x3, y3):

    """ Calculation the area of
        triangle. We have skipped
        multiplication with 0.5 to
        avoid floating point computations """
    a = x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2)
```

```
if (a == 0):
    print "Yes"
else:
    print "No"

# Driver Code
x1, x2, x3, y1, y2, y3 = 1, 1, 1, 1, 4, 5
collinear(x1, y1, x2, y2, x3, y3)

# This code is contributed
# by Sachin Bisht
```

C#

```
// C# program to check if
// three points are collinear
// or not using area of triangle.
using System;

class GFG
{

    /* function to check if
    point collinear or not */
    static void collinear(int x1, int y1, int x2,
                          int y2, int x3, int y3)
    {

        /* Calculation the area of
        triangle. We have skipped
        multiplication with 0.5 to
        avoid floating point computations */
        int a = x1 * (y2 - y3) +
               x2 * (y3 - y1) +
               x3 * (y1 - y2);

        if (a == 0)
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }

    // Driver code
    public static void Main ()
    {
        int x1 = 1, x2 = 1, x3 = 1,
            y1 = 1, y2 = 4, y3 = 5;
```

```
    collinear(x1, y1, x2, y2, x3, y3);
}
}

// This code is contributed by Sam007.
```

PHP

```
<?php
// PHP or not using area of triangle.

/* function to check if
point collinear or not */
function collinear($x1, $y1, $x2,
                   $y2, $x3, $y3)
{
    /* Calculation the area of
triangle. We have skipped
multiplication with 0.5 to
avoid floating point computations */
    $a = $x1 * ($y2 - $y3) +
        $x2 * ($y3 - $y1) +
        $x3 * ($y1 - $y2);

    if ($a == 0)
        printf("Yes");
    else
        printf("No");
}

// Driver Code
$x1 = 1; $x2 = 1; $x3 = 1;
$y1 = 1; $y2 = 4; $y3 = 5;
collinear($x1, $y1, $x2, $y2, $x3, $y3);

// This code is contributed by Sam007.
?>
```

Output :

Yes

Second approach

For three points, slope of any pair of points must be same as other pair.

For example, slope of line joining (x_2, y_2) and (x_3, y_3) , and line joining (x_1, y_1) and (x_2, y_2) must be same.

$$(y_3 - y_2)/(x_3 - x_2) = (y_2 - y_1)/(x_2 - x_1)$$

In other words,

$$(y_3 - y_2)(x_2 - x_1) = (y_2 - y_1)(x_3 - x_2)$$

If this equals zero then points lie on a straight line

C

```
// Slope based solution to check
// if three points are collinear.
#include <stdio.h>
#include <math.h>

/* function to check if
point collinear or not*/
void collinear(int x1, int y1, int x2,
              int y2, int x3, int y3)
{
    if ((y3 - y2) * (x2 - x1) ==
        (y2 - y1) * (x3 - x2))
        printf("Yes");
    else
        printf("No");
}

// Driver Code
int main()
{
    int x1 = 1, x2 = 1, x3 = 0,
        y1 = 1, y2 = 6, y3 = 9;
    collinear(x1, y1, x2, y2, x3, y3);
    return 0;
}
```

Python

```
# Slope based solution to check if three
# points are collinear.
```

```
# function to check if
# point collinear or not
def collinear(x1, y1, x2, y2, x3, y3):

    if ((y3 - y2)*(x2 - x1) == (y2 - y1)*(x3 - x2)):
        print ("Yes")
    else:
        print ("No")

# Driver Code
x1, x2, x3, y1, y2, y3 = 1, 1, 0, 1, 6, 9
collinear(x1, y1, x2, y2, x3, y3);

# This code is contributed
# by Sachin Bisht
```

Output :

No

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/program-check-three-points-collinear/>

Chapter 184

Program to check if water tank overflows when n solid balls are dipped in the water tank

Program to check if water tank overflows when n solid balls are dipped in the water tank - GeeksforGeeks

Given the dimensions of cylindrical water tank, spherical solid balls and the amount of water present in the tank check if water tank will overflow when balls are dipped in the water tank.

Examples :

```
input : H = 10, r = 5
        h = 5
        N = 2, R = 2
output : Not in overflow state
Explanation :
water tank capacity = 3.14 * r * r * H
                      = 3.14 * 5 * 5 * 10
                      = 785

volume of water in tank = 3.14 * r * r * h
                        = 3.14 * 5 * 5 * 5
                        = 392.5

Volume of balls = N * (4/3) * 3.14 * R * R * R
                  = 2 * (4/3) * 3.14 * 2 * 2 * 2
                  = 67.02

Total volume of water + dip balls = 392.5 + 67.02
                                    = 459.52
```

Total volume (459.02) < tank capacity (785)
So, there is no overflow in tank

```
input : H = 5, r = 3
       h = 3
       N = 3, R = 2
output : Overflow
Explanation:
water tank capacity = 3.14 * r * r * H
                      = 3.14 * 3 * 3 * 5
                      = 141.3
```

```
volume of water in tank = 3.14 * r * r * h
                          = 3.14 * 3 * 3 * 3
                          = 84.78
```

```
volume of balls = N * (4/3) * 3.14 * R * R * R
                  = 3 * (4/3) * 3.14 * 2 * 2 * 2
                  = 100.48
```

```
Total volume of water + dip balls = 84.78 + 100.48
                                    = 185.26
```

Total volume (185.26) > tank capacity (141.3)
So, tank will overflow

Approach:

When solid balls are dipped in water, level of water increases, hence volume of water will also increase.

Increasing in water volume = Total volume of dip balls

Volume of Cylinder = $3.14 * r * r * h$

where: r: radius of tank

h: height of tank

Number of balls are n

Balls have shape of Sphere

Volume of Sphere = $(4/3) * 3.14 * R * R * R$

Where R: Sphere's(solid ball) radius

After dipping all balls, if the total volume of water and all balls is less than or equal to the total volume of tank capacity then there will no overflow in tank, otherwise there will be overflow.

Below is the implementation of above approach:

C++

```
// C++ Program to check if water tank
```

```
// overflows when n solid balls are
// dipped in the water tank
#include <bits/stdc++.h>
using namespace std;

// function to find if tank will
// overflow or not
void overflow(int H, int r, int h,
              int N, int R)
{
    // cylinder capacity
    float tank_cap = 3.14 * r * r * H;

    // volume of water in tank
    float water_vol = 3.14 * r * r * h;

    // volume of n balls
    float balls_vol = N * (4 / 3) * 3.14 * R * R * R;

    // total volume of water
    // and n dipped balls
    float vol = water_vol + balls_vol;

    /* condition to check if tank is in
    overflow state or not */
    if (vol > tank_cap) {
        cout << "Overflow" << endl;
    }
    else {
        cout << "Not in overflow state"
            << endl;
    }
}

// main function
int main()
{
    // giving dimensions
    int H = 10, r = 5, h = 5,
        N = 2, R = 2;

    // calling function
    overflow(H, r, h, N, R);
    return 0;
}
```

Java

```
// JAVA Code for Program to check if
// water tank overflows
import java.util.*;

class GFG {

    // function to find if tank will
    // overflow or not
    static void overflow(int H, int r, int h,
                         int N, int R)
    {
        // cylinder capacity
        double tank_cap = 3.14 * r * r * H;

        // volume of water in tank
        double water_vol = 3.14 * r * r * h;

        // volume of n balls
        double balls_vol = N * (4 / 3) * 3.14 * R * R * R;

        // total volume of water
        // and n dipped balls
        double vol = water_vol + balls_vol;

        /* condition to check if tank is in
        overflow state or not */
        if (vol > tank_cap) {
            System.out.println("Overflow");
        }
        else {
            System.out.println("Not in overflow state");
        }
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        // giving dimensions
        int H = 10, r = 5, h = 5,
            N = 2, R = 2;

        // calling function
        overflow(H, r, h, N, R);
    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Python3

```
# Python code to check if water tank
# overflows when n solid balls are
# dipped in the water tank

# function to find if tak will
# overflow or not
def overflow( H, r, h, N, R ):

    # cylinder capacity
    tank_cap = 3.14 * r * r * H

    # volume of water in tank
    water_vol = 3.14 * r * r * h

    # volume of n balls
    balls_vol = N * (4 / 3) * 3.14 * R * R * R

    # total volume of water
    # and n dipped balls
    vol = water_vol + balls_vol

    # condition to check if tank is in
    # overflow state or not
    if vol > tank_cap:
        print("Overflow")
    else:
        print("Not in overflow state")

# Driver code

# giving dimensions
H = 10
r = 5
h = 5
N = 2
R = 2

# calling function
overflow (H, r, h, N, R)

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# Code for Program to check if
```

```
// water tank overflows
using System;

class GFG {

    // function to find if tank will
    // overflow or not
    static void overflow(int H, int r, int h,
                         int N, int R)
    {
        // cylinder capacity
        double tank_cap = 3.14 * r * r * H;

        // volume of water in tank
        double water_vol = 3.14 * r * r * h;

        // volume of n balls
        double balls_vol = N * (4 / 3) * 3.14 * R * R * R;

        // total volume of water
        // and n dipped balls
        double vol = water_vol + balls_vol;

        /* condition to check if tank is in
        overflow state or not */
        if (vol > tank_cap) {
            Console.WriteLine("Overflow");
        }
        else {
            Console.WriteLine("Not in overflow state");
        }
    }

    /* Driver program to test above function */
    public static void Main()
    {
        // giving dimensions
        int H = 10, r = 5, h = 5,
            N = 2, R = 2;

        // calling function
        overflow(H, r, h, N, R);
    }
}

// This code is contributed by vt_M.
```

```
<?php
// PHP Program to check if water tank
// overflows when n solid balls are
// dipped in the water tank

// function to find if tank
// will overflow or not

function overflow($H, $r, $h,
                  $N, $R)
{
    // cylinder capacity
    $tank_cap = 3.14 * $r *
                $r * $H;

    // volume of water in tank
    $water_vol = 3.14 * $r *
                 $r * $h;

    // volume of n balls
    $balls_vol = $N * (4/3) *
                 3.14 * $R *
                 $R * $R;

    // total volume of water
    // and n dipped balls
    $vol = $water_vol + $balls_vol;

    // condition to check if tank
    // is in overflow state or not
    if($vol > $tank_cap)
    {
        echo "Overflow", "\n";
    }
    else
    {
        echo "Not in overflow state", "\n";
    }
}

// Driver Code
// giving dimensions
$H = 10; $r = 5; $h = 5;
$N = 2; $R = 2;

// calling function
overflow ($H, $r, $h, $N, $R);
```

```
// This code is contributed by aj_36  
?>
```

Output :

Not in overflow state

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/program-check-water-tank-overflows-n-solid-balls-dipped-water-tank/>

Chapter 185

Program to check similarity of given two triangles

Program to check similarity of given two triangles - GeeksforGeeks

Given four array of 3 numbers each which represents sides and angles of two triangles. The task is to check if two triangles are similar or not. If it is similar, print the theorem by which it is.

Examples:

```
Input : side1 = [2, 3, 3] angle1 = [80, 60, 40]
        side2 = [4, 6, 6]   angle2 = [40, 60, 80]
Output: Triangles are similar by SSS AAA SAS
```

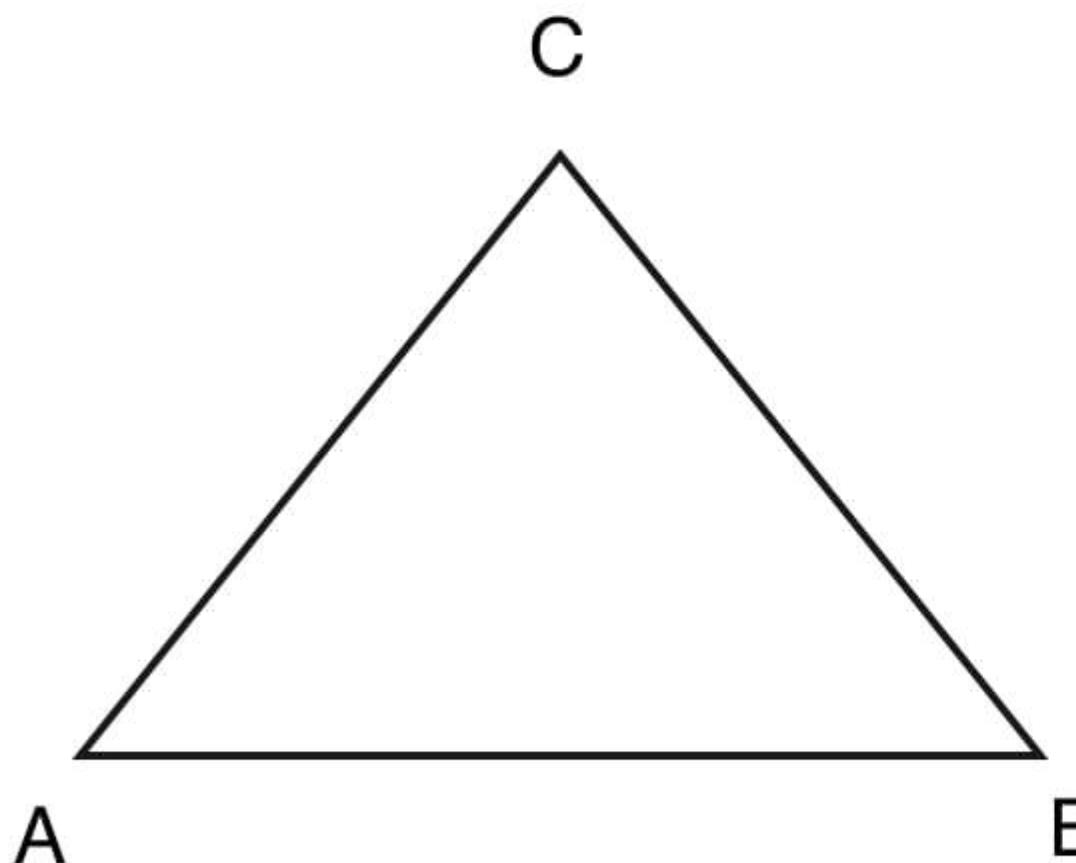
```
Input : side1 = [2, 3, 4] angle1 = [85, 45, 50]
        side2 = [4, 6, 6]   angle2 = [40, 60, 80]
Output: Triangles are not similar
```

Similar triangles are two or more triangles that have all corresponding angles that are equal and all corresponding sides that are proportionate. It does not matter what direction the triangles are facing. Their size does not matter as long as each side is proportionate. The similarity of triangles can be proved by the following theorems:

1. Side-Side-Side (SSS) similarity criteria :

If all the sides of a triangle are proportional to the corresponding sides of another triangle then the triangles are said to be similar by the property of *Side-Side-Side* (SSS).

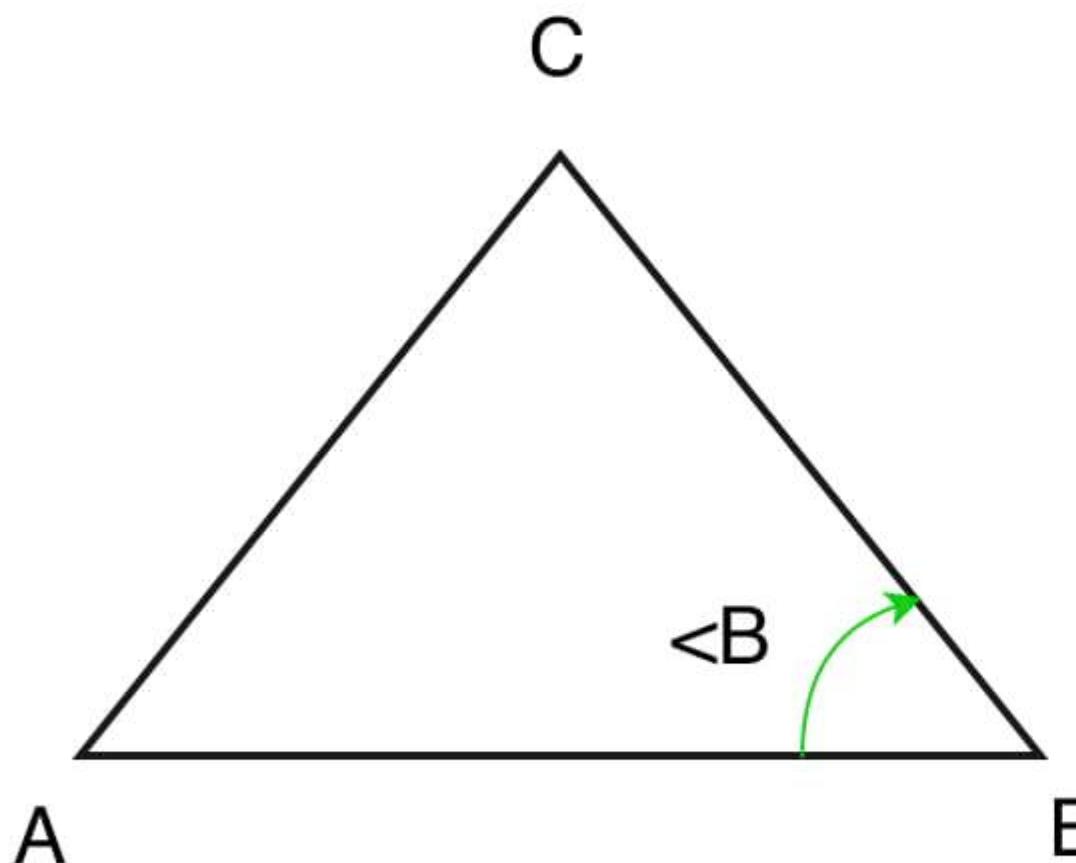
In a triangle ABC and PQR if, $AB/PQ = BC/QR = CA/RP$ triangles are similar.



2. Side-Angle-Side (SAS) similarity criteria :

If two sides of the two triangles are proportional and the angle between them is same in both triangle then the triangles are said to be similar by the property of *Side-Angle-Side* (SAS).

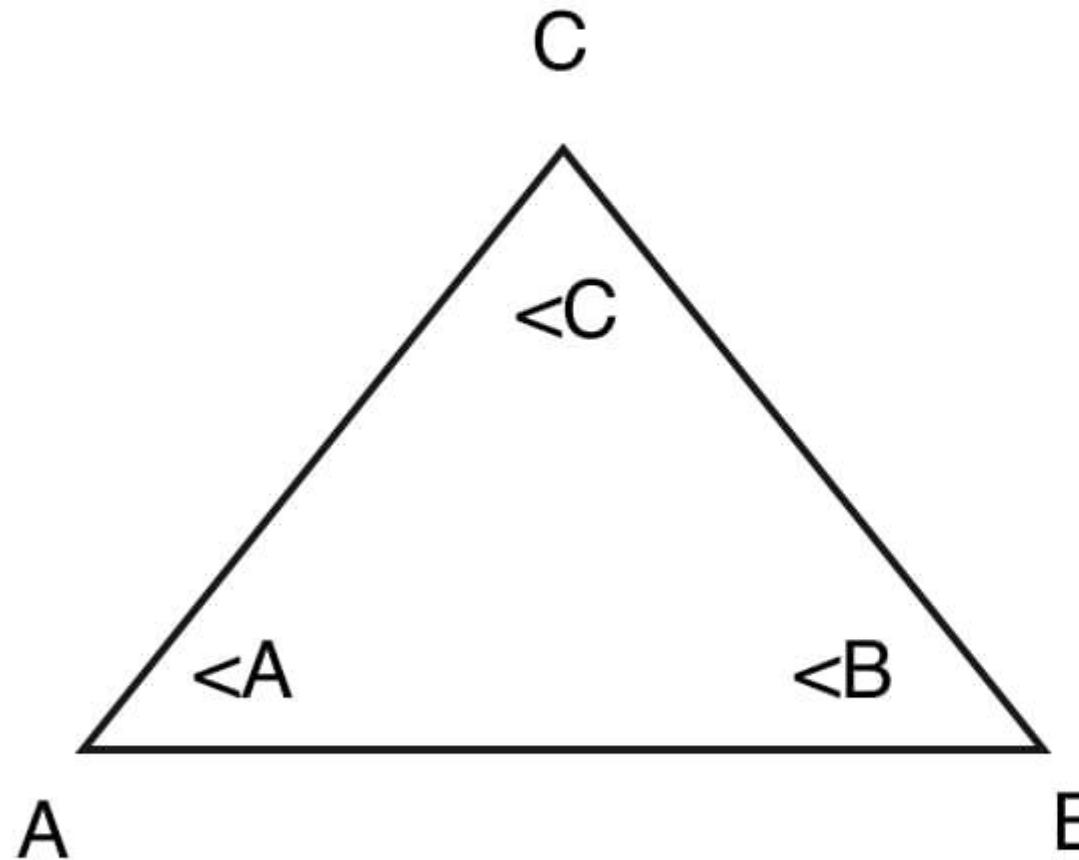
In a triangle ABC and PQR if, $AB/PQ = BC/QR$ and $\angle A = \angle P$ then triangles are similar.



3. Angle-Angle-Angle (AAA) similarity criteria :

If all the angles of a triangle are equal to the corresponding angles of another triangle then the triangles are said to be similar by the property of *Angle-Angle-Angle* (AAA).

In a triangle ABC and PQR if $\angle A = \angle P$, $\angle B = \angle Q$ and $\angle C = \angle R$ then triangles are similar.



Below is the implementation of the above approach:

C++

```
// C++ program to check
// similarity between
// two triangles.
#include<bits/stdc++.h>
using namespace std;

//Function for AAA similarity
int simi_aaa(int a1[], int a2[])
{
    sort(a1, a1 + 3);
    sort(a2, a2 + 3);

    // Check for AAA
    if (a1[0] == a2[0] &&
        a1[1] == a2[1] &&
        a1[2] == a2[2])
        return 1;
    else
        return 0;
}

// Function for
// SAS similarity
int simi_sas(int s1[], int s2[],
             int a1[], int a2[])
{
    sort(a1, a1 + 3);
    sort(a2, a2 + 3);
    sort(s1, s1 + 3);
    sort(s2, s2 + 3);

    // Check for SAS

    // angle b / w two smallest
    // sides is largest.
    if( s1[0] / s2[0] == s1[1] /
        s2[1])
    {
        // since we take angle
        // b / w the sides.
        if (a1[2] == a2[2])
            return 1;
    }
}
```

```
if (s1[1] / s2[1] == s1[2] /
    s2[2])
{
    if (a1[0] == a2[0])
        return 1;
}
if (s1[2] / s2[2] == s1[0] /
    s2[0])
{
    if(a1[1] == a2[1])
        return 1;
}
return 0;
}

// Function for SSS similarity
int simi_sss(int s1[], int s2[])
{
    sort(s1, s1 + 3);
    sort(s2, s2 + 3);

    // Check for SSS
    if(s1[0] / s2[0] == s1[1] / s2[1] &&
       s1[1] / s2[1] == s1[2] / s2[2] &&
       s1[2] / s2[2] == s1[0] / s2[0])
        return 1;

    return 0;
}

// Driver Code
int main()
{
    int s1[] = {2, 3, 3};
    int s2[] = {4, 6, 6};

    int a1[] = {80, 60, 40};
    int a2[] = {40, 60, 80};

    // function call for
    // AAA similarity
    int aaa = simi_aaa(a1, a2);

    // function call for
    // SSS similarity
    int sss = simi_sss(s1, s2) ;

    // function call for
```

```
// SAS similarity
int sas = simi_sas(s1, s2,
                     a1, a2) ;

// Check if triangles
// are similar or not
if(aaa == 1 ||
   sss == 1 || sas == 1)
{
    cout << "Triangles are "
        << "similar by ";
    if(aaa == 1) cout << "AAA ";
    if(sss == 1) cout << "SSS ";
    if(sas == 1) cout << "SAS.";
}

else
    cout << "Triangles are "
        << "not similar";
return 0;
}

// This code is contributed
// by Arnab Kundu
```

Java

```
// Java program to check
// similarity between
// two triangles.
import java.util.*;
class GFG1
{

    // Function for
    // AAA similarity
    static int simi_aaa(int a1[],
                         int a2[])
    {
        Arrays.sort(a1);
        Arrays.sort(a2);

        // Check for AAA
        if (a1[0] == a2[0] &&
            a1[1] == a2[1] &&
            a1[2] == a2[2])
            return 1;
        else
```

```
    return 0;

}

// Function for
// SAS similarity
static int simi_sas(int s1[], int s2[],
                     int a1[], int a2[])
{
    Arrays.sort(a1);
    Arrays.sort(a2);
    Arrays.sort(s1);
    Arrays.sort(s2);

    // Check for SAS

    // angle b / w two smallest
    // sides is largest.
    if(s1[0] / s2[0] == s1[1] / s2[1])
    {
        // since we take angle
        // b / w the sides.
        if (a1[2] == a2[2])
            return 1;
    }
    if (s1[1] / s2[1] == s1[2] / s2[2])
    {
        if (a1[0] == a2[0])
            return 1;
    }
    if (s1[2] / s2[2] == s1[0] / s2[0])
    {
        if(a1[1] == a2[1])
            return 1;
    }
    return 0;
}

// Function for
// SSS similarity
static int simi_sss(int s1[],
                     int s2[])
{
    Arrays.sort(s1);
    Arrays.sort(s2);

    // Check for SSS
    if(s1[0] / s2[0] == s1[1] / s2[1] &&
```

```
s1[1] / s2[1] == s1[2] / s2[2] &&
s1[2] / s2[2] == s1[0] / s2[0])
    return 1;

    return 0;
}

// Driver Code
public static void main(String args[])
{
    int s1[] = {2, 3, 3};
    int s2[] = {4, 6, 6};

    int a1[] = {80, 60, 40};
    int a2[] = {40, 60, 80};

    // function call for
    // AAA similarity
    int aaa = simi_aaa(a1, a2);

    // function call for
    // SSS similarity
    int sss = simi_sss(s1, s2) ;

    // function call for
    // SAS similarity
    int sas = simi_sas(s1, s2,
                        a1, a2) ;

    // Check if triangles
    // are similar or not
    if(aaa == 1 ||
       sss == 1 || sas == 1)
    {
        System.out.print("Triangles are " +
                         "similar by ");
        if(aaa == 1) System.out.print("AAA ");
        if(sss == 1) System.out.print("SSS ");
        if(sas == 1) System.out.print("SAS.");
    }
    else
        System.out.println("Triangles are " +
                           "not similar");
}
}

// This code is contributed
// by Arnab Kundu
```

Python

```
# Python program to check
# similarity between two triangles.

# Function for AAA similarity
def simi_aaa(a1, a2):
    a1 = [float(i) for i in a1]
    a2 = [float(i) for i in a2]
    a1.sort()
    a2.sort()

    # Check for AAA
    if a1[0] == a2[0] and a1[1] == a2[1] and a1[2] == a2[2]:
        return 1
    return 0

# Function for SAS similarity
def simi_sas(s1, s2, a1, a2):

    s1 = [float(i) for i in s1]
    s2 = [float(i) for i in s2]
    a1 = [float(i) for i in a1]
    a2 = [float(i) for i in a2]

    s1.sort()
    s2.sort()
    a1.sort()
    a2.sort()

    # Check for SAS

    # angle b / w two smallest sides is largest.
    if s1[0] / s2[0] == s1[1] / s2[1]:

        # since we take angle b / w the sides.
        if a1[2] == a2[2]:
            return 1

        if s1[1] / s2[1] == s1[2] / s2[2]:
            if a1[0] == a2[0]:
                return 1

        if s1[2] / s2[2] == s1[0] / s2[0]:
            if a1[1] == a2[1]:
                return 1

    return 0
```

```
# Function for SSS similarity
def simi_sss(s1, s2):

    s1 = [float(i) for i in s1]
    s2 = [float(i) for i in s2]
    s1.sort()
    s2.sort()

    # Check for SSS
    if(s1[0] / s2[0] == s1[1] / s2[1]
       and s1[1] / s2[1] == s1[2] / s2[2]
       and s1[2] / s2[2] == s1[0] / s2[0]):
        return 1

    return 0

# Driver Code
s1 = [2, 3, 3]
s2 = [4, 6, 6]

a1 = [80, 60, 40]
a2 = [40, 60, 80]

# function call for AAA similarity
aaa = simi_aaa(a1, a2)

# function call for SSS similarity
sss = simi_sss(s1, s2)

# function call for SAS similarity
sas = simi_sas(s1, s2, a1, a2)

# Check if triangles are similar or not
if aaa or sss or sas:
    print "Triangles are similar by",
    if aaa: print "AAA",
    if sss: print "SSS",
    if sas: print "SAS"
else: print "Triangles are not similar"
```

Output:

Triangles are similar by AAA SSS SAS

Improved By : [andrew1234](#)

Source

<https://www.geeksforgeeks.org/program-to-check-similarity-of-given-two-triangles/>

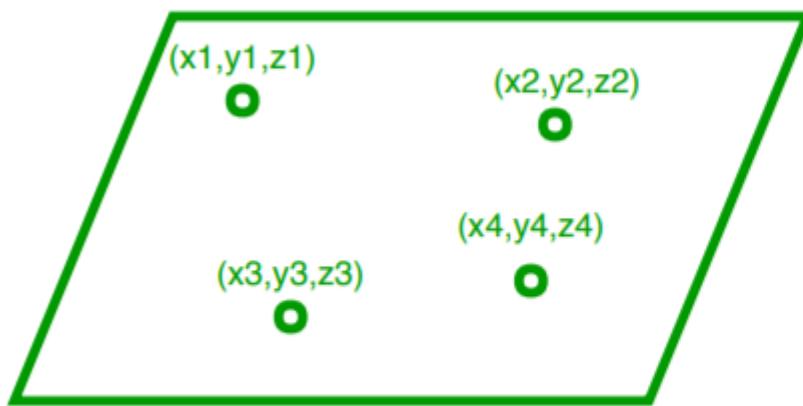
Chapter 186

Program to check whether 4 points in a 3-D plane are Coplanar

Program to check whether 4 points in a 3-D plane are Coplanar - GeeksforGeeks

Given 4 points (x_1, y_1, z_1) , (x_2, y_2, z_2) , (x_3, y_3, z_3) , (x_4, y_4, z_4) . The task is to write a program to check whether these 4 points are coplanar or not.

Note: 4 points in a 3-D plane are said to be coplanar if they lie in the same plane.



Coplanar Points

Non-Coplanar Points

Examples:

Input:

$x_1 = 3, y_1 = 2, z_1 = -5$
 $x_2 = -1, y_2 = 4, z_2 = -3$
 $x_3 = -3, y_3 = 8, z_3 = -5$
 $x_4 = -3, y_4 = 2, z_4 = 1$

Output: Coplanar

Input:

$x_1 = 0, y_1 = -1, z_1 = -1$
 $x_2 = 4, y_2 = 5, z_2 = 1$
 $x_3 = 3, y_3 = 9, z_3 = 4$
 $x_4 = -4, y_4 = 4, z_4 = 3$

Output: Not Coplanar

Approach:

1. To check whether 4 points are coplanar or not, first of all, find the equation of the plane passing through any three of the given points.
[Approach to find equation of a plane passing through 3 points.](#)
2. Then, check whether the 4th point satisfies the equation obtained in step 1. That is, putting the value of 4th point in the equation obtained. If it satisfies the equation then the 4 points are Coplanar otherwise not.

Below is the implementation of the above idea:

```
# Python program to check if 4 points
# in a 3-D plane are Coplanar

# Function to find equation of plane.
def equation_plane(x1, y1, z1, x2, y2, z2, x3,
                    y3, z3, x, y, z):

    a1 = x2 - x1
    b1 = y2 - y1
    c1 = z2 - z1
    a2 = x3 - x1
    b2 = y3 - y1
    c2 = z3 - z1
    a = b1 * c2 - b2 * c1
    b = a2 * c1 - a1 * c2
    c = a1 * b2 - b1 * a2
    d = (- a * x1 - b * y1 - c * z1)

    # equation of plane is: a*x + b*y + c*z = 0 #

    # checking if the 4th point satisfies
    # the above equation
    if(a * x + b * y + c * z + d == 0):
        print("Coplanar")
    else:
        print("Not Coplanar")

# Driver Code
x1 = 3
y1 = 2
z1 = -5
x2 = -1
y2 = 4
z2 = -3
x3 = -3
```

```
y3 = 8
z3 = -5
x4 = -3
y4 = 2
z4 = 1
equation_plane(x1, y1, z1, x2, y2, z2, x3,
                 y3, z3, x4, y4, z4)
```

Output:

Coplanar

Source

<https://www.geeksforgeeks.org/program-to-check-whether-4-points-in-a-3-d-plane-are-coplanar/>

Chapter 187

Program to determine the octant of the axial plane

Program to determine the octant of the axial plane - GeeksforGeeks

Given 3 coordinates x, y and z, the task is to determine the octant of the axial plane.

Examples:

Input: 2, 3, 4

Output: Point lies in 1st octant

Input: -4, 2, -8

Output: Point lies in 6th octant

Input: -6, -2, 8

Output: Point lies in 3rd octant

Approach: Given below are the conditions which need to be checked in order to determine the octant of the axial plane.

- Check if $x \geq 0$ and $y \geq 0$ and $z \geq 0$, then Point lies in 1st octant.
- Check $x < 0$ and $y \geq 0$ and $z \geq 0$, then Point lies in 2nd octant.
- Check if $x < 0$ and $y < 0$ and $z \geq 0$, then Point lies in 3rd octant.
- Check if $x \geq 0$ and $y < 0$ and $z \geq 0$, then Point lies in 4th octant.
- Check if $x \geq 0$ and $y \geq 0$ and $z < 0$, then Point lies in 5th octant.
- Check if $x < 0$ and $y \geq 0$ and $z < 0$, then Point lies in 6th octant.
- Check if $x < 0$ and $y < 0$ and $z < 0$, then Point lies in 7th octant.
- Check if $x \geq 0$ and $y < 0$ and $z < 0$, then Point lies in 8th octant.

Below is the implementation of the above approach:

C

```
// C program to print octant
// of a given point.
#include <stdio.h>

// Function to print octant
void octant(float x, float y,
            float z)
{
    if (x >= 0 && y >= 0 && z >= 0)
        printf("Point lies in 1st octant\n");

    else if (x < 0 && y >= 0 && z >= 0)
        printf("Point lies in 2nd octant\n");

    else if (x < 0 && y < 0 && z >= 0)
        printf("Point lies in 3rd octant\n");

    else if (x >= 0 && y < 0 && z >= 0)
        printf("Point lies in 4th octant\n");

    else if (x >= 0 && y >= 0 && z < 0)
        printf("Point lies in 5th octant\n");

    else if (x < 0 && y >= 0 && z < 0)
        printf("Point lies in 6th octant\n");

    else if (x < 0 && y < 0 && z < 0)
        printf("Point lies in 7th octant\n");

    else if (x >= 0 && y < 0 && z < 0)
        printf("Point lies in 8th octant\n");
}

// Driver Code
int main()
{
    float x = 2, y = 3, z = 4;
    octant(x, y, z);

    x = -4, y = 2, z = -8;
    octant(x, y, z);

    x = -6, y = -2, z = 8;
    octant(x, y, z);
}

// This code is contributed
// by Amber_Saxena.
```

Python

```
# Python program to print octant of a
# given point.

# Function to print octant
def octant(x, y, z):

    if x >= 0 and y >= 0 and z >= 0:
        print "Point lies in 1st octant"

    elif x < 0 and y >= 0 and z >= 0:
        print "Point lies in 2nd octant"

    elif x < 0 and y < 0 and z >= 0:
        print "Point lies in 3rd octant"

    elif x >= 0 and y < 0 and z >= 0:
        print "Point lies in 4th octant"

    elif x >= 0 and y >= 0 and z < 0:
        print "Point lies in 5th octant"

    elif x < 0 and y >= 0 and z < 0:
        print "Point lies in 6th octant"

    elif x < 0 and y < 0 and z < 0:
        print "Point lies in 7th octant"

    elif x >= 0 and y < 0 and z < 0:
        print "Point lies in 8th octant"

# Driver Code
x, y, z = 2, 3, 4
octant(x, y, z)

x, y, z = -4, 2, -8
octant(x, y, z)

x, y, z = -6, -2, 8
octant(x, y, z)
```

PHP

```
<?php
// PHP program to print octant
```

```
// of a given point.

// Function to print octant
function octant($x, $y, $z)
{
    if ($x >= 0 && $y >= 0 && $z >= 0)
        echo "Point lies in 1st octant\n";

    else if ($x < 0 && $y >= 0 && $z >= 0)
        echo "Point lies in 2nd octant\n";

    else if ($x < 0 && $y < 0 && $z >= 0)
        echo "Point lies in 3rd octant\n";

    else if ($x >= 0 && $y < 0 && $z >= 0)
        echo "Point lies in 4th octant\n";

    else if ($x >= 0 && $y >= 0 && $z < 0)
        echo "Point lies in 5th octant\n";

    else if ($x < 0 && $y >= 0 && $z < 0)
        echo "Point lies in 6th octant\n";

    else if ($x < 0 && $y < 0 && $z < 0)
        echo "Point lies in 7th octant\n";

    else if ($x >= 0 && $y < 0 && $z < 0)
        echo "Point lies in 8th octant\n";
}

// Driver Code
$x = 2;
$y = 3;
$z = 4;
octant($x, $y, $z);

$x = -4;
$y = 2;
$z = -8;
octant($x, $y, $z);

$x = -6;
$y = -2;
$z = 8;
octant($x, $y, $z);

// This code is contributed
// by Amber_Saxena.
```

?>

Output:

```
Point lies in 1st octant
Point lies in 6th octant
Point lies in 3rd octant
```

Improved By : [Amber_Saxena](#)

Source

<https://www.geeksforgeeks.org/program-to-determine-the-octant-of-the-axial-plane/>

Chapter 188

Program to find third side of triangle using law of cosines

Program to find third side of triangle using law of cosines - GeeksforGeeks

Given two sides A, B and angle C. Find the third side of the triangle using law of cosines.

Examples:

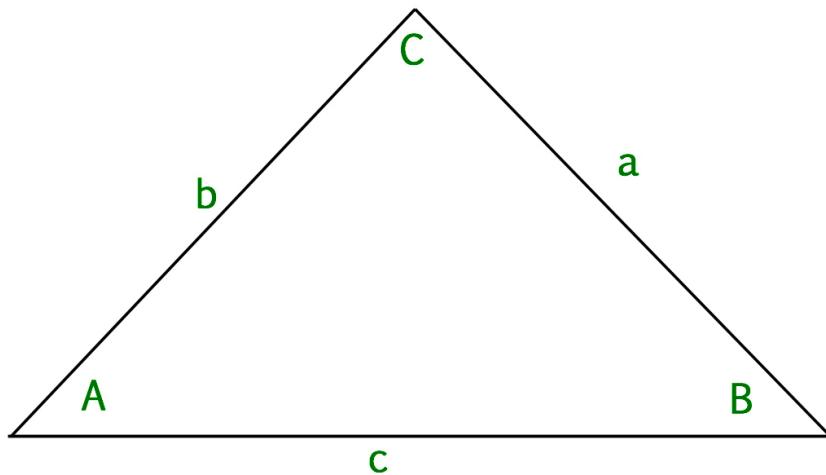
```
Input : a = 5, b = 8, c = 49
Output : 6.04339
```

In particular, the Law of Cosines can be used to find the length of the third side of a triangle when you know the length of two sides and the angle in between. See [here](#) to learn to how to find the value of cos.

Let us assume a, b, c are the sides of triangle where c is the side across from angle C. Then,

```
c^2 = a^2 + b^2 - 2*a*b*cos(c)
OR
c = sqrt(a^2 + b^2 - 2*a*b*cos(c))
```

$$C^2 = a^2 + b^2 - 2*a*b*\cos(C)$$



C++

```
// CPP program to find third
// side of triangle using
// law of cosines

#include <bits/stdc++.h>
using namespace std;

// Function to calculate cos value of angle c
float cal_cos(float n)
{
    float accuracy = 0.0001, x1, denominator, cosx, cosval;

    // Converting degrees to radian
    n = n * (3.142 / 180.0);

    x1 = 1;

    // Maps the sum along the series
    cosx = x1;

    // Holds the actual value of sin(n)
```

```
cosval = cos(n);
int i = 1;
do {
    denominator = 2 * i * (2 * i - 1);
    x1 = -x1 * n * n / denominator;
    cosx = cosx + x1;
    i = i + 1;
} while (accuracy <= fabs(cosval - cosx));

return cosx;
}

// Function to find third side
float third_side(int a, int b, float c)
{
    float angle = cal_cos(c);
    return sqrt((a * a) + (b * b) - 2 * a * b * angle);
}
// Driver program to check the above function
int main()
{
    float c = 49;
    int a = 5, b = 8;
    // function call
    cout << third_side(a, b, c);

    return 0;
}
```

Java

```
// Java program to find third
// side of triangle using
// law of cosines
class GFG
{
    // Function to calculate
    // cos value of angle c
    static float cal_cos(float n)
    {
        float accuracy = 0.0001f, x1;
        float denominator, cosx, cosval;

        // Converting degrees to radian
        n = n * (3.142f / 180.0f);

        x1 = 1;
```

```
// Maps the sum along the series
cosx = x1;

// Holds the actual value of sin(n)
cosval = (float)Math.cos(n);
int i = 1;
do {
    denominator = 2 * i * (2 * i - 1);
    x1 = -x1 * n * n / denominator;
    cosx = cosx + x1;
    i = i + 1;
} while (accuracy <=
         Math.abs(cosval - cosx));

return cosx;
}

// Function to find third side
static float third_side(int a,
                        int b, float c)
{
    float angle = cal_cos(c);

    return (float)Math.sqrt((a * a) +
                           (b * b) - 2 * a * b * angle);
}

// Driver code
public static void main (String[] args)
{
    float c = 49;
    int a = 5, b = 8;

    // function call
    System.out.print(Math.round(third_side(a,
                                           b, c)*100000.0)/100000.0);
}
}

// This code is contributed by Anant Agarwal.
```

C#

```
// C# program to find third
// side of triangle using
// law of cosines
using System;
```

```
class GFG
{
    // Function to calculate
    // cos value of angle c
    static float cal_cos(float n)
    {
        float accuracy = 0.0001f, x1;
        float denominator, cosx, cosval;

        // Converting degrees to radian
        n = n * (3.142f / 180.0f);

        x1 = 1;

        // Maps the sum along the series
        cosx = x1;

        // Holds the actual value of sin(n)
        cosval = (float)Math.Cos(n);
        int i = 1;
        do {
            denominator = 2 * i * (2 * i - 1);
            x1 = -x1 * n * n / denominator;
            cosx = cosx + x1;
            i = i + 1;
        } while (accuracy <=
            Math.Abs(cosval - cosx));

        return cosx;
    }

    // Function to find third side
    static float third_side(int a,
                           int b, float c)
    {
        float angle = cal_cos(c);

        return (float)Math.Sqrt((a * a) +
                               (b * b) - 2 * a * b * angle);
    }

    // Driver code
    public static void Main ()
    {
        float c = 49;
        int a = 5, b = 8;

        // function call
```

```
        Console.WriteLine(Math.Round(third_side(a,
                                              b, c)*100000.0)/100000.0);
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find third
// side of triangle using
// law of cosines

// Function to calculate
// cos value of angle n
function cal_cos( $n )
{
    $accuracy = 0.0001;
    $x1; $denominator;
    $cosx; $cosval;

    // Converting degrees
    // to radian
    $n = $n * (3.142 / 180.0);

    $x1 = 1;

    // Maps the sum
    // along the series
    $cosx = $x1;

    // Holds the actual
    // value of sin(n)
    $cosval = cos($n);
    $i = 1;
    do
    {
        $denominator = 2 * $i *
                      (2 * $i - 1);
        $x1 = -$x1 * $n * $n /
              $denominator;
        $cosx = $cosx + $x1;
        $i = $i + 1;
    } while ($accuracy <= ($cosval -
                           $cosx));

    return $cosx;
```

```
}

// Function to find third side
function third_side($a, $b, $c)
{
    $angle = cal_cos($c);
    return sqrt(($a * $a) +
                ($b * $b) - 2 *
                $a * $b * $angle);
}

// Driver Code
$c = 49;
$a = 5;
$b = 8;

// function call
echo third_side($a, $b, $c);

// This code is contributed
// by ajit
?>
```

Output:

6.04339

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/program-find-third-side-triangle-using-law-cosines/>

Chapter 189

Program to find Circumcenter of a Triangle

Program to find Circumcenter of a Triangle - GeeksforGeeks

Given 3 non-collinear points in the 2D Plane P, Q and R with their respective x and y coordinates, find the circumcenter of the triangle.

Note: Circumcenter of a triangle is the centre of the circle, formed by the three vertices of a triangle. Note that three points can uniquely determine a circle.

Examples:

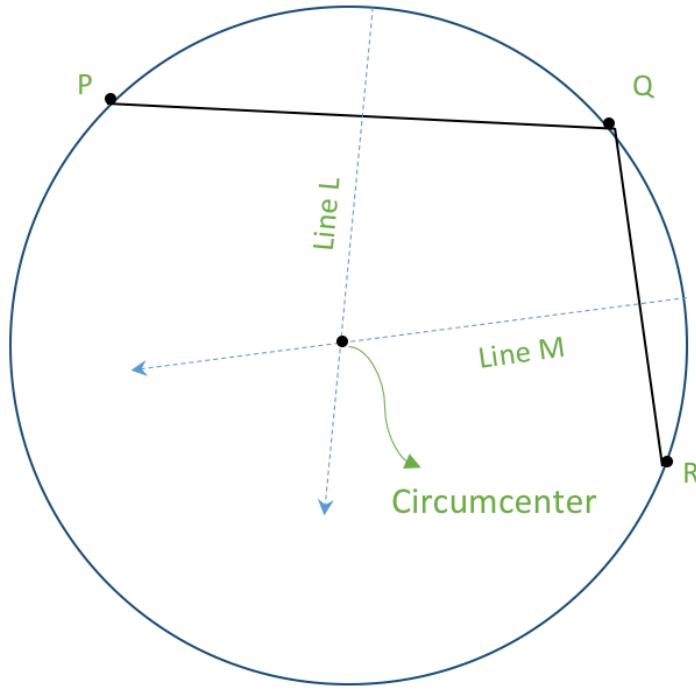
```
Input : P(6, 0)
        Q(0, 0)
        R(0, 8)
Output : The circumcenter of the triangle PQR
         is: (3, 4)

Input : P(1, 1)
        Q(0, 0)
        R(2, 2)
Output : The two perpendicular bisectors found
         come parallel. Thus, the given points
         do not form a triangle and are collinear
```

Given, three points of the triangle, we can easily find the sides of the triangle. Now, we have the equations of the lines for the three sides of the triangle. After getting these, we can find the circumcenter of the triangle by a simple property stated as under:

The circumcenter of the triangle is point where all the perpendicular bisectors of the sides of the triangle intersect.

This is well explained in the following diagram.



Note here that, there is no need to find all of the three sides of the triangle. Finding two sides is sufficient as we can uniquely find the point of intersection using just two perpendicular bisectors. The third perpendicular bisector will itself pass through the so found circumcenter.

The things to be done can be divided as under:

1. Find 2 lines (say PQ and QR) which form the sides of the triangle.
2. Find the perpendicular bisectors of PQ and QR (say lines L and M respectively).
3. Find the point of intersection of lines L and M as the circumcenter of the given triangle.

STEP 1

Refer this post [Program to find line passing through 2 Points](#)

STEP 2

Let PQ be represented as $ax + by = c$

A line perpendicular to this line is represented as $-bx + ay = d$ for some d .

However, we are interested in the perpendicular bisector. So, we find the mid-point of P and Q and putting this value in the standard equation, we get the value of d .

Similarly, we repeat the process for QR.

$$d = -bx + ay$$

$$\text{where, } x = (x_p + x_q)/2$$

$$\text{AND } y = (y_p + y_q)/2$$

STEP 3

Refer this post [Program for Point of Intersection of Two Lines](#)

```
// C++ program to find the CIRCUMCENTER of a
// triangle
#include <iostream>
#include <cfloat>
using namespace std;

// This pair is used to store the X and Y
// coordinate of a point respectively
#define pdd pair<double, double>

// Function to find the line given two points
void lineFromPoints(pdd P, pdd Q, double &a,
                     double &b, double &c)
{
    a = Q.second - P.second;
    b = P.first - Q.first;
    c = a*(P.first) + b*(P.second);
}

// Function which converts the input line to its
// perpendicular bisector. It also inputs the points
// whose mid-point lies on the bisector
void perpendicularBisectorFromLine(pdd P, pdd Q,
                                    double &a, double &b, double &c)
{
    pdd mid_point = make_pair((P.first + Q.first)/2,
                               (P.second + Q.second)/2);

    // c = -bx + ay
    c = -b*(mid_point.first) + a*(mid_point.second);

    double temp = a;
    a = -b;
    b = temp;
}

// Returns the intersection point of two lines
pdd lineLineIntersection(double a1, double b1, double c1,
                        double a2, double b2, double c2)
{
    double determinant = a1*b2 - a2*b1;
    if (determinant == 0)
    {
        // The lines are parallel. This is simplified
        // by returning a pair of FLT_MAX
```

```

        return make_pair(FLT_MAX, FLT_MAX);
    }

    else
    {
        double x = (b2*c1 - b1*c2)/determinant;
        double y = (a1*c2 - a2*c1)/determinant;
        return make_pair(x, y);
    }
}

void findCircumCenter(pdd P, pdd Q, pdd R)
{
    // Line PQ is represented as ax + by = c
    double a, b, c;
    lineFromPoints(P, Q, a, b, c);

    // Line QR is represented as ex + fy = g
    double e, f, g;
    lineFromPoints(Q, R, e, f, g);

    // Converting lines PQ and QR to perpendicular
    // vbisectors. After this, L = ax + by = c
    // M = ex + fy = g
    perpendicularBisectorFromLine(P, Q, a, b, c);
    perpendicularBisectorFromLine(Q, R, e, f, g);

    // The point of intersection of L and M gives
    // the circumcenter
    pdd circumcenter =
        lineLineIntersection(a, b, c, e, f, g);

    if (circumcenter.first == FLT_MAX &&
        circumcenter.second == FLT_MAX)
    {
        cout << "The two perpendicular bisectors "
            "found come parallel" << endl;
        cout << "Thus, the given points do not form "
            "a triangle and are collinear" << endl;
    }

    else
    {
        cout << "The circumcenter of the triangle PQR is: ";
        cout << "(" << circumcenter.first << ", "
            << circumcenter.second << ")" << endl;
    }
}

```

```
// Driver code.  
int main()  
{  
    pdd P = make_pair(6, 0);  
    pdd Q = make_pair(0, 0);  
    pdd R = make_pair(0, 8);  
    findCircumCenter(P, Q, R);  
    return 0;  
}
```

Output:

The circumcenter of the triangle PQR is: (3, 4)

Source

<https://www.geeksforgeeks.org/program-find-circumcenter-triangle-2/>

Chapter 190

Program to find Circumference of a Circle

Program to find Circumference of a Circle - GeeksforGeeks

Given radius of a circle, write a program to find its circumference.

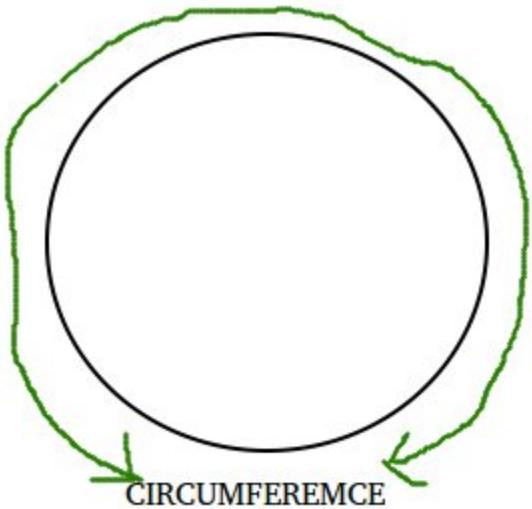
Examples :

Input : 2
Output : Circumference = 12.566

Input : 8
Output : Circumference = 50.264

In a circle, points lie in the boundary of a circle are at same distance from its center. This distance is called radius. Circumference of a circle can simply be evaluated using following formula.

Circumference = $2\pi r$
where r is the radius of circle
and value of π = 3.1415.



C++

```
// CPP program to find circumference of circle
#include<bits/stdc++.h>
using namespace std;

#define PI 3.1415

double circumference(double r)
{
    double cir = 2*PI*r;
    return cir;
}

// driver function
int main()
{
    double r = 5;
    cout << "Circumference = "
        << circumference(r);
    return 0;
}
```

Python3

```
# Python3 code to find
# circumference of circle
```

```
PI = 3.1415

# utility function
def circumference(r):
    return (2 * PI * r)

# driver function
print ('%.3f' % circumference(5))

# This code is contributed by Saloni Gupta
```

Java

```
// Java program to find circumference of circle
import java.io.*;

class Geometry {

    // utility function
    static double circumference(double r){

        double PI = 3.1415;
        double cir = 2*PI*r;
        return cir;
    }

    // driver function
    public static void main (String[] args) {

        double r = 5;
        double result = Math.round(circumference(r) * 1000) / 1000.0;
        System.out.println("Circumference = "+ result);
    }
}

// This article is contributed by Chinmoy Lenka
```

C#

```
// C# program to find circumference of circle
using System;

class GFG {

    // utility function
    static double circumference(double r){
```

```
double PI = 3.1415;
double cir = 2*PI*r;
return cir;
}

// driver function
public static void Main () {

    double r = 5;
    double result =
        Math.Round(circumference(r)
                   * 1000) / 1000.0;

    Console.WriteLine("Circumference = "
                      + result);
}
}

// This article is contributed by anuj_67.
```

PHP

```
<?php
// PHP program to find
// circumference of circle

$PI= 3.1415;

function circumference($r)
{
    global $PI;
    $cir = 2 * $PI * $r;
    return $cir;
}

// Driver Code
$r = 5;
echo "Circumference = ",
circumference($r);

// This code is contributed aj_36
?>
```

Output :

Circumference = 31.415

Improved By : jit_t, vt_m

Source

<https://www.geeksforgeeks.org/program-find-circumference-circle/>

Chapter 191

Program to find Perimeter / Circumference of Square and Rectangle

Program to find Perimeter / Circumference of Square and Rectangle - GeeksforGeeks

The circumference of a figure is the sum of all the side lengths. To calculate the circumference of square, length of one of the side is required as all sides are equal. To calculate the circumference of rectangle, length and breadth of rectangle is required.

Circumference of a Square:



The circumference of a square is given by the formula:

$$C = 4 * a$$

where a is the side length.

Examples :

input: 4

output: 16

input: 3
output: 12

C++

```
// CPP program to find
// Circumference of a square
#include <bits/stdc++.h>
using namespace std;

int Circumference(int a)
{
    return 4 * a;
}

// Driver Code
int main()
{
    int a = 5;
    cout << "Circumference of"
         << " a square is "
         << Circumference(a);
    return 0;
}

// This code is contributed
// by mohitw16
```

Java

```
// Java program to find
// Circumference of a square

import java.io.*;
class GFG
{
    int Circumference(int a)
    {
        return 4 * a;
    }

    // Driver code
    public static void main(String args[])
    {
        GFG obj = new GFG();
```

```
        int a = 5;
        System.out.println("Circumference of " +
                            "a square is " +
                            obj.Circumference(a));
    }
}

// This code is contributed
// by Anshika Goyal.
```

Python3

```
# Python3 Program to find
# Circumference of a square

def Circumference(a):
    return (4 * a)

# Driver code
a = 5
c = Circumference(a)
print("Circumference of a " +
      "square is % d" % (c))
```

C#

```
// C# program to find Circumference
// of a square
using System;

class GFG
{

    static int Circumference(int a)
    {
        return 4 * a;
    }

    // Driver Code
    public static void Main()
    {
        int a = 5;

        Console.WriteLine("Circumference" +
                          " of a square is " +
                          Circumference(a));
    }
}
```

```
}
```

```
// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find
// Circumference of a square

function Circumference($a)
{
    return 4 * $a;
}

// Driver Code
$a = 5;
echo "Circumference of a ".
      "square is ",
      Circumference($a);

// This code is contributed by ajit
?>
```

Output :

Circumference Of a square is 20

Circumference of a rectangle:



The circumference of a rectangle is given by the formula:

$C = 2 * (l + W)$
where l is the length and W is the width.

Examples :

input: 2 4
output: 12

input: 4 6
output: 20

C++

```
// C++ Program to find
// Circumference of a rectangle
#include <iostream>
using namespace std;

int Circumference(int l, int w)
{
    return (2 * (l + w));
}

// Driver code
int main()
{
    int l = 8, w = 4;

    int c = Circumference(l, w);

    cout << "Circumference of a"
        << " rectangle is "
        << c << endl;

    return 0;
}

// This code is contributed by vt_m.
```

Python3

```
# Python Program to find
# Circumference of a rectangle

def Circumference(l, w):
    return (2 * (l + w))

# Driver code
l = 8
w = 4
c = Circumference(l, w)
```

```
print("Circumference of a" +
      " rectangle is % d" % (c))
```

Java

```
// java Program to find
// Circumference of a rectangle
import java.io.*;

class GFG
{

    static int Circumference(int l,
                           int w)
    {
        return (2 * (l + w));
    }

    // Driver code
    static public void main(String[] args)
    {
        int l = 8, w = 4;

        int c = Circumference(l, w);

        System.out.println("Circumference of " +
                           "a rectangle is " + c);
    }
}

// This code is contributed by vt_m.
```

C#

```
// C# Program to find
// circumference of a rectangle
using System;

class GFG
{

    static int Circumference(int l,
                           int w)
    {
        return (2 * (l + w));
    }
}
```

```
// Driver code
static public void Main()
{
    int l = 8, w = 4;

    int c = Circumference(l, w);

    Console.WriteLine("Circumference of " +
                      "a rectangle is " + c);
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// Php Program to find
// Circumference of a rectangle

function Circumference($l,$w)
{
    return (2 * ($l + $w));
}

// Driver code
$l = 8; $w = 4;

$c = Circumference($l, $w);

echo "Circumference of a ".
     "rectangle is " , $c , "\n";

// This code is contributed by aj_36.
?>
```

Output :

Circumference of a rectangle is 24

Improved By : [vt_m, jit_t](#)

Source

<https://www.geeksforgeeks.org/python-program-find-perimeter-circumference-square-rectangle/>

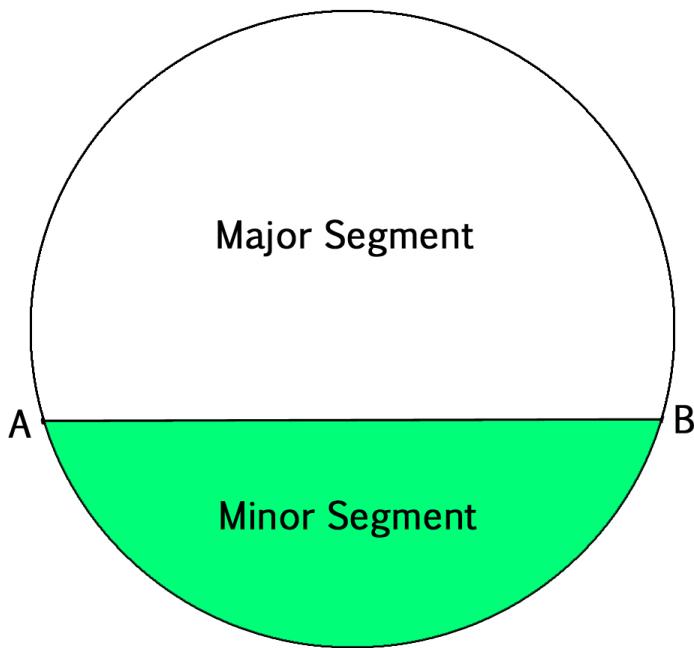
Chapter 192

Program to find area of a Circular Segment

Program to find area of a Circular Segment - GeeksforGeeks

In a circle, if a chord is drawn then that chord divides the whole circle into two parts. These two parts of the circle are called segments of the circle. The smaller area is known as the **Minor segment** and the larger area is called as the **Major segment**.

In the figure below, the chord AB divides the circle into minor and major segments.



We are given radius of circle and angle that forms minor segment. We need to find areas of two segments.

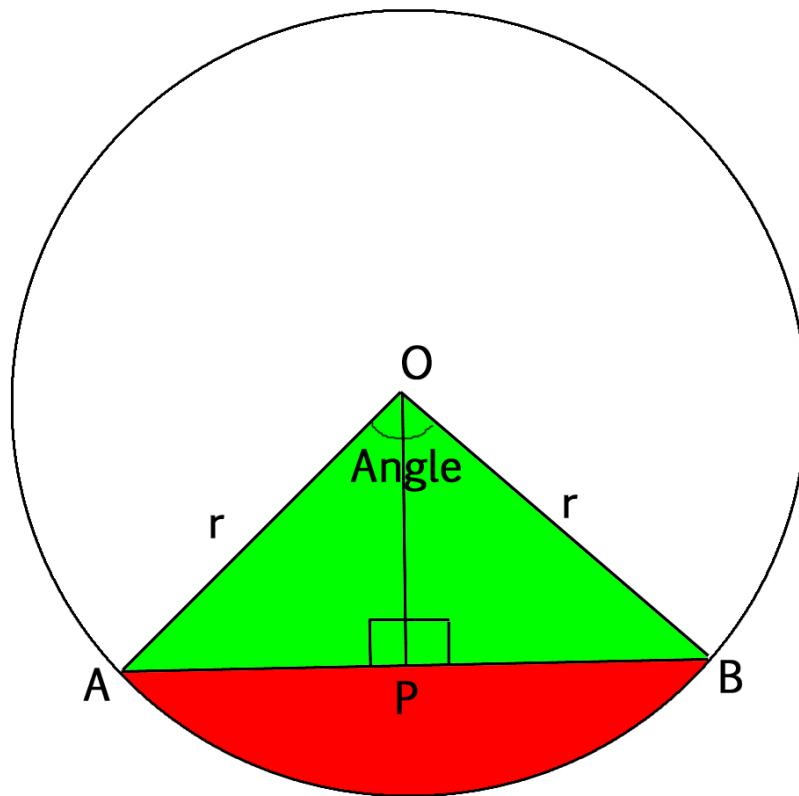
Examples :

Input :
radius = 21.0
angle = 120.0
Output :
Area of minor segment 270.855
Area of major segment 1114.59

Input :
radius = 10.0
angle = 90.0
Output :
Area of minor segment 28.5397
Area of major segment 285.619

Area of the segment :

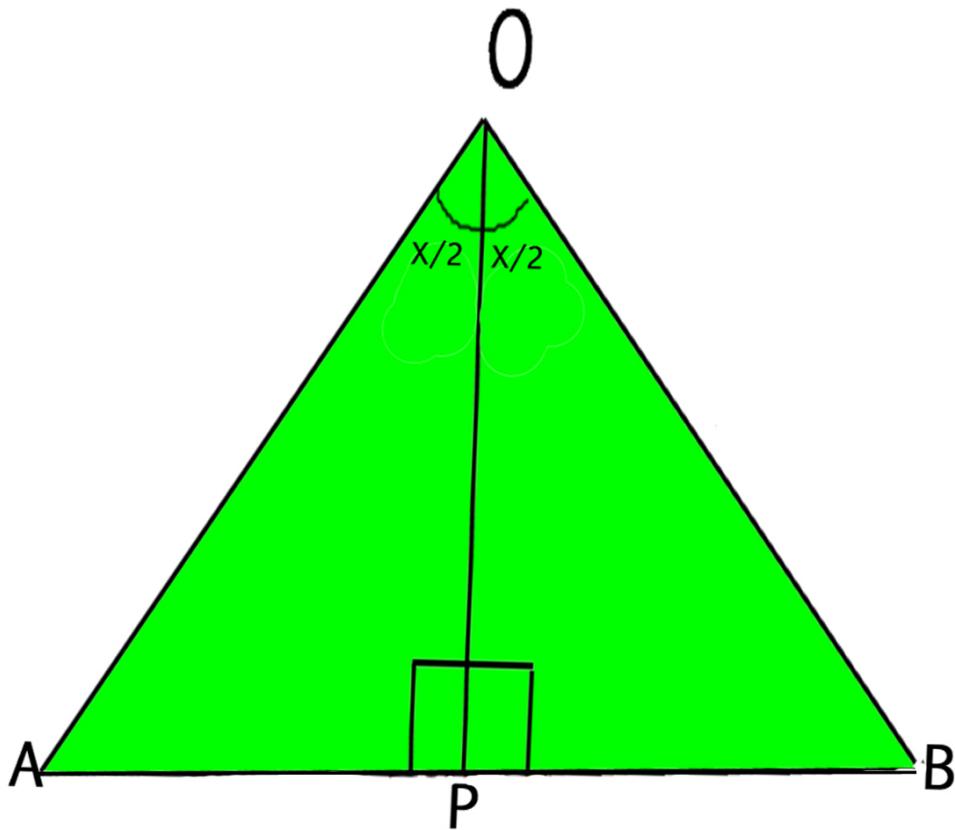
For that, we join the end points of the chord with the center of the circle resulting in a sector which subtends some ‘angle’ at the center. And a perpendicular is drawn from the center of the circle on the chord AB. By congruence of triangles, we obtain that the $\angle AOP = \angle BOP = 1/2(\text{angle})$.



Formula for Area of Segment :

$$\begin{aligned}\text{Area of Segment} &= \text{Area of sector} - \text{Area of Triangle OAB} \\ &= \pi * r^2 * (\text{angle}/360) - \\ &\quad \text{Area of Triangle OAB}\end{aligned}$$

For detailed information about formula of Area of Sector, refer <https://www.geeksforgeeks.org/area-of-a-sector/>.



In the figure above, assume angle made by sector = X ,
so $\angle AOP = \angle BOP = X/2$

$$\begin{aligned}\text{Area of Triangle } AOB &= 1/2 * \text{base} * \text{height} \\ &= 1/2 * AB * OP\end{aligned}$$

$$\begin{aligned}\text{Now in Triangle } AOP, \text{ By trigonometry} \\ \cos(X/2) &= OP/AO \quad \text{i.e. } OP = AO * \cos(X/2) \\ &\quad OP = r * \cos(X/2) \\ \sin(X/2) &= AP/AO \quad \text{i.e. } AP = AO * \sin(X/2) \\ &\quad AP = r * \sin(X/2)\end{aligned}$$

$$\begin{aligned}\text{So,} \\ \text{Base} &= AB = AP + PB \\ &= 2 * AP \\ &= 2 * r * \sin(X/2)\end{aligned}$$

```
Height = OP = r * Cos(X/2)

Area of triangle = 1/2 * (2 * r * Sin(X/2)) * (r * Cos(X/2))
                  = 1/2 * r2 * Sin(X)
                  [Using identity 2 * Sin(A) * Cos(A)]
                  = Sin(2 * A)
```

Hence Area of Segment = $\pi * r^2 * (\text{angle}/360) - 1/2 * r^2 * \text{Sin}(\text{angle})$

C++

```
// C++ Program to
// find area of
// segment of a
// circle
#include <bits/stdc++.h>
using namespace std;

float pi = 3.14159;

// Function to find
// area of segment
float area_of_segment(float radius,
                      float angle)
{
    // Calculating area of sector
    float area_of_sector = pi *
                           (radius * radius)
                           *(angle / 360);

    // Calculating area of triangle
    float area_of_triangle = (float)1 / 2 *
                             (radius * radius) *
                             sin((angle * pi) / 180);

    return area_of_sector - area_of_triangle;
}

// Driver Code
int main()
{
    float radius = 10.0, angle = 90.0;
    cout << "Area of minor segment = "
         << area_of_segment(radius, angle) << endl;

    cout << "Area of major segment = "
         << area_of_segment(radius, (360 - angle));
}
```

Java

```
// Java Program to find area of
// segment of a circle
class GFG {
    static float pi = 3.14159f;

    static float area_of_segment(float radius,
                                float angle)
    {
        // Calculating area of sector
        float area_of_sector = pi *
        (radius * radius) * (angle / 360);

        // Calculating area of triangle
        float area_of_triangle =
            (float)1 / 2 * (radius * radius) *
            (float)Math.sin((angle * pi) / 180);

        return area_of_sector - area_of_triangle;
    }

    // Driver Function
    public static void main(String[] args)
    {
        float radius = 10.0f, angle = 90.0f;
        System.out.println("Area of minor segment = " +
                           area_of_segment(radius, angle));

        System.out.println("Area of major segment = " +
                           area_of_segment(radius, (360 - angle)));
    }
}

// This code is contributed by Anant Agarwal.
```

Python

```
# Python3 Program
# to find area of
# segment of a
# circle
import math

pi = 3.14159

# Function to find
```

```
# area of segment
def area_of_segment(radius, angle):
    # Calculating area of sector
    area_of_sector = pi *
        (radius * radius)
        * (angle / 360)

    # Calculating area of triangle
    area_of_triangle = 1 / 2 *
        (radius * radius) *
        math.sin((angle * pi) / 180)

    return area_of_sector - area_of_triangle;

# Driver Code
radius = 10.0
angle = 90.0
print("Area of minor segment =", area_of_segment(radius, angle))
print("Area of major segment =", area_of_segment(radius, (360 - angle)))

# This code is contributed by Smitha Dinesh Semwal
```

C#

```
// C# Program to find area
// of segment of a circle
using System;

class GFG {
    static float pi = 3.14159f;

    static float area_of_segment(float radius,
                                float angle)
    {
        // Calculating area of sector
        float area_of_sector = pi * (radius * radius)
            * (angle / 360);

        // Calculating area of triangle
        float area_of_triangle =(float)1 / 2 * (radius * radius)
            *(float)Math.Sin((angle * pi) / 180);

        return area_of_sector - area_of_triangle;
    }
}
```

```
// Driver Function
public static void Main()
{
    float radius = 10.0f, angle = 90.0f;
    Console.WriteLine("Area of minor segment = " +
                      area_of_segment(radius, angle));

    Console.WriteLine("Area of major segment = " +
                      area_of_segment(radius, (360 - angle)));
}
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP Program to
// find area of
// segment of a
// circle

// Function to find
// area of segment
function area_of_segment($radius,
                        $angle)

{
    $pi = 3.14159;

    // Calculating area of sector
    $area_of_sector = $pi * ($radius * $radius) *
                     ($angle / 360);

    // Calculating area of triangle
    $area_of_triangle = 1 / 2 *   ($radius * $radius)
                       * sin(($angle * $pi) / 180);

    return $area_of_sector - $area_of_triangle;
}

// Driver Code
$radius = 10.0;
$angle = 90.0;
echo ("Area of minor segment = ");
echo( area_of_segment($radius, $angle));
echo("\n");
```

```
echo("Area of major segment = ");
echo(area_of_segment($radius, (360 - $angle)));
// This code is contributed by vt_m.
?>
```

Output :

```
Area of minor segment = 28.5397
Area of major segment = 285.619
```

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/program-find-area-circular-segment/>

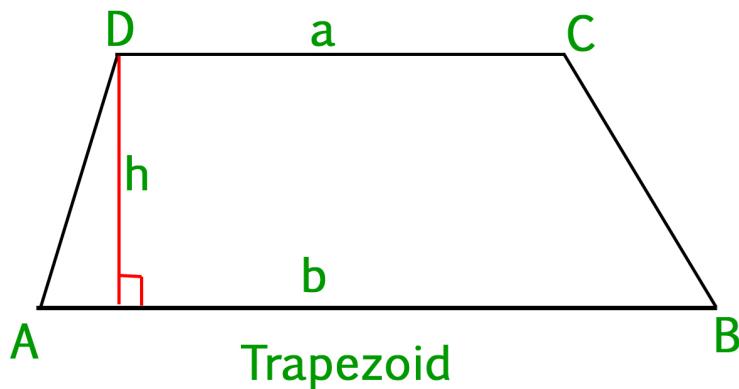
Chapter 193

Program to find area of a Trapezoid

Program to find area of a Trapezoid - GeeksforGeeks

Definition of Trapezoid :

A Trapezoid is a convex quadrilateral with at least one pair of parallel sides. The parallel sides are called the bases of the trapezoid and the other two sides which are not parallel are referred to as the legs. There can also be two pairs of bases.



In the above figure $CD \parallel AB$, so they form the bases and the other two sides i.e., AD and BC form the legs.

The area of a trapezoid can be found by using this simple formula :

$a = \text{base}$

$b = \text{base}$

$h = \text{height}$

Examples :

```
Input : base1 = 8, base2 = 10, height = 6
Output : Area is: 54.0
```

```
Input :base1 = 4, base2 = 20, height = 7
Output :Area is: 84.0
```

C

```
// CPP program to calculate
// area of a trapezoid
#include <stdio.h>

// Function for the area
double Area(int b1, int b2,
            int h)
{
    return ((b1 + b2) / 2) * h;
}

// Driver Code
int main()
{
    int base1 = 8, base2 = 10,
        height = 6;
    double area = Area(base1, base2,
                       height);
    printf("Area is: %.1lf", area);
    return 0;
}
```

Java

```
// Java program to calculate
// area of a trapezoid
import java.io.*;

class GFG
{

    // Function for the area
    static double Area(int b1,
                      int b2,
                      int h)
    {
        return ((b1 + b2) / 2) * h;
    }
}
```

```
// Driver Code
public static void main (String[] args)
{
    int base1 = 8, base2 = 10,
        height = 6;
    double area = Area(base1, base2,
                       height);
    System.out.println("Area is: " + area);
}
}
```

Python3

```
# Python program to calculate
# area of a trapezoid

# Function for the area
def Area(b1, b2, h):
    return ((b1 + b2) / 2) * h

# Driver Code
base1 = 8; base2 = 10; height = 6
area = Area(base1, base2, height)
print("Area is:", area)
```

C#

```
// C# program to calculate
// area of a trapezoid
using System;

class GFG
{

    // Function for the area
    static double Area(int b1,
                      int b2,
                      int h)
    {
        return ((b1 + b2) / 2) * h;
    }

    // Driver Code
    public static void Main ()
    {
        int base1 = 8, base2 = 10,
```

```
        height = 6;
    double area = Area(base1, base2,
                        height);
    Console.WriteLine("Area is: " + area);
}
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP program to calculate
// area of a trapezoid

// Function for the area
function Area( $b1, $b2, $h)
{
    return (( $b1 + $b2 ) / 2) * $h;
}

// Driver Code

$base1 = 8; $base2 = 10;
$height = 6;
$area = Area($base1, $base2, $height);
echo("Area is: ");
echo($area);

// This code is contributed by vt_m.
?>
```

Output :

Area is: 54.0

Improved By : [vt_m](#)

Source

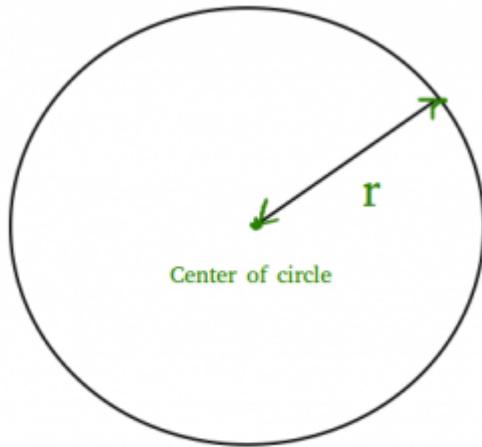
<https://www.geeksforgeeks.org/program-find-area-trapezoid/>

Chapter 194

Program to find area of a circle

Program to find area of a circle - GeeksforGeeks

Area of a circle can simply be evaluated using following formula.



$$\text{Area} = \text{PI} * r * r$$

Area = pi * r²
where r is radius of circle

C

```
#include <stdio.h>
#include <math.h>
#define PI 3.142
```

```
double findArea(int r)
{
    return PI * pow(r, 2);
}

int main()
{
    printf("Area is %f", findArea(5));
    return 0;
}
```

Java

```
// Java program to find area
// of circle

class Test
{
    static final double PI = Math.PI;

    static double findArea(int r)
    {
        return PI * Math.pow(r, 2);
    }

    // Driver method
    public static void main(String[] args)
    {
        System.out.println("Area is " + findArea(5));
    }
}
```

Python3

```
# Python program to find Area of a circle

def findArea(r):
    PI = 3.142
    return PI * (r*r);

# Driver method
print("Area is %.6f" % findArea(5));

# This code is contributed by Chinmoy Lenka
```

C#

```
// C# program to find area of circle
using System;

class GFG
{
    static double PI = Math.PI;

    static double findArea(int r)
    {
        return PI * Math.Pow(r, 2);
    }

    // Driver method
    static void Main()
    {
        Console.WriteLine("Area is " + findArea(5));
    }
}

// This code is contributed by Sam007.
```

PHP

```
<?php
// PHP program to find area
// of circle

function findArea( $r )
{
    $PI = 3.142;
    return $PI * pow($r, 2);
}

// Driver Code
echo("Area is ");
echo(findArea(5));
return 0;

// This code is contributed by vt_m.
?>
```

Output:

Area is 78.550000

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/c-program-find-area-circle/>

Chapter 195

Program to find area of a triangle

Program to find area of a triangle - GeeksforGeeks

Finding area using given sides:

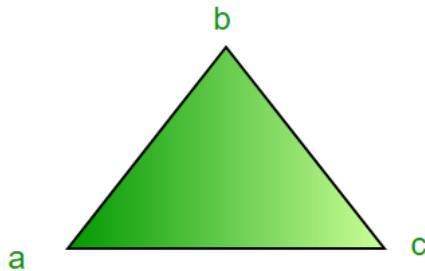
Examples :

```
Input : a = 5, b = 7, c = 8
Output : Area of a triangle is 17.320508
```

```
Input : a = 3, b = 4, c = 5
Output : Area of a triangle is 6.000000
```

Area of a [triangle](#) can simply be evaluated using following formula.

```
Area = sqrt(s*(s-a)*(s-b)*(s-c))
where a, b and c are lengths of sides of
triangle and s = (a+b+c)/2
```



$$\text{Area of triangle} = \sqrt{s(s-a)(s-b)(s-c)}$$

$$s = \frac{a+b+c}{2}$$

C++

```
#include <stdio.h>
#include <stdlib.h>

float findArea(int a, int b, int c)
{
    // Length of sides must be positive and sum of any two sides
    // must be smaller than third side.
    if (a < 0 || b < 0 || c < 0 || (a+b <= c) ||
        a+c <= b || b+c <= a)
    {
        printf("Not a valid triangle");
        exit(0);
    }
    int s = (a+b+c)/2;
    return sqrt(s*(s-a)*(s-b)*(s-c));
}

int main()
{
    int a = 3;
    int b = 4;
    int c = 5;

    printf("Area is %f", findArea(a, b, c));
    return 0;
}
```

Java

```
// Java program to print
```

```
// Floyd's triangle

class Test
{
    static float findArea(int a, int b, int c)
    {
        // Length of sides must be positive and sum of any two sides
        // must be smaller than third side.
        if (a < 0 || b < 0 || c < 0 || (a+b <= c) ||
            a+c <=b || b+c <=a)
        {
            System.out.println("Not a valid triangle");
            System.exit(0);
        }
        int s = (a+b+c)/2;
        return (float)Math.sqrt(s*(s-a)*(s-b)*(s-c));
    }

    // Driver method
    public static void main(String[] args)
    {
        int a = 3;
        int b = 4;
        int c = 5;

        System.out.println("Area is " + findArea(a, b, c));
    }
}
```

Python

```
# Python Program to find the area
# of triangle

# Length of sides must be positive
# and sum of any two sides
def findArea(a,b,c):

    # must be smaller than third side.
    if (a < 0 or b < 0 or c < 0 or (a+b <= c) or (a+c <=b) or (b+c <=a) ):
        print('Not a valid triangle')
        return

    # calculate the semi-perimeter
    s = (a + b + c) / 2

    # calculate the area
    area = (s * (s - a) * (s - b) * (s - c)) ** 0.5
```

```
print('Area of a traingle is %f' %area)

# Initialize first side of traingle
a = 3
# Initialize second side of traingle
b = 4
# Initialize Third side of traingle
c = 5
findArea(a,b,c)

# This code is contributed by Shariq Raza
```

C#

```
// C# program to print
// Floyd's triangle
using System;

class Test {

    // Function to find area
    static float findArea(int a, int b,
                          int c)
    {

        // Length of sides must be positive
        // and sum of any two sides
        // must be smaller than third side.
        if (a < 0 || b < 0 || c < 0 ||
            (a + b <= c) || a + c <=b ||
            b + c <=a)
        {
            Console.WriteLine("Not a valid triangle");
            System.Environment.Exit(0);
        }
        int s = (a + b + c) / 2;
        return (float)Math.Sqrt(s * (s - a) *
                               (s - b) * (s - c));
    }

    // Driver code
    public static void Main()
    {
        int a = 3;
        int b = 4;
        int c = 5;
```

```
        Console.WriteLine("Area is " + findArea(a, b, c));
    }
}

// This code is contributed Nitin Mittal.
```

PHP

```
<?php
function findArea($a, $b, $c)
{
    // Length of sides must be positive
    // and sum of any two sides must
    // be smaller than third side.
    if ($a < 0 or $b < 0 or
        $c < 0 or ($a + $b <= $c) or
        $a + $c <= $b or $b + $c <= $a)
    {
        echo "Not a valid trianglen";
        exit(0);
    }
    $s = ($a + $b + $c) / 2;
    return sqrt($s * ($s - $a) *
                ($s - $b) * ($s - $c));
}

// Driver Code
$a = 3;
$b = 4;
$c = 5;

echo "Area is ", findArea($a, $b, $c);

// This code is contributed anuJ_67.
?>
```

Output :

```
Area is 6
```

Finding area using coordinates:

If we are given coordinates of three corners, we can apply below [Shoelace formula](#) for area.

Area

$$= | \frac{1}{2} [(x_1y_2 + x_2y_3 + \dots + x_{n-1}y_n + x_ny_1) - (x_2y_1 + x_3y_2 + \dots + x_ny_{n-1} + x_1y_n)] |$$

CPP

```
// C++ program to evaluate area of a polygon using
// shoelace formula
#include <bits/stdc++.h>
using namespace std;

// (X[i], Y[i]) are coordinates of i'th point.
double polygonArea(double X[], double Y[], int n)
{
    // Initialize area
    double area = 0.0;

    // Calculate value of shoelace formula
    int j = n - 1;
    for (int i = 0; i < n; i++)
    {
        area += (X[j] + X[i]) * (Y[j] - Y[i]);
        j = i; // j is previous vertex to i
    }

    // Return absolute value
    return abs(area / 2.0);
}

// Driver program to test above function
int main()
{
    double X[] = {0, 2, 4};
    double Y[] = {1, 3, 7};

    int n = sizeof(X)/sizeof(X[0]);

    cout << polygonArea(X, Y, n);
}
```

Java

```
// Java program to evaluate area of
// a polygon using shoelace formula
import java.io.*;
import java.math.*;

class GFG {

    // (X[i], Y[i]) are coordinates of i'th point.
    static double polygonArea(double X[], double Y[], int n)
```

```
{  
    // Initialize area  
    double area = 0.0;  
  
    // Calculate value of shoelace formula  
    int j = n - 1;  
    for (int i = 0; i < n; i++)  
    {  
        area += (X[j] + X[i]) * (Y[j] - Y[i]);  
  
        // j is previous vertex to i  
        j = i;  
    }  
  
    // Return absolute value  
    return Math.abs(area / 2.0);  
}  
  
// Driver program  
public static void main (String[] args)  
{  
    double X[] = {0, 2, 4};  
    double Y[] = {1, 3, 7};  
  
    int n = X.length;  
    System.out.println(polygonArea(X, Y, n));  
}  
}  
  
// This code is contributed  
// by Nikita Tiwari.
```

Python3

```
# Python 3 program to evaluate  
# area of a polygon using  
# shoelace formula  
  
# (X[i], Y[i]) are coordinates of i'th point.  
def polygonArea(X,Y, n) :  
  
    # Initialize area  
    area = 0.0  
  
    # Calculate value of shoelace formula  
    j = n - 1  
    for i in range( 0, n) :
```

```
area = area + (X[j] + X[i]) * (Y[j] - Y[i])
j = i  # j is previous vertex to i

# Return absolute value
return abs(area // 2.0)

# Driver program to test above function
X = [0, 2, 4]
Y = [1, 3, 7]

n = len(X)
print(polygonArea(X, Y, n))

# This code is contributed
# by Nikita Tiwari.
```

C#

```
// C# program to evaluate area of
// a polygon using shoelace formula
using System;

class GFG {

    // (X[i], Y[i]) are coordinates
    // of i'th point.
    static double polygonArea(double []X,
                               double []Y, int n)
    {
        // Initialize area
        double area = 0.0;

        // Calculate value of shoelace
        // formula
        int j = n - 1;
        for (int i = 0; i < n; i++)
        {
            area += (X[j] + X[i]) *
                    (Y[j] - Y[i]);

            // j is previous vertex to i
            j = i;
        }

        // Return absolute value
        return Math.Abs(area / 2);
    }
}
```

```
        return Math.Abs(area / 2.0);
    }

    // Driver program
    public static void Main ()
    {
        double []X = {0, 2, 4};
        double []Y = {1, 3, 7};

        int n = X.Length;
        Console.WriteLine(
            polygonArea(X, Y, n));
    }
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP program to evaluate area of a
// polygon using shoelace formula

// (X[i], Y[i]) are coordinates
// of i'th point.
function polygonArea( $X, $Y, $n)
{

    // Initialize area
    $area = 0.0;

    // Calculate value of
    // shoelace formula
    $j = $n - 1;
    for ( $i = 0; $i < $n; $i++)
    {
        $area += ($X[$j] + $X[$i]) *
            ($Y[$j] - $Y[$i]);

        // j is previous vertex to i
        $j = $i;
    }

    // Return absolute value
    return abs($area / 2.0);
}

// Driver Code
```

```
$X = array(0, 2, 4);
$Y = array(1, 3, 7);
$n = count($X);
echo polygonArea($X, $Y, $n);

// This code is contributed by anuj_67.
?>
```

Output :

2

Improved By : [nitin mittal](#), [vt_m](#)

Source

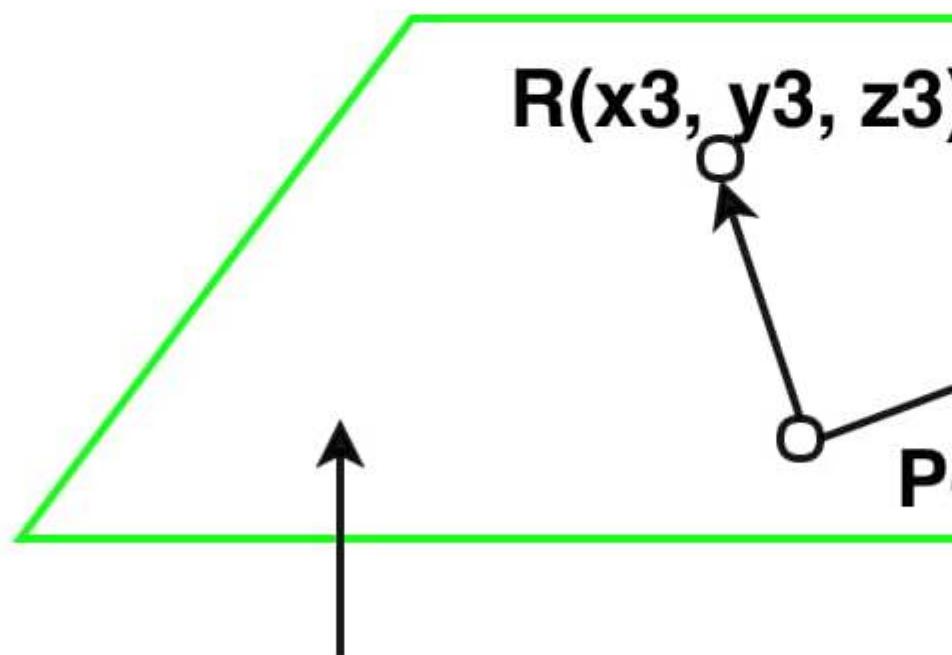
<https://www.geeksforgeeks.org/c-program-find-area-triangle/>

Chapter 196

Program to find equation of a plane passing through 3 points

Program to find equation of a plane passing through 3 points - GeeksforGeeks

Given three points (x_1, y_1, z_1) , (x_2, y_2, z_2) , (x_3, y_3, z_3) . The task is to find the equation of the plane passing through these 3 points.



$$a^*x + b^*y + c^*z + d = 0$$

Examples:

Input: $x_1 = -1$ $y_1 = w$ $z_1 = 1$

$x_2 = 0$ $y_2 = -3$ $z_2 = 2$

$x_3 = 1$ $y_3 = 1$ $z_3 = -4$

Output: equation of plane is $26x + 7y + 9z + 3 = 0$.

Input: $x_1 = 2$, $y_1 = 1$, $z_1 = -1$, 1

$x_2 = 0$, $y_2 = -2$, $z_2 = 0$

$x_3 = 1$, $y_3 = -1$, $z_3 = 2$

Output: equation of plane is $-7x + 5y + 1z + 10 = 0$.

Approach: Let P, Q and R be the three points with coordinates (x_1, y_1, z_1) , (x_2, y_2, z_2) , (x_3, y_3, z_3) respectively. Then the equation of plane is $a * (x - x_0) + b * (y - y_0) + c * (z - z_0) = 0$, where a, b, c are direction ratios of normal to the plane and (x_0, y_0, z_0) are co-ordinates of any point(i.e P, Q, or R) passing through the plane. For finding direction ratios of normal to the plane, take any two vectors in plane, let it be vector PQ, vector PR.

$$\Rightarrow \text{Vector } PQ = (x_2 - x_1, y_2 - y_1, z_2 - z_1) = (a_1, b_1, c_1).$$

$$\Rightarrow \text{Vector } PR = (x_3 - x_1, y_3 - y_1, z_3 - z_1) = (a_2, b_2, c_2).$$

Normal vector to this plane will be vector PQ x vector PR.

$$\begin{aligned} \Rightarrow PQ \times PR &= (b_1 * c_2 - b_2 * c_1) i \\ &\quad + (a_2 * c_1 - a_1 * c_2) j \\ &\quad + (a_1 * b_2 - a_2 * b_1) k = ai + bj + ck. \end{aligned}$$

Direction ratios of normal vector will be a, b, c. Taking any one point from P, Q, or R, let its co-ordinate be (x_0, y_0, z_0) . Then the equation of plane passing through a point (x_0, y_0, z_0) and having direction ratios a, b, c will be

$$\Rightarrow a * (x - x_0) + b * (y - y_0) + c * (z - z_0) = 0.$$

$$\Rightarrow a * x - a * x_0 + b * y - b * y_0 + c * z - c * z_0 = 0.$$

$$\Rightarrow a * x + b * y + c * z + (-a * x_0 - b * y_0 - c * z_0) = 0.$$

Below is the implementation of the above approach:

C

```
// C program to find equation of a plane
// passing through given 3 points.
```

```
#include<stdio.h>
```

```
// Function to find equation of plane.
void equation_plane(float x1, float y1, float z1, float x2,
                     float y2, float z2, float x3, float y3, float z3)
{
    float a1 = x2 - x1;
    float b1 = y2 - y1;
    float c1 = z2 - z1;
    float a2 = x3 - x1;
    float b2 = y3 - y1;
    float c2 = z3 - z1;
    float a = b1 * c2 - b2 * c1;
    float b = a2 * c1 - a1 * c2;
    float c = a1 * b2 - b1 * a2;
    float d = (- a * x1 - b * y1 - c * z1);
    printf("equation of plane is %.2f x + %.2f"
          " y + %.2f z + %.2f = 0.", a, b, c, d);
    return;
}

// Driver Code
int main()
{
    float x1 = -1;
    float y1 = 2;
    float z1 = 1;
    float x2 = 0;
    float y2 = -3;
    float z2 = 2;
    float x3 = 1;
    float y3 = 1;
    float z3 = -4;
    equation_plane(x1, y1, z1, x2, y2, z2, x3, y3, z3);
    return 0;
}
// This code is contributed
// by Amber_Saxena.
```

Java

```
// Java program to find equation
// of a plane passing through
// given 3 points.
import java .io.*;

class GFG
{
```

```
// Function to find equation of plane.
static void equation_plane(float x1, float y1,
                           float z1, float x2,
                           float y2, float z2,
                           float x3, float y3,
                           float z3)
{
    float a1 = x2 - x1;
    float b1 = y2 - y1;
    float c1 = z2 - z1;
    float a2 = x3 - x1;
    float b2 = y3 - y1;
    float c2 = z3 - z1;
    float a = b1 * c2 - b2 * c1;
    float b = a2 * c1 - a1 * c2;
    float c = a1 * b2 - b1 * a2;
    float d = (- a * x1 - b * y1 - c * z1);
    System.out.println("equation of plane is " + a +
                        " x + " + b + " y + " + c +
                        " z + " + d + " = 0.");
}

// Driver code
public static void main(String[] args)
{
    float x1 = -1;
    float y1 = 2;
    float z1 = 1;
    float x2 = 0;
    float y2 = -3;
    float z2 = 2;
    float x3 = 1;
    float y3 = 1;
    float z3 = -4;
    equation_plane(x1, y1, z1, x2,
                   y2, z2, x3, y3, z3);
}
}

// This code is contributed
// by Amber_Saxena.
```

Python

```
# Python program to find equation of a plane
# passing through given 3 points.

# Function to find equation of plane.
```

```
def equation_plane(x1, y1, z1, x2, y2, z2, x3, y3, z3):  
  
    a1 = x2 - x1  
    b1 = y2 - y1  
    c1 = z2 - z1  
    a2 = x3 - x1  
    b2 = y3 - y1  
    c2 = z3 - z1  
    a = b1 * c2 - b2 * c1  
    b = a2 * c1 - a1 * c2  
    c = a1 * b2 - b1 * a2  
    d = (- a * x1 - b * y1 - c * z1)  
    print "equation of plane is ",  
    print a, "x +",  
    print b, "y +",  
    print c, "z +",  
    print d, "= 0."  
  
# Driver Code  
x1 = -1  
y1 = 2  
z1 = 1  
x2 = 0  
y2 = -3  
z2 = 2  
x3 = 1  
y3 = 1  
z3 = -4  
equation_plane(x1, y1, z1, x2, y2, z2, x3, y3, z3)
```

C#

```
// C# program to find equation  
// of a plane passing through  
// given 3 points.  
using System;  
  
class GFG  
{  
  
    // Function to find equation of plane.  
    static void equation_plane(float x1, float y1,  
    float z1, float x2,  
    float y2, float z2,  
    float x3, float y3,  
    float z3)  
    {  
        float a1 = x2 - x1;  
        float b1 = y2 - y1;
```

```
float c1 = z2 - z1;
float a2 = x3 - x1;
float b2 = y3 - y1;
float c2 = z3 - z1;
float a = b1 * c2 - b2 * c1;
float b = a2 * c1 - a1 * c2;
float c = a1 * b2 - b1 * a2;
float d = (- a * x1 - b * y1 - c * z1);
Console.WriteLine("equation of plane is " + a +
"x + " + b + "y + " + c +
"z + " + d + " = 0 ");
}

// Driver code
public static void Main()
{
    float x1 = -1;
    float y1 = 2;
    float z1 = 1;
    float x2 = 0;
    float y2 = -3;
    float z2 = 2;
    float x3 = 1;
    float y3 = 1;
    float z3 = -4;
    equation_plane(x1, y1, z1,
x2, y2, z2,
x3, y3, z3);
}
}

// This code is contributed
// by ChitraNayal
```

PHP

```
<?php
// PHP program to find equation
// of a plane passing through
// given 3 points.

// Function to find equation of plane.
function equation_plane($x1, $y1, $z1,
                      $x2, $y2, $z2,
                      $x3, $y3, $z3)
{
    $a1 = $x2 - $x1;
    $b1 = $y2 - $y1;
    $c1 = $z2 - $z1;
```

```
$a2 = $x3 - $x1;
$b2 = $y3 - $y1;
$c2 = $z3 - $z1;
$a = $b1 * $c2 - $b2 * $c1;
$b = $a2 * $c1 - $a1 * $c2;
$c = $a1 * $b2 - $b1 * $a2;
$d = (- $a * $x1 - $b * $y1 - $c * $z1);
echo sprintf("equation of the plane is %.2fx" .
    " + %.2fy + %.2fz + %.2f = 0",
    $a, $b, $c, $d);
}

// Driver Code
$x1 = -1;
$y1 = 2;
$z1 = 1;
$x2 = 0;
$y2 = -3;
$z2 = 2;
$x3 = 1;
$y3 = 1;
$z3 = -4;
equation_plane($x1, $y1, $z1, $x2,
    $y2, $z2, $x3, $y3, $z3);

// This code is contributed
// by Amber_Saxena.
?>
```

Output:

```
equation of plane is 26 x + 7 y + 9 z + 3 = 0.
```

Improved By : [Amber_Saxena](#), [ChitraNayal](#)

Source

<https://www.geeksforgeeks.org/program-to-find-equation-of-a-plane-passing-through-3-points/>

Chapter 197

Program to find line passing through 2 Points

Program to find line passing through 2 Points - GeeksforGeeks

Given two points P and Q in the coordinate plane, find the equation of the line passing through both the points.

This kind of conversion is very useful in many geometric algorithms like intersection of lines, finding the [circumcenter](#) of a triangle, finding the [incenter](#) of a triangle and many more...

Examples:

```
Input : P(3, 2)
        Q(2, 6)
Output : 4x + 1y = 14
```

```
Input : P(0, 1)
        Q(2, 4)
Output : 3x + -2y = -2
```

Let the given two points be $P(x_1, y_1)$ and $Q(x_2, y_2)$. Now, we find the equation of line formed by these points.

Any line can be represented as,

$$ax + by = c$$

Let the two points satisfy the given line. So, we have,

$$ax_1 + by_1 = c$$

$$ax_2 + by_2 = c$$

We can set the following values so that all the equations hold true,

$$a = y_2 - y_1$$

$$\begin{aligned} b &= x_1 - x_2 \\ c &= ax_1 + by_1 \end{aligned}$$

These can be derived by first getting the slope directly and then finding the intercept of the line. OR these can also be derived cleverly by a simple observation as under:

Derivation :

```

ax1 + by1 = c ... (i)
ax2 + by2 = c ... (ii)
Equating (i) and (ii),
ax1 + by1 = ax2 + by2
=> a(x1 - x2) = b(y2 - y1)
Thus, for equating LHS and RHS, we can simply have,
a = (y2 - y1)
AND
b = (x1 - x2)
so that we have,
(y2 - y1)(x1 - x2) = (x1 - x2)(y2 - y1)
AND
Putting these values in (i), we get,
c = ax1 + by1

```

Thus, we now have the values of a, b and c which means that we have the line in the coordinate plane.

```

// C++ Implementation to find the line passing
// through two points
#include <iostream>
using namespace std;

// This pair is used to store the X and Y
// coordinate of a point respectively
#define pdd pair<double, double>

// Function to find the line given two points
void lineFromPoints(pdd P, pdd Q)
{
    double a = Q.second - P.second;
    double b = P.first - Q.first;
    double c = a*(P.first) + b*(P.second);

    if(b<0)
    {
        cout << "The line passing through points P and Q is: "
            << a << "x " << b << "y = " << c << endl;
    }
}

```

```
else
{
    cout << "The line passing through points P and Q is: "
        << a << "x + " << b << "y = " << c << endl;
}
}

// Driver code
int main()
{
    pdd P = make_pair(3, 2);
    pdd Q = make_pair(2, 6);
    lineFromPoints(P, Q);
    return 0;
}
```

Output:

The line passing through points P and Q is: 4x + 1y = 14

Source

<https://www.geeksforgeeks.org/program-find-line-passing-2-points/>

Chapter 198

Program to find slope of a line

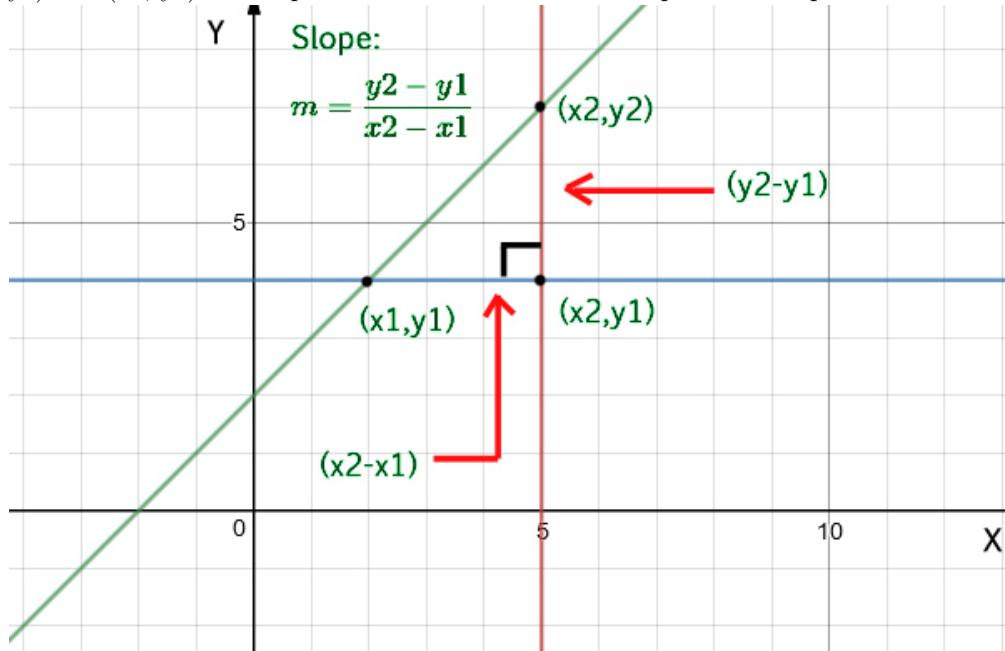
Program to find slope of a line - GeeksforGeeks

Given two co-ordinates, find the [slope](#) of a straight line.

Examples:

Input : $x_1 = 4, y_1 = 2,$
 $x_2 = 2, y_2 = 5$
Output : Slope is -1.5

Approach: To calculate the slope of a line you need only two points from that line, (x_1, y_1) and (x_2, y_2) . The equation used to calculate the slope from two points is:



Below is the implementation of the above approach:

c++

```
// C program for slope of line
#include <bits/stdc++.h>
using namespace std;

// function to find the slope of a straight line
float slope(float x1, float y1, float x2, float y2)
{
    return (y2 - y1) / (x2 - x1);
}

// driver code to check the above function
int main()
{
    float x1 = 4, y1 = 2;
    float x2 = 2, y2 = 5;
    cout << "Slope is: "
         << slope(x1, y1, x2, y2);
    return 0;
}
```

Java

```
// Java program for slope of line
import java.io.*;

class GFG {
    static float slope(float x1, float y1,
                      float x2, float y2)
    {
        return (y2 - y1) / (x2 - x1);
    }
    public static void main(String[] args)
    {
        float x1 = 4, y1 = 2;
        float x2 = 2, y2 = 5;
        System.out.println("Slope is " +
                           slope(x1, y1, x2, y2));
    }
}
```

Python

```
# Python program for slope of line
```

```
def slope(x1, y1, x2, y2):
    return (float)(y2-y1)/(x2-x1)

# driver code
x1 = 4
y1 = 2
x2 = 2
y2 = 5
print "Slope is :", slope(x1, y1, x2, y2)
```

C#

```
// C# program for slope of line
using System;

class GFG
{
    static float slope(float x1, float y1,
                       float x2, float y2)
    {
        return (y2 - y1) / (x2 - x1);
    }

    // Driver code
    public static void Main()
    {
        float x1 = 4, y1 = 2;
        float x2 = 2, y2 = 5;
        Console.WriteLine("Slope is " +
                          slope(x1, y1, x2, y2));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program for
// slope of line

// function to find the
// slope of a straight line
function slope($x1, $y1, $x2, $y2)
{
    return ($y2 - $y1) /
           ($x2 - $x1);
```

```
}

// Driver Code
$x1 = 4;
$y1 = 2;
$x2 = 2;
$y2 = 5;
echo "Slope is: "
    , slope($x1, $y1,
            $x2, $y2);

// This code is contributed by anuj_67.
?>
```

Output:

Slope is: -1.5

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/program-find-slope-line/>

Chapter 199

Program to find smallest difference of angles of two parts of a given circle

Program to find smallest difference of angles of two parts of a given circle - GeeksforGeeks

Given a division of circle into n pieces as an array of size n. The i-th element of the array denotes the angle of one piece. Our task is to make two continuous parts from these pieces so that the difference between angles of these two parts is minimum.

Examples :

Input : arr[] = {90, 90, 90, 90}
Output : 0
In this example, we can take 1 and 2 pieces and 3 and 4 pieces. Then the answer is $|90 + 90| - |90 + 90| = 0$.

Input : arr[] = {170, 30, 150, 10}
Output : 0
In this example, we can take 1 and 4, and 2 and 3 pieces. So the answer is $|170 + 10| - |30 + 150| = 0$.

Input : arr[] = {100, 100, 160}
Output : 40

We can notice that if one of the part is continuous then all the remaining pieces also form a continuous part. If angle of the first part is equal to x then difference between angles of first and second parts is $|x - (360 - x)| = |2 * x - 360| = 2 * |x - 180|$. So for each possible

continuous part we can count it's angle and update answer.

C++

```
// CPP program to find minimum
// difference of angles of two
// parts of given circle.
#include <bits/stdc++.h>
using namespace std;

// Returns the minimum difference
// of angles.
int findMinimumAngle(int arr[], int n)
{
    int l = 0, sum = 0, ans = 360;
    for (int i = 0; i < n; i++) {
        // sum of array
        sum += arr[i];

        while (sum >= 180) {

            // calculating the difference of
            // angles and take minimum of
            // previous and newly calculated
            ans = min(ans, 2 * abs(180 - sum));
            sum -= arr[l];
            l++;
        }
        ans = min(ans, 2 * abs(180 - sum));
    }
    return ans;
}

// driver code
int main()
{
    int arr[] = { 100, 100, 160 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findMinimumAngle(arr, n) << endl;
    return 0;
}

// This code is contributed by "Abhishek Sharma 44"
```

Java

```
// java program to find minimum
```

```
// difference of angles of two
// parts of given circle.
import java.util.*;

class Count{
    public static int findMinimumAngle(int arr[], int n)
    {
        int l = 0, sum = 0, ans = 360;
        for (int i = 0; i < n; i++)
        {
            // sum of array
            sum += arr[i];

            while (sum >= 180)
            {

                // calculating the difference of
                // angles and take minimum of
                // previous and newly calculated
                ans = Math.min(ans,
                               2 * Math.abs(180 - sum));
                sum -= arr[l];
                l++;
            }

            ans = Math.min(ans,
                           2 * Math.abs(180 - sum));
        }

        return ans;
    }

    public static void main(String[] args)
    {
        int arr[] = { 100, 100, 160 };
        int n = 3;
        System.out.print(findMinimumAngle(arr, n));
    }
}

// This code is contributed by rishabh_jain
```

Python3

```
# java program to find minimum
# difference of angles of two
# parts of given circle.
```

```
import math

# function returns the minimum
# difference of angles.
def findMinimumAngle (arr, n):
    l = 0
    _sum = 0
    ans = 360
    for i in range(n):

        #sum of array
        _sum += arr[i]

        while _sum >= 180:

            # calculating the difference of
            # angles and take minimum of
            # previous and newly calculated
            ans = min(ans, 2 * abs(180 - _sum))
            _sum -= arr[l]
            l+=1
        ans = min(ans, 2 * abs(180 - _sum))
    return ans

# driver code
arr = [100, 100, 160]
n = len(arr)
print(findMinimumAngle (arr, n))

# This code is contributed by "Abhishek Sharma 44"
```

C#

```
// C# program to find minimum
// difference of angles of two
// parts of given circle.
using System;

class GFG
{
    public static int findMinimumAngle(int []arr, int n)
    {
        int l = 0, sum = 0, ans = 360;
        for (int i = 0; i < n; i++)
        {
            // sum of array
            sum += arr[i];
```

```
        while (sum >= 180)
        {

            // calculating the difference of
            // angles and take minimum of
            // previous and newly calculated
            ans = Math.Min(ans,
                2 * Math.Abs(180 - sum));
            sum -= arr[1];
            l++;
        }

        ans = Math.Min(ans,
            2 * Math.Abs(180 - sum));
    }

    return ans;
}

// Driver code
public static void Main()
{
    int []arr = { 100, 100, 160 };
    int n = 3;
    Console.WriteLine(findMinimumAngle(arr, n));
}
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP program to find minimum
// difference of angles of two
// parts of given circle.

// Returns the minimum difference
// of angles.
function findMinimumAngle($arr, $n)
{
    $l = 0; $sum = 0; $ans = 360;
    for ($i = 0; $i < $n; $i++)
    {
        // sum of array
        $sum += $arr[$i];
```

```
while ($sum >= 180)
{
    // calculating the difference of
    // angles and take minimum of
    // previous and newly calculated
    $ans = min($ans, 2 *
               abs(180 - $sum));
    $sum -= $arr[$l];
    $l++;
}

$ans = min($ans, 2 * abs(180 - $sum));
}
return $ans;
}

// Driver Code
$arr = array( 100, 100, 160 );
$n = sizeof($arr);
echo findMinimumAngle($arr, $n), "\n" ;

// This code is contributed by m_kit
?>
```

Output :

40

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/program-find-smallest-difference-angles-two-parts-given-circle/>

Chapter 200

Program to find the Volume of a Triangular Prism

Program to find the Volume of a Triangular Prism - GeeksforGeeks

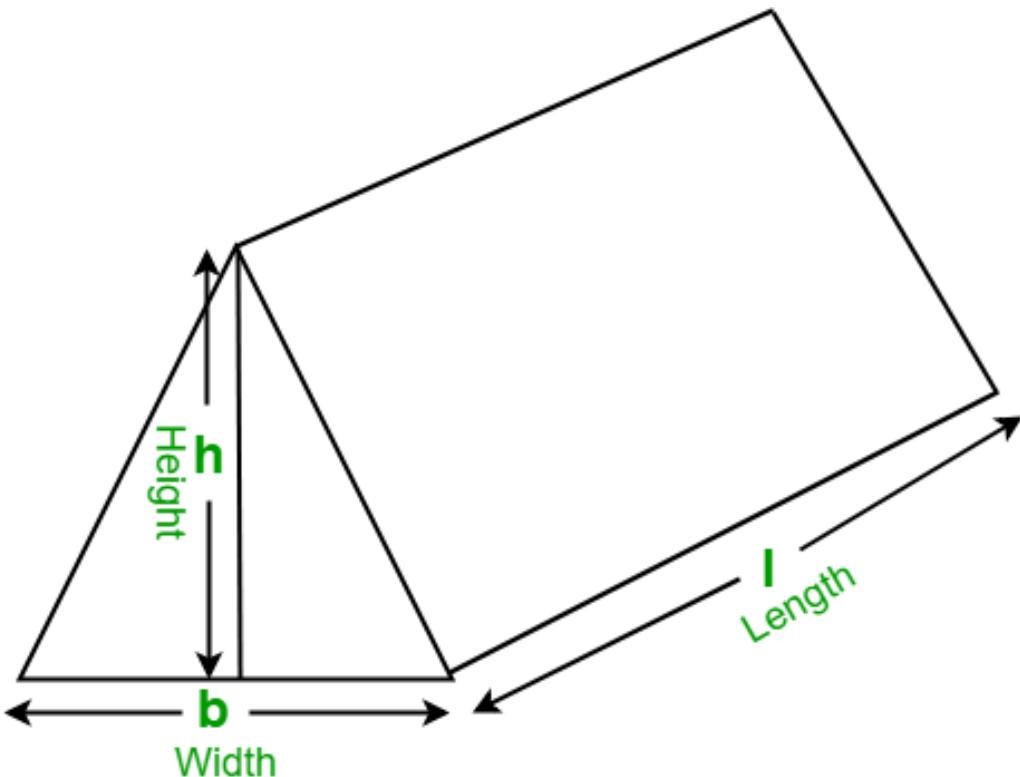
Given the length, width, and height of a triangular prism, the task is to find the volume of the triangular prism.

Examples:

```
Input: l = 18, b = 12, h = 9
Output: Volume of triangular prism: 972
```

```
Input: l = 10, b = 8, h = 6
Output: Volume of triangular prism: 240
```

In mathematics, a triangular prism is a three-dimensional solid shape with two identical ends connected by equal parallel lines. The triangular prism contains 5 faces, 9 edges, and 6 vertices.



Formula to find the volume of triangular prism:

$$\text{Volume} = (l * b * h) / 2$$

C++

```
// CPP program to find the volume
// of the triangular prism
#include <bits/stdc++.h>
using namespace std;

// function to find the Volume
// of triangular prism
float findVolume(float l, float b, float h)
{
    // formula to find Volume
    float volume = (l * b * h) / 2;

    return volume;
}

// Driver Code
```

```
int main()
{
    float l = 18, b = 12, h = 9;

    // function calling
    cout << "Volume of triangular prism: "
        << findVolume(l, b, h);

    return 0;
}
```

Java

```
// Java program to find the volume
// of the triangular prism
import java.io.*;

class GFG {

    // function to find the Volume
    // of triangular prism
    static float findVolume(float l, float b, float h)
    {
        // formula to find Volume
        float volume = (l * b * h) / 2;

        return volume;
    }

    // Driver code
    public static void main(String[] args)
    {
        float l = 18, b = 12, h = 9;

        // function calling
        System.out.println("Volume of triangular prism: "
            + findVolume(l, b, h));
    }
}
```

Python3

```
# Python3 program to find the volume
# of the triangular prism

# function to find the Volume
# of triangular prism
```

```
def findVolume(l, b, h) :  
  
    # formula to find Volume  
    return ((l * b * h) / 2)  
  
# Driver Code  
l = 18  
b = 12  
h = 9  
  
# function calling  
print("Volume of triangular prism: ",  
      findVolume(l, b, h))
```

C#

```
// C# program to find the volume  
// of the triangular prism  
using System;  
  
class GFG {  
  
    // function to find the Volume  
    // of triangular prism  
    static float findVolume(float l, float b, float h)  
    {  
        // formula to find Volume  
        float volume = (l * b * h) / 2;  
  
        return volume;  
    }  
  
    // Driver code  
    static public void Main()  
    {  
        float l = 18, b = 12, h = 9;  
  
        // function calling  
        Console.WriteLine("Volume of triangular prism: "  
                          + findVolume(l, b, h));  
    }  
}
```

PHP

```
<?php  
// PHP program to find the volume
```

```
// of the triangular prism

// function to find the Volume
// of triangular prism
function findVolume($l, $b, $h)
{
    // formula to find Volume
    $volume = ($l * $b * $h) / 2;

    return $volume;
}

// Driver Code
$l = 18; $b = 12; $h = 9;

// function calling
echo "Volume of triangular prism: "
    . findVolume($l, $b, $h);

?>
```

Output:

```
Volume of triangular prism: 972
```

Source

<https://www.geeksforgeeks.org/program-to-find-the-volume-of-a-triangular-prism/>

Chapter 201

Program to find the mid-point of a line

Program to find the mid-point of a line - GeeksforGeeks

Given two coordinates of a line starting is (x_1, y_1) and ending is (x_2, y_2) find out the mid-point of a line.

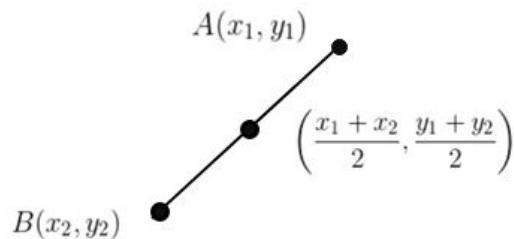
Examples :

Input : $x_1 = -1, y_1 = 2,$
 $x_2 = 3, y_2 = -6$

Output : 1,-2

Input : $x_1 = 6.4, y_1 = 3$
 $x_2 = -10.7, y_2 = 4$

Output : -2.15, 3.5



The Midpoint Formula: The midpoint of two points, (x_1, y_1) and (x_2, y_2) is the point M found by the following formula: $M = ((x_1+x_2)/2, (y_1+y_2)/2)$

C++

```
// C++ program to find
// the midpoint of a line
#include<iostream>
using namespace std;

// function to find the
// midpoint of a line
void midpoint(int x1, int x2,
              int y1, int y2)
{
    cout << (float)(x1+x2)/2 <<
        " , " << (float)(y1+y2)/2 ;
}

// Driver Function to test above
int main()
{
    int x1 = -1, y1 = 2 ;
    int x2 = 3, y2 = -6 ;
    midpoint(x1, x2, y1, y2);
    return 0;
}
```

Java

```
// Java program to find
// the midpoint of a line
import java.io.*;

class GFG
{
    // function to find the
    // midpoint of a line
    static void midpoint(int x1, int x2,
                         int y1, int y2)
    {
        System.out.print((x1 + x2) / 2 +
                         " , " + (y1 + y2) / 2) ;
    }

    // Driver code
    public static void main (String[] args)
    {
        int x1 = -1, y1 = 2 ;
        int x2 = 3, y2 = -6 ;
        midpoint(x1, x2, y1, y2);

    }
}
```

```
}
```

```
// This code is contributed by vt_m.
```

Python3

```
# Python3 program to find
# the midpoint of a line

# Function to find the
# midpoint of a line
def midpoint(x1, x2, y1, y2):

    print((x1 + x2) // 2, " , ",
          (y1 + y2) // 2)

# Driver Code
x1, y1, x2, y2 = -1, 2, 3, -6
midpoint(x1, x2, y1, y2)

# This code is contributed by Anant Agarwal.
```

C#

```
// C# program to find
// the midpoint of a line
using System;

class GFG
{
    // function to find the
    // midpoint of a line
    static void midpoint(int x1, int x2,
                         int y1, int y2)
    {
        Console.WriteLine((x1 + x2) / 2 +
                          " , " + (y1 + y2) / 2) ;
    }

    // Driver code
    public static void Main ()
    {
        int x1 = -1, y1 = 2 ;
        int x2 = 3, y2 = -6 ;

        midpoint(x1, x2, y1, y2);
    }
}
```

```
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program to find
// the midpoint of a line

// function to find the
// midpoint of a line
function midpoint($x1, $x2, $y1, $y2)
{
    echo((float)($x1 + $x2)/2 . " , " .
          (float)($y1 + $y2)/2) ;
}

// Driver Code
$x1 = -1; $y1 = 2 ;
$x2 = 3; $y2 = -6 ;
midpoint($x1, $x2, $y1, $y2);

// This code is contributed by Ajit.
?>
```

Output :

1 , -2

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/program-find-mid-point-line/>

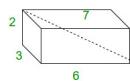
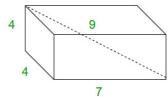
Chapter 202

Pythagorean Quadruple

Pythagorean Quadruple - GeeksforGeeks

Given four points, check whether they form Pythagorean Quadruple.

It is defined as a tuple of integers a, b, c, d such that $a^2 + b^2 + c^2 = d^2$. They are basically the solutions of **Diophantine Equations**. In the geometric interpretation it represents a **cuboid** with integer side lengths $|a|, |b|, |c|$ and whose space diagonal is $|d|$.



The cuboids sides shown here are examples of pythagorean quadruples.

It is primitive when their greatest common divisor is 1. Every Pythagorean quadruple is an integer multiple of a primitive quadruple. We can generate the set of primitive pythagorean quadruples for which a is odd can be generated by formula :

$$\begin{aligned} a &= m^2 + n^2 - p^2 - q^2, \\ b &= 2(mq + np), \\ c &= 2(nq - mp), \\ d &= m^2 + n^2 + p^2 + q^2 \end{aligned}$$

where m, n, p, q are non-negative integers with greatest common divisor 1 such that $m + n + p + q$ are odd. Thus, all primitive Pythagorean quadruples are characterized by **Lebesgue's identity**.

$$(m^2 + n^2 + p^2 + q^2)^2 = (2mq + 2nq)^2 + 2(nq - mp)^2 + (m^2 + n^2 - p^2 - q^2)m^2 + n^2 - p^2 - q^2$$

C++

```
// C++ code to detect Pythagorean Quadruples.
#include <bits/stdc++.h>
using namespace std;

// function for checking
bool pythagorean_quadruple(int a, int b, int c,
                           int d)
{
    int sum = a * a + b * b + c * c;
    if (d * d == sum)
        return true;
    else
        return false;
}

// Driver Code
int main()
{
    int a = 1, b = 2, c = 2, d = 3;
    if (pythagorean_quadruple(a, b, c, d))
        cout << "Yes" << endl;
    else
        cout << "No" << endl;
}
```

Java

```
// Java code to detect Pythagorean Quadruples.
import java.io.*;
import java.util.*;

class GFG {

    // function for checking
    static Boolean pythagorean_quadruple(int a, int b,
                                         int c, int d)
    {
        int sum = a * a + b * b + c * c;
        if (d * d == sum)
            return true;
        else
            return false;
    }
}
```

```
}

// Driver function
public static void main (String[] args) {
    int a = 1, b = 2, c = 2, d = 3;
    if (pythagorean_quadruple(a, b, c, d))
        System.out.println("Yes");
    else
        System.out.println("No" );
}

}

// This code is contributed by Gitanjali.
```

Python3

```
# Python  code to detect
# Pythagorean Quadruples.
import math

# function for checking
def pythagorean_quadruple(a,b, c, d):

    sum = a * a + b * b + c * c;
    if (d * d == sum):
        return True
    else:
        return False

#driver code
a = 1
b = 2
c = 2
d = 3
if (pythagorean_quadruple(a, b, c, d)):
    print("Yes")
else:
    print("No" )

# This code is contributed
# by Gitanjali.
```

C#

```
// C# code to detect
// Pythagorean Quadruples.
using System;
```

```
class GFG {

    // function for checking
    static Boolean pythagorean_quadruple(int a,
                                         int b, int c, int d)
    {
        int sum = a * a + b * b + c * c;
        if (d * d == sum)
            return true;
        else
            return false;
    }

    // Driver function
    public static void Main ()
    {
        int a = 1, b = 2, c = 2, d = 3;

        if (pythagorean_quadruple(a, b, c, d))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No" );
    }
}

// This code is contributed by vt_M.
```

PHP

```
<?php
// php code to detect Pythagorean Quadruples.

// function for checking
function pythagorean_quadruple($a, $b, $c, $d)
{
    $sum = $a * $a + $b * $b + $c * $c;

    if ($d * $d == $sum)
        return true;
    else
        return false;
}

// Driver Code
$a = 1; $b = 2; $c = 2; $d = 3;
```

```
if (pythagorean_quadruple($a, $b, $c, $d))
    echo "Yes" ;
else
    echo "No" ;

// This code is contributed by anuj_67.
?>
```

Output:

Yes

References

[Wiki](#)

[mathworld](#)

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/pythagorean-quadruple/>

Chapter 203

Queries on count of points lie inside a circle

Queries on count of points lie inside a circle - GeeksforGeeks

Given n coordinate (x, y) of points on 2D plane and Q queries. Each query contains an integer r , the task is to count the number of points lying inside or on the circumference of the circle having radius r and centered at the origin.

Examples :

Input : $n = 5$

Coordinates:

```
1 1  
2 2  
3 3  
-1 -1  
4 4
```

Query 1: 3

Query 2: 32

Output :

```
3  
5
```

For first query radius = 3, number of points lie inside or on the circumference are $(1, 1)$, $(-1, -1)$, $(2, 2)$. There are only 3 points lie inside or on the circumference of the circle.

For second query radius = 32, all five points are inside the circle.

The equation for the circle centered at origin (0, 0) with radius r, $x^2 + y^2 = r^2$. And condition for a point at (x_1, y_1) to lie inside or on the circumference, $x_1^2 + y_1^2 \leq r^2$.

A **Naive approach** can be for each query, traverse through all points and check the condition. This take $O(n*Q)$ time complexity.

An **Efficient approach** is to precompute $x^2 + y^2$ for each point coordinate and store them in an array p[]. Now, sort the array p[]. Then apply binary search on the array to find last index with condition $p[i] \leq r^2$ for each query.

Below is the implementation of this approach:

C++

```
// C++ program to find number of points lie inside or
// on the circumference of circle for Q queries.
#include <bits/stdc++.h>
using namespace std;

// Computing the x^2 + y^2 for each given points
// and sorting them.
void preprocess(int p[], int x[], int y[], int n)
{
    for (int i = 0; i < n; i++)
        p[i] = x[i] * x[i] + y[i] * y[i];

    sort(p, p + n);
}

// Return count of points lie inside or on circumference
// of circle using binary search on p[0..n-1]
int query(int p[], int n, int rad)
{
    int start = 0, end = n - 1;
    while ((end - start) > 1) {
        int mid = (start + end) / 2;
        double tp = sqrt(p[mid]);

        if (tp > (rad * 1.0))
            end = mid - 1;
        else
            start = mid;
    }

    double tp1 = sqrt(p[start]), tp2 = sqrt(p[end]);

    if (tp1 > (rad * 1.0))
        return 0;
    else if (tp2 <= (rad * 1.0))
```

```
        return end + 1;
    else
        return start + 1;
}

// Driven Program
int main()
{
    int x[] = { 1, 2, 3, -1, 4 };
    int y[] = { 1, 2, 3, -1, 4 };
    int n = sizeof(x) / sizeof(x[0]);

    // Compute distances of all points and keep
    // the distances sorted so that query can
    // work in O(logn) using Binary Search.
    int p[n];
    preprocess(p, x, y, n);

    // Print number of points in a circle of radius 3.
    cout << query(p, n, 3) << endl;

    // Print number of points in a circle of radius 32.
    cout << query(p, n, 32) << endl;
    return 0;
}
```

Java

```
// JAVA Code for Queries on count of
// points lie inside a circle
import java.util.*;

class GFG {

    // Computing the x^2 + y^2 for each given points
    // and sorting them.
    public static void preprocess(int p[], int x[],
                                  int y[], int n)
    {
        for (int i = 0; i < n; i++)
            p[i] = x[i] * x[i] + y[i] * y[i];

        Arrays.sort(p);
    }

    // Return count of points lie inside or on
    // circumference of circle using binary
    // search on p[0..n-1]
```

```
public static int query(int p[], int n, int rad)
{
    int start = 0, end = n - 1;
    while ((end - start) > 1) {
        int mid = (start + end) / 2;
        double tp = Math.sqrt(p[mid]);

        if (tp > (rad * 1.0))
            end = mid - 1;
        else
            start = mid;
    }

    double tp1 = Math.sqrt(p[start]);
    double tp2 = Math.sqrt(p[end]);

    if (tp1 > (rad * 1.0))
        return 0;
    else if (tp2 <= (rad * 1.0))
        return end + 1;
    else
        return start + 1;
}

/* Driver program to test above function */
public static void main(String[] args)
{
    int x[] = { 1, 2, 3, -1, 4 };
    int y[] = { 1, 2, 3, -1, 4 };
    int n = x.length;

    // Compute distances of all points and keep
    // the distances sorted so that query can
    // work in O(logn) using Binary Search.
    int p[] = new int[n];
    preprocess(p, x, y, n);

    // Print number of points in a circle of
    // radius 3.
    System.out.println(query(p, n, 3));

    // Print number of points in a circle of
    // radius 32.
    System.out.println(query(p, n, 32));
}
}

// This code is contributed by Arnav Kr. Mandal.
```

C#

```
// C# Code for Queries on count of
// points lie inside a circle
using System;

class GFG {

    // Computing the x^2 + y^2 for each
    // given points and sorting them.
    public static void preprocess(int[] p, int[] x,
                                  int[] y, int n)
    {
        for (int i = 0; i < n; i++)
            p[i] = x[i] * x[i] + y[i] * y[i];

        Array.Sort(p);
    }

    // Return count of points lie inside or on
    // circumference of circle using binary
    // search on p[0..n-1]
    public static int query(int[] p, int n, int rad)
    {
        int start = 0, end = n - 1;
        while ((end - start) > 1) {
            int mid = (start + end) / 2;
            double tp = Math.Sqrt(p[mid]);

            if (tp > (rad * 1.0))
                end = mid - 1;
            else
                start = mid;
        }

        double tp1 = Math.Sqrt(p[start]);
        double tp2 = Math.Sqrt(p[end]);

        if (tp1 > (rad * 1.0))
            return 0;
        else if (tp2 <= (rad * 1.0))
            return end + 1;
        else
            return start + 1;
    }

    /* Driver program to test above function */
    public static void Main()
}
```

```
{  
    int[] x = { 1, 2, 3, -1, 4 };  
    int[] y = { 1, 2, 3, -1, 4 };  
    int n = x.Length;  
  
    // Compute distances of all points and keep  
    // the distances sorted so that query can  
    // work in O(logn) using Binary Search.  
    int[] p = new int[n];  
    preprocess(p, x, y, n);  
  
    // Print number of points in a circle of  
    // radius 3.  
    Console.WriteLine(query(p, n, 3));  
  
    // Print number of points in a circle of  
    // radius 32.  
    Console.WriteLine(query(p, n, 32));  
}  
}  
  
// This code is contributed by vt_m.
```

Output:

```
3  
5
```

Time Complexity: $O(n \log n)$ for preprocessing and $O(Q \log n)$ for Q queries.

Improved By : [vt_m](#)

Source

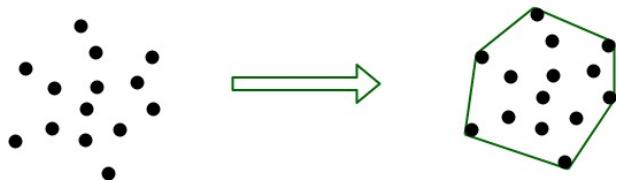
<https://www.geeksforgeeks.org/queries-count-point-lie-inside-circle/>

Chapter 204

Quickhull Algorithm for Convex Hull

Quickhull Algorithm for Convex Hull - GeeksforGeeks

Given a set of points, a Convex hull is the smallest convex polygon containing all the given points.



Input is an array of points specified by their x and y coordinates. Output is a convex hull of this set of points in ascending order of x coordinates.

Example :

```
Input : points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},  
                    {0, 0}, {1, 2}, {3, 1}, {3, 3}};
```

```
Output : The points in convex hull are:  
        (0, 0) (0, 3) (3, 1) (4, 4)
```

```
Input : points[] = {{0, 3}, {1, 1}}  
Output : Not Possible  
There must be at least three points to form a hull.
```

```
Input : points[] = {(0, 0), (0, 4), (-4, 0), (5, 0),  
                    (0, -6), (1, 0)};  
Output : (-4, 0), (5, 0), (0, -6), (0, 4)
```

We have discussed following algorithms for Convex Hull problem.

[Convex Hull | Set 1 \(Jarvis's Algorithm or Wrapping\)](#)

[Convex Hull | Set 2 \(Graham Scan\)](#)

The QuickHull algorithm is a Divide and Conquer algorithm similar to [QuickSort](#). Let $a[0...n-1]$ be the input array of points. Following are the steps for finding the convex hull of these points.

1. Find the point with minimum x-coordinate lets say, min_x and similarly the point with maximum x-coordinate, max_x .
2. Make a line joining these two points, say L . This line will divide the the whole set into two parts. Take both the parts one by one and proceed further.
3. For a part, find the point P with maximum distance from the line L . P forms a triangle with the points min_x , max_x . It is clear that the points residing inside this triangle can never be the part of convex hull.
4. The above step divides the problem into two sub-problems (solved recursively). Now the line joining the points P and min_x and the line joining the points P and max_x are new lines and the points residing outside the triangle is the set of points. Repeat point no. 3 till there no point left with the line. Add the end points of this point to the convex hull.

Below is C++ implementation of above idea. The implementation uses [set](#) to store points so that points can be printed in sorted order. A point is represented as a [pair](#).

```
// C++ program to implement Quick Hull algorithm
// to find convex hull.
#include<bits/stdc++.h>
using namespace std;

// iPair is integer pairs
#define iPair pair<int, int>

// Stores the result (points of convex hull)
set<iPair> hull;

// Returns the side of point p with respect to line
// joining points p1 and p2.
int findSide(iPair p1, iPair p2, iPair p)
{
    int val = (p.second - p1.second) * (p2.first - p1.first) -
              (p2.second - p1.second) * (p.first - p1.first);

    if (val > 0)
        return 1;
    if (val < 0)
        return -1;
    return 0;
}
```

```

// Returns the square of distance between
// p1 and p2.
int dist(iPair p, iPair q)
{
    return (p.second - q.second) * (p.second - q.second) +
           (p.first - q.first) * (p.first - q.first);
}

// returns a value proportional to the distance
// between the point p and the line joining the
// points p1 and p2
int lineDist(iPair p1, iPair p2, iPair p)
{
    return abs ((p.second - p1.second) * (p2.first - p1.first) -
                (p2.second - p1.second) * (p.first - p1.first));
}

// End points of line L are p1 and p2. side can have value
// 1 or -1 specifying each of the parts made by the line L
void quickHull(iPair a[], int n, iPair p1, iPair p2, int side)
{
    int ind = -1;
    int max_dist = 0;

    // finding the point with maximum distance
    // from L and also on the specified side of L.
    for (int i=0; i<n; i++)
    {
        int temp = lineDist(p1, p2, a[i]);
        if (findSide(p1, p2, a[i]) == side && temp > max_dist)
        {
            ind = i;
            max_dist = temp;
        }
    }

    // If no point is found, add the end points
    // of L to the convex hull.
    if (ind == -1)
    {
        hull.insert(p1);
        hull.insert(p2);
        return;
    }

    // Recur for the two parts divided by a[ind]
    quickHull(a, n, a[ind], p1, -findSide(a[ind], p1, p2));
}

```

```

        quickHull(a, n, a[ind], p2, -findSide(a[ind], p2, p1));
    }

void printHull(iPair a[], int n)
{
    // a[i].second -> y-coordinate of the ith point
    if (n < 3)
    {
        cout << "Convex hull not possible\n";
        return;
    }

    // Finding the point with minimum and
    // maximum x-coordinate
    int min_x = 0, max_x = 0;
    for (int i=1; i<n; i++)
    {
        if (a[i].first < a[min_x].first)
            min_x = i;
        if (a[i].first > a[max_x].first)
            max_x = i;
    }

    // Recursively find convex hull points on
    // one side of line joining a[min_x] and
    // a[max_x]
    quickHull(a, n, a[min_x], a[max_x], 1);

    // Recursively find convex hull points on
    // other side of line joining a[min_x] and
    // a[max_x]
    quickHull(a, n, a[min_x], a[max_x], -1);

    cout << "The points in Convex Hull are:\n";
    while (!hull.empty())
    {
        cout << "(" <<(*hull.begin()).first << ", "
              << (*hull.begin()).second << ") ";
        hull.erase(hull.begin());
    }
}

// Driver code
int main()
{
    iPair a[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},
                 {0, 0}, {1, 2}, {3, 1}, {3, 3}};
    int n = sizeof(a)/sizeof(a[0]);
}

```

```
    printHull(a, n);
    return 0;
}
```

Input :

The points in Convex Hull are:

(0, 0) (0, 3) (3, 1) (4, 4)

Time Complexity: The analysis is similar to Quick Sort. On average, we get time complexity as $O(n \log n)$, but in worst case, it can become $O(n^2)$

Source

<https://www.geeksforgeeks.org/quickhull-algorithm-convex-hull/>

Chapter 205

Reflection of a point about a line in C++

Reflection of a point about a line in C++ - GeeksforGeeks

Let's first consider a general case where the line is nothing but the X-Axis. We can now definitely say that the conjugate of a point is the reflection of the point about X-Axis.

Now, using the methods of translation and rotation of coordinate axes we will find out the reflection of a point about the generic line.

The idea of translation was described in the previous post. Here we describe the idea of rotation.

What is Rotation?

In Euclidean geometry, a rotation of axes in two dimensions is a mapping from an xy-Cartesian coordinate system to an x'y'-Cartesian coordinate system in which the origin is kept fixed and the x' and y' axes are obtained by rotating the x and y axes through an angle .

How to Perform Rotation?

Rotation can be interpreted as multiplying (rotating in anticlockwise direction) or dividing (rotating in clockwise direction) every point of the coordinate system by a constant vector. Note here that if we want to rotate a point by in the anticlockwise direction about the origin, we multiply it by polar $(1.0, \theta)$ as discussed in [SET 1](#). Similarly, we divide by polar $(1.0, \theta)$ to rotate the point by in the clockwise direction.

After the rotation, required computations are performed and rotation is nullified by dividing or multiplying every point by the constant vector respectively.

So, we have to reflect a point P about a line specified by points A and B denoted as AB. Since, we know that the conjugate of a point is the reflection of the point about X-Axis. In order to be able to use this fact, we will first perform translation (making A as the origin in the new system) and then rotating the coordinate axes in such a way that the line becomes the X-Axis in the new coordinate system.

Now we can simply apply the formula for reflection about X-Axis and then nullify the effects of rotation and translation to get the final result.

These steps can be described as under:

1. **Translation (Shifting origin at A):** Subtract A from all points.

```
Pt = P - A  
Bt = B - A  
At is origin
```

2. **Rotation (Shifting $B_t A_t$ to the X-Axis):** Divide all points by B_t (dividing means rotating in clockwise direction which is the requirement here to bring on X-Axis).

```
Pr = Pt/Bt
```

3. **Reflection of P_r about $B_r A_r$ (which is nothing but the X-Axis):** Simply take the conjugate of the point.

```
Prreflected = conj(Pr)
```

4. **Restoring back from Rotation:** Multiply all points by B_t .

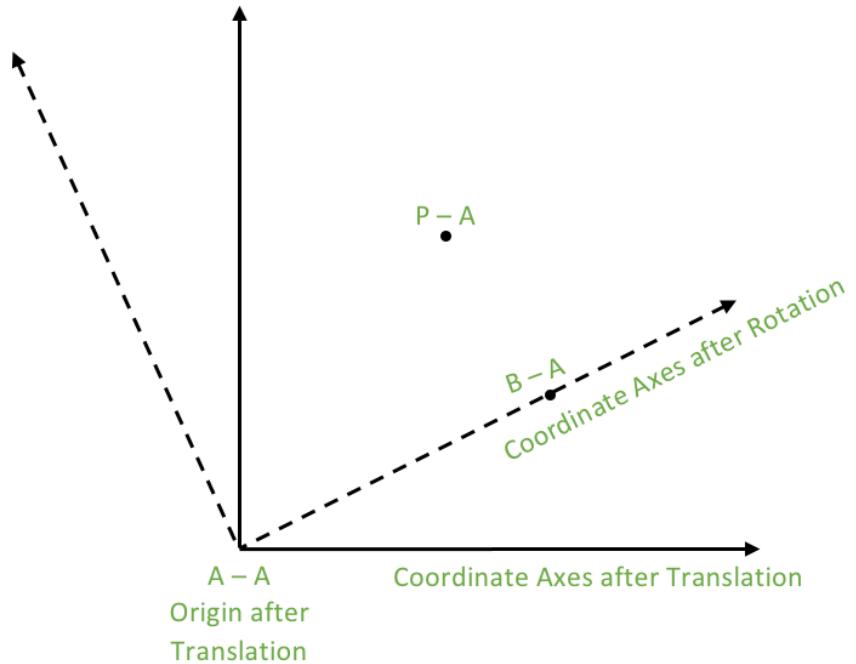
```
Ptreflected= conj(Pr)*Bt
```

5. **Restoring back from Translation:** Add A to all points.

```
P reflected = conj(Pr)*Bt + A
```

Thus,

```
return conj(Pr)*Bt + A  
where, Bt = B - A  
Pt = P - A  
Pr = Pt/Bt
```



```

// CPP example to illustrate the
// reflection of a point about a line
#include <iostream>
#include <complex>

using namespace std;

typedef complex<double> point;
#define x real()
#define y imag()

// Constant PI for providing angles in radians
#define PI 3.1415926535897932384626

// Function used to display X and Y coordinates of a point
void displayPoint(point P)
{
    cout << "(" << P.x << ", " << P.y << ")" << endl;
}

// Function for Reflection of P about line AB
point reflect(point P, point A, point B)

```

```
{  
    // Performing translation and shifting origin at A  
    point Pt = P-A;  
    point Bt = B-A;  
  
    // Performing rotation in clockwise direction  
    // BtAt becomes the X-Axis in the new coordinate system  
    point Pr = Pt/Bt;  
  
    // Reflection of Pr about the new X-Axis  
    // Followed by restoring from rotation  
    // Followed by restoring from translation  
  
    return conj(Pr)*Bt + A;  
}  
  
int main()  
{  
    // Rotate P about line AB  
    point P(4.0, 7.0);  
    point A(1.0, 1.0);  
    point B(3.0, 3.0);  
  
    point P_reflected = reflect(P, A, B);  
    cout << "The point P on reflecting about AB becomes:";  
    cout << "P_reflected"; displayPoint(P_reflected);  
  
    return 0;  
}
```

Output:

The point P on reflecting about AB becomes: P_reflected(7, 4)

Source

<https://www.geeksforgeeks.org/reflection-point-line-c/>

Chapter 206

Reflection of a point at 180 degree rotation of another point

Reflection of a point at 180 degree rotation of another point - GeeksforGeeks

Given two points coordinates (x_1, y_1) and (x_2, y_2) on 2D plane. The task is to find the reflection of (x_1, y_1) at 180 degree rotation of (x_2, y_2) .

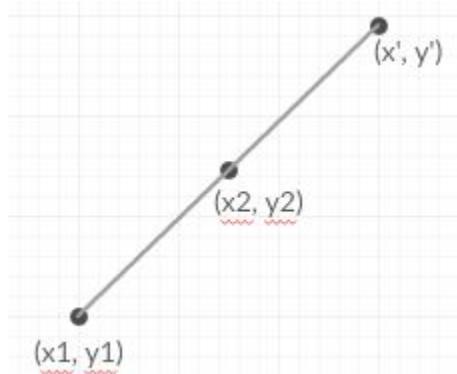
Examples:

```
Input : x1 = 0, y1 = 0, x2 = 1, y2 = 1
Output : (2, 2)
```

```
Input : x1 = 1, y1 = 1, x2 = 2, y2 = 2
Output : (3, 3)
```

Let the reflection point of point (x_1, y_1) about (x_2, y_2) be (x', y') .

For (x', y') be the 180 degree rotation of point (x_1, y_1) around point (x_2, y_2) , they all must be collinear i.e all the three point must lie on a same straight line. Also, observe (x_2, y_2) will became mid point between (x_1, y_1) and (x', y') .



So,

$$x' - x_2 = x_2 - x_1$$

$$y' - y_2 = y_2 - y_1$$

$$x' = 2 * x_2 - x_1$$

$$y' = 2 * y_2 - y_1$$

Below is the implementation of this approach:

C++

```
// CPP Program to find the 180 degree reflection
// of one point around another point.
#include <bits/stdc++.h>
using namespace std;

void findPoint(int x1, int y1, int x2, int y2)
{
    cout << "(" << 2 * x2 - x1 << ", "
        << 2 * y2 - y1 << ")";
}

int main()
{
    int x1 = 0, y1 = 0, x2 = 1, y2 = 1;
    findPoint(x1, y1, x2, y2);
    return 0;
}
```

Java

```
// Java Program to find the 180 degree
// reflection of one point around
// another point.
class GFG {
```

```
static void findPoint(int x1, int y1,
                      int x2, int y2)
{
    System.out.println("(" + (int)(2 * x2 - x1)
                       + "," + (int)(2 * y2 - y1 ) + " )");
}

// Driver code
public static void main(String args[])
{
    int x1 = 0, y1 = 0, x2 = 1, y2 = 1;

    findPoint(x1, y1, x2, y2);
}
}

// This code is contributed by Arnab Kundu.
```

C#

```
// C# Program to find the 180 degree reflection
// of one point around another point.
using System;

public class GFG {

    static void findPoint(int x1, int y1,
                          int x2, int y2)
    {
        Console.WriteLine("(" + (int)(2 * x2 - x1)
                           + "," + (int)(2 * y2 - y1 ) + " )");
    }

    // Driver code
    static public void Main(String []args)
    {
        int x1 = 0, y1 = 0, x2 = 1, y2 = 1;

        findPoint(x1, y1, x2, y2);
    }
}

// This code is contributed by Arnab Kundu.
```

PHP

```
<?php
```

```
// PHP Program for find the 180
// degree reflection of one point
// around another point.

function findPoint($x1, $y1, $x2, $y2)
{
    echo "(" , 2 * $x2 - $x1 , ", "
          , 2 * $y2 - $y1 , ")";
}

// Driver Code
$x1 = 0;
$y1 = 0;
$x2 = 1;
$y2 = 1;
findPoint($x1, $y1, $x2, $y2);

// This code is contributed by anuj_67
?>
```

Output:

(2, 2)

Time Complexity : O(1)

Improved By : [andrew1234](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/reflection-point-180-degree-rotation-another-point/>

Chapter 207

Regular polygon using only 1s in a binary numbered circle

Regular polygon using only 1s in a binary numbered circle - GeeksforGeeks

Given an array of binary integers, suppose these values are kept on the circumference of a circle at an equal distance. We need to tell whether it is possible to draw a regular polygon using only 1s as its vertices and if it is possible then print the maximum number of sides that regular polygon has.

Example:

Input : arr[] = [1, 1, 1, 0, 1, 0]
Output : Polygon possible with side length 3
We can draw a regular triangle having 1s as its vertices as shown in below diagram (a).

Input : arr[] = [1, 0, 1, 0, 1, 0, 1, 0, 1, 1]
Output : Polygon possible with side length 5
We can draw a regular pentagon having 1s as its vertices as shown in below diagram (b).

We can solve this problem by getting a relation between the number of vertices a possible polygon can have and total number of values in the array. Let a possible regular polygon in circle has K vertices or K sides then it should satisfy two things to be the answer, If given array size is N, then K should divide N otherwise K vertices can't divide N vertices in an equally manner to be a regular polygon.

Next thing is there should be one at every vertex of chosen polygon.

After above points, we can see that for solving this problem we need to iterate over

divisors of N and then check whether every value of the array at chosen divisor's distance is 1 or not. If it is 1 then we found our solution. We can iterate over all divisors in $O(\sqrt{N})$ time just by iterating over from 1 to \sqrt{N} . you can read more about that [here](#).

C++

```
// C++ program to find whether a regular polygon
// is possible in circle with 1s as vertices
#include <bits/stdc++.h>
using namespace std;

// method returns true if polygon is possible with
// 'midpoints' number of midpoints
bool checkPolygonWithMidpoints(int arr[], int N,
                                int midpoints)
{
    // loop for getting first vertex of polygon
    for (int j = 0; j < midpoints; j++)
    {
        int val = 1;

        // loop over array values at 'midpoints' distance
        for (int k = j; k < N; k += midpoints)
        {
            // and(&) all those values, if even one of
            // them is 0, val will be 0
            val &= arr[k];
        }

        /* if val is still 1 and (N/midpoints) or (number
         of vertices) are more than two (for a polygon
         minimum) print result and return true */
        if (val && N/midpoints > 2)
        {
            cout << "Polygon possible with side length " <<
                (N/midpoints) << endl;
            return true;
        }
    }
    return false;
}

// method prints sides in the polygon or print not
// possible in case of no possible polygon
void isPolygonPossible(int arr[], int N)
{
    // limit for iterating over divisors
    int limit = sqrt(N);
```

```
for (int i = 1; i <= limit; i++)
{
    // If i divides N then i and (N/i) will
    // be divisors
    if (N % i == 0)
    {
        // check polygon for both divisors
        if (checkPolygonWithMidpoints(arr, N, i) ||
            checkPolygonWithMidpoints(arr, N, (N/i)))
            return;
    }
}

cout << "Not possiblen";
}

// Driver code to test above methods
int main()
{
    int arr[] = {1, 0, 1, 0, 1, 0, 1, 0, 1, 1};
    int N = sizeof(arr) / sizeof(arr[0]);
    isPolygonPossible(arr, N);
    return 0;
}
```

Java

```
// Java program to find whether a regular polygon
// is possible in circle with 1s as vertices

class Test
{
    // method returns true if polygon is possible with
    // 'midpoints' number of midpoints
    static boolean checkPolygonWithMidpoints(int arr[], int N,
                                              int midpoints)
    {
        // loop for getting first vertex of polygon
        for (int j = 0; j < midpoints; j++)
        {
            int val = 1;

            // loop over array values at 'midpoints' distance
            for (int k = j; k < N; k += midpoints)
            {
                // and(&) all those values, if even one of
                // them is 0, val will be 0
                val &= arr[k];
            }
        }
    }
}
```

```
}

/* if val is still 1 and (N/midpoints) or (number
   of vertices) are more than two (for a polygon
   minimum) print result and return true */
if (val != 0 && N/midpoints > 2)
{
    System.out.println("Polygon possible with side length " +
                       N/midpoints);
    return true;
}
return false;
}

// method prints sides in the polygon or print not
// possible in case of no possible polygon
static void isPolygonPossible(int arr[], int N)
{
    // limit for iterating over divisors
    int limit = (int)Math.sqrt(N);
    for (int i = 1; i <= limit; i++)
    {
        // If i divides N then i and (N/i) will
        // be divisors
        if (N % i == 0)
        {
            // check polygon for both divisors
            if (checkPolygonWithMidpoints(arr, N, i) ||
                checkPolygonWithMidpoints(arr, N, (N/i)))
                return;
        }
    }
    System.out.println("Not possible");
}

// Driver method
public static void main(String args[])
{
    int arr[] = {1, 0, 1, 0, 1, 0, 1, 0, 1, 1};

    isPolygonPossible(arr, arr.length);
}
```

C#

```
// C# program to find whether
// a regular polygon is possible
// in circle with 1s as vertices
using System;

class GFG
{

    // method returns true if
    // polygon is possible
    // with 'midpoints' number
    // of midpoints
    static bool checkPolygonWithMidpoints(int []arr, int N,
                                         int midpoints)
    {
        // loop for getting first
        // vertex of polygon
        for (int j = 0; j < midpoints; j++)
        {
            int val = 1;

            // loop over array values
            // at 'midpoints' distance
            for (int k = j; k < N; k += midpoints)
            {
                // and(&) all those values,
                // if even one of them is 0,
                // val will be 0
                val &= arr[k];
            }

            /* if val is still 1 and
               (N/midpoints) or (number
               of vertices) are more than
               two (for a polygon minimum)
               print result and return true */
            if (val != 0 && N / midpoints > 2)
            {
                Console.WriteLine("Polygon possible with " +
                                  "side length " +
                                  N / midpoints);
                return true;
            }
        }
        return false;
    }

    // method prints sides in the
```

```
// polygon or print not possible
// in case of no possible polygon
static void isPolygonPossible(int []arr,
                               int N)
{
    // limit for iterating
    // over divisors
    int limit = (int)Math.Sqrt(N);
    for (int i = 1; i <= limit; i++)
    {
        // If i divides N then i
        // and (N/i) will be divisors
        if (N % i == 0)
        {
            // check polygon for
            // both divisors
            if (checkPolygonWithMidpoints(arr, N, i) ||
                checkPolygonWithMidpoints(arr, N, (N / i)))
                return;
        }
    }

    Console.WriteLine("Not possible");
}

// Driver Code
static public void Main ()
{
    int []arr = {1, 0, 1, 0, 1,
                0, 1, 0, 1, 1};

    isPolygonPossible(arr, arr.Length);
}
}

// This code is contributed by jit_t
```

PHP

```
<?php
// PHP program to find whether
// a regular polygon is possible
// in circle with 1s as vertices
// method returns true if polygon
// is possible with 'midpoints'
// number of midpoints

function checkPolygonWithMidpoints($arr, $N,
```

```
$midpoints)
{
    // loop for getting first
    // vertex of polygon
    for ($j = 0; $j < $midpoints; $j++)
    {
        $val = 1;

        // loop over array values
        // at 'midpoints' distance
        for ($k = $j; $k < $N; $k += $midpoints)
        {
            // and(&) all those values,
            // if even one of them is 0,
            // val will be 0
            $val &= $arr[$k];
        }

        /* if val is still 1 and
        (N/midpoints) or (number
        of vertices) are more than
        two (for a polygon minimum)
        print result and return true */
        if ($val && $N / $midpoints > 2)
        {
            echo "Polygon possible with side length " ,
                ($N / $midpoints) , "\n";
            return true;
        }
    }
    return false;
}

// method prints sides in
// the polygon or print not
// possible in case of no
// possible polygon
function isPolygonPossible($arr, $N)
{
    // limit for iterating
    // over divisors
    $limit = sqrt($N);
    for ($i = 1; $i <= $limit; $i++)
    {
        // If i divides N then
        // i and (N/i) will be
        // divisors
        if ($N % $i == 0)
```

```
{  
    // check polygon for  
    // both divisors  
    if (checkPolygonWithMidpoints($arr, $N, $i) ||  
        checkPolygonWithMidpoints($arr, $N, ($N / $i)))  
        return;  
}  
}  
  
echo "Not possiblen";  
}  
  
// Driver Code  
$arr = array(1, 0, 1, 0, 1,  
            0, 1, 0, 1, 1);  
$N = sizeof($arr);  
isPolygonPossible($arr, $N);  
  
// This code is contributed by ajit  
?>
```

Output:

Polygon possible with side length 5

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/regular-polygon-using-1s-binary-numbered-circle/>

Chapter 208

Represent a given set of points by the best possible straight line

Represent a given set of points by the best possible straight line - GeeksforGeeks

Find the value of m and c such that a straight line $y = mx + c$, best represents the equation of a given set of points $(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_n, y_n)$, given $n \geq 2$.

Examples:

```
Input : n = 5
        x = 1, x = 2, x = 3,
        x = 4, x = 5
        y = 14, y = 27, y = 40,
        y = 55, y = 68
```

```
Output : m = 13.6
         c = 0
```

If we take any pair of number (x, y) from the given data, these value of m and c should make it best fit into the equation for a straight line, $y = mx + c$. Take $x = 1$ and $y = 14$, then using values of m and c from the output, and putting it in the following equation,
 $y = mx + c$,
L.H.S.: $y = 14$, R.H.S: $mx + c = 13.6 \times 1 + 0 = 13.6$
So, they are approximately equal.
Now, take $x = 3$ and $y = 40$,
L.H.S.: $y = 40$, R.H.S: $mx + c = 13.6 \times 3 + 0 = 40.8$
So, they are also approximately equal, and so on for all other values.

```

Input : n = 6
        x = 1, x = 2, x = 3,
        x = 4, x = 5, x = 6
        y = 1200, y = 900, y = 600,
        y = 200, y = 110, y = 50
Output : m = -243.42
         c = 1361.97
    
```

Approach

To best fit a set of points in an equation for a straight line, we need to find the value of two variables, m and c . Now, since there are 2 unknown variables and depending upon the value of n , two cases are possible –

Case 1 – When $n = 2$: There will be two equations and two unknown variables to find, so, there will be a unique solution .

Case 2 – When $n > 2$: In this case, there may or may not exist values of m and c , which satisfy all the n equations, but we can find the best possible values of m and c which can fit a straight line in the given points .

So, if we have n different pairs of x and y , then, we can form n no. of equations from them for a straight line, as follows

```

f = mx + c,
f = mx + c,
f = mx + c,
.....,
.....,
f = mx + c,
where, f, is the value
obtained by putting x in equation
mx + c.
    
```

Then, since ideally f_i should be same as y_i , but still we can find the f_i closest to y_i in all the cases, if we take a new quantity, $U = \sum (y_i - f_i)^2$, and make this quantity minimum for all value of i from 1 to n .

Note: $(y_i - f_i)^2$ is used in place of $(y_i - f_i)$, as we want to consider both the cases when f_i or when y_i is greater, and we want their difference to be minimum, so if we would not square the term, then situations in which f_i is greater and situation in which y_i is greater will cancel each other to an extent, and this is not what we want. So, we need to square the term.

Now, for U to be minimum, it must satisfy the following two equations –

$$\begin{aligned} &= 0 \text{ and} \\ &= 0. \end{aligned}$$

On solving the above two equations, we get two equations, as follows :

```
?y = nc + m?x, and
?xy = c?x + m?x, which can be rearranged as -
m = (n * ?xy - ?x?y) / (n * ?x - (?x)), and
c = (?y - m?x) / n,
```

So, this is how values of m and c for both the cases are obtained, and we can represent a given set of points, by the best possible straight line.

The following code implements the above given algorithm –

C

```
// C Program to find m and c for a straight line given,
// x and y
#include <stdio.h>

// function to calculate m and c that best fit points
// represented by x[] and y[]
void bestApproximate(int x[], int y[], int n)
{
    int i, j;
    float m, c, sum_x = 0, sum_y = 0, sum_xy = 0, sum_x2 = 0;
    for (i = 0; i < n; i++) {
        sum_x += x[i];
        sum_y += y[i];
        sum_xy += x[i] * y[i];
        sum_x2 += (x[i] * x[i]);
    }

    m = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - (sum_x * sum_x));
    c = (sum_y - m * sum_x) / n;

    printf("m =% f", m);
    printf("\nc =% f", c);
}

// Driver main function
int main()
{
    int x[] = { 1, 2, 3, 4, 5 };
    int y[] = { 14, 27, 40, 55, 68 };
    int n = sizeof(x) / sizeof(x[0]);
    bestApproximate(x, y, n);
    return 0;
}
```

C++

```
// C++ Program to find m and c for a straight line given,
// x and y
#include <cmath>
#include <iostream>
using namespace std;

// function to calculate m and c that best fit points
// represented by x[] and y[]
void bestApproximate(int x[], int y[], int n)
{
    float m, c, sum_x = 0, sum_y = 0, sum_xy = 0, sum_x2 = 0;
    for (int i = 0; i < n; i++) {
        sum_x += x[i];
        sum_y += y[i];
        sum_xy += x[i] * y[i];
        sum_x2 += pow(x[i], 2);
    }

    m = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - pow(sum_x, 2));
    c = (sum_y - m * sum_x) / n;

    cout << "m =" << m;
    cout << "\nc =" << c;
}

// Driver main function
int main()
{
    int x[] = { 1, 2, 3, 4, 5 };
    int y[] = { 14, 27, 40, 55, 68 };
    int n = sizeof(x) / sizeof(x[0]);
    bestApproximate(x, y, n);
    return 0;
}
```

Java

```
// Java Program to find m and c for a straight line given,
// x and y
import java.io.*;
import static java.lang.Math.pow;

public class A {
    // function to calculate m and c that best fit points
    // represented by x[] and y[]
```

```
static void bestApproximate(int x[], int y[])
{
    int n = x.length;
    double m, c, sum_x = 0, sum_y = 0,
           sum_xy = 0, sum_x2 = 0;
    for (int i = 0; i < n; i++) {
        sum_x += x[i];
        sum_y += y[i];
        sum_xy += x[i] * y[i];
        sum_x2 += pow(x[i], 2);
    }

    m = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - pow(sum_x, 2));
    c = (sum_y - m * sum_x) / n;

    System.out.println("m = " + m);
    System.out.println("c = " + c);
}

// Driver main function
public static void main(String args[])
{
    int x[] = { 1, 2, 3, 4, 5 };
    int y[] = { 14, 27, 40, 55, 68 };
    bestApproximate(x, y);
}
}
```

Python3

```
# python Program to find m and c for
# a straight line given, x and y

# function to calculate m and c that
# best fit points represented by x[]
# and y[]
def bestApproximate(x, y, n):

    sum_x = 0
    sum_y = 0
    sum_xy = 0
    sum_x2 = 0

    for i in range (0, n):
        sum_x += x[i]
        sum_y += y[i]
        sum_xy += x[i] * y[i]
        sum_x2 += pow(x[i], 2)
```

```
m = (float)((n * sum_xy - sum_x * sum_y)
           / (n * sum_x2 - pow(sum_x, 2)));

c = (float)(sum_y - m * sum_x) / n;

print("m = ", m);
print("c = ", c);

# Driver main function
x = [1, 2, 3, 4, 5]
y = [ 14, 27, 40, 55, 68]
n = len(x)

bestApproximate(x, y, n)

# This code is contributed by Sam007.

C#
// C# Program to find m and c for a
// straight line given, x and y
using System;

class GFG {

    // function to calculate m and c that
    // best fit points represented by x[] and y[]
    static void bestApproximate(int[] x, int[] y)
    {
        int n = x.Length;
        double m, c, sum_x = 0, sum_y = 0,
               sum_xy = 0, sum_x2 = 0;

        for (int i = 0; i < n; i++) {
            sum_x += x[i];
            sum_y += y[i];
            sum_xy += x[i] * y[i];
            sum_x2 += Math.Pow(x[i], 2);
        }

        m = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - Math.Pow(sum_x, 2));

        c = (sum_y - m * sum_x) / n;

        Console.WriteLine("m = " + m);
        Console.WriteLine("c = " + c);
    }
}
```

```
}

// Driver main function
public static void Main()
{
    int[] x = { 1, 2, 3, 4, 5 };
    int[] y = { 14, 27, 40, 55, 68 };

    // Function calling
    bestApproximate(x, y);
}
}

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP Program to find m and c
// for a straight line given,
// x and y

// function to calculate m and
// c that best fit points
// represented by x[] and y[]
function bestApproximate($x, $y, $n)
{
    $i; $j;
    $m; $c;
    $sum_x = 0;
    $sum_y = 0;
    $sum_xy = 0;
    $sum_x2 = 0;
    for ($i = 0; $i < $n; $i++)
    {
        $sum_x += $x[$i];
        $sum_y += $y[$i];
        $sum_xy += $x[$i] * $y[$i];
        $sum_x2 += ($x[$i] * $x[$i]);
    }

    $m = ($n * $sum_xy - $sum_x * $sum_y) /
        ($n * $sum_x2 - ($sum_x * $sum_x));
    $c = ($sum_y - $m * $sum_x) / $n;

    echo "m =", $m;
    echo "\nc =", $c;
}
```

```
// Driver Code
$x =array(1, 2, 3, 4, 5);
$y =array (14, 27, 40, 55, 68);
$n = sizeof($x);
bestApproximate($x, $y, $n);

// This code is contributed by ajit
?>
```

Output:

```
m=13.6
c=0.0
```

Analysis of above code-

Auxiliary Space : O(1)

Time Complexity : O(n). We have one loop which iterates n times, and each time it performs constant no. of computations.

Reference-

1-Higher Engineering Mathematics by B.S. Grewal.

Improved By : [Sam007, jit_t](#)

Source

<https://www.geeksforgeeks.org/represent-given-set-points-best-possible-straight-line/>

Chapter 209

Rotation of a point about another point in C++

Rotation of a point about another point in C++ - GeeksforGeeks

We have already discussed the rotation of a point P about the origin in the [Set 1](#) and [Set 2](#). The rotation of point P about origin with an angle θ in the anti-clockwise direction is given as under:

```
Rotation of P about origin: P * polar(1.0, theta)
```

Rotation of P about point Q

Now, we have to rotate the point P not about origin but about a general point Q. This can be easily understood by the method of translation which is quite a common technique adopted in geometric analysis.

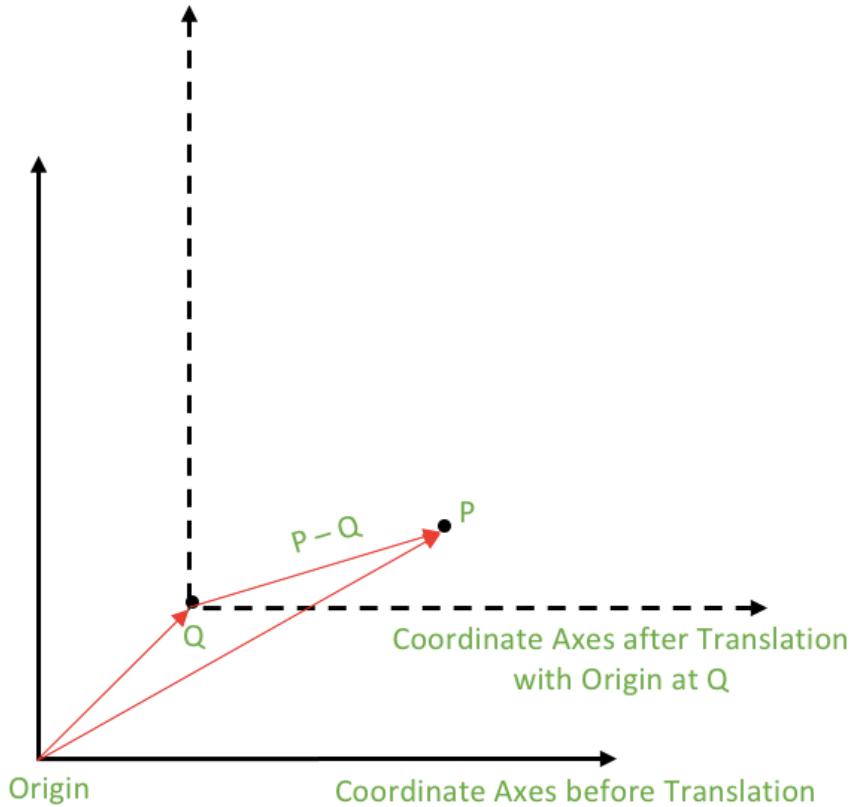
What is Translation?

In Euclidean geometry, translation is a geometric transformation that moves every point of a figure or a space by the same amount in a given direction.

How to Perform Translation?

Translation can also be interpreted as the addition of a constant vector to every point, or as shifting the origin of the coordinate system.

After the translation, required computations are made and the translation is nullified by subtracting the constant vector to every point or shifting the origin back.



So, for rotating P about Q, we shift the origin at Q i.e. we subtract the vector equivalent of Q from every point of the coordinate plane. Now the new point $P - Q$ has to be rotated about the origin and then translation has to be nullified.

These steps can be described as under:

1. **Translation (Shifting origin at Q):** Subtract Q from all points. Thus, P becomes $P - Q$
2. **Rotation of ($P - Q$) about origin:** $(P - Q) * \text{polar}(1.0, \theta)$
3. **Restoring back the Origin:** Add Q to all the points.

Thus,

Rotation of P about Q : $(P - Q) * \text{polar}(1.0, \theta) + Q$

```
// CPP example to illustrate the rotation
// of a point about another point
#include <iostream>
```

```
#include <complex>

using namespace std;

typedef complex<double> point;
#define x real()
#define y imag()

// Constant PI for providing angles in radians
#define PI 3.1415926535897932384626

// Function used to display X and Y coordinates of a point
void displayPoint(point P)
{
    cout << "(" << P.x << ", " << P.y << ")" << endl;
}

//Function for Rotation of P about Q by angle theta
point rotate(point P, point Q, double theta)
{
    return (P-Q) * polar(1.0, theta) + Q;
}

int main()
{
    // Rotate P about Q
    point P(4.0, 3.0);
    point Q(2.0, 2.0);

    // Angle of rotation = 90 degrees
    double theta = PI/2;

    point P_rotated = rotate(P, Q, theta);
    cout << "The point P on rotating 90 degrees anti-clockwise about Q becomes:";
    cout << "P_rotated"; displayPoint(P_rotated);

    return 0;
}
```

Output:

The point P on rotating 90 degrees anti-clockwise about Q becomes: P_rotated(1, 4)

Source

<https://www.geeksforgeeks.org/rotation-of-a-point-about-another-point-in-cpp/>

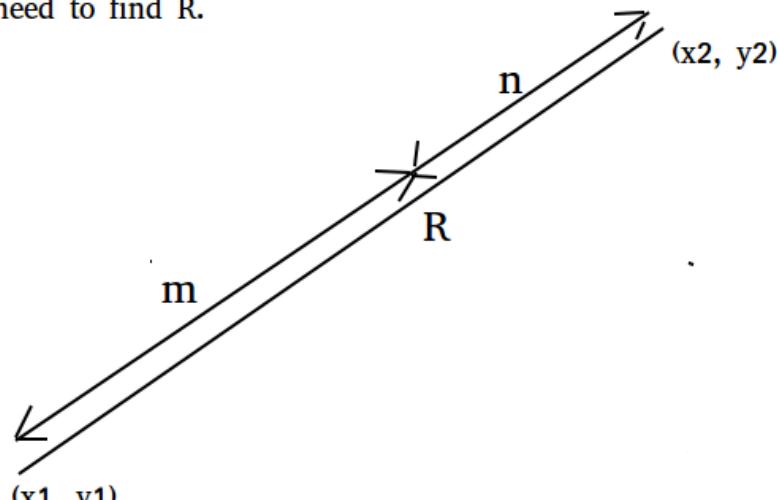
Chapter 210

Section formula (Point that divides a line in given ratio)

Section formula (Point that divides a line in given ratio) - GeeksforGeeks

Given two coordinates (x_1, y_1) and (x_2, y_2) , and m and n , find the co-ordinates that divides that divides the line joining (x_1, y_1) and (x_2, y_2) in the ratio $m : n$

Given (x_1, y_1) , (x_2, y_2) , m and n ,
we need to find R .



Examples:

Input : $x_1 = 1$, $y_1 = 0$, $x_2 = 2$ $y_2 = 5$,
 $m = 1$, $n = 1$

Output : $(1.5, 2.5)$

Explanation: co-ordinates $(1.5, 2.5)$

```

divides the line in ratio 1 : 1

Input : x1 = 2, y1 = 4, x2 = 4, y2 = 6,
        m = 2, n = 3
Output : (2.8, 4.8)
Explanation: (2.8, 4.8) divides the line
in the ratio 2:3

```

The section formula tells us the coordinates of the point which divides a given line segment into two parts such that their lengths are in the ratio $m : n$

$$\left[\frac{mx_2 + nx_1}{m+n}, \frac{my_2 + ny_1}{m+n} \right]$$

C++

```

// CPP program to find point that divides
// given line in given ratio.
#include <iostream>
using namespace std;

// Function to find the section of the line
void section(double x1, double x2, double y1,
             double y2, double m, double n)
{
    // Applying section formula
    double x = ((n * x1) + (m * x2)) /
               (m + n);
    double y = ((n * y1) + (m * y2)) /
               (m + n);

    // Printing result
    cout << "(" << x << ", ";
    cout << y << ")" << endl;
}

// Driver code
int main()
{
    double x1 = 2, x2 = 4, y1 = 4,
           y2 = 6, m = 2, n = 3;
    section(x1, x2, y1, y2, m, n);
}

```

```
    return 0;
}
```

Java

```
// Java program to find point that divides
// given line in given ratio.
import java.io.*;

class sections {
    static void section(double x1, double x2,
                        double y1, double y2,
                        double m, double n)
    {
        // Applying section formula
        double x = ((n * x1) + (m * x2)) /
                   (m + n);
        double y = ((n * y1) + (m * y2)) /
                   (m + n);

        // Printing result
        System.out.println("(" + x + ", " + y + ")");
    }

    public static void main(String[] args)
    {
        double x1 = 2, x2 = 4, y1 = 4,
               y2 = 6, m = 2, n = 3;
        section(x1, x2, y1, y2, m, n);
    }
}
```

Python

```
# Python program to find point that divides
# given line in given ratio.
def section(x1, x2, y1, y2, m, n):

    # Applying section formula
    x = (float)((n * x1)+(m * x2))/(m + n)
    y = (float)((n * y1)+(m * y2))/(m + n)

    # Printing result
    print (x, y)

x1 = 2
```

```
x2 = 4
y1 = 4
y2 = 6
m = 2
n = 3
section(x1, x2, y1, y2, m, n)
```

C#

```
// C# program to find point that divides
// given line in given ratio.
using System;

class GFG {

    static void section(double x1, double x2,
                        double y1, double y2,
                        double m, double n)
    {

        // Applying section formula
        double x = ((n * x1) + (m * x2)) /
                   (m + n);

        double y = ((n * y1) + (m * y2)) /
                   (m + n);

        // Printing result
        Console.WriteLine("(" + x + ", " + y + ")");
    }

    // Driver code
    public static void Main()
    {

        double x1 = 2, x2 = 4, y1 = 4,
               y2 = 6, m = 2, n = 3;

        section(x1, x2, y1, y2, m, n);
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
```

```
// PHP program to find point that
// divides given line in given ratio.

// Function to find the
// section of the line
function section($x1, $x2, $y1,
                 $y2, $m, $n)
{
    // Applying section formula
    $x = (($n * $x1) + ($m * $x2))
        / ($m + $n);

    $y = (($n * $y1) + ($m * $y2))
        / ($m + $n);

    // Printing result
    echo("(" . $x . ", ");
    echo($y . ")");
}

// Driver code
$x1 = 2; $x2 = 4; $y1 = 4;
$y2 = 6; $m = 2; $n = 3;
section($x1, $x2, $y1, $y2, $m, $n);

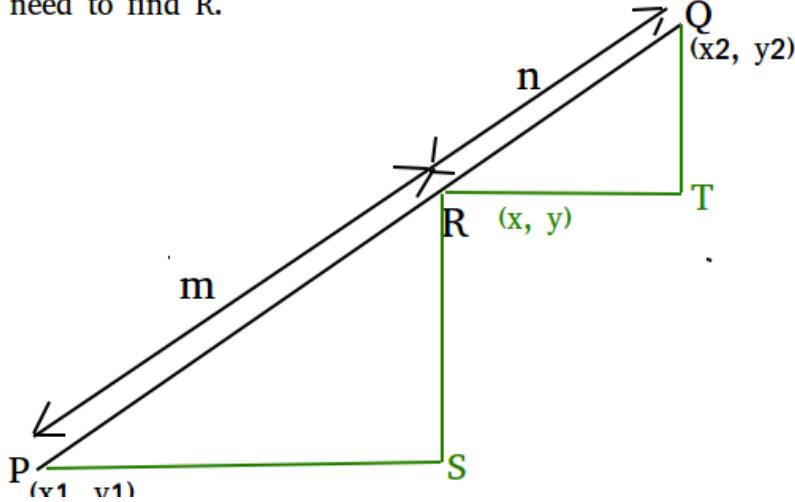
// This code is contributed by Ajit.
?>
```

Output:

(2.8, 4.8)

How does this work?

Given (x_1, y_1) , (x_2, y_2) , m and n ,
we need to find R .



From our diagram, we can see,
 $PS = x - x_1$ and $RT = x_2 - x$

We are given,

$$PR/QR = m/n$$

Using similarity, we can write
 $RS/QT = PS/RT = PR/QR$

Therefore, we can write

$$\begin{aligned} PS/RR &= m/n \\ (x - x_1) / (x_2 - x) &= m/n \end{aligned}$$

From above, we get

$$x = (mx_2 + nx_1) / (m + n)$$

Similarly, we can solve for y .

References:

<http://doubleroot.in/lessons/coordinate-geometry-basics/section-formula/#.WjYXQvhU8o>

Improved By : jit_t

Source

<https://www.geeksforgeeks.org/section-formula-point-divides-line-given-ratio/>

Chapter 211

Shortest distance between a Line and a Point in a 3-D plane

Shortest distance between a Line and a Point in a 3-D plane - GeeksforGeeks

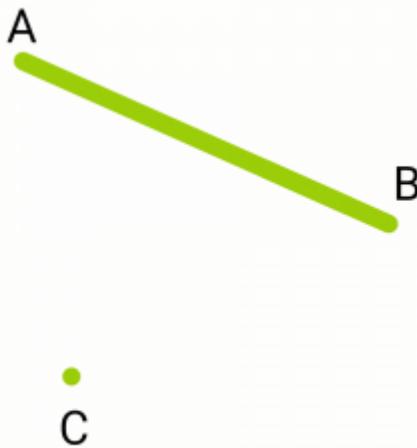
Given a line passing through two points A and B and an arbitrary point C in a 3-D plane, the task is to find the shortest distance between the point C and the line passing through the points A and B.

Examples:

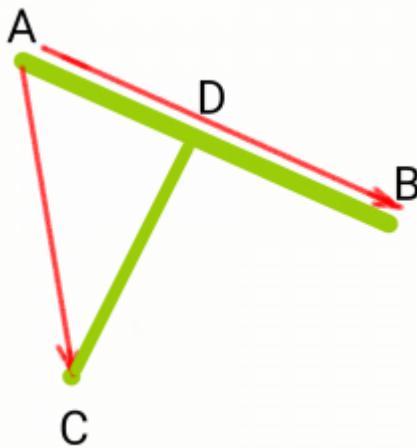
Input: A = (5, 2, 1), B = (3, 1, -1), C = (0, 2, 3)
Output: Shortest Distance is 5

Input: A = (4, 2, 1), B = (3, 2, 1), C = (0, 2, 0)
Output: Shortest Distance is 1

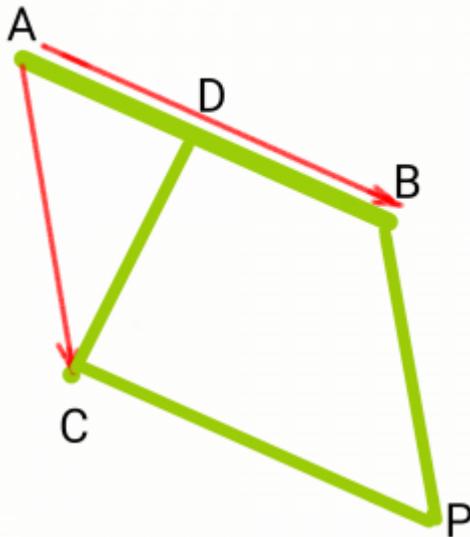
Consider a point C and a line that passes through A and B as shown in the below figure.



Now Consider the vectors, \mathbf{AB} and \mathbf{AC} and the shortest distance as CD . The Shortest Distance is always the perpendicular distance. The point D is taken on AB such that CD is perpendicular to AB .



Construct BP and CP as shown in the figure to form a Parallelogram. Now C is a vertex of parallelogram ABPC and CD is perpendicular to Side AB. Hence CD is the height of the parallelogram.



Note: In the case when D does not fall on line segment AB there will be a point D' such that PD' is perpendicular to AB and D' lies on line segment AB with CD = PD'.

The magnitude of cross product $\mathbf{AB} \times \mathbf{AC}$ gives the Area of the parallelogram. Also, the area of a parallelogram is $\text{Base} * \text{Height} = \mathbf{AB} * \mathbf{CD}$. So,

$$CD = |\mathbf{AB} \times \mathbf{AC}| / |\mathbf{AB}|$$

Below is the CPP program to find the shortest distance:

```
// C++ program to find the Shortest
// Distance Between A line and a
// Given point.
#include<bits/stdc++.h>
using namespace std;

class Vector {
private:
    int x, y, z;
    // 3D Coordinates of the Vector

public:
    Vector(int x, int y, int z)
    {
        // Constructor
        this->x = x;
        this->y = y;
        this->z = z;
    }
}
```

```
}

Vector operator+(Vector v); // ADD 2 Vectors
Vector operator-(Vector v); // Subtraction
int operator^(Vector v); // Dot Product
Vector operator*(Vector v); // Cross Product
float magnitude()
{
    return sqrt(pow(x, 2) + pow(y, 2) + pow(z, 2));
}
friend ostream& operator<<(ostream& out, const Vector& v);
// To output the Vector
};

// ADD 2 Vectors
Vector Vector::operator+(Vector v)
{
    int x1, y1, z1;
    x1 = x + v.x;
    y1 = y + v.y;
    z1 = z + v.z;
    return Vector(x1, y1, z1);
}

// Subtract 2 vectors
Vector Vector::operator-(Vector v)
{
    int x1, y1, z1;
    x1 = x - v.x;
    y1 = y - v.y;
    z1 = z - v.z;
    return Vector(x1, y1, z1);
}

// Dot product of 2 vectors
int Vector::operator^(Vector v)
{
    int x1, y1, z1;
    x1 = x * v.x;
    y1 = y * v.y;
    z1 = z * v.z;
    return (x1 + y1 + z1);
}

// Cross product of 2 vectors
Vector Vector::operator*(Vector v)
{
    int x1, y1, z1;
    x1 = y * v.z - z * v.y;
```

```
y1 = z * v.x - x * v.z;
z1 = x * v.y - y * v.x;
return Vector(x1, y1, z1);
}

// Display Vector
ostream& operator<<(ostream& out,
                      const Vector& v)
{
    out << v.x << "i ";
    if (v.y >= 0)
        out << "+ ";
    out << v.y << "j ";
    if (v.z >= 0)
        out << "+ ";
    out << v.z << "k" << endl;
    return out;
}

// calculate shortest dist. from point to line
float shortDistance(Vector line_point1, Vector line_point2,
                     Vector point)
{
    Vector AB = line_point2 - line_point1;
    Vector AC = point - line_point1;
    float area = Vector(AB * AC).magnitude();
    float CD = area / AB.magnitude();
    return CD;
}

// Driver program
int main()
{
    // Taking point C as (2, 2, 2)
    // Line Passes through A(4, 2, 1)
    // and B(8, 4, 2).
    Vector line_point1(4, 2, 1), line_point2(8, 4, 2);
    Vector point(2, 2, 2);

    cout << "Shortest Distance is : "
         << shortDistance(line_point1, line_point2, point);

    return 0;
}
```

Output:

Shortest Distance is : 1.63299

Source

<https://www.geeksforgeeks.org/shortest-distance-between-a-line-and-a-point-in-a-3-d-plane/>

Chapter 212

Slope of perpendicular to line

Slope of perpendicular to line - GeeksforGeeks

You are given the slope of one line (m_1) and you have to find the slope of another lie which is perpendicular to the given line.

Examples:

Input : 5

Output : Slope of perpendicular line is : -0.20

Input : 4

Output : Slope of perpendicular line is : -0.25

Suppose we are given two perpendicular line segments AB and CD. The slope of AB is m_1 and line CD is m_2 .

$$m_1 * m_2 = -1$$

From above, we can say

$$m_2 = -1/(m_1)$$

How does above formula work?

Let slope of line AB be m_1 and we need to find slope of line CD. Below diagram gives an idea about working of formula.

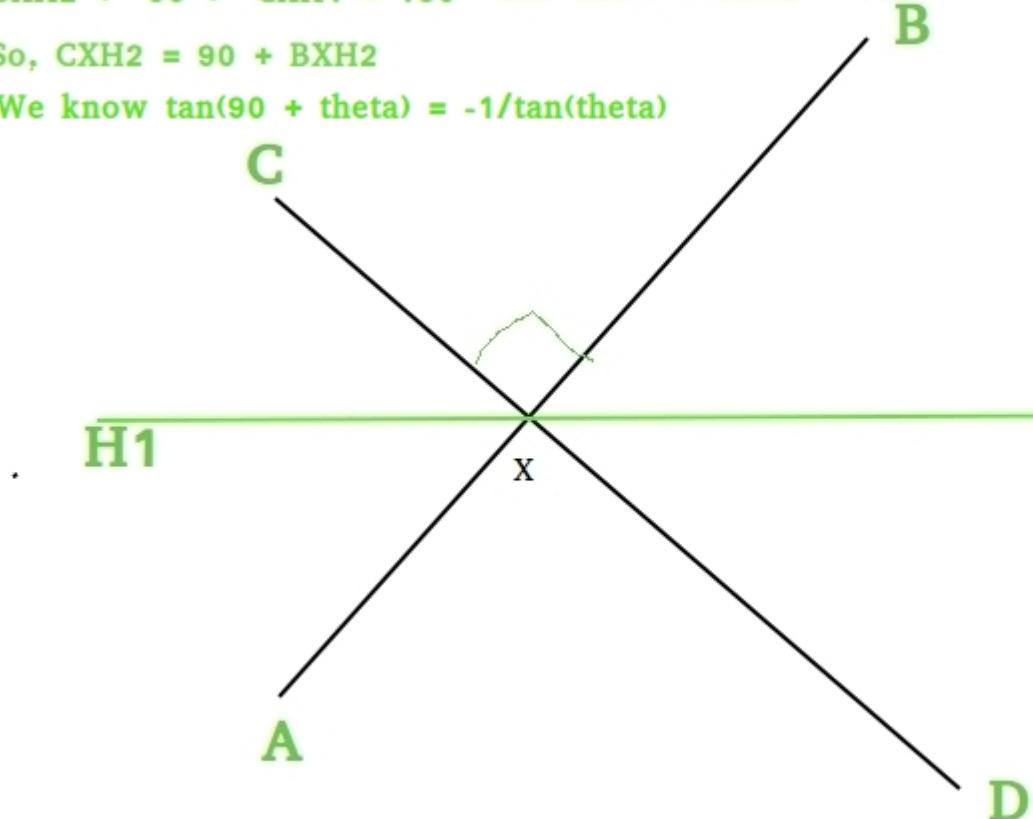
Let AB and CD be two perpendicular lines.

Let H₁H₂ be a horizontal line passing through intersection point of AB and CD. We can say

$$BXH_2 + 90 + CXH_1 = 180 \text{ and } CXH_1 + CXH_2 = 180$$

$$\text{So, } CXH_2 = 90 + BXH_2$$

$$\text{We know } \tan(90 + \theta) = -1/\tan(\theta)$$



C++

```
// C++ program find slope of perpendicular line
#include <bits/stdc++.h>
using namespace std;

// Function to find
// the Slope of other line
double findPCslope(double m)
{
```

```
        return -1.0 / m;
    }

int main()
{
    double m = 2.0;
    cout << findPCSSlope(m);
    return 0;
}
```

Java

```
// Java program to illustrate ...

import java.io.*;
import java.util.*;

class GFG {

    // Function to find
    // the Slope of other line
    static double findPCSSlope(double m)
    {
        return -1.0 / m;
    }

    public static void main(String[] args)
    {

        double m = 2.0;
        System.out.println(findPCSSlope(m));
    }
}
```

Python 3

```
# Python 3 program find
# slope of perpendicular line

# Function to find
# the Slope of other line
def findPCSSlope(m):

    return -1.0 / m

m = 2.0
print(findPCSSlope(m))
```

```
# This code is contributed
# by Smitha

C#

// C# Program to find Slope
// of perpendicular to line
using System;

class GFG {

    // Function to find
    // the Slope of other line
    static double findPCSlope(double m)
    {
        return -1.0 / m;
    }

    // Driver Code
    public static void Main()
    {

        double m = 2.0;
        Console.WriteLine(findPCSlope(m));
    }
}

// This code is contributed by nitin mittal
```

PHP

```
<?php
// PHP program find slope
// of perpendicular line

// Function to find the
// Slope of other line
function findPCSlope($m)
{
    return -1.0 / $m;
}

// Driver Code
$m = 2.0;
echo findPCSlope($m);
```

```
// This code is contributed by anuj_67  
?>
```

Output:

-0.5

Improved By : [vt_m](#), [nitin mittal](#), [Smitha Dinesh Semwal](#)

Source

<https://www.geeksforgeeks.org/slope-perpendicular-line/>

Chapter 213

Smallest square formed with given rectangles

Smallest square formed with given rectangles - GeeksforGeeks

Given a rectangle of length **l** and breadth **b**, we need to find the area of the smallest square which can be formed with the rectangles of these given dimensions.

Examples:

```
Input : 1 2
Output : 4
We can form a 2 x 2 square
using two rectangles of size
1 x 2.
```

```
Input : 7 10
Output :4900
```

Let's say we want to make a square of side length **a** from rectangles of length **l** & **b**. This means that **a** is a multiple of both **l** & **b**. Since we want the smallest square, it has to be the **lowest common multiple** (LCM) of **l** & **b**.

Program 1:

```
// C++ Program to find the area
// of the smallest square which
// can be formed with rectangles
// of given dimensions
#include <bits/stdc++.h>
using namespace std;
// Recursive function to return gcd of a and b
```

```
int gcd(int a, int b)
{
    // Everything divides 0
    if (a == 0 || b == 0)
        return 0;

    // Base case
    if (a == b)
        return a;

    // a is greater
    if (a > b)
        return gcd(a - b, b);
    return gcd(a, b - a);
}

// Function to find the area
// of the smallest square
int squarearea(int l, int b)
{

    // the length or breadth or side
    // cannot be negative
    if (l < 0 || b < 0)
        return -1;

    // LCM of length and breadth
    int n = (l * b) / gcd(l, b);

    // squaring to get the area
    return n * n;
}

// Driver code
int main()
{
    int l = 6, b = 4;
    cout << squarearea(l, b) << endl;
    return 0;
}
```

Output:

Source

<https://www.geeksforgeeks.org/smallest-square-formed-with-given-rectangles/>

Chapter 214

Sum of Areas of Rectangles possible for an array

Sum of Areas of Rectangles possible for an array - GeeksforGeeks

Given an array, task is to compute the sum of all possible maximum area rectangle which can be formed from the array elements. Also, you can reduce the elements of the array by at most 1.

Examples :

```
Input :a = {10, 10, 10, 10, 11,
           10, 11, 10}
Output : 210
Explanation :
We can form two rectangles one square (10 * 10)
and one (11 * 10). Hence, total area = 100 + 110 = 210.
```

```
Input : a = { 3, 4, 5, 6 }
Output : 15
Explanation :
We can reduce 4 to 3 and 6 to 5 so that we got
rectangle of (3 * 5). Hence area = 15.
```

```
Input : a = { 3, 2, 5, 2 }
Output : 0
```

Naive Approach : Check for all possible four elements of the array and then whichever can form a rectangle. In these rectangles, separate all those rectangles which are of maximum area formed by these elements. After getting the rectangles and their areas, sum them all to get our desired solution.

Efficient Approach : To get the maximum area rectangle, first sort the elements of the array in non-increasing array. After sorting, start the procedure to select the elements of the array. Here, selection of two elements of array (as length of rectangle) is possible if elements of array are equal ($a[i] == a[i+1]$) or if length of smaller element $a[i+1]$ is one less than $a[i]$ (*in this case we have our length $a[i+1]$ because $a[i]$ is decreased by 1*). One flag variable is maintained to check that *whether we get length and breadth both*. After getting length, set the flag variable, now calculate the breadth in the same way as we have done for length and sum the area of rectangle. After getting length and breadth both, again set the flag variable false so that we will now search for a new rectangle. This process is repeated and lastly, final sum of area is returned.

C++

```
// CPP code to find sum of all
// area rectangle possible
#include <bits/stdc++.h>
using namespace std;

// Function to find
// area of rectangles
int MaxTotalRectangleArea(int a[],
                           int n)
{
    // sorting the array in
    // descending order
    sort(a, a + n, greater<int>());

    // store the final sum of
    // all the rectangles area
    // possible
    int sum = 0;
    bool flag = false;

    // temporary variable to store
    // the length of rectangle
    int len;

    for (int i = 0; i < n; i++)
    {
        // Selecting the length of
        // rectangle so that difference
        // between any two number is 1
        // only. Here length is selected
        // so flag is set
        if ((a[i] == a[i + 1] || a[i] -
             a[i + 1] == 1) && (!flag))
        {
            // flag is set means
            // that we have found
            // a pair of numbers
            // which can form a
            // rectangle with
            // length as a[i]
            // and breadth as
            // a[i + 1]
            // so calculate
            // their product
            // and add it to
            // sum
            sum += a[i] * a[i + 1];
            flag = true;
        }
    }
}
```

```
// we have got length of
// rectangle
flag = true;

// length is set to
// a[i+1] so that if
// a[i] a[i+1] is less
// than by 1 then also
// we have the correct
// choice for length
len = a[i + 1];

// incrementing the counter
// one time more as we have
// considered a[i+1] element
// also so.
i++;
}

// Selecting the width of rectangle
// so that difference between any
// two number is 1 only. Here width
// is selected so now flag is again
// unset for next rectangle
else if ((a[i] == a[i + 1] ||
         a[i] - a[i + 1] == 1) && (flag))
{
    // area is calculated for
    // rectangle
    sum = sum + a[i + 1] * len;

    // flag is set false
    // for another rectangle
    // which we can get from
    // elements in array
    flag = false;

    // incrementing the counter
    // one time more as we have
    // considered a[i+1] element
    // also so.
    i++;
}
}

return sum;
}
```

```
// Driver code
int main()
{
    int a[] = { 10, 10, 10, 10,
                11, 10, 11, 10,
                9, 9, 8, 8 };
    int n = sizeof(a) / sizeof(a[0]);

    cout << MaxTotalRectangleArea(a, n);

    return 0;
}
```

Java

```
// Java code to find sum of
// all area rectangle possible
import java.io.*;
import java.util.Arrays;

class GFG
{
    // Function to find
    // area of rectangles
    static int MaxTotalRectangleArea(int []a,
                                    int n)
    {

        // sorting the array in
        // descending order
        Arrays.sort(a);

        // store the final sum of
        // all the rectangles area
        // possible
        int sum = 0;
        boolean flag = false;

        // temporary variable to
        // store the length of rectangle
        int len = 0;

        for (int i = 0; i < n; i++)
        {

            // Selecting the length of
            // rectangle so that difference
            // between any two number is 1
```

```
// only. Here length is selected
// so flag is set
if ((a[i] == a[i + 1] ||
     a[i] - a[i + 1] == 1) &&
    !flag)
{
    // flag is set means
    // we have got length of
    // rectangle
    flag = true;

    // length is set to
    // a[i+1] so that if
    // a[i] a[i+1] is less
    // than by 1 then also
    // we have the correct
    // choice for length
    len = a[i + 1];

    // incrementing the counter
    // one time more as we have
    // considered a[i+1] element
    // also so.
    i++;
}

// Selecting the width of rectangle
// so that difference between any
// two number is 1 only. Here width
// is selected so now flag is again
// unset for next rectangle
else if ((a[i] == a[i + 1] ||
          a[i] - a[i + 1] == 1) &&
          (flag))
{
    // area is calculated for
    // rectangle
    sum = sum + a[i + 1] * len;

    // flag is set false
    // for another rectangle
    // which we can get from
    // elements in array
    flag = false;

    // incrementing the counter
    // one time more as we have
    // considered a[i+1] element
```

```
// also so.
    i++;
}
}

return sum;
}

// Driver code
public static void main (String args[])
{
int []a = { 10, 10, 10, 10,
           11, 10, 11, 10,
           9, 9, 8, 8 };
int n = a.length;

System.out.print(MaxTotalRectangleArea(a, n));
}
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Python3

```
# Python3 code to find sum
# of all area rectangle
# possible

# Function to find
# area of rectangles
def MaxTotalRectangleArea(a, n) :

    # sorting the array in
    # descending order
    a.sort(reverse = True)

    # store the final sum of
    # all the rectangles area
    # possible
    sum = 0
    flag = False

    # temporary variable to store
    # the length of rectangle
    len = 0
    i = 0

    while (i < n-1) :
```

```
if(i != 0) :
    i = i + 1

# Selecting the length of
# rectangle so that difference
# between any two number is 1
# only. Here length is selected
# so flag is set
if ((a[i] == a[i + 1] or
     a[i] - a[i + 1] == 1)
     and flag == False) :

    # flag is set means
    # we have got length of
    # rectangle
    flag = True

    # length is set to
    # a[i+1] so that if
    # a[i+1] is less than a[i]
    # by 1 then also we have
    # the correct choice for length
    len = a[i + 1]

    # incrementing the counter
    # one time more as we have
    # considered a[i+1] element
    # also so.
    i = i + 1

# Selecting the width of rectangle
# so that difference between any
# two number is 1 only. Here width
# is selected so now flag is again
# unset for next rectangle
elif ((a[i] == a[i + 1] or
      a[i] - a[i + 1] == 1)
      and flag == True) :

    # area is calculated for
    # rectangle
    sum = sum + a[i + 1] * len

    # flag is set false
    # for another rectangle
    # which we can get from
    # elements in array
    flag = False
```

```
# incrementing the counter
# one time more as we have
# considered a[i+1] element
# also so.
i = i + 1

return sum

# Driver code
a = [ 10, 10, 10, 10, 11, 10,
      11, 10, 9, 9, 8, 8 ]
n = len(a)

print (MaxTotalRectangleArea(a, n))

# This code is contributed by
# Manish Shaw (manishshaw1)

C#
// C# code to find sum of all area rectangle
// possible
using System;

class GFG {

    // Function to find
    // area of rectangles
    static int MaxTotalRectangleArea(int []a,
                                      int n)
    {

        // sorting the array in descending
        // order
        Array.Sort(a);

        // store the final sum of all the
        // rectangles area possible
        int sum = 0;
        bool flag = false;

        // temporary variable to store the
        // length of rectangle
        int len = 0;

        for (int i = 0; i < n; i++)
        {
```

```
// Selecting the length of
// rectangle so that difference
// between any two number is 1
// only. Here length is selected
// so flag is set
if ((a[i] == a[i + 1] || a[i] -
    a[i + 1] == 1) && (!flag))
{
    // flag is set means
    // we have got length of
    // rectangle
    flag = true;

    // length is set to
    // a[i+1] so that if
    // a[i] a[i+1] is less
    // than by 1 then also
    // we have the correct
    // choice for length
    len = a[i + 1];

    // incrementing the counter
    // one time more as we have
    // considered a[i+1] element
    // also so.
    i++;
}

// Selecting the width of rectangle
// so that difference between any
// two number is 1 only. Here width
// is selected so now flag is again
// unset for next rectangle
else if ((a[i] == a[i + 1] ||
          a[i] - a[i + 1] == 1) && (flag))
{
    // area is calculated for
    // rectangle
    sum = sum + a[i + 1] * len;

    // flag is set false
    // for another rectangle
    // which we can get from
    // elements in array
    flag = false;

    // incrementing the counter
```

```
// one time more as we have
// considered a[i+1] element
// also so.
i++;
}
}

return sum;
}

// Driver code
static public void Main ()
{
    int []a = { 10, 10, 10, 10,
                11, 10, 11, 10,
                9, 9, 8, 8 };
    int n = a.Length;

    Console.WriteLine(
        MaxTotalRectangleArea(a, n));
}
}

// This code is contributed by anuj_67.
```

PHP

```
<?php
// PHP code to find sum
// of all area rectangle
// possible

// Function to find
// area of rectangles
function MaxTotalRectangleArea( $a, $n)
{
    // sorting the array in
    // descending order
    rsort($a);

    // store the final sum of
    // all the rectangles area
    // possible
    $sum = 0;
    $flag = false;

    // temporary variable to store
    // the length of rectangle
```

```
$len;

for ( $i = 0; $i < $n; $i++)
{
    // Selecting the length of
    // rectangle so that difference
    // between any two number is 1
    // only. Here length is selected
    // so flag is set
    if (($a[$i] == $a[$i + 1] or $a[$i] -
        $a[$i + 1] == 1) and (!$flag))
    {
        // flag is set means
        // we have got length of
        // rectangle
        $flag = true;

        // length is set to
        // a[i+1] so that if
        // a[i+1] is less than a[i]
        // by 1 then also we have
        // the correct choice for length
        $len = $a[$i + 1];

        // incrementing the counter
        // one time more as we have
        // considered a[i+1] element
        // also so.
        $i++;
    }

    // Selecting the width of rectangle
    // so that difference between any
    // two number is 1 only. Here width
    // is selected so now flag is again
    // unset for next rectangle
    else if (($a[$i] == $a[$i + 1] or
        $a[$i] - $a[$i + 1] == 1) and
        ($flag))
    {
        // area is calculated for
        // rectangle
        $sum = $sum + $a[$i + 1] * $len;

        // flag is set false
        // for another rectangle
        // which we can get from
```

```
// elements in array
$flag = false;

// incrementing the counter
// one time more as we have
// considered a[i+1] element
// also so.
$i++;
}

}

return $sum;
}

// Driver code
$a = array( 10, 10, 10, 10,
           11, 10, 11, 10,
           9, 9, 8, 8 );
$n = count($a);

echo MaxTotalRectangleArea($a, $n);

//This code is contributed by anuj_67.
?>
```

Output :

282

Time Complexity : **O(nlog(n))**
Auxiliary Space : **O(1)**

Improved By : [vt_m](#), [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/sum-area-rectangles-possible-array/>

Chapter 215

Sum of Manhattan distances between all pairs of points

Sum of Manhattan distances between all pairs of points - GeeksforGeeks

Given n integer coordinates. The task is to find sum of manhattan distance between all pairs of coordinates.

Manhattan Distance between two points (x_1, y_1) and (x_2, y_2) is:

$$|x_1 - x_2| + |y_1 - y_2|$$

Examples :

```
Input : n = 4
        point1 = { -1, 5 }
        point2 = { 1, 6 }
        point3 = { 3, 5 }
        point4 = { 2, 3 }

Output : 22
```

Distance of { 1, 6 }, { 3, 5 }, { 2, 3 } from
{ -1, 5 } are 3, 4, 5 respectively.
Therefore, sum = 3 + 4 + 5 = 12

Distance of { 3, 5 }, { 2, 3 } from { 1, 6 }
are 3, 4 respectively.
Therefore, sum = 12 + 3 + 4 = 19

Distance of { 2, 3 } from { 3, 5 } is 3.
Therefore, sum = 19 + 3 = 22.

Method 1: (Brute Force)

The idea is to run two nested loop i.e for each each point, find manhattan distance for all other points.

```
for (i = 1; i < n; i++)
    for (j = i + 1; j < n; j++)
        sum += ((xi - xj) + (yi - yj))
```

Below is the implementation of this approach:

C++

```
// CPP Program to find sum of Manhattan distance
// between all the pairs of given points
#include <bits/stdc++.h>
using namespace std;

// Return the sum of distance between all
// the pair of points.
int distancesum(int x[], int y[], int n)
{
    int sum = 0;

    // for each point, finding distance to
    // rest of the point
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            sum += (abs(x[i] - x[j]) +
                    abs(y[i] - y[j]));

    return sum;
}

// Driven Program
int main()
{
    int x[] = { -1, 1, 3, 2 };
    int y[] = { 5, 6, 5, 3 };
    int n = sizeof(x) / sizeof(x[0]);
    cout << distancesum(x, y, n) << endl;
    return 0;
}
```

Java

```
// Java Program to find sum of Manhattan distance
// between all the pairs of given points

import java.io.*;
```

```
class GFG {

    // Return the sum of distance between all
    // the pair of points.
    static int distancesum(int x[], int y[], int n)
    {
        int sum = 0;

        // for each point, finding distance to
        // rest of the point
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                sum += (Math.abs(x[i] - x[j]) +
                        Math.abs(y[i] - y[j]));

        return sum;
    }

    // Driven Program
    public static void main(String[] args)
    {
        int x[] = { -1, 1, 3, 2 };
        int y[] = { 5, 6, 5, 3 };
        int n = x.length;

        System.out.println(distancesum(x, y, n));
    }
}

// This code is contributed by vt_m.
```

Python3

```
# Python3 code to find sum of
# Manhattan distance between all
# the pairs of given points

# Return the sum of distance
# between all the pair of points.
def distancesum (x, y, n):
    sum = 0

    # for each point, finding distance
    # to rest of the point
    for i in range(n):
        for j in range(i+1,n):
            sum += (abs(x[i] - x[j]) +
                    abs(y[i] - y[j]))
```

```
return sum

# Driven Code
x = [ -1, 1, 3, 2 ]
y = [ 5, 6, 5, 3 ]
n = len(x)
print(distancesum(x, y, n))

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# Program to find sum of Manhattan distance
// between all the pairs of given points
using System;

class GFG {

    // Return the sum of distance between all
    // the pair of points.
    static int distancesum(int []x, int []y, int n)
    {
        int sum = 0;

        // for each point, finding distance to
        // rest of the point
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                sum += (Math.Abs(x[i] - x[j]) +
                        Math.Abs(y[i] - y[j]));

        return sum;
    }

    // Driven Program
    public static void Main()
    {
        int []x = { -1, 1, 3, 2 };
        int []y = { 5, 6, 5, 3 };
        int n = x.Length;

        Console.WriteLine(distancesum(x, y, n));
    }
}

// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP Program to find sum
// of Manhattan distance
// between all the pairs
// of given points

// Return the sum of distance
// between all the pair of points.
function distancesum( $x, $y, $n)
{
    $sum = 0;

    // for each point, finding
    // distance to rest of
    // the point
    for($i = 0; $i < $n; $i++)
        for ($j = $i + 1; $j < $n; $j++)
            $sum += (abs($x[$i] - $x[$j]) +
                      abs($y[$i] - $y[$j]));

    return $sum;
}

// Driver Code
$x = array(-1, 1, 3, 2);
$y = array(5, 6, 5, 3);
$n = count($x);
echo distancesum($x, $y, $n);

// This code is contributed by anuj_67.
?>
```

Output:

22

Time Complexity: $O(n^2)$

Method 2: (Efficient Approach)

The idea is to use Greedy Approach. First observe, the manhattan formula can be decomposed into two independent sums, one for the difference between x coordinates and the second between y coordinates. If we know how to compute one of them we can use the same method to compute the other. So now we will stick to compute the sum of x coordinates distance.

Let's assume that we know all distances from a point x_i to all values of x 's smaller than x_i . Let's consider other points, the first one not smaller than x_i , and call it x_j . How to compute the distances from x_j to all smaller points ? We can use the corresponding distances from from x_i . Notice that each distance from x_j to some x_k , where $x_k < x_j$ equals the distance

from x_i to x_k plus the distance between x_j and x_i . If there are A points smaller than x_j and S is the sum of distances from x_i to smaller points, then the sum of distances from x_j to smaller points equals $S + (x_j - x_i) * A$.

If we sort all points in non-decreasing order, we can easily compute the desired sum of distances along one axis between each pair of coordinates in $O(N)$ time, processing points from left to right and using the above method.

Also, we don't have to concern if two points are equal coordinates, after sorting points in non-decreasing order, we say that a point x_i is smaller x_j if and only if it appears earlier in the sorted array.

Below is the implementation of this approach:

C++

```
// CPP Program to find sum of Manhattan
// distances between all the pairs of
// given points
#include <bits/stdc++.h>
using namespace std;

// Return the sum of distance of one axis.
int distancesum(int arr[], int n)
{
    // sorting the array.
    sort(arr, arr + n);

    // for each point, finding the distance.
    int res = 0, sum = 0;
    for (int i = 0; i < n; i++) {
        res += (arr[i] * i - sum);
        sum += arr[i];
    }

    return res;
}

int totaldistancesum(int x[], int y[], int n)
{
    return distancesum(x, n) + distancesum(y, n);
}

// Driven Program
int main()
{
    int x[] = { -1, 1, 3, 2 };
    int y[] = { 5, 6, 5, 3 };
    int n = sizeof(x) / sizeof(x[0]);
    cout << totaldistancesum(x, y, n) << endl;
```

```
    return 0;
}
```

Java

```
// Java Program to find sum of Manhattan
// distances between all the pairs of
// given points

import java.io.*;
import java.util.*;

class GFG {

    // Return the sum of distance of one axis.
    static int distancesum(int arr[], int n)
    {

        // sorting the array.
        Arrays.sort(arr);

        // for each point, finding the distance.
        int res = 0, sum = 0;
        for (int i = 0; i < n; i++) {
            res += (arr[i] * i - sum);
            sum += arr[i];
        }

        return res;
    }

    static int totaldistancesum(int x[],
                                int y[], int n)
    {
        return distancesum(x, n) +
               distancesum(y, n);
    }

    // Driven Program
    public static void main(String[] args)
    {

        int x[] = { -1, 1, 3, 2 };
        int y[] = { 5, 6, 5, 3 };
        int n = x.length;
        System.out.println(totaldistancesum(x,
                                             y, n));
    }
}
```

```
}
```

```
// This code is contributed by vt_m.
```

Python3

```
# Python3 code to find sum of Manhattan
# distances between all the pairs of
# given points

# Return the sum of distance of one axis.
def distancesum (arr, n):

    # sorting the array.
    arr.sort()

    # for each point, finding
    # the distance.
    res = 0
    sum = 0
    for i in range(n):
        res += (arr[i] * i - sum)
        sum += arr[i]

    return res

def totaldistancesum( x , y , n ):
    return distancesum(x, n) + distancesum(y, n)

# Driven Code
x = [ -1, 1, 3, 2 ]
y = [ 5, 6, 5, 3 ]
n = len(x)
print(totaldistancesum(x, y, n) )

# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// C# Program to find sum of Manhattan
// distances between all the pairs of
// given points

using System;

class GFG {
```

```
// Return the sum of distance of one axis.  
static int distancesum(int []arr, int n)  
{  
  
    // sorting the array.  
    Array.Sort(arr);  
  
    // for each point, finding the distance.  
    int res = 0, sum = 0;  
    for (int i = 0; i < n; i++) {  
        res += (arr[i] * i - sum);  
        sum += arr[i];  
    }  
  
    return res;  
}  
  
static int totaldistancesum(int []x,  
                           int []y, int n)  
{  
    return distancesum(x, n) +  
           distancesum(y, n);  
}  
  
// Driven Program  
public static void Main()  
{  
  
    int []x = { -1, 1, 3, 2 };  
    int []y = { 5, 6, 5, 3 };  
    int n = x.Length;  
    Console.WriteLine(totaldistancesum(x,  
                                       y, n));  
}  
}  
  
// This code is contributed by vt_m.
```

PHP

```
<?php  
// PHP Program to find sum of  
// Manhattan distances between  
// all the pairs of given points  
  
// Return the sum of  
// distance of one axis.  
function distancesum($arr, $n)
```

```
{  
    // sorting the array.  
    sort($arr);  
  
    // for each point,  
    // finding the distance.  
    $res = 0; $sum = 0;  
    for ($i = 0; $i < $n; $i++)  
    {  
        $res += ($arr[$i] * $i - $sum);  
        $sum += $arr[$i];  
    }  
  
    return $res;  
}  
  
function totaldistancesum($x, $y, $n)  
{  
    return distancesum($x, $n) +  
           distancesum($y, $n);  
}  
  
// Driver Code  
$x = array(-1, 1, 3, 2);  
$y = array(5, 6, 5, 3);  
$n = sizeof($x);  
echo totaldistancesum($x, $y, $n), "\n";  
  
// This code is contributed by m_kit  
?>
```

Output :

22

Time Complexity : O(nlogn)

Improved By : vt_m, jit_t

Source

<https://www.geeksforgeeks.org/sum-manhattan-distances-pairs-points/>

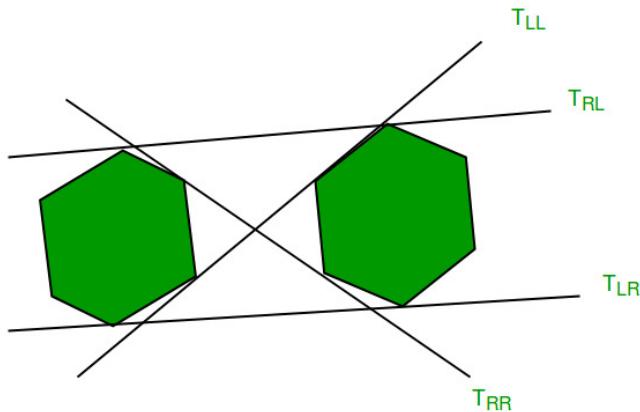
Chapter 216

Tangents between two Convex Polygons

Tangents between two Convex Polygons - GeeksforGeeks

Given two convex polygons, we need to find the lower and upper tangents to these polygons.

As shown in the figure below, $\overrightarrow{T_{RL}}$ and $\overrightarrow{T_{LR}}$ show upper and lower tangent respectively.

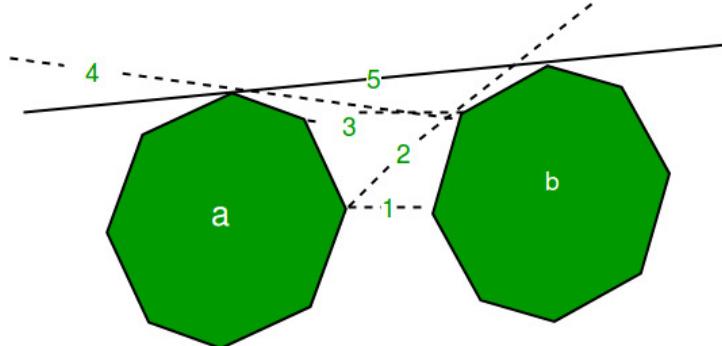


Examples:

```
Input : First Polygon : {(2, 2), (3, 3), (5, 2), (4, 0), (3, 1)}  
        Second Polygon : {(-1, 0), (0, 1), (1, 0), (0, -2)}.  
Output : Upper Tangent - line joining (0,1) and (3,3)  
        Lower Tangent - line joining (0,-2) and (4,0)
```

Overview:

Let's have two convex polygons as shown,



For finding the upper tangent, we start by taking two points. The rightmost point of a and leftmost point of b. The line joining them is labelled as 1. As this line passes through the polygon b (is not above polygon b) so we take the anti-clockwise next point on b, the line is labelled 2. Now the line is above the polygon b, fine! But the line is crossing the polygon a, so we move to the clockwise next point, labelled as 3 in the picture. This again crossing the polygon a so we move to line 4. This line is crossing b so we move to line 5. Now this line is crossing neither of the points. So this is the upper tangent for the given polygons.
For finding the lower tangent we need to move inversely through the polygons i.e. if the line is crossing the polygon b we move to clockwise next and to anti-clockwise next if the line is crossing the polygon a.

Algorithm for upper tangent:

L
Algorithm for lower tangent:

```

L
// C++ program to find upper tangent of two polygons.
#include<bits/stdc++.h>
using namespace std;

// stores the center of polygon (It is made
// global because it is used in compare function)
pair<int,int> mid;

// determines the quadrant of a point
// (used in compare())
int quad(pair<int,int> p)
{
    if (p.first >= 0 && p.second >= 0)
        return 1;
    if (p.first <= 0 && p.second >= 0)
        return 2;
    if (p.first <= 0 && p.second <= 0)
        return 3;
    if (p.first >= 0 && p.second <= 0)
        return 4;
}
```

```

        return 3;
        return 4;
    }

// Checks whether the line is crossing the polygon
int orientation(pair<int,int> a, pair<int,int> b,
                pair<int,int> c)
{
    int res = (b.second-a.second)*(c.first-b.first) -
              (c.second-b.second)*(b.first-a.first);

    if (res == 0)
        return 0;
    if (res > 0)
        return 1;
    return -1;
}

// compare function for sorting
bool compare(pair<int,int> p1, pair<int,int> q1)
{
    pair<int,int> p = make_pair(p1.first - mid.first,
                                p1.second - mid.second);
    pair<int,int> q = make_pair(q1.first - mid.first,
                                q1.second - mid.second);

    int one = quad(p);
    int two = quad(q);

    if (one != two)
        return (one < two);
    return (p.second*q.first < q.second*p.first);
}

// Finds upper tangent of two polygons 'a' and 'b'
// represented as two vectors.
void findUpperTangent(vector<pair<int,int> > a,
                      vector<pair<int,int> > b)
{
    // n1 -> number of points in polygon a
    // n2 -> number of points in polygon b
    int n1 = a.size(), n2 = b.size();

    // To find a point inside the convex polygon(centroid),
    // we sum up all the coordinates and then divide by
    // n(number of points). But this would be a floating-point
    // value. So to get rid of this we multiply points
    // initially with n1 and then find the centre and
}

```

```

// then divided it by n1 again.
// Similarly we do divide and multiply for n2 (i.e.,
// elements of b)

// maxa and minb are used to check if polygon a
// is left of b.
int maxa = INT_MIN;
for (int i=0; i<n1; i++)
{
    maxa = max(maxa, a[i].first);
    mid.first += a[i].first;
    mid.second += a[i].second;
    a[i].first *= n1;
    a[i].second *= n1;
}

// sorting the points in counter clockwise order
// for polygon a
sort(a.begin(), a.end(), compare);

for (int i=0; i<n1; i++)
{
    a[i].first /= n1;
    a[i].second /= n1;
}

mid = {0, 0};

int minb = INT_MAX;
for (int i=0; i<n2; i++)
{
    mid.first += b[i].first;
    mid.second += b[i].second;
    minb = min(minb, b[i].first);
    b[i].first *= n2;
    b[i].second *= n2;
}

// sorting the points in counter clockwise
// order for polygon b
sort(b.begin(), b.end(), compare);

for (int i=0; i<n2; i++)
{
    b[i].first/=n2;
    b[i].second/=n2;
}

```

```

// If a is to the right of b, swap a and b
// This makes sure a is left of b.
if (minb < maxa)
{
    a.swap(b);
    n1 = a.size();
    n2 = b.size();
}

// ia -> rightmost point of a
int ia = 0, ib = 0;
for (int i=1; i<n1; i++)
    if (a[i].first > a[ia].first)
        ia = i;

// ib -> leftmost point of b
for (int i=1; i<n2; i++)
    if (b[i].first < b[ib].first)
        ib=i;

// finding the upper tangent
int inda = ia, indb = ib;
bool done = 0;
while (!done)
{
    done = 1;
    while (orientation(b[indb], a[inda], a[(inda+1)%n1]) > 0)
        inda = (inda + 1) % n1;

    while (orientation(a[inda], b[indb], b[(n2+indb-1)%n2]) < 0)
    {
        indb = (n2+indb-1)%n2;
        done = 0;
    }
}

cout << "upper tangent (" << a[inda].first << ","
     << a[inda].second << ") (" << b[indb].first
     << "," << b[indb].second << ")\n";
}

// Driver code
int main()
{
    vector<pair<int,int> > a;
    a.push_back({2, 2});
    a.push_back({3, 1});
    a.push_back({3, 3});
}

```

```
a.push_back({5, 2});  
a.push_back({4, 0});  
  
vector<pair<int,int> > b;  
b.push_back({0, 1});  
b.push_back({1, 0});  
b.push_back({0, -2});  
b.push_back({-1, 0});  
  
findUpperTangent(a, b);  
  
return 0;  
}
```

Output:

Upper tangent (0,1) (3,3)

Note that the above code only finds upper tangent. We can similarly find lower tangent.

Source

<https://www.geeksforgeeks.org/tangents-two-convex-polygons/>

Chapter 217

Tetradecagonal number

Tetradecagonal number - GeeksforGeeks

Given a number n , the task is to find the n th tetradecagonal number. A tetradecagonal number is 14-sided polygon called tetrakaidecagon or tetradecagon and belongs to the figurative number. The n th tetradecagonal number dotted with some dots and create a series of the pattern. They have a common sharing corner point and dotted with their spaces to each other. The dots continue with n th nested loop.

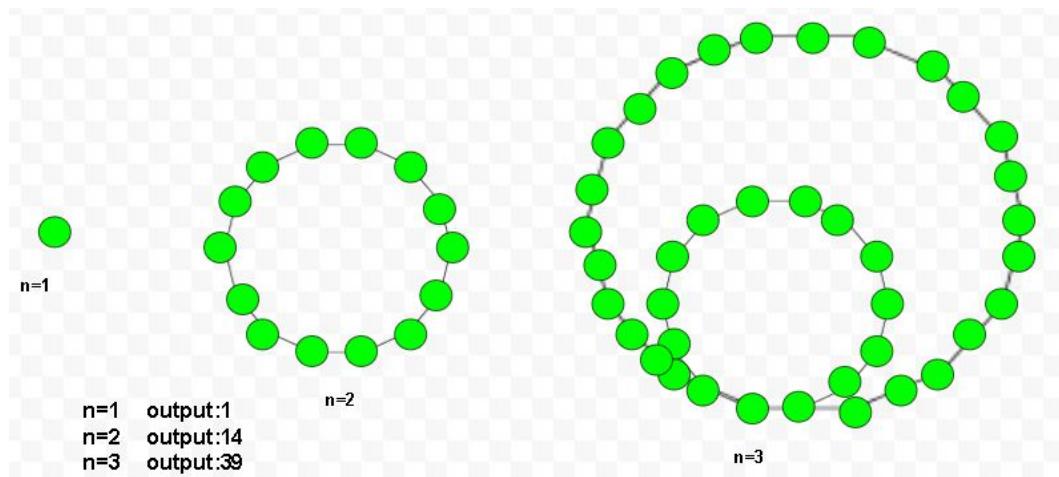
Examples :

Input : 5

Output :125

Input :7

Output :259



Formula for n th tetradecagonal number :-

C++

```
// Program to find nth
// Tetradecagonal number
#include <bits/stdc++.h>
using namespace std;

// Function to find
// Tetradecagonal number
int tetradecagonal_num(int n)
{
    // Formula to calculate nth
    // tetradecagonal number=
    return (12 * n * n - 10 * n) / 2;
}

// Driver Code
int main()
{
    int n = 2;
    cout << n << " th Tetradecagonal number: ";
    cout << tetradecagonal_num(n);
    cout << endl;
    n = 6;
    cout << n << " th Tetradecagonal number: ";
    cout << tetradecagonal_num(n);

    return 0;
}
```

Java

```
// Java Program to find nth
// Tetradecagonal number
import java.io.*;

class GFG
{

    // Function to find
    // Tetradecagonal number
    static int tetradecagonal_num(int n)
    {
        // Formula to calculate nth
```

```
// tetradecagonal number=
return (12 * n * n - 10 * n) / 2;
}

// Driver Code
public static void main (String[] args)
{
int n = 2;
System.out.print(n + " th Tetradecagonal" +
" number: ");
System.out.println(tetradecagonal_num(n));

n = 6;
System.out.print(n + " th Tetradecagonal" +
" number: ");
System.out.print(tetradecagonal_num(n));

}
}
// This code is contributed by m_kit
```

Python 3

```
# Program to find nth
# Tetradecagonal number

# Tetradecagonal number
# number function
def tetradecagonal_num(n) :

    # Formula to calculate
    # nth Tetradecagonal
    # number return it
    # into main function.
    return (12 * n * n -
           10 * n) // 2

# Driver Code
if __name__ == '__main__' :

    n = 2
    print(n,"th Tetradecagonal " +
          "number : " ,
          tetradecagonal_num(n))

    n = 6
    print(n,"th Tetradecagonal " +
```

```
        "number : " ,
tetradecagonal_num(n))

# This code is contributed ajit

C#

// C# Program to find nth
// Tetradecagonal number
using System;

class GFG
{

    // Function to find
    // Tetradecagonal number
    static int tetradecagonal_num(int n)
    {

        // Formula to calculate nth
        // tetradecagonal number
        return (12 * n * n -
               10 * n) / 2;
    }

    // Driver Code
    static public void Main ()
    {
        int n = 2;
        Console.Write(n + "th Tetradecagonal" +
                      " number: ");
        Console.WriteLine(tetradecagonal_num(n));

        n = 6;
        Console.Write(n + "th Tetradecagonal" +
                      " number: ");
        Console.WriteLine(tetradecagonal_num(n));
    }
}

// This code is contributed by ajit
```

PHP

```
<?php
// Program to find nth
// Tetradecagonal number
```

```
// Function to find
// Tetradecagonal number
function tetradecagonal_num($n)
{
    // Formula to calculate nth
    // tetradecagonal number=
    return (12 * $n * $n - 10 * $n) / 2;
}

// Driver Code
$n = 2;
echo $n , " th Tetradecagonal number: ";
echo tetradecagonal_num($n), "\n";

$n = 6;
echo $n , " th Tetradecagonal number: ";
echo tetradecagonal_num($n);

// This code is contributed by aj_36
?>
```

Output :

```
2 th Tetradecagonal number: 14
6 th Tetradecagonal number: 186
```

Reference:
https://en.wikipedia.org/wiki/Polygonal_number

Improved By : [jit_t](#)

Source

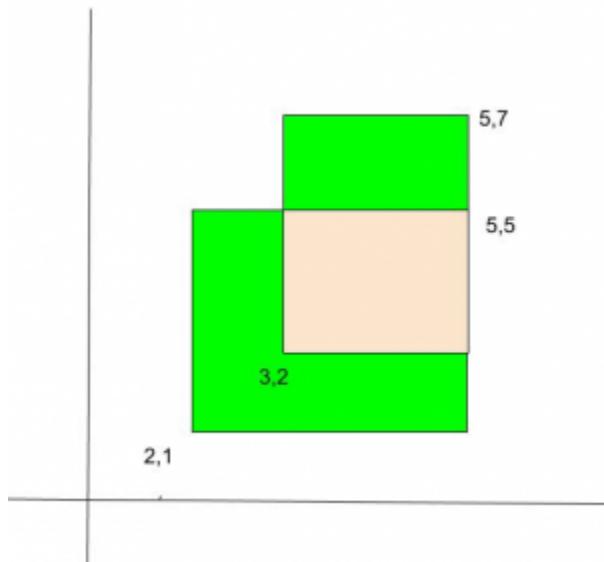
<https://www.geeksforgeeks.org/tetradecagonal-number/>

Chapter 218

Total area of two overlapping rectangles

Total area of two overlapping rectangles - GeeksforGeeks

Given two overlapping rectangles on a plane. We are given bottom left and top right points of the two rectangles. We need to find the total area (Green and pink areas in below diagram).



Examples:

```
Input : Point l1 = {2, 2}, r1 = {5, 7};  
        Point l2 = {3, 4}, r2 = {6, 9};
```

Output : Total Area = 24

```
Input :Point l1 = {2, 1}, r1 = {5, 5};  
        Point l2 = {3, 2}, r2 = {5, 7};  
Output : Total Area = 16
```

Asked in [Juniper](#)

We basically add areas of two rectangles. This includes intersecting part twice, so we subtract area of intersecting part.

```
Total Area = (Area of 1st rectangle +  
              Area of 2nd rectangle) -  
              Area of Intersecting part
```

Area of Rectangle = x_distance * y_distance

Where,

x_distance for 1st rectangle = $\text{abs}(l1.x - r1.x)$
y_distance for 1st rectangle = $\text{abs}(l1.y - r1.y)$

Similarly, we can compute area of 2nd rectangle.

For area of intersecting part,
x_distance for intersecting rectangle =
 $\min(r1.x, r2.x) - \max(l1.x, l2.x)$
y_distance for 1st rectangle =
 $\min(r1.y, r2.y) - \max(l1.y, l2.y)$

```
// C++ program to find total area of two  
// overlapping Rectangles  
#include <bits/stdc++.h>  
using namespace std;  
  
struct Point {  
    int x, y;  
};  
  
// Returns Total Area of two overlap  
// rectangles  
int overlappingArea(Point l1, Point r1,  
                    Point l2, Point r2)  
{  
    // Area of 1st Rectangle  
    int area1 = abs(l1.x - r1.x) *  
               abs(l1.y - r1.y);  
  
    // Area of 2nd Rectangle  
    int area2 = abs(l2.x - r2.x) *
```

```
abs(l2.y - r2.y);

// Length of intersecting part i.e
// start from max(l1.x, l2.x) of
// x-coordinate and end at min(r1.x,
// r2.x) x-coordinate by subtracting
// start from end we get required
// lengths
int areaI = (min(r1.x, r2.x) -
             max(l1.x, l2.x)) *
             (min(r1.y, r2.y) -
              max(l1.y, l2.y));

return (area1 + area2 - areaI);
}

// Driver's Code
int main()
{
    Point l1 = { 2, 2 }, r1 = { 5, 7 };
    Point l2 = { 3, 4 }, r2 = { 6, 9 };
    cout << overlappingArea(l1, r1, l2, r2);
    return 0;
}
```

Output:

24

Source

<https://www.geeksforgeeks.org/total-area-two-overlapping-rectangles/>

Chapter 219

Triangle with no point inside

Triangle with no point inside - GeeksforGeeks

Given N points in 2-dimensional space, we need to find three points such that triangle made by choosing these points should not contain any other points inside. All given points will not lie on the same line so solution will always exist.

Examples:

In above diagram possible triangle with no point inside can be formed by choosing these triplets,

`[(0, 0), (2, 0), (1, 1)]
[(0, 0), (1, 1), (0, 2)]
[(1, 1), (2, 0), (2, 2)]
[(1, 1), (0, 2), (2, 2)]`

So any of the above triplets can be the final answer.

The solution is based on the fact that if there exist triangle(s) with no points inside, then we can form a triangle with any random point among all points.

We can solve this problem by searching all three points one by one. The first point can be chosen randomly. After choosing the first point, we need two points such that their slope should be different and no point should lie inside the triangle of these three points. We can do this by choosing the second point as nearest point to the first and third point as second nearest point with the different slope. For doing this, we first iterate over all points and choose the point which is nearest to the first one and designate that as second point of the required triangle. Then we iterate one more time to find point which has different slope and has the smallest distance, which will be the third point of our triangle.

C++

```
// C/C++ program to find triangle with no point inside
#include <bits/stdc++.h>
using namespace std;

// method to get square of distance between
// (x1, y1) and (x2, y2)
int getDistance(int x1, int y1, int x2, int y2)
{
    return (x2 - x1)*(x2 - x1) +
           (y2 - y1)*(y2 - y1);
}

// Method prints points which make triangle with no
// point inside
void triangleWithNoPointInside(int points[][][2], int N)
{
    //      any point can be chosen as first point of triangle
    int first = 0;
    int second, third;
    int minD = INT_MAX;

    // choose nearest point as second point of triangle
    for (int i = 0; i < N; i++)
    {
        if (i == first)
            continue;

        // Get distance from first point and choose
        // nearest one
        int d = getDistance(points[i][0], points[i][1],
                            points[first][0], points[first][1]);
        if (minD > d)
        {
            minD = d;
            second = i;
        }
    }

    // Pick third point by finding the second closest
    // point with different slope.
    minD = INT_MAX;
    for (int i = 0; i < N; i++)
    {
        // if already chosen point then skip them
        if (i == first || i == second)
            continue;

        // get distance from first point
```

```

int d = getDistance(points[i][0], points[i][1],
                    points[first][0], points[first][1]);

/* the slope of the third point with the first
point should not be equal to the slope of
second point with first point (otherwise
they'll be collinear) and among all such
points, we choose point with the smallest
distance */

// here cross multiplication is compared instead
// of division comparison
if ((points[i][0] - points[first][0]) *
    (points[second][1] - points[first][1]) !=
    (points[second][0] - points[first][0]) *
    (points[i][1] - points[first][1]) &&
    minD > d)
{
    minD = d;
    third = i;
}
}

cout << points[first][0] << ", "
    << points[first][1] << endl;
cout << points[second][0] << ", "
    << points[second][1] << endl;
cout << points[third][0] << ", "
    << points[third][1] << endl;
}

// Driver code to test above methods
int main()
{
    int points[][][2] = {{0, 0}, {0, 2}, {2, 0},
                        {2, 2}, {1, 1}};
    int N = sizeof(points) / sizeof(points[0]);
    triangleWithNoPointInside(points, N);
    return 0;
}

```

Java

```

// Java program to find triangle
// with no point inside
import java.io.*;

class GFG
{

```

```
// method to get square of distance between
// (x1, y1) and (x2, y2)
static int getDistance(int x1, int y1, int x2, int y2)
{
    return (x2 - x1)*(x2 - x1) +
           (y2 - y1)*(y2 - y1);
}

// Method prints points which make triangle with no
// point inside
static void triangleWithNoPointInside(int points[][] , int N)
{
    // any point can be chosen as first point of triangle
    int first = 0;
    int second =0;
    int third =0;
    int minD = Integer.MAX_VALUE;

    // choose nearest point as second point of triangle
    for (int i = 0; i < N; i++)
    {
        if (i == first)
            continue;

        // Get distance from first point and choose
        // nearest one
        int d = getDistance(points[i][0], points[i][1],
                            points[first][0], points[first][1]);
        if (minD > d)
        {
            minD = d;
            second = i;
        }
    }

    // Pick third point by finding the second closest
    // point with different slope.
    minD = Integer.MAX_VALUE;
    for (int i = 0; i < N; i++)
    {
        // if already chosen point then skip them
        if (i == first || i == second)
            continue;

        // get distance from first point
        int d = getDistance(points[i][0], points[i][1],
                            points[first][0], points[first][1]);
    }
}
```

```

/* the slope of the third point with the first
   point should not be equal to the slope of
   second point with first point (otherwise
   they'll be collinear) and among all such
   points, we choose point with the smallest
   distance */
// here cross multiplication is compared instead
// of division comparison
if ((points[i][0] - points[first][0]) *
    (points[second][1] - points[first][1]) !=
    (points[second][0] - points[first][0]) *
    (points[i][1] - points[first][1])) &&
    minD > d)
{
    minD = d;
    third = i;
}
}

System.out.println(points[first][0] + " , "
+ points[first][1]);
System.out.println(points[second][0]+ " , "
+ points[second][1]) ;
System.out.println(points[third][0] +", "
+ points[third][1]);
}

// Driver code
public static void main (String[] args)
{
    int points[][] = {{0, 0}, {0, 2}, {2, 0},
                      {2, 2}, {1, 1}};
    int N = points.length;
    triangleWithNoPointInside(points, N);
}
}

// This article is contributed by vt_m.

```

Output:

```

0, 0
1, 1
0, 2

```

Source

<https://www.geeksforgeeks.org/triangle-no-point-inside/>

Chapter 220

Triangular Matchstick Number

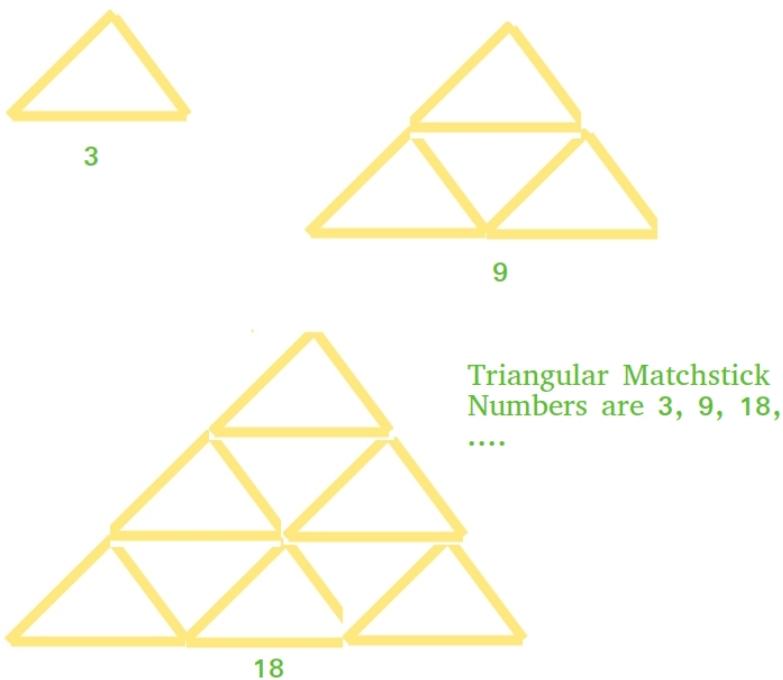
Triangular Matchstick Number - GeeksforGeeks

Given a number X which represents the floor of a matchstick pyramid, write a program to print the total number of matchstick required to form pyramid of matchsticks of x floors.

Examples:

```
Input : X = 1
Output : 3
Input : X = 2
Output : 9
```

This is mainly an extension of [triangular numbers](#). For a number X, the matchstick required will be three times of X-th [triangular numbers](#), i.e., $(3*X*(X+1))/2$



C++

```
// C++ program to find X-th triangular
// matchstick number

#include <bits/stdc++.h>
using namespace std;

int numberOfSticks(int x)
{
    return (3 * x * (x + 1)) / 2;
}

int main()
{
    cout<<numberOfSticks(7);
    return 0;
}
```

Java

```
// Java program to find X-th triangular
// matchstick number
public class TriangularPyramidNumber {
    public static int numberOfSticks(int x)
    {
```

```
        return (3 * x * (x + 1)) / 2;
    }
    public static void main(String[] args)
    {
        System.out.println(numberOfSticks(7));
    }
}
```

Python3

```
# Python program to find X-th triangular
# matchstick number

def numberOfSticks(x):
    return (3 * x * (x + 1)) / 2

# main()
print(int(numberOfSticks(7)))
```

C#

```
// C# program to find X-th triangular
// matchstick number
using System;

class GFG
{
    // Function to ind missing number
    static int numberOfSticks(int x)
    {
        return (3 * x * (x + 1)) / 2;
    }

    public static void Main()
    {
        Console.WriteLine(numberOfSticks(7));
    }
}

// This code is contributed by _omg
```

PHP

```
<?php
// PHP program to find
// X-th triangular
```

```
// matchstick number

function numberOfSticks($x)
{
    return (3 * $x * ($x + 1)) / 2;
}

// Driver code
echo(numberOfSticks(7));

// This code is contributed by Ajit.
?>
```

Output:

84

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/triangular-matchstick-number/>

Chapter 221

Ways to choose three points with distance between the most distant points $\leq L$

Ways to choose three points with distance between the most distant points $\leq L$ - Geeks-forGeeks

Given a set of n distinct points $x_1, x_2, x_3 \dots x_n$ all lying on the X-axis and an integer L, the task is to find the number of ways of selecting three points such that the distance between the most distant points is less than or equal to L

Note: Order is not important i.e the points {3, 2, 1} and {1, 2, 3} represent the same set of three points

Examples:

Input : $x = \{1, 2, 3, 4\}$

$L = 3$

Output : 4

Explanation:

Ways to select three points such that the distance between the most distant points $\leq L$ are:

- 1) {1, 2, 3} Here distance between farthest points = $3 - 1 = 2 \leq L$
- 2) {1, 2, 4} Here distance between farthest points = $4 - 1 = 3 \leq L$
- 3) {1, 3, 4} Here distance between farthest points = $4 - 1 = 3 \leq L$
- 4) {2, 3, 4} Here distance between farthest points = $4 - 2 = 2 \leq L$

Thus, total number of ways = 4

Naive Approach:

First of all, sort the array of points to generate triplets {a, b, c} such that a and c are the farthest points of the triplet and $a < b < c$, since all the points are distinct. We can

generate all the possible triplets and check for the condition if the distance between the two most distant points in $\leq L$. If it holds we count this way, else we don't

C++

```
// C++ program to count ways to choose
// triplets such that the distance
// between the farthest points <= L
#include<bits/stdc++.h>
using namespace std;

// Returns the number of triplets with
// distance between farthest points <= L
int countTripletsLessThanL(int n, int L, int* arr)
{
    // sort to get ordered triplets so that we can
    // find the distance between farthest points
    // belonging to a triplet
    sort(arr, arr + n);

    int ways = 0;

    // generate and check for all possible
    // triplets: {arr[i], arr[j], arr[k]}
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            for (int k = j + 1; k < n; k++) {

                // Since the array is sorted the
                // farthest points will be a[i]
                // and a[k];
                int mostDistantDistance = arr[k] - arr[i];
                if (mostDistantDistance <= L) {
                    ways++;
                }
            }
        }
    }

    return ways;
}

// Driver Code
int main()
{
    // set of n points on the X axis
    int arr[] = { 1, 2, 3, 4 };
```

```
int n = sizeof(arr) / sizeof(arr[0]);
int L = 3;
int ans = countTripletsLessThanL(n, L, arr);
cout << "Total Number of ways = " << ans << "\n";
return 0;
}
```

Java

```
// Java program to count ways to choose
// triplets such that the distance
// between the farthest points  $\leq L$ 
import java .io.*;
import java .util.Arrays;
class GFG {

    // Returns the number of triplets with
    // distance between farthest points  $\leq L$ 
    static int countTripletsLessThanL(int n, int L,
                                      int []arr)
    {

        // sort to get ordered triplets
        // so that we can find the
        // distance between farthest
        // points belonging to a triplet
        Arrays.sort(arr);

        int ways = 0;

        // generate and check for all possible
        // triplets: {arr[i], arr[j], arr[k]}
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                for (int k = j + 1; k < n; k++) {

                    // Since the array is sorted the
                    // farthest points will be a[i]
                    // and a[k];
                    int mostDistantDistance =
                        arr[k] - arr[i];
                    if (mostDistantDistance  $\leq L$ )
                    {
                        ways++;
                    }
                }
            }
        }
    }
}
```

```
        return ways;
    }

// Driver Code
static public void main (String[] args)
{
    // set of n points on the X axis
    int []arr = {1, 2, 3, 4};

    int n = arr.length;
    int L = 3;
    int ans = countTripletsLessThanL(n, L, arr);
    System.out.println("Total Number of ways = "
                        + ans);
}
}

// This code is contributed by anuj_67.
```

C#

```
// C# program to count ways to choose
// triplets such that the distance
// between the farthest points <= L
using System;
class GFG {

    // Returns the number of triplets with
    // distance between farthest points <= L
    static int countTripletsLessThanL(int n, int L,
                                      int []arr)
    {

        // sort to get ordered triplets
        // so that we can find the
        // distance between farthest
        // points belonging to a triplet
        Array.Sort(arr);

        int ways = 0;

        // generate and check for all possible
        // triplets: {arr[i], arr[j], arr[k]}
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                for (int k = j + 1; k < n; k++) {
```

```
// Since the array is sorted the
// farthest points will be a[i]
// and a[k];
int mostDistantDistance = arr[k] - arr[i];
if (mostDistantDistance <= L)
{
    ways++;
}
}

return ways;
}

// Driver Code
static public void Main ()
{
    // set of n points on the X axis
    int []arr = {1, 2, 3, 4};

    int n = arr.Length;
    int L = 3;
    int ans = countTripletsLessThanL(n, L, arr);
    Console.WriteLine("Total Number of ways = " + ans);
}
}

// This code is contributed by anuj_67.
```

Output

Total Number of ways = 4

Time Complexity: $O(n^3)$ for generating all possible triplets.

Efficient Approach:

- This problem can be solved by using Binary search.
- First of all, sort the array.
- Now, for each element of the array we find the number of elements which are greater than it (by maintaining a sorted order of points) and lie in the range $(x_i + 1, x_i + L)$ both inclusive (Note that here all points are distinct so we need consider the elements equal to x_i itself).

- Doing so we find all such points where the distance between the farthest points will always be less than or equal to L .
- Now let's say for the i^{th} point, we have M such points which are less than or equal to $x_i + L$, then the number of ways we can select 2 points from M such points is simply $M * (M - 1) / 2$

```

// C++ program to count ways to choose
// triplets such that the distance between
// the farthest points <= L */
#include<bits/stdc++.h>
using namespace std;

// Returns the number of triplets with the
// distance between farthest points <= L
int countTripletsLessThanL(int n, int L, int* arr)
{
    // sort the array
    sort(arr, arr + n);

    int ways = 0;
    for (int i = 0; i < n; i++) {

        // find index of element greater than arr[i] + L
        int indexGreater = upper_bound(arr, arr + n,
                                         arr[i] + L) - arr;

        // find Number of elements between the ith
        // index and indexGreater since the Numbers
        // are sorted and the elements are distinct
        // from the points btw these indices represent
        // points within range (a[i] + 1 and a[i] + L)
        // both inclusive

        int numberOfElements = indexGreater - (i + 1);

        // if there are at least two elements in between
        // i and indexGreater find the Number of ways
        // to select two points out of these

        if (numberOfElements >= 2) {
            ways += (numberOfElements
                      * (numberOfElements - 1) / 2);
        }
    }

    return ways;
}

```

```
// Driver Code
int main()
{
    // set of n points on the X axis
    int arr[] = { 1, 2, 3, 4 };

    int n = sizeof(arr) / sizeof(arr[0]);
    int L = 4;

    int ans = countTripletsLessThanL(n, L, arr);

    cout << "Total Number of ways = " << ans << "\n";

    return 0;
}
```

Output

Total Number of ways = 4

Time Complexity:**O(NlogN)** where N is the number of points.

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/ways-choose-three-points-distance-distant-points-l/>

Chapter 222

n'th Pentagonal Number

n'th Pentagonal Number - GeeksforGeeks

Given an integer n, find the nth Pentagonal number. First three pentagonal numbers are 1, 5 and 12 (Please see below diagram).

The n'th pentagonal number P_n is the number of distinct dots in a pattern of dots consisting of the outlines of regular pentagons with sides up to n dots, when the pentagons are overlaid so that they share one vertex [Source [Wiki](#)]

Examples :

Input: n = 1
Output: 1

Input: n = 2
Output: 5

Input: n = 3
Output: 12

In general, a [polygonal number](#)(triangular number, square number, etc) is a number represented as dots or pebbles arranged in the shape of a regular polygon. The first few pentagonal numbers are: 1, 5, 12, etc.

If s is the number of sides in a polygon, the formula for the nth s-gonal number $P(s, n)$ is

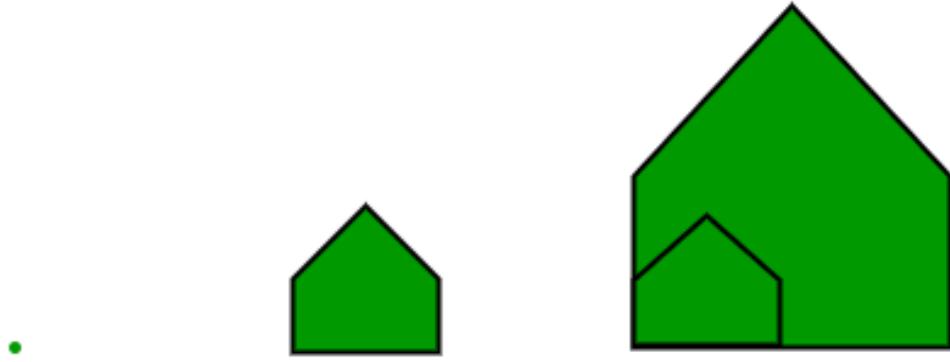
$$\text{nth s-gonal number } P(s, n) = (s - 2)n(n-1)/2 + n$$

If we put s = 5, we get

$$\text{n'th Pentagonal number } P_n = 3*n*(n-1)/2 + n$$

Examples:

Pentagonal Number



Below are the implementations of above idea in different programming languages.

C/C++

```
// C program for above approach
#include <stdio.h>
#include <stdlib.h>

// Finding the nth Pentagonal Number
int pentagonalNum(int n)
{
    return (3*n*n - n)/2;
}

// Driver program to test above function
int main()
{
    int n = 10;
    printf("10th Pentagonal Number is = %d \n \n",
           pentagonalNum(n));

    return 0;
}
```

Java

```
// Java program for above approach
class Pentagonal
{
    int pentagonalNum(int n)
```

```
{  
    return (3*n*n - n)/2;  
}  
}  
  
public class GeeksCode  
{  
    public static void main(String[] args)  
    {  
        Pentagonal obj = new Pentagonal();  
        int n = 10;  
        System.out.printf("10th pentagonal number is = "  
                          + obj.pentagonalNum(n));  
    }  
}
```

Python

```
# Python program for finding pentagonal numbers  
def pentagonalNum( n ):  
    return (3*n*n - n)/2  
#Script Begins  
  
n = 10  
print "10th Pentagonal Number is = ", pentagonalNum(n)  
  
#Scripts Ends
```

C#

```
// C# program for above approach  
using System;  
  
class GFG {  
  
    static int pentagonalNum(int n)  
    {  
        return (3 * n * n - n) / 2;  
    }  
  
    public static void Main()  
    {  
        int n = 10;  
  
        Console.WriteLine("10th pentagonal"  
                          + " number is = " + pentagonalNum(n));  
    }  
}
```

```
}
```

```
// This code is contributed by vt_m.
```

PHP

```
<?php
// PHP program for above approach

// Finding the nth Pentagonal Number
function pentagonalNum($n)
{
    return (3 * $n * $n - $n) / 2;
}

// Driver Code
$n = 10;
echo "10th Pentagonal Number is = ",
      pentagonalNum($n);

// This code is contributed by ajit
?>
```

Output :

```
10th Pentagonal Number is = 145
```

Reference:

https://en.wikipedia.org/wiki/Polygonal_number

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/nth-pentagonal-number/>