

Contents

1 1 to n bit numbers with no consecutive 1s in binary representation.	12
Source	15
2 Alexander Bogomolny's UnOrdered Permutation Algorithm	16
Source	21
3 All possible binary numbers of length n with equal sum in both halves	22
Source	24
4 Backtracking Introduction	25
Source	28
5 Balanced expressions such that given positions have opening brackets Set 2	29
Source	33
6 Bank Of America (BA Continuum India Pvt. Ltd.) Campus Recruitment	34
Source	35
7 Binary to Gray code using recursion	36
Source	41
8 Bitwise recursive addition of two integers	42
Source	45
9 Check for balanced parenthesis without using stack	46
Source	48
10 Check if a destination is reachable from source with two movements allowed	49
Source	54
11 Check if a given string is sum-string	55
Source	58
12 Check if a number is Palindrome	59
Source	68
13 Check if linked list is sorted (Iterative and Recursive)	69

Source	72
14 Combinational Sum	73
Source	76
15 Combinations in a String of Digits	77
Source	80
16 Convert a Binary Tree such that every node stores the sum of all nodes in its right subtree	81
Source	84
17 Count all possible paths from top left to bottom right of a mXn matrix	85
Source	91
18 Count consonants in a string (Iterative and recursive methods)	92
Source	99
19 Count number of ways to partition a set into k subsets	100
Source	105
20 Count occurrences of a substring recursively	106
Source	110
21 Count subtrees that sum up to a given value x	111
Source	117
22 Count ways to express a number as sum of powers	118
Source	123
23 Decimal to binary number using recursion	124
Source	128
24 Decode a string recursively encoded as count followed by substring	129
Source	135
25 Delete a linked list using recursion	136
Source	137
26 Delete middle element of a stack	138
Source	144
27 Diagonal Sum of a Binary Tree	145
Source	150
28 Equal sum array partition excluding a given element	151
Source	160
29 Euler Tour Subtree Sum using Segment Tree	161
Source	168

30 Find Nth term (A matrix exponentiation example)	169
Source	172
31 Find all even length binary sequences with same sum of first and second half bits	173
Source	179
32 Find m-th summation of first n natural numbers.	180
Source	184
33 Find maximum level product in Binary Tree	185
Source	188
34 Find middle of singly linked list Recursively	189
Source	191
35 Find n-th lexicographically permutation of a string Set 2	192
Source	199
36 Find n-th node in Postorder traversal of a Binary Tree	200
Source	202
37 Find profession in a special family	203
Source	211
38 Find ways an Integer can be expressed as sum of n-th power of unique natural numbers	212
Source	218
39 First uppercase letter in a string (Iterative and Recursive)	219
Source	226
40 Flatten a binary tree into linked list	227
Source	231
41 Flood fill Algorithm – how to implement fill() in paint?	232
Source	235
42 Function to copy string (Iterative and Recursive)	236
Source	238
43 Generate all binary strings without consecutive 1's	239
Source	241
44 Generate all passwords from given character set	242
Source	246
45 Generate all possible sorted arrays from alternate elements of two given sorted arrays	247
Source	253

46 Generating all possible Subsequences using Recursion	254
Source	256
47 Generating subarrays using recursion	257
Source	258
48 Given a matrix of ‘O’ and ‘X’, replace ‘O’ with ‘X’ if surrounded by ‘X’	259
Source	270
49 Given a string, print all possible palindromic partitions	271
Source	273
50 Given an array A[] and a number x, check for pair in A[] with sum as x	274
Source	288
51 Happy Number	289
Source	295
52 Highest power of 2 less than or equal to given number	296
Source	306
53 How to print maximum number of A’s using given four keys	307
Source	316
54 How will you print numbers from 1 to 100 without using loop?	317
Source	320
55 How will you print numbers from 1 to 100 without using loop? Set-2	321
Source	322
56 Identify all Grand-Parent Nodes of each Node in a Map	323
Source	324
57 Java 8 Arrays parallelSort() method with Examples	325
Source	329
58 Java Applet Implementing Flood Fill algorithm	330
Source	343
59 Josephus problem Set 1 (A O(n) Solution)	344
Source	348
60 Last non-zero digit of a factorial	349
Source	354
61 Leaf nodes from Preorder of a Binary Search Tree (Using Recursion)	355
Source	358
62 Level order traversal in spiral form	359
Source	368

63 Level order traversal with direction change after every two levels Recursive Approach	369
Source	372
64 Lexicographically Kth smallest way to reach given coordinate from origin	373
Source	378
65 Longest consecutive sequence in Binary tree	379
Source	381
66 Max sum of M non-overlapping subarrays of size K	382
Source	388
67 Maximize array elements upto given number	389
Source	406
68 Maximum length of segments of 0's and 1's	407
Source	409
69 Memoization (1D, 2D and 3D)	410
Source	422
70 Merge Sort for Linked Lists in JavaScript	423
Source	427
71 Minimum steps to reach a destination	428
Source	434
72 Minimum tiles of sizes in powers of two to cover whole area	435
Source	440
73 Modular exponentiation (Recursive)	441
Source	446
74 Moser-de Bruijn Sequence	447
Source	458
75 Mutual Recursion with example of Hofstadter Female and Male sequences	459
Source	464
76 N Queen in O(n) space	465
Source	468
77 Non-crossing lines to connect points in a circle	469
Source	476
78 Number of handshakes such that a person shakes hands only once	477
Source	484
79 Number of non-negative integral solutions of sum equation	485
Source	490

80	Number of ways to divide a given number as a set of integers in decreasing order	491
	Source	493
81	Partition given string in such manner that i'th substring is sum of (i-1)'th and (i-2)'th substring	494
	Source	497
82	Perform n steps to convert every digit of a number in the format [count][digit]	498
	Source	500
83	Power Set in Lexicographic order	501
	Source	503
84	Practice Questions for Recursion Set 1	504
	Source	505
85	Practice Questions for Recursion Set 2	506
	Source	507
86	Practice Questions for Recursion Set 3	508
	Source	509
87	Practice Questions for Recursion Set 4	510
	Source	512
88	Practice Questions for Recursion Set 5	513
	Source	515
89	Practice Questions for Recursion Set 6	516
	Source	518
90	Practice Questions for Recursion Set 7	519
	Source	521
91	Practice questions for Linked List and Recursion	522
	Source	525
92	Print 1 to 100 in C++, without loop and recursion	526
	Source	528
93	Print N-bit binary numbers having more 1's than 0's in all prefixes	529
	Source	533
94	Print a pattern without using any loop	534
	Source	543
95	Print all combinations of factors (Ways to factorize)	544
	Source	547

96 Print all distinct integers that can be formed by K numbers from a given array of N numbers	548
Source	551
97 Print all increasing sequences of length k from first n natural numbers	552
Source	561
98 Print all leaf nodes of a Binary Tree from left to right	562
Source	564
99 Print all longest common sub-sequences in lexicographical order	565
Source	568
100 Print all n-digit strictly increasing numbers	569
Source	573
101 Print all non-increasing sequences of sum equal to a given number x	574
Source	583
102 Print all possible combinations of r elements in a given array of size n	584
Source	598
103 Print all possible expressions that evaluate to a target	599
Source	602
104 Print all possible strings of length k that can be formed from a set of n characters	603
Source	607
105 Print all possible strings that can be made by placing spaces	608
Source	613
106 Print all possible words from phone digits	614
Source	617
107 Print all sequences starting with n and consecutive difference limited to k	618
Source	625
108 Print all subsequences of a string	626
Source	629
109 Print all the combinations of a string in lexicographical order	630
Source	633
110 Print alternate nodes of a linked list using recursion	634
Source	636
111 Print sums of all subsets of a given set	637
Source	642

112 Product of 2 Numbers using Recursion	643
Source	647
113 Program for Chocolate and Wrapper Puzzle	648
Source	658
114 Program for Sum the digits of a given number	659
Source	667
115 Program for length of a string using recursion	668
Source	672
116 Program to find amount of water in a given glass	673
Source	682
117 Program to find the minimum (or maximum) element of an array	683
Source	689
118 Program to implement Collatz Conjecture	690
Source	694
119 Program to reverse a string (Iterative and Recursive)	695
Source	702
120 Python Convert a nested list into a flat list	703
Source	704
121 Recaman's sequence	705
Source	712
122 Recursion	713
Source	720
123 Recursive Approach to find nth node from the end in the linked list	721
Source	725
124 Recursive Bubble Sort	726
Source	731
125 Recursive Functions	732
Source	739
126 Recursive Implementation of atoi()	740
Source	741
127 Recursive Insertion Sort	742
Source	748
128 Recursive Practice Problems with Solutions	749
129 Category Archives: Recursion (Recent articles based on Recursion)	752

130 Practice Problems on Geeks for Geeks!	753
Source	753
131 Recursive Tower of Hanoi using 4 pegs / rods	754
Source	760
132 Recursive approach for alternating split of Linked List	761
Source	763
133 Recursive function to delete k-th node from linked list	764
Source	766
134 Recursive insertion and traversal linked list	767
Source	769
135 Recursive program for prime number	770
Source	774
136 Recursive program to generate power set	775
Source	778
137 Recursive program to print formula for GCD of n integers	779
Source	782
138 Recursive solution to count substrings with same first and last characters	783
Source	788
139 Recursively Reversing a linked list (A simple implementation)	789
Source	791
140 Recursively remove all adjacent duplicates	792
Source	799
141 Reduce a number to 1 by performing given operations	800
Source	802
142 Remove duplicates from a sorted linked list using recursion	803
Source	805
143 Reverse a Doubly linked list using recursion	806
Source	809
144 Reverse a stack using recursion	810
Source	820
145 Reversing a queue using recursion	821
Source	827
146 Shuffle 2n integers in format {a1, b1, a2, b2, a3, b3,, an, bn} without using extra space	828
Source	834

147 Smallest number in BST which is greater than or equal to N	835
Source	838
148 Solve the Crossword Puzzle	839
Source	844
149 Sort a stack using a temporary stack	845
Source	851
150 Sort a stack using recursion	852
Source	859
151 String with additive sequence	860
Source	864
152 Sum of all elements of N-ary Tree	865
Source	868
153 Sum of digit of a number using recursion	869
Source	873
154 Sum of elements of all partitions of number such that no element is less than K	874
Source	879
155 Sum of natural numbers using recursion	880
Source	883
156 Sum triangle from array	884
Source	884
157 Tail Recursion	885
Source	891
158 Tail recursion to calculate sum of array elements.	892
Source	893
159 Tiling with Dominoes	894
Source	900
160 Time Complexity Analysis Tower Of Hanoi (Recursion)	901
Source	902
161 Two Dimensional Segment Tree Sub-Matrix Sum	903
Source	910
162 Vertical width of Binary tree Set 2	911
Source	914
163 Water Jug Problem using Memoization	915
Source	917

164 Word Break Problem using Backtracking	918
Source	920
165 Write a program to print all permutations of a given string	921
Source	927
166 Write a program to reverse digits of a number	928
Source	933
167 nth Rational number in Calkin-Wilf sequence	934
Source	938

Chapter 1

1 to n bit numbers with no consecutive 1s in binary representation.

1 to n bit numbers with no consecutive 1s in binary representation. - GeeksforGeeks

Given a number n, our task is to find all 1 to n bit numbers with no consecutive 1s in their binary representation.

Examples:

```
Input : n = 4
Output : 1 2 4 5 8 9 10
These are numbers with 1 to 4
bits and no consecutive ones in
binary representation.
```

```
Input : n = 3
Output : 1 2 4 5
```

We add bits one by one and recursively print numbers. For every last bit, we have two choices.

```
if last digit in sol is 0 then
    we can insert 0 or 1 and recur.
else if last digit is 1 then
    we can insert 0 only and recur.
```

We will use recursion-

1. We make a solution vector sol and insert first bit 1 in it which will be the first number.
2. Now we check whether length of solution vector is less than or equal to n or not.
3. If it is so then we calculate the decimal number and store it into a map as it store numbers in sorted order.
4. Now we will have two conditions-
 - if last digit in sol is 0 the we can insert 0 or 1 and recur.
 - else if last digit is 1 then we can insert 0 only and recur.

```
numberWithNoConsecutiveOnes(n, sol)
{
    if sol.size() <= n

        // calculate decimal and store it
        if last element of sol is 1
            insert 0 in sol
            numberWithNoConsecutiveOnes(n, sol)
        else
            insert 1 in sol
            numberWithNoConsecutiveOnes(n, sol)

        // because we have to insert zero
        // also in place of 1
        sol.pop_back();
        insert 0 in sol
        numberWithNoConsecutiveOnes(n, sol)
    }

    // CPP program to find all numbers with no
    // consecutive 1s in binary representation.
    #include <bits/stdc++.h>

    using namespace std;
    map<int, int> h;

    void numberWithNoConsecutiveOnes(int n, vector<int>
                                     sol)
    {
        // If it is in limit i.e. of n lengths in
        // binary
        if (sol.size() <= n) {
            int ans = 0;
            for (int i = 0; i < sol.size(); i++)
                ans += pow((double)2, i) *
                    sol[sol.size() - 1 - i];
            h[ans] = 1;
        }
    }
}
```

```
// Last element in binary
int last_element = sol[sol.size() - 1];

// if element is 1 add 0 after it else
// If 0 you can add either 0 or 1 after that
if (last_element == 1) {
    sol.push_back(0);
    numberWithNoConsecutiveOnes(n, sol);
} else {
    sol.push_back(1);
    numberWithNoConsecutiveOnes(n, sol);
    sol.pop_back();
    sol.push_back(0);
    numberWithNoConsecutiveOnes(n, sol);
}
}

// Driver program
int main()
{
    int n = 4;
    vector<int> sol;

    // Push first number
    sol.push_back(1);

    // Generate all other numbers
    numberWithNoConsecutiveOnes(n, sol);

    for (map<int, int>::iterator i = h.begin();
         i != h.end(); i++)
        cout << i->first << " ";
    return 0;
}
```

Output:

1 2 4 5 8 9 10

Related Post :

[Count number of binary strings without consecutive 1's](#)

Improved By : [Kishore Srinivas](#)

Source

<https://www.geeksforgeeks.org/1-to-n-bit-numbers-with-no-consecutive-1s-in-binary-representation/>

Chapter 2

Alexander Bogomolny's UnOrdered Permutation Algorithm

Alexander Bogomolny's UnOrdered Permutation Algorithm - [GeeksforGeeks](#)

The Alexander Bogomolyn's algorithm is used to permute first N natural numbers. Given the value of N we have to output all the permutations of numbers from 1 to N.

Examples:

```
Input : 2
Output : 1 2
         2 1
```

```
Input : 3
Output : 1 2 3
         1 3 2
         2 1 3
         3 1 2
         2 3 1
         3 2 1
```

The idea is to maintain an array to store the current permutation. A static integer level variable is used to define these permutations.

1. It initializes the value of current level and permutes the remaining values to the higher levels.
2. As the assigning action of the values reaches to the highest level, it prints the permutation obtained.

3. This approach is recursively implemented to obtain all possible permutations.

C++

```
// CPP program to implement Alexander
// Bogomolny's UnOrdered Permutation Algorithm
#include <iostream>
using namespace std;

// A function to print the permutation.
void print(int perm[], int N)
{
    for (int i = 0; i < N; i++)
        cout << " " << perm[i];
    cout << "\n";
}

// A function implementing Alexander Bogomolyn
// algorithm.
void AlexanderBogomolyn(int perm[], int N, int k)
{
    static int level = -1;

    // Assign level to zero at start.
    level = level + 1;
    perm[k] = level;

    if (level == N)
        print(perm, N);
    else
        for (int i = 0; i < N; i++)

            // Assign values to the array
            // if it is zero.
            if (perm[i] == 0)
                AlexanderBogomolyn(perm, N, i);

    // Decrement the level after all possible
    // permutation after that level.
    level = level - 1;

    perm[k] = 0;
}

// Driver Function
int main()
{
    int i, N = 3;
```

```
    int perm[N] = { 0 };
    AlexanderBogomolyn(perm, N, 0);
    return 0;
}
```

Java

```
// Java program to implement
// Alexander Bogomolny UnOrdered
// Permutation Algorithm
import java.io.*;

class GFG
{
    static int level = -1;

    // A function to print
    // the permutation.
    static void print(int perm[], int N)
    {
        for (int i = 0; i < N; i++)
            System.out.print(" " + perm[i]);
        System.out.println();
    }

    // A function implementing
    // Alexander Bogomolyn algorithm.
    static void AlexanderBogomolyn(int perm[],
                                    int N, int k)
    {

        // Assign level to
        // zero at start.
        level = level + 1;
        perm[k] = level;

        if (level == N)
            print(perm, N);
        else
            for (int i = 0; i < N; i++)

                // Assign values
                // to the array
                // if it is zero.
                if (perm[i] == 0)
                    AlexanderBogomolyn(perm, N, i);

        // Decrement the level
    }
}
```

```
        // after all possible
        // permutation after
        // that level.
        level = level - 1;

        perm[k] = 0;
    }

    // Driver Code
    public static void main (String[] args)
    {
        int i, N = 3;
        int perm[] = new int[N];
        AlexanderBogomolyn(perm, N, 0);
    }
}

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

C#

```
// C# program to implement
// Alexander Bogomolny UnOrdered
// Permutation Algorithm
using System;

class GFG
{
    static int level = -1;

    // A function to print
    // the permutation.
    static void print(int []perm,
                      int N)
    {
        for (int i = 0; i < N; i++)
            Console.Write(" " + perm[i]);
        Console.WriteLine();
    }

    // A function implementing
    // Alexander Bogomolyn algorithm.
    static void AlexanderBogomolyn(int []perm,
                                    int N, int k)
    {
        // Assign level to
        // zero at start.
```

```
    level = level + 1;
    perm[k] = level;

    if (level == N)
        print(perm, N);
    else
        for (int i = 0; i < N; i++)

            // Assign values
            // to the array
            // if it is zero.
            if (perm[i] == 0)
                AlexanderBogomolyn(perm, N, i);

        // Decrement the level
        // after all possible
        // permutation after
        // that level.
        level = level - 1;

    perm[k] = 0;
}

// Driver Code
public static void Main ()
{
    int N = 3;
    int []perm = new int[N];
    AlexanderBogomolyn(perm, N, 0);
}
}
```

// This code is contributed
// by anuj_67.

Output:

```
1 2 3
1 3 2
2 1 3
3 1 2
2 3 1
3 2 1
```

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/alexander-bogomolnys-unordered-permutation-algorithm/>

Chapter 3

All possible binary numbers of length n with equal sum in both halves

All possible binary numbers of length n with equal sum in both halves - GeeksforGeeks

Given a number n, we need to print all n-digit binary numbers with equal sum in left and right halves. If n is odd, then mid element can be either 0 or 1.

Examples:

```
Input  : n = 4
Output : 1001 1010 1111
```

```
Input : n = 5
Output : 10001 10101 10010 10110 11011 11111
```

The idea is to recursively build left and right halves and keep track of difference between counts of 1s in them. We print a string when difference becomes 0 and there are no more characters to add.

```
// C++ program to generate all binary strings with
// equal sums in left and right halves.
#include <bits/stdc++.h>
using namespace std;

/* Default values are used only in initial call.
   n is number of bits remaining to be filled
   di is current difference between sums of
   left and right halves.
```

```
    left and right are current half substrings. */
void equal(int n, string left="", string right="",
           int di=0)
{
    // TWO BASE CASES
    // If there are no more characters to add (n is 0)
    if (n == 0)
    {
        // If difference between counts of 1s and
        // 0s is 0 (di is 0)
        if (di == 0)
            cout << left + right << " ";
        return;
    }

    /* If 1 remains than string length was odd */
    if (n == 1)
    {
        // If difference is 0, we can put remaining
        // bit in middle.
        if (di == 0)
        {
            cout << left + "0" + right << " ";
            cout << left + "1" + right << " ";
        }
        return;
    }

    /* If difference is more than what can be
    be covered with remaining n digits
    (Note that lengths of left and right
    must be same) */
    if ((2 * abs(di) <= n))
    {
        /*binary number would not start with 0*/
        if (left != "")
        {
            /* add 0 to end in both left and right
            half. Sum in both half will not
            change*/
            equal(n-2, left+"0", right+"0", di);

            /* add 0 to end in both left and right
            half* subtract 1 from di as right
            sum is increased by 1*/
            equal(n-2, left+"0", right+"1", di-1);
        }
    }
}
```

```
        /* add 1 in end in left half and 0 to the
           right half. Add 1 to di as left sum is
           increased by 1*/
        equal(n-2, left+"1", right+"0", di+1);

        /* add 1 in end to both left and right
           half the sum will not change*/
        equal(n-2, left+"1", right+"1", di);
    }
}

/* driver function */
int main()
{
    int n = 5; // n-bits
    equal(n);
    return 0;
}
```

Output:

10001 10101 10010 10110 11011 11111

Source

<https://www.geeksforgeeks.org/all-possible-binary-numbers-of-length-n-with-equal-sum-in-both-halves/>

Chapter 4

Backtracking | Introduction

Backtracking | Introduction - GeeksforGeeks

Prerequisites :

- [Recursion](#)
- [Complexity Analysis](#)

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

According to the wiki definition,

Backtracking can be defined as a general algorithmic technique that considers searching every possible combination in order to solve an optimization problem.

How to determine if a problem can be solved using Backtracking?

Generally, every [constraint satisfaction problem](#) which has clear and well-defined constraints on any objective solution, that incrementally builds candidate to the solution and abandons a candidate (“backtracks”) as soon as it determines that the candidate cannot possibly be completed to a valid solution, can be solved by Backtracking. However, most of the problems that are discussed, can be solved using other known algorithms like *Dynamic Programming* or *Greedy Algorithms* in logarithmic, linear, linear-logarithmic time complexity in order of input size, and therefore, outshine the backtracking algorithm in every respect (since backtracking algorithms are generally exponential in both time and space). However, a few problems still remain, that only have backtracking algorithms to solve them until now.

Consider a situation that you have three boxes in front of you and only one of them has a gold coin in it but you do not know which one. So, in order to get the coin, you will have to open all of the boxes one by one. You will first check the first box, if it does not contain the coin, you will have to close it and check the second box and so on until you find the

coin. This is what backtracking is, that is solving all sub-problems one by one in order to reach the best possible solution.

Consider the below example to understand the Backtracking approach more formally,

Given an instance of any computational problem P and data D corresponding to the instance, all the constraints that need to be satisfied in order to solve the problem are represented by C . A backtracking algorithm will then work as follows:

The Algorithm begins to build up a solution, starting with an empty solution set S . $S = \{\}$

1. Add to S the first move that is still left (All possible moves are added to S one by one). This now creates a new sub-tree S in the search tree of the algorithm.
2. Check if $S + c$ satisfies each of the constraints in C .
 - If Yes, then the sub-tree S is “eligible” to add more “children”.
 - Else, the entire sub-tree S is useless, so recurs back to step 1 using argument S .
3. In the event of “eligibility” of the newly formed sub-tree S , recurs back to step 1, using argument $S + c$.
4. If the check for $S + c$ returns that it is a solution for the entire data D . Output and terminate the program.
If not, then return that no solution is possible with the current S and hence discard it.

Pseudo Code for Backtracking :

1. Recursive backtracking solution.

```
void findSolutions(n, other params) :
    if (found a solution) :
        solutionsFound = solutionsFound + 1;
        displaySolution();
        if (solutionsFound >= solutionTarget) :
            System.exit(0);
        return

    for (val = first to last) :
        if (isValid(val, n)) :
            applyValue(val, n);
            findSolutions(n+1, other params);
            removeValue(val, n);
```

2. Finding whether a solution exists or not

```

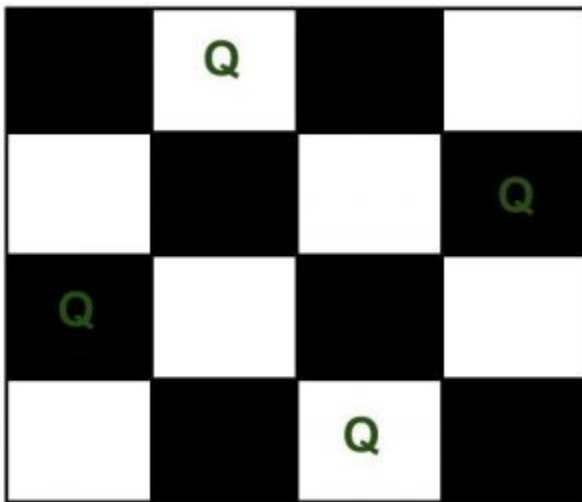
boolean findSolutions(n, other params) :
    if (found a solution) :
        displaySolution();
        return true;

    for (val = first to last) :
        if (isValid(val, n)) :
            applyValue(val, n);
            if (findSolutions(n+1, other params))
                return true;
            removeValue(val, n);
    return false;

```

Let us try to solve a standard Backtracking problem, **N-Queen Problem**.

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.



The expected output is a binary matrix which has 1s for the blocks where queens are placed. For example, following is the output matrix for the above 4 queen solution.

```

{ 0, 1, 0, 0}
{ 0, 0, 0, 1}
{ 1, 0, 0, 0}
{ 0, 0, 1, 0}

```

Backtracking Algorithm: The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes

with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

- 1) Start in the leftmost column
- 2) If all queens are placed
 return true
- 3) Try all rows in the current column. Do following for every tried row.
 - a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
 - b) If placing the queen in [row, column] leads to a solution then return true.
 - c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
- 3) If all rows have been tried and nothing worked, return false to trigger backtracking.

You may refer to the article on [Backtracking | Set 3 \(N Queen Problem\)](#) for complete implementation of the above approach.

More Backtracking Problems:

- [Backtracking | Set 1 \(The Knight's tour problem\)](#)
- [Backtracking | Set 2 \(Rat in a Maze\)](#)
- [Backtracking | Set 4 \(Subset Sum\)](#)
- [Backtracking | Set 5 \(m Coloring Problem\)](#)
- [-> Click Here for More](#)

Source

<https://www.geeksforgeeks.org/backtracking-introduction/>

Chapter 5

Balanced expressions such that given positions have opening brackets | Set 2

Balanced expressions such that given positions have opening brackets | Set 2 - GeeksforGeeks

Given an integer n and an array of positions 'position[]' ($1 \leq \text{length}(\text{position}[]) \leq 2n$), find the number of ways of proper bracket expressions that can be formed of length $2n$ such that given positions have the opening bracket.

Note: position[] array is given in the form of (1-based indexing) [0, 1, 1, 0]. Here 1 denotes the positions at which open bracket should be placed. At positions with value 0, either of opening and closing bracket can be placed.

Examples:

Input: $n = 3$, position[] = [0, 1, 0, 0, 0, 0]

Output: 3

The proper bracket sequences of length 6 and opening bracket at position 2 are:

```
[ [ ] ] [ ]  
[ [ [ ] ] ]  
[ [ ] ] [ ]
```

Input: $n = 2$, position[] = [1, 0, 1, 0]

Output: 1

The only possibility is:

```
[ ] [ ]
```

Dynamic Programming approach of this problem has been already discussed [here](#). In this post, recursive and recursion using memoization approach will be discussed.

Algorithm–

1. Mark all the positions with open brackets in the given array adj as 1.
2. Run a recursive loop, such that –
 - If count of total brackets(opening brackets subtracted from closing brackets is less than zero), return 0.
 - If the index reaches till n and if the total brackets=0, then a solution is obtained and return 1, otherwise return 0.
 - If the index has 1 pre-assigned to it, return the function recursively with index+1 and increment the total brackets.
 - Otherwise Return the function recursively by inserting open brackets at that index and incrementing total brackets by 1 + inserting closed brackets at that index and decrementing total brackets by 1 and move on to the next index till n.

Below is the **Recursive solution** for above algorithm:

```
// C++ implementation of above
// approach using Recursion
#include <bits/stdc++.h>
using namespace std;

// Function to find Number of
// proper bracket expressions
int find(int index, int openbrk, int n, int adj[])
{
    // If open-closed brackets < 0
    if (openbrk < 0)
        return 0;

    // If index reaches the end of expression
    if (index == n) {

        // IF brackets are balanced
        if (openbrk == 0)
            return 1;
        else
            return 0;
    }

    // If the current index has assigned open bracket
    if (adj[index] == 1) {

        // Move forward increasing the
        // length of open brackets
        return find(index + 1, openbrk + 1, n, adj);
    }

    else {
```

```
        // Move forward by inserting open as well
        // as closed brackets on that index
        return find(index + 1, openbrk + 1, n, adj)
            + find(index + 1, openbrk - 1, n, adj);
    }
}
// Driver Code
int main()
{
    int n = 2;

    // Open brackets at position 1
    int adj[4] = { 1, 0, 0, 0 };

    // Calling the find function to calculate the answer
    cout << find(0, 0, 2 * n, adj) << endl;

    return 0;
}
```

Output:

2

Memoized Approach: Time complexity of the above algorithm can be optimized by using **Memorization**. The only thing to be done is to use an array to store the results of previous iterations so that there is no need to recursively call the same function more than once, if the value is already calculated.

Below is the required implementation:

```
// C++ implementation of above
// approach using memoization
#include <bits/stdc++.h>
using namespace std;

#define N 1000

// Function to find Number
// of proper bracket expressions
int find(int index, int openbrk, int n,
        int dp[N][N], int adj[])
{
    // If open-closed brackets < 0
    if (openbrk < 0)
        return 0;
```

```
// If index reaches the end of expression
if (index == n) {

    // If brackets are balanced
    if (openbrk == 0)
        return 1;

    else
        return 0;
}

// If already stored in dp
if (dp[index][openbrk] != -1)
    return dp[index][openbrk];

// If the current index has assigned open bracket
if (adj[index] == 1) {

    // Move forward increasing the
    // length of open brackets
    dp[index][openbrk] = find(index + 1,
                              openbrk + 1, n, dp, adj);
}
else {
    // Move forward by inserting open as
    // well as closed brackets on that index
    dp[index][openbrk] = find(index + 1, openbrk + 1, n, dp, adj)
        + find(index + 1, openbrk - 1, n, dp, adj);
}
// return the answer
return dp[index][openbrk];
}

// Driver Code
int main()
{
    // DP array to precompute the answer
    int dp[N][N];
    int n = 2;

    memset(dp, -1, sizeof(dp));

    // Open brackets at position 1
    int adj[4] = { 1, 0, 0, 0 };

    // Calling the find function to calculate the answer
    cout << find(0, 0, 2 * n, dp, adj) << endl;
```



```
    return 0;  
}
```

Output:

2

Time complexity: $O(N^2)$

Source

<https://www.geeksforgeeks.org/balanced-expressions-such-that-given-positions-have-opening-brackets-set-2/>

Chapter 6

Bank Of America (BA Continuum India Pvt. Ltd.) Campus Recruitment

Bank Of America (BA Continuum India Pvt. Ltd.) Campus Recruitment - GeeksforGeeks
Approved Offer.

Bank Of America has visited our college for on campus recruitment . The recruitment consisted of 4 Rounds in total.

The recruitment was for BA Continuum India Pvt Ltd. the technical field of BOA

Round 1:

This round was a general aptitude test, English proficiency, quantitative analysis and the logical questions. This was time specific for each and every section. Try practicing from GFG, indiabix and careerride youtube videos.

Round 2: TECHNICAL 1

This was a face to face technical interview. I was asked to submit my resume and then was asked to wait until my name was called out. The interview went around 30-40 minutes.

The interviewer started with a very basic aptitude problem(dices probability) and then he asked me a puzzle of water jug problem. Then he asked me to explain my projects in the resume and about the experience I had in the previous organization.

After that he went forward with the technical questions consisting of CODING problems .

1. What is recursion? Find the length of a linked list using recursion
This was to test the basics of the datastructure
2. Balanced Parentheses problem
3. Find the position of the first unbalanced parentheses?

Round 3: TECHNICAL 2

It was another technical round. This round was based on advanced coding.

1. Base conversion. Given a number in 10 format convert it into base of 6?
2. Matrix generation.
3. Backtracking question

The interviewers above solely focused on the approach or the pseudo code and helped if we got stuck somewhere.

Round 4: HR ROUND

Here the HR was very friendly and asked about whether you are able to relocate if asked?

At last I was given a feedback by the interviewers as strong logical and coding skills . I received the offer letter on that day itself at night.

Position: Senior Tech Associate, Bank Of America

Source

<https://www.geeksforgeeks.org/bank-of-america-ba-continuum-india-pvt-ltd-campus-recruitment/>

Chapter 7

Binary to Gray code using recursion

Binary to Gray code using recursion - GeeksforGeeks

Given Binary code of a number as a decimal number, we need to convert this into its equivalent [Gray Code](#).

Examples :

Input : 1001
Output : 1101

Input : 11
Output : 10

In gray code, only one bit is changed in 2 consecutive numbers.

Algorithm :

```
binary_to_grey(n)
  if n == 0
    grey = 0;
  else if last two bits are opposite to each other
    grey = 1 + 10 * binary_to_gray(n/10)
  else if last two bits are same
    grey = 10 * binary_to_gray(n/10)
```

Below is implementation of above approach :

CPP

```
// CPP program to convert Binary to
// Gray code using recursion
#include <bits/stdc++.h>
using namespace std;

// Function to change Binary to
// Gray using recursion
int binary_to_gray(int n)
{
    if (!n)
        return 0;

    // Taking last digit
    int a = n % 10;

    // Taking second last digit
    int b = (n / 10) % 10;

    // If last digit are opposite bits
    if ((a && !b) || (!a && b))
        return (1 + 10 * binary_to_gray(n / 10));

    // If last two bits are same
    return (10 * binary_to_gray(n / 10));
}

// Driver Function
int main()
{
    int binary_number = 1011101;

    printf("%d", binary_to_gray(binary_number));
    return 0;
}
```

Java

```
// Java program to convert
// Binary code to Gray code
import static java.lang.StrictMath.pow;
import java.util.Scanner;

class bin_gray
{
    // Function to change Binary to
    // Gray using recursion
    int binary_to_gray(int n, int i)
    {
```

```
int a, b;
int result = 0;
if (n != 0)
{
    // Taking last digit
    a = n % 10;

    n = n / 10;

    // Taking second last digit
    b = n % 10;

    if ((a & ~ b) == 1 || (~ a & b) == 1)
    {
        result = (int) (result + pow(10,i));
    }

    return binary_to_gray(n, ++i) + result;
}
return 0;
}

// Driver Function
public static void main(String[] args)
{
    int binary_number;
    int result = 0;
    binary_number = 1011101;

    bin_gray obj = new bin_gray();
    result = obj.binary_to_gray(binary_number,0);

    System.out.print(result);
}

// This article is contributed by Anshika Goyal.
```

Python3

```
# Python3 code to convert Binary
# to Gray code using recursion

# Function to change Binary to Gray using recursion
def binary_to_gray( n ):
    if not(n):
        return 0
```

```
# Taking last digit
a = n % 10

# Taking second last digit
b = int(n / 10) % 10

# If last digit are opposite bits
if (a and not(b)) or (not(a) and b):
    return (1 + 10 * binary_to_gray(int(n / 10)))

# If last two bits are same
return (10 * binary_to_gray(int(n / 10)))

# Driver Code
binary_number = 1011101
print( binary_to_gray(binary_number), end='')

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

C#

```
// C# program to convert
// Binary code to Gray code
using System;

class GFG {

    // Function to change Binary to
    // Gray using recursion
    static int binary_to_gray(int n, int i)
    {
        int a, b;
        int result = 0;
        if (n != 0)
        {

            // Taking last digit
            a = n % 10;

            n = n / 10;

            // Taking second last digit
            b = n % 10;

            if ((a & ~ b) == 1 || (~ a & b) == 1)
            {
                result = (int) (result + Math.Pow(10,i));
            }
        }
    }
}
```

```
        return binary_to_gray(n, ++i) + result;
    }

    return 0;
}

// Driver Function
public static void Main()
{
    int binary_number;
    binary_number = 1011101;

    Console.WriteLine(binary_to_gray(binary_number,0));
}

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

PHP

```
<?php
// PHP program to convert Binary to
// Gray code using recursion

// Function to change Binary to
// Gray using recursion
function binary_to_gray($n)
{
    if (!$n)
        return 0;

    // Taking last digit
    $a = $n % 10;

    // Taking second
    // last digit
    $b = ($n / 10) % 10;

    // If last digit are
    // opposite bits
    if (($a && !$b) || (!$a && $b))
        return (1 + 10 * binary_to_gray($n / 10));

    // If last two
    // bits are same
    return (10 * binary_to_gray($n / 10));
}
```



```
// Driver Code
$binary_number = 1011101;
echo binary_to_gray($binary_number);

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

Output :

1110011

Assume that the binary number is in the range of integer. For larger value we can take binary number as string.

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/program-convert-binary-code-equivalent-gray-code-using-recursion/>

Chapter 8

Bitwise recursive addition of two integers

Bitwise recursive addition of two integers - GeeksforGeeks

When adding two binary numbers by hand we keep the carry bits in mind and add it at the same time. But to do same thing in program we need a lot of checks. Recursive solution can be imagined as addition of *carry* and $a \oplus b$ (two inputs) until *carry* becomes 0.

Examples :

Input : int x = 45, y = 45
Output : 90

Input : int x = 4, y = 78
Output : 82

Sum of two bits can be obtained by performing XOR (\oplus) of the two bits. Carry bit can be obtained by performing AND ($\&$) of two bits.

Above is simple [Half Adder](#) logic that can be used to add 2 single bits. We can extend this logic for integers. If x and y don't have set bits at same position(s), then bitwise XOR (\oplus) of x and y gives the sum of x and y. To incorporate common set bits also, bitwise AND ($\&$) is used. Bitwise AND of x and y gives all carry bits. We calculate $(x \& y) \ll 1$ and add it to $x \oplus y$ to get the required result.

One important observation is, if $(x \& y)$ becomes 0, then result is $x \oplus y$.

C

```
// CPP program to do recursive addition
// of two integers
#include <stdio.h>
```

```
int add(int x, int y) {
    int keep = (x & y) << 1;
    int res = x^y;

    // If bitwise & is 0, then there
    // is not going to be any carry.
    // Hence result of XOR is addition.
    if (keep == 0)
        return res;

    add(keep, res);
}

// Driver code
int main(){
    printf("%d", add(15, 38));
    return 0;
}
```

Java

```
// Java program to do recursive addition
// of two integers
import java.io.*;

class GFG {

    static int add(int x, int y)
    {
        int keep = (x & y) << 1;
        int res = x^y;

        // If bitwise & is 0, then there
        // is not going to be any carry.
        // Hence result of XOR is addition.
        if (keep == 0)
            return res;

        return add(keep, res);
    }

    // Driver code
    public static void main (String[] args)
    {
        System.out.println(add(15, 38));
    }
}
```

// This code is contributed by Ajit.

C#

```
// C# program to do recursive
// addition of two integers
using System;

class GFG {

    static int add(int x, int y)
    {
        int keep = (x & y) << 1;
        int res = x^y;

        // If bitwise & is 0, then there
        // is not going to be any carry.
        // Hence result of XOR is addition.
        if (keep == 0)
            return res;

        return add(keep, res);
    }

    // Driver code
    public static void Main ()
    {
        Console.WriteLine(add(15, 38));
    }
}
```

// This code is contributed by Smitha.

PHP

```
<?php
// php program to do recursive addition
// of two integers

function add($x, $y) {
    $keep = ($x & $y) << 1;
    $res = $x^$y;

    // If bitwise & is 0, then there
    // is not going to be any carry.
    // Hence result of XOR is addition.
    if ($keep == 0)
```

```
{
    echo $res;
    exit(0);
}

add($keep, $res);
}

// Driver code
$k= add(15, 38);

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

Output:

53

Improved By : [jit_t](#), [Smitha Dinesh Semwal](#), [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/bitwise-recursive-addition-two-integers/>

Chapter 9

Check for balanced parenthesis without using stack

Check for balanced parenthesis without using stack - GeeksforGeeks

Given an expression string exp, write a program to examine whether the pairs and the orders of “{, }”, “(,)”, “[,]” are correct in exp.

Examples:

Input : exp = "[()]{}{[()()]()}"
Output : true

Input : exp = "[()]"
Output : false

We have discussed a [stack based solution](#). Here we are not allowed to use stack. Looks like this problem cannot be solved without extra space (please see comments at the end). We use [recursion](#) to solve the problem.

Below is the implementation of above algorithm:

```
// CPP program to check if parenthesis are
// balanced or not in an expression.
#include <bits/stdc++.h>
using namespace std;

char findClosing(char c)
{
    if (c == '(')
        return ')';
    if (c == '{')
```

```
        return '}';
    if (c == '[')
        return ']';
    return -1;
}

// function to check if parenthesis are
// balanced.
bool check(char expr[], int n)
{
    // Base cases
    if (n == 0)
        return true;
    if (n == 1)
        return false;
    if (expr[0] == ')' || expr[0] == '}' || expr[0] == ']')
        return false;

    // Search for closing bracket for first
    // opening bracket.
    char closing = findClosing(expr[0]);

    // count is used to handle cases like
    // "((()))". We basically need to
    // consider matching closing bracket.
    int i, count = 0;
    for (i = 1; i < n; i++) {
        if (expr[i] == expr[0])
            count++;
        if (expr[i] == closing) {
            if (count == 0)
                break;
            count--;
        }
    }

    // If we did not find a closing
    // bracket
    if (i == n)
        return false;

    // If closing bracket was next
    // to open
    if (i == 1)
        return check(expr + 2, n - 2);

    // If closing bracket was somewhere
    // in middle.
```

```
    return check(expr + 1, i - 1) && check(expr + i + 1, n - i - 1);
}

// Driver program to test above function
int main()
{
    char expr[] = "[()]";
    int n = strlen(expr);
    if (check(expr, n))
        cout << "Balanced";
    else
        cout << "Not Balanced";
    return 0;
}
```

Output:

Not Balanced

The above solution is very inefficient compared to stack based solution. This seems to only useful for recursion practice problems.

Source

<https://www.geeksforgeeks.org/check-for-balanced-parenthesis-without-using-stack/>

Chapter 10

Check if a destination is reachable from source with two movements allowed

Check if a destination is reachable from source with two movements allowed - GeeksforGeeks

Given coordinates of a source point (x_1, y_1) determine if it is possible to reach the destination point (x_2, y_2) . From any point (x, y) there only two types of valid movements: $(x, x + y)$ and $(x + y, y)$. Return a boolean true if it is possible else return false.

Note: All coordinates are positive.

Asked in: Expedia, Telstra

Examples:

```
Input : (x1, y1) = (2, 10)
        (x2, y2) = (26, 12)
Output : True
(2, 10)->(2, 12)->(14, 12)->(26, 12)
is a valid path.
```

```
Input : (x1, y1) = (20, 10)
        (x2, y2) = (6, 12)
Output : False
No such path is possible because  $x_1 > x_2$ 
and coordinates are positive
```

The problem can be solved using simple recursion. Base case would be to check if current x or y coordinate is greater than that of destination, in which case we return false. If it is not the destination point yet we make two calls for both valid movements from that point. If any of them yields a path we return true else return false.

C++

```
// C++ program to check if a destination is reachable
// from source with two movements allowed
#include <bits/stdc++.h>
using namespace std;

bool isReachable(int sx, int sy, int dx, int dy)
{
    // base case
    if (sx > dx || sy > dy)
        return false;

    // current point is equal to destination
    if (sx == dx && sy == dy)
        return true;

    // check for other 2 possibilities
    return (isReachable(sx + sy, sy, dx, dy) ||
            isReachable(sx, sy + sx, dx, dy));
}

// Driver code
int main()
{
    int source_x = 2, source_y = 10;
    int dest_x = 26, dest_y = 12;
    if (isReachable(source_x, source_y, dest_x, dest_y))
        cout << "True\n";
    else
        cout << "False\n";
    return 0;
}
```

Java

```
// Java program to check if a destination is
// reachable from source with two movements
// allowed

class GFG {

    static boolean isReachable(int sx, int sy,
                               int dx, int dy)
    {

        // base case
        if (sx > dx || sy > dy)
            return false;
```

```
// current point is equal to destination
if (sx == dx && sy == dy)
    return true;

// check for other 2 possibilities
return (isReachable(sx + sy, sy, dx, dy) ||
        isReachable(sx, sy + sx, dx, dy));
}

//driver code
public static void main(String arg[])
{
    int source_x = 2, source_y = 10;
    int dest_x = 26, dest_y = 12;
    if (isReachable(source_x, source_y, dest_x,
                    dest_y))
        System.out.print("True\n");
    else
        System.out.print("False\n");
}
}

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

Python3

```
# Python3 program to check if
# a destination is reachable
# from source with two movements allowed

def isReachable(sx, sy, dx, dy):

    # base case
    if (sx > dx or sy > dy):
        return False

    # current point is equal to destination
    if (sx == dx and sy == dy):
        return True

    # check for other 2 possibilities
    return (isReachable(sx + sy, sy, dx, dy) or
            isReachable(sx, sy + sx, dx, dy))

# Driver code
source_x, source_y = 2, 10
dest_x, dest_y = 26, 12
```

```
if (isReachable(source_x, source_y, dest_x, dest_y)):
    print("True")
else:
    print("False")
```

This code is contributed by Anant Agarwal.

C#

```
// C# program to check if a destination is
// reachable from source with two movements
// allowed
using System;

class GFG {

    static bool isReachable(int sx, int sy,
                           int dx, int dy)
    {

        // base case
        if (sx > dx || sy > dy)
            return false;

        // current point is equal to destination
        if (sx == dx && sy == dy)
            return true;

        // check for other 2 possibilities
        return (isReachable(sx + sy, sy, dx, dy) ||
                isReachable(sx, sy + sx, dx, dy));
    }

    //driver code
    public static void Main()
    {
        int source_x = 2, source_y = 10;
        int dest_x = 26, dest_y = 12;
        if (isReachable(source_x, source_y, dest_x,
                        dest_y))

            Console.WriteLine("True\n");
        else
            Console.WriteLine("False\n");
    }
}
```

// This code is contributed by Anant Agarwal.

PHP

```
<?php
// PHP program to check if a
// destination is reachable
// from source with two movements
// allowed

function isReachable($sx, $sy, $dx, $dy)
{
    // base case
    if ($sx > $dx || $sy > $dy)
        return false;

    // current point is equal
    // to destination
    if ($sx == $dx && $sy == $dy)
        return true;

    // check for other 2 possibilities
    return (isReachable($sx + $sy, $sy, $dx, $dy) ||
            isReachable($sx, $sy + $sx, $dx, $dy));
}

// Driver code
$source_x = 2;
$source_y = 10;
$dest_x = 26;
$dest_y = 12;
if (isReachable($source_x, $source_y,
                $dest_x, $dest_y))
    echo "True\n";
else
    echo "False\n";

// This code is contributed by Sam007
?>
```

Output:

True

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/check-destination-reachable-source-two-movements-allowed/>

Chapter 11

Check if a given string is sum-string

Check if a given string is sum-string - GeeksforGeeks

Given a string of digits, determine whether it is a 'sum-string'. A string S is called a sum-string if a rightmost substring can be written as sum of two substrings before it and same is recursively true for substrings before it.

Examples :

"12243660" is a sum string.

Explanation : $24 + 36 = 60$, $12 + 24 = 36$

"1111112223" is a sum string.

Explanation: $111+112 = 223$, $1+111 = 112$

"2368" is not a sum string

In general a string S is called sum-string if it satisfies the following properties:

```
sub-string(i, x) + sub-string(x+1, j)
= sub-string(j+1, l)
and
sub-string(x+1, j)+sub-string(j+1, l)
= sub-string(l+1, m)
and so on till end.
```

From the examples, we can see that our decision depends on first two chosen numbers. So we choose all possible first two number for given string. Then for every chosen two

numbers we check whether it is sum-string or not? So the approach is very simple. We generate all possible first two numbers using two substrings s_1 and s_2 using two loops. then we check whether it is possible to make number $s_3 = (s_1 + s_2)$ or not. If we can make s_3 then we recursively check for $s_2 + s_3$ so on.

```
// C++ program to check if a given string
// is sum-string or not
#include <bits/stdc++.h>
using namespace std;

// this is function for finding sum of two
// numbers as string
string string_sum(string str1, string str2)
{
    if (str1.size() < str2.size())
        swap(str1, str2);

    int m = str1.size();
    int n = str2.size();
    string ans = "";

    // sum the str2 with str1
    int carry = 0;
    for (int i = 0; i < n; i++) {

        // Sum of current digits
        int ds = ((str1[m - 1 - i] - '0') +
                  (str2[n - 1 - i] - '0') +
                  carry) % 10;

        carry = ((str1[m - 1 - i] - '0') +
                  (str2[n - 1 - i] - '0') +
                  carry) / 10;

        ans = char(ds + '0') + ans;
    }

    for (int i = n; i < m; i++) {
        int ds = (str1[m - 1 - i] - '0' +
                  carry) % 10;
        carry = (str1[m - 1 - i] - '0' +
                  carry) / 10;
        ans = char(ds + '0') + ans;
    }

    if (carry)
        ans = char(carry + '0') + ans;
    return ans;
}
```



```
}

// Returns true of two substrings of given
// lengths of str[beg..] can cause a positive
// result.
bool checkSumStrUtil(string str, int beg,
                    int len1, int len2)
{
    // Finding two substrings of given lengths
    // and their sum
    string s1 = str.substr(beg, len1);
    string s2 = str.substr(beg + len1, len2);
    string s3 = string_sum(s1, s2);

    int s3_len = s3.size();

    // if number of digits s3 is greater than
    // the available string size
    if (s3_len > str.size() - len1 - len2 - beg)
        return false;

    // we got s3 as next number in main string
    if (s3 == str.substr(beg + len1 + len2, s3_len)) {

        // if we reach at the end of the string
        if (beg + len1 + len2 + s3_len == str.size())
            return true;

        // otherwise call recursively for n2, s3
        return checkSumStrUtil(str, beg + len1, len2,
                               s3_len);
    }

    // we do not get s3 in main string
    return false;
}

// Returns true if str is sum string, else false.
bool isSumStr(string str)
{
    int n = str.size();

    // choosing first two numbers and checking
    // whether it is sum-string or not.
    for (int i = 1; i < n; i++)
        for (int j = 1; i + j < n; j++)
            if (checkSumStrUtil(str, 0, i, j))
```

```
        return true;

    return false;
}

// Driver code
int main()
{
    cout << isSumStr("1212243660") << endl;
    cout << isSumStr("123456787");
    return 0;
}
```

Output:

```
1
0
```

Source

<https://www.geeksforgeeks.org/check-given-string-sum-string/>

Chapter 12

Check if a number is Palindrome

Check if a number is Palindrome - GeeksforGeeks

Given an integer, write a function that returns true if the given number is palindrome, else false. For example, 12321 is palindrome, but 1451 is not palindrome.

Palindrome number - 12321

Let the given number be *num*. A simple method for this problem is to first [reverse digits of *num*](#), then compare the reverse of *num* with *num*. If both are same, then return true, else false.

Following is an interesting method inspired from method#2 of [this](#) post. The idea is to create a copy of *num* and recursively pass the copy by reference, and pass *num* by value. In the recursive calls, divide *num* by 10 while moving down the recursion tree. While moving up the recursion tree, divide the copy by 10. When they meet in a function for which all child calls are over, the last digit of *num* will be *i*th digit from the beginning and the last digit of copy will be *i*th digit from the end.

C++

```
// A recursive C++ program to check whether a given number is
// palindrome or not
#include <stdio.h>

// A function that returns true only if num contains one digit
int oneDigit(int num)
{
    // comparison operation is faster than division operation.
    // So using following instead of "return num / 10 == 0;"
```

```
    return (num >= 0 && num < 10);
}

// A recursive function to find out whether num is palindrome
// or not. Initially, dupNum contains address of a copy of num.
bool isPalUtil(int num, int* dupNum)
{
    // Base case (needed for recursion termination): This statement
    // mainly compares the first digit with the last digit
    if (oneDigit(num))
        return (num == (*dupNum) % 10);

    // This is the key line in this method. Note that all recursive
    // calls have a separate copy of num, but they all share same copy
    // of *dupNum. We divide num while moving up the recursion tree
    if (!isPalUtil(num/10, dupNum))
        return false;

    // The following statements are executed when we move up the
    // recursion call tree
    *dupNum /= 10;

    // At this point, if num%10 contains i'th digit from beginning,
    // then (*dupNum)%10 contains i'th digit from end
    return (num % 10 == (*dupNum) % 10);
}

// The main function that uses recursive function isPalUtil() to
// find out whether num is palindrome or not
int isPal(int num)
{
    // If num is negative, make it positive
    if (num < 0)
        num = -num;

    // Create a separate copy of num, so that modifications made
    // to address dupNum don't change the input number.
    int *dupNum = new int(num); // *dupNum = num

    return isPalUtil(num, dupNum);
}

// Driver program to test above functions
int main()
{
    int n = 12321;
    isPal(n)? printf("Yesn"): printf("Non");
}
```

```
n = 12;
isPal(n)? printf("Yesn"): printf("Non");

n = 88;
isPal(n)? printf("Yesn"): printf("Non");

n = 8999;
isPal(n)? printf("Yesn"): printf("Non");
return 0;
}
```

Java

```
// A recursive Java program to
// check whether a given number
// is palindrome or not
import java.io.*;
import java.util.*;

class GFG
{
    // A function that returns true
    // only if num contains one digit
    public static int oneDigit(int num)
    {
        // comparison operation is
        // faster than division
        // operation. So using
        // following instead of
        // "return num / 10 == 0;"
        if((num >= 0) &&(num < 10))
            return 1;
        else
            return 0;
    }

    // A recursive function to
    // find out whether num is
    // palindrome or not.
    // Initially, dupNum contains
    // address of a copy of num.
    public static int isPalUtil(int num,
                                int dupNum)
    {
        // Base case (needed for recursion
        // termination): This statement
        // mainly compares the first digit
```

```
// with the last digit
if (oneDigit(num) == 1)
    if(num == (dupNum) % 10)
        return 1;
    else
        return 0;

// This is the key line in
// this method. Note that
// all recursive calls have
// a separate copy of num,
// but they all share same
// copy of *dupNum. We divide
// num while moving up the
// recursion tree
if (isPalUtil((int)(num / 10), dupNum) == 0)
    return -1;

// The following statements
// are executed when we move
// up the recursion call tree
dupNum = (int)(dupNum / 10);

// At this point, if num%10
// contains i'th digit from
// beginning, then (*dupNum)%10
// contains i'th digit from end
if(num % 10 == (dupNum) % 10)
    return 1;
else
    return 0;
}

// The main function that uses
// recursive function isPalUtil()
// to find out whether num is
// palindrome or not
public static int isPal(int num)
{
    // If num is negative,
    // make it positive
    if (num < 0)
        num = (-num);

    // Create a separate copy
    // of num, so that modifications
    // made to address dupNum
    // don't change the input number.
```

```
        int dupNum = (num); // *dupNum = num

        return isPalUtil(num, dupNum);
    }

    // Driver Code
    public static void main(String args[])
    {
        int n = 12321;
        if(isPal(n) == 0)
            System.out.println("Yes");
        else
            System.out.println("No");

        n = 12;
        if(isPal(n) == 0)
            System.out.println("Yes");
        else
            System.out.println("No");

        n = 88;
        if(isPal(n) == 1)
            System.out.println("Yes");
        else
            System.out.println("No");

        n = 8999;
        if(isPal(n) == 0)
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
```

```
// This code is contributed
// by Akanksha Rai(Abby_akku)
```

C#

```
// A recursive C# program to
// check whether a given number
// is palindrome or not
using System;

class GFG
{
    // A function that returns true
```

```
// only if num contains one digit
public static int oneDigit(int num)
{
    // comparison operation is
    // faster than division
    // operation. So using
    // following instead of
    // "return num / 10 == 0;"
    if((num >= 0) &&(num < 10))
        return 1;
    else
        return 0;
}

// A recursive function to
// find out whether num is
// palindrome or not.
// Initially, dupNum contains
// address of a copy of num.
public static int isPalUtil(int num,
                           int dupNum)
{
    // Base case (needed for recursion
    // termination): This statement
    // mainly compares the first digit
    // with the last digit
    if (oneDigit(num) == 1)
        if(num == (dupNum) % 10)
            return 1;
        else
            return 0;

    // This is the key line in
    // this method. Note that
    // all recursive calls have
    // a separate copy of num,
    // but they all share same
    // copy of *dupNum. We divide
    // num while moving up the
    // recursion tree
    if (isPalUtil((int)(num / 10), dupNum) == 0)
        return -1;

    // The following statements
    // are executed when we move
    // up the recursion call tree
    dupNum = (int)(dupNum / 10);
}
```



```
// At this point, if num%10
// contains i'th digit from
// beginning, then (*dupNum)%10
// contains i'th digit from end
if(num % 10 == (dupNum) % 10)
    return 1;
else
    return 0;
}

// The main function that uses
// recursive function isPalUtil()
// to find out whether num is
// palindrome or not
public static int isPal(int num)
{
    // If num is negative,
    // make it positive
    if (num < 0)
        num = (-num);

    // Create a separate copy
    // of num, so that modifications
    // made to address dupNum
    // don't change the input number.
    int dupNum = (num); // *dupNum = num

    return isPalUtil(num, dupNum);
}

// Driver Code
public static void Main()
{
    int n = 12321;
    if(isPal(n) == 0)
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");

    n = 12;
    if(isPal(n) == 0)
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");

    n = 88;
    if(isPal(n) == 1)
        Console.WriteLine("Yes");
```

```
else
    Console.WriteLine("No");

n = 8999;
if(isPal(n) == 0)
    Console.WriteLine("Yes");
else
    Console.WriteLine("No");
}
}

// This code is contributed by mits
```

PHP

```
<?php
// A recursive PHP program to
// check whether a given number
// is palindrome or not

// A function that returns true
// only if num contains one digit
function oneDigit($num)
{
    // comparison operation is faster
    // than division operation. So
    // using following instead of
    // "return num / 10 == 0;"
    return (($num >= 0) &&
            ($num < 10));
}

// A recursive function to find
// out whether num is palindrome
// or not. Initially, dupNum
// contains address of a copy of num.
function isPalUtil($num, $dupNum)
{
    // Base case (needed for recursion
    // termination): This statement
    // mainly compares the first digit
    // with the last digit
    if (oneDigit($num))
        return ($num == ($dupNum % 10));

    // This is the key line in this
    // method. Note that all recursive
    // calls have a separate copy of
```

```
// num, but they all share same
// copy of *dupNum. We divide num
// while moving up the recursion tree
if (!isPalUtil((int)($num / 10),
               $dupNum))
    return -1;

// The following statements are
// executed when we move up the
// recursion call tree
$dupNum = (int)($dupNum / 10);

// At this point, if num%10
// contains i'th digit from
// beginning, then (*dupNum)%10
// contains i'th digit from end
return ($num % 10 == ($dupNum) % 10);
}

// The main function that uses
// recursive function isPalUtil()
// to find out whether num is
// palindrome or not
function isPal($num)
{
    // If num is negative,
    // make it positive
    if ($num < 0)
        $num = (-$num);

    // Create a separate copy of
    // num, so that modifications
    // made to address dupNum
    // don't change the input number.
    $dupNum = ($num); // *dupNum = num

    return isPalUtil($num, $dupNum);
}

// Driver Code
$n = 12321;
if(isPal($n) == 0)
    echo "Yes\n";
else
    echo "No\n";

$n = 12;
if(isPal($n) == 0)
```

```
        echo "Yes\n";
else
    echo "No\n";

$n = 88;
if(isPal($n) == 1)
    echo "Yes\n";
else
    echo "No\n";

$n = 8999;
if(isPal($n) == 0)
    echo "Yes\n";
else
    echo "No\n";

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

Output:

```
Yes
No
Yes
No
```

To check a number is palindrome or not without using any extra space

This article is compiled by [Aashish Barnwal](#). Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : [jit_t](#), [Mithun Kumar](#), [Abby_akku](#)

Source

<https://www.geeksforgeeks.org/check-if-a-number-is-palindrome/>

Chapter 13

Check if linked list is sorted (Iterative and Recursive)

Check if linked list is sorted (Iterative and Recursive) - GeeksforGeeks

Given a Linked List, task is to check whether the Linked List is sorted in Descending order or not?

Examples :

Input : 8 -> 7 -> 5 -> 2 -> 1

Output : Yes

Explanation :

In given linked list, starting from head,
8 > 7 > 5 > 2 > 1. So, it is sorted in reverse order

Input : 24 -> 12 -> 9 -> 11 -> 8 -> 2

Output : No

Iterative Approach : Traverse the linked list from head to end. For every newly encountered element, check **node -> data > node -> next -> data**. If True, do same for each node else return 0 and Print “No”.

```
// C++ program to check Linked List is sorted
// in descending order or not
#include <bits/stdc++.h>
using namespace std;

/* Linked list node */
struct Node
{
```

```
    int data;
    struct Node* next;
};

// function to Check Linked List is
// sorted in descending order or not
bool isSortedDesc(struct Node *head)
{
    if (head == NULL)
        return true;

    // Traverse the list till last node and return
    // false if a node is smaller than or equal
    // its next.
    for (Node *t=head; t->next != NULL; t=t->next)
        if (t->data <= t->next->data)
            return false;
    return true;
}

Node *newNode(int data)
{
    Node *temp = new Node;
    temp->next = NULL;
    temp->data = data;
}

// Driver program to test above
int main()
{
    struct Node *head = newNode(7);
    head->next = newNode(5);
    head->next->next = newNode(4);
    head->next->next->next = newNode(3);

    isSortedDesc(head) ? cout << "Yes" :
                        cout << "No";

    return 0;
}
```

Output:

Yes

Time Complexity : $O(N)$, where N is the length of linked list.

Recursive Approach :

Check Recursively that **node -> data > node -> next -> data**, If not, return 0 that is our terminated condition to come out from recursion else Call Check_List Function Recursively for next node.

```
// C++ program to recursively check Linked List
// is sorted in descending order or not
#include <bits/stdc++.h>
using namespace std;

/* Linked list node */
struct Node
{
    int data;
    struct Node* next;
};

// function to Check Linked List is
// sorted in descending order or not
bool isSortedDesc(struct Node *head)
{
    // Base cases
    if (head == NULL || head->next == NULL)
        return true;

    // Check first two nodes and recursively
    // check remaining.
    return (head->data > head->next->data &&
            isSortedDesc(head->next));
}

Node *newNode(int data)
{
    Node *temp = new Node;
    temp->next = NULL;
    temp->data = data;
}

// Driver program to test above
int main()
{
    struct Node *head = newNode(7);
    head->next = newNode(5);
    head->next->next = newNode(4);
    head->next->next->next = newNode(3);

    isSortedDesc(head) ? cout << "Yes" :
                        cout << "No";
}
```

```
    return 0;  
}
```

Output:

Yes

Source

<https://www.geeksforgeeks.org/check-linked-list-sorting-order/>

Chapter 14

Combinational Sum

Combinational Sum - GeeksforGeeks

Given an array of positive integers `arr[]` and a sum `x`, find all unique combinations in `arr[]` where the sum is equal to `x`. The same repeated number may be chosen from `arr[]` unlimited number of times. Elements in a combination (`a1, a2, ..., ak`) must be printed in non-descending order. (ie, `a1 <= a2 <= ... <= ak`).

The combinations themselves must be sorted in ascending order, i.e., the combination with smallest first element should be printed first. If there is no combination possible the print "Empty" (without quotes).

Examples:

Input : `arr[] = 2, 4, 6, 8`
 `x = 8`

Output : `[2, 2, 2, 2]`
 `[2, 2, 4]`
 `[2, 6]`
 `[4, 4]`
 `[8]`

Since the problem is to get all the possible results, not the best or the number of result, thus we don't need to consider DP(dynamic programming), recursion is needed to handle it.

We should use the following algorithm.

1. Sort the array(non-decreasing).
2. First remove all the duplicates from array.
3. Then use recursion and backtracking to solve the problem.
 - (A) If at any time sub-problem sum == 0 then add that array to the result (vector of

- vectors).
- (B) Else if sum is negative then ignore that sub-problem.
- (C) Else insert the present array in that index to the current vector and call the function with `sum = sum - ar[index]` and `index = index + 1`, then pop that element from current index (backtrack) and call the function with `sum = sum` and `index = index + 1`

Below is C++ implementation of above steps.

```
// C++ program to find all combinations that
// sum to a given value
#include <bits/stdc++.h>
using namespace std;

// Print all members of ar[] that have given
void findNumbers(vector<int>& ar, int sum,
                 vector<vector<int>>& res,
                 vector<int>& r, int i)
{
    // If current sum becomes negative
    if (sum < 0)
        return;

    // if we get exact answer
    if (sum == 0)
    {
        res.push_back(r);
        return;
    }

    // Recur for all remaining elements that
    // have value smaller than sum.
    while (i < ar.size() && sum - ar[i] >= 0)
    {
        // Till every element in the array starting
        // from i which can contribute to the sum
        r.push_back(ar[i]); // add them to list

        // recur for next numbers
        findNumbers(ar, sum - ar[i], res, r, i);
        i++;

        // remove number from list (backtracking)
        r.pop_back();
    }
}
```

```
    }  
}  
  
// Returns all combinations of ar[] that have given  
// sum.  
vector<vector<int> > combinationSum(vector<int>& ar,  
                                   int sum)  
{  
    // sort input array  
    sort(ar.begin(), ar.end());  
  
    // remove duplicates  
    ar.erase(unique(ar.begin(), ar.end()), ar.end());  
  
    vector<int> r;  
    vector<vector<int> > res;  
    findNumbers(ar, sum, res, r, 0);  
  
    return res;  
}  
  
// Driver code  
int main()  
{  
    vector<int> ar;  
    ar.push_back(2);  
    ar.push_back(4);  
    ar.push_back(6);  
    ar.push_back(8);  
    int n = ar.size();  
  
    int sum = 8; // set value of sum  
    vector<vector<int> > res = combinationSum(ar, sum);  
  
    // If result is empty, then  
    if (res.size() == 0)  
    {  
        cout << "Emptyn";  
        return 0;  
    }  
  
    // Print all combinations stored in res.  
    for (int i = 0; i < res.size(); i++)  
    {  
        if (res[i].size() > 0)  
        {  
            cout << " ( ";  
            for (int j = 0; j < res[i].size(); j++)
```

```
        cout << res[i][j] << " ";  
        cout << ")\n";  
    }  
}
```

Output:

(2 2 2 2) (2 2 4) (2 6) (4 4) (8)

Source

<https://www.geeksforgeeks.org/combinational-sum/>

Chapter 15

Combinations in a String of Digits

Combinations in a String of Digits - GeeksforGeeks

Given an input string of numbers, find all combinations of numbers that can be formed using digits in the same order.

Examples:

Input : 123

Output : 1 2 3
1 23
12 3
123

Input : 1234

Output : 1 2 3 4
1 2 34
1 23 4
1 234
12 3 4
12 34
123 4
1234

The problem can be solved using recursion. We keep track of current index in given input string and length of output string so far. In each call to the function, if there are no digits remaining in the input string print the current output string and return. Otherwise, copy current digit to output. From here make two calls, one considering next digit as part of next number(including a space in output string) and one considering next digit as part of current number(no space included). If there are no digits remaining after current digit the second call to the function is omitted because a trailing space doesn't count as a new combination.

```
// CPP program to find all combination of numbers
// from a given string of digits
#include <iostream>
#include <cstring>
using namespace std;

// function to print combinations of numbers
// in given input string
void printCombinations(char* input, int index,
                      char* output, int outLength)
{
    // no more digits left in input string
    if (input[index] == '\0')
    {
        // print output string & return
        output[outLength] = '\0';
        cout << output << endl;
        return;
    }

    // place current digit in input string
    output[outLength] = input[index];

    // separate next digit with a space
    output[outLength + 1] = ' ';

    printCombinations(input, index + 1, output,
                      outLength + 2);

    // if next digit exists make a
    // call without including space
    if(input[index + 1] != '\0')
        printCombinations(input, index + 1, output,
                          outLength + 1);
}

// driver function to test above function
int main()
{
    char input[] = "1214";
    char *output = new char[100];

    // initialize output with empty string
    output[0] = '\0';

    printCombinations(input, 0, output, 0);
    return 0;
}
```

```
}
```

Output:

```
1 2 1 4
1 2 14
1 21 4
1 214
12 1 4
12 14
121 4
1214
```

Alternative Solution:

```
// CPP program to find all combination of
// numbers from a given string of digits
// using bit algorithm used same logic
// as to print power set of string
#include <bits/stdc++.h>
using namespace std;

// function to print combinations of
// numbers in given input string
void printCombinations(char s[]){

    // find length of char array
    int l = strlen(s);

    // we can give space between characters
    // ex. ('1' & '2') or ('2' & '3') or
    // ('3' & '4') or ('3' & '4') or all
    // that's why here we have maximum
    // space length - 1
    for(int i = 0; i < pow(2, l - 1); i++){
        int k = i, x = 0;

        // first character will be printed
        // as well
        cout << s[x];
        x++;
        for(int j = 0; j < strlen(s) - 1; j++){

            // if bit is set, means provide
            // space
            if(k & 1)
```

```
        cout << " ";
        k = k >> 1;
        cout << s[x];

        // always increment index of
        // input string
        x++;
    }
    cout << "\n";
}

// driver code
int main() {

    char input[] = "1214";
    combination(input);

    return 0;
}
// This code is contributed by PRINCE Gupta 2
```

Output:

```
1214
1 214
12 14
1 2 14
121 4
1 21 4
12 1 4
1 2 1 4
```

Source

<https://www.geeksforgeeks.org/combinations-string-digits/>

Chapter 16

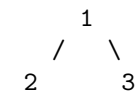
Convert a Binary Tree such that every node stores the sum of all nodes in its right subtree

Convert a Binary Tree such that every node stores the sum of all nodes in its right subtree
- GeeksforGeeks

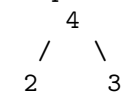
Given a binary tree, change the value in each node to sum of all the values in the nodes in the right subtree including its own.

Examples:

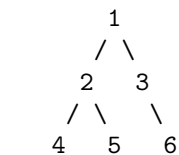
Input :



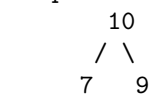
Output :

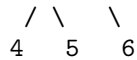


Input :



Output :





Approach : The idea is to traverse the given binary tree in **bottom up** manner. Recursively compute the sum of nodes in right and left subtrees. Accumulate sum of nodes in the right subtree to the current node and return sum of nodes under current subtree.

Below is the implementation of above approach.

```
// C++ program to store sum of nodes in
// right subtree in every node
#include <bits/stdc++.h>
using namespace std;

// Node of tree
struct Node {
    int data;
    Node *left, *right;
};

// Function to create a new node
struct Node* createNode(int item)
{
    Node* temp = new Node;
    temp->data = item;
    temp->left = NULL;
    temp->right = NULL;

    return temp;
}

// Function to build new tree with
// all nodes having the sum of all
// nodes in its right subtree
int updateBTree(Node* root)
{
    // Base cases
    if (!root)
        return 0;
    if (root->left == NULL && root->right == NULL)
        return root->data;

    // Update right and left subtrees
    int rightsum = updateBTree(root->right);
    int leftsum = updateBTree(root->left);

    // Add rightsum to current node
    root->data += rightsum;
```

```
// Return sum of values under root
return root->data + leftsum;
}

// Function to traverse tree in inorder way
void inorder(struct Node* node)
{
    if (node == NULL)
        return;
    inorder(node->left);
    cout << node->data << " ";
    inorder(node->right);
}

// Driver code
int main()
{
    /* Let us construct a binary tree
        1
       / \
      2  3
     / \  \
    4  5  6    */
    struct Node* root = NULL;
    root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->right = createNode(6);

    // new tree construction
    updateBTree(root);

    cout << "Inorder traversal of the modified tree is \n";
    inorder(root);

    return 0;
}
```

Output:

```
Inorder traversal of the modified tree is
4 7 5 10 9 6
```

Time Complexity: $O(n)$

Source

<https://www.geeksforgeeks.org/convert-a-binary-tree-such-that-every-node-stores-the-sum-of-all-nodes-in-its-right-subtree/>

Chapter 17

Count all possible paths from top left to bottom right of a mXn matrix

Count all possible paths from top left to bottom right of a mXn matrix - GeeksforGeeks

The problem is to count all the possible paths from top left to bottom right of a mXn matrix with the constraints that *from each cell you can either move only to right or down*

Examples :

Input : m = 2, n = 2;

Output : 2

There are two paths

(0, 0) -> (0, 1) -> (1, 1)

(0, 0) -> (1, 0) -> (1, 1)

Input : m = 2, n = 3;

Output : 3

There are three paths

(0, 0) -> (0, 1) -> (0, 2) -> (1, 2)

(0, 0) -> (0, 1) -> (1, 1) -> (1, 2)

(0, 0) -> (1, 0) -> (1, 1) -> (1, 2)

We have discussed a [solution to print all possible paths](#), counting all paths is easier. Let NumberOfPaths(m, n) be the count of paths to reach row number m and column number n in the matrix, NumberOfPaths(m, n) can be recursively written as following.

C++

```
#include <iostream>
```

```
using namespace std;

// Returns count of possible paths to reach cell at row
// number m and column number n from the topmost leftmost
// cell (cell at 1, 1)
int numberOfPaths(int m, int n)
{
    // If either given row number is first or given column
    // number is first
    if (m == 1 || n == 1)
        return 1;

    // If diagonal movements are allowed then the last
    // addition is required.
    return numberOfPaths(m-1, n) + numberOfPaths(m, n-1);
    // + numberOfPaths(m-1,n-1);
}

int main()
{
    cout << numberOfPaths(3, 3);
    return 0;
}
```

Java

```
// A Java program to count all possible paths
// from top left to bottom right

class GFG
{
    // Returns count of possible paths to reach
    // cell at row number m and column number n
    // from the topmost leftmost cell (cell at 1, 1)
    static int numberOfPaths(int m, int n)
    {
        // If either given row number is first or
        // given column number is first
        if (m == 1 || n == 1)
            return 1;

        // If diagonal movements are allowed then
        // the last addition is required.
        return numberOfPaths(m-1, n) + numberOfPaths(m, n-1);
        // + numberOfPaths(m-1,n-1);
    }
}
```

```
public static void main(String args[])
{
    System.out.println(numberOfPaths(3, 3));
}

// This code is contributed by Sumit Ghosh
```

Python

```
# Python program to count all possible paths
# from top left to bottom right

# function to return count of possible paths
# to reach cell at row number m and column
# number n from the topmost leftmost
# cell (cell at 1, 1)
def numberOfPaths(m, n):
    # If either given row number is first
    # or given column number is first
    if(m == 1 or n == 1):
        return 1

    # If diagonal movements are allowed
    # then the last addition
    # is required.
    return numberOfPaths(m-1, n) + numberOfPaths(m, n-1)

# Driver program to test above function
m = 3
n = 3
print(numberOfPaths(m, n))

# This code is contributed by Aditi Sharma
```

Output:

6

The time complexity of above recursive solution is exponential. There are many overlapping subproblems. We can draw a recursion tree for `numberOfPaths(3, 3)` and see many overlapping subproblems. The recursion tree would be similar to [Recursion tree for Longest Common Subsequence problem](#).

So this problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem.

Like other typical [Dynamic Programming\(DP\) problems](#), recomputations of same subproblems can be avoided by constructing a temporary array `count[][]` in bottom up manner using the above recursive formula.

C

```
#include <iostream>
using namespace std;

// Returns count of possible paths to reach cell at
// row number m and column number n from the topmost
// leftmost cell (cell at 1, 1)
int numberOfPaths(int m, int n)
{
    // Create a 2D table to store results of subproblems
    int count[m][n];

    // Count of paths to reach any cell in first column is 1
    for (int i = 0; i < m; i++)
        count[i][0] = 1;

    // Count of paths to reach any cell in first column is 1
    for (int j = 0; j < n; j++)
        count[0][j] = 1;

    // Calculate count of paths for other cells in
    // bottom-up manner using the recursive solution
    for (int i = 1; i < m; i++)
    {
        for (int j = 1; j < n; j++)

            // By uncommenting the last part the code calculatest he total
            // possible paths if the diagonal Movements are allowed
            count[i][j] = count[i-1][j] + count[i][j-1]; //+ count[i-1][j-1];

    }
    return count[m-1][n-1];
}

// Driver program to test above functions
int main()
{
    cout << numberOfPaths(3, 3);
    return 0;
}
```

Java

```
// A Java program to count all possible paths
```



```
// from top left to bottom right
class GFG
{
    // Returns count of possible paths to reach
    // cell at row number m and column number n from
    // the topmost leftmost cell (cell at 1, 1)
    static int numberOfPaths(int m, int n)
    {
        // Create a 2D table to store results
        // of subproblems
        int count[] [] = new int[m] [n];

        // Count of paths to reach any cell in
        // first column is 1
        for (int i = 0; i < m; i++)
            count[i][0] = 1;

        // Count of paths to reach any cell in
        // first column is 1
        for (int j = 0; j < n; j++)
            count[0][j] = 1;

        // Calculate count of paths for other
        // cells in bottom-up manner using
        // the recursive solution
        for (int i = 1; i < m; i++)
        {
            for (int j = 1; j < n; j++)

                // By uncommenting the last part the
                // code calculate the total possible paths
                // if the diagonal Movements are allowed
                count[i][j] = count[i-1][j] + count[i][j-1]; //+ count[i-1][j-1];

        }
        return count[m-1][n-1];
    }

    // Driver program to test above function
    public static void main(String args[])
    {
        System.out.println(numberOfPaths(3, 3));
    }
}

// This code is contributed by Sumit Ghosh
```

Python

```
# Python program to count all possible paths
# from top left to bottom right

# Returns count of possible paths to reach cell
# at row number m and column number n from the
# topmost leftmost cell (cell at 1, 1)
def numberOfPaths(m, n):
    # Create a 2D table to store
    # results of subproblems
    count = [[0 for x in range(m)] for y in range(n)]

    # Count of paths to reach any
    # cell in first column is 1
    for i in range(m):
        count[i][0] = 1;

    # Count of paths to reach any
    # cell in first column is 1
    for j in range(n):
        count[0][j] = 1;

    # Calculate count of paths for other
    # cells in bottom-up
    # manner using the recursive solution
    for i in range(1, m):
        for j in range(n):
            count[i][j] = count[i-1][j] + count[i][j-1]
    return count[m-1][n-1]

# Driver program to test above function
m = 3
n = 3
print( numberOfPaths(m, n))

# This code is contributed by Aditi Sharma
```

Output:

6

Time complexity of the above dynamic programming solution is $O(mn)$.

The space complexity of the above solution is $O(mn)$.

Space Optimization of DP solution.

Above solution is more intuitive but we can also reduce the space by $O(n)$; where n is column size.

Java

```
class GFG
{
    // Returns count of possible paths to reach
    // cell at row number m and column number n from
    // the topmost leftmost cell (cell at 1, 1)
    static int numberOfPaths(int m, int n)
    {
        // Create a 1D array to store results of subproblems
        int[] dp = new int[n];
        dp[0] = 1;

        for (int i = 0; i < m; i++) {
            for (int j = 1; j < n; j++) {
                dp[j] += dp[j - 1];
            }
        }

        return dp[n - 1];
    }

    // Driver program to test above function
    public static void main(String args[])
    {
        System.out.println(numberOfPaths(3, 3));
    }
}
```

Output:

6

This code is contributed by [Vivek Singh](#)

Note the count can also be calculated using the formula $(m-1 + n-1)! / ((m-1)!(n-1)!)$.

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

Improved By : [viveksingh14](#)

Source

<https://www.geeksforgeeks.org/count-possible-paths-top-left-bottom-right-nxm-matrix/>

Chapter 18

Count consonants in a string (Iterative and recursive methods)

Count consonants in a string (Iterative and recursive methods) - GeeksforGeeks

Given a string, count total number of consonants in it. A consonant is a English alphabet character that is not vowel (a, e, i, o and u). Examples of constants are b, c, d, f, g, ..

Examples :

Input : abc de
Output : 3
There are three consonants b, c and d.

Input : geeksforgeeks portal
Output : 12

1. Iterative Method

C++

```
// Iterative CPP program to count total number
// of consonants
#include <iostream>
using namespace std;

// Function to check for consonant
bool isConsonant(char ch)
{
```

```
// To handle lower case
ch = toupper(ch);

return !(ch == 'A' || ch == 'E' ||
        ch == 'I' || ch == 'O' ||
        ch == 'U') && ch >= 65 && ch <= 90;
}

int totalConsonants(string str)
{
    int count = 0;
    for (int i = 0; i < str.length(); i++)

        // To check is character is Consonant
        if (isConsonant(str[i]))
            ++count;
    return count;
}

// Driver code
int main()
{
    string str = "abc de";
    cout << totalConsonants(str);
    return 0;
}
```

Java

```
// Iterative Java program
// to count total number
// of consonants

import java.io.*;

class GFG {

    // Function to check for consonant
    static boolean isConsonant(char ch)
    {
        // To handle lower case
        ch = Character.toUpperCase(ch);

        return !(ch == 'A' || ch == 'E' ||
                ch == 'I' || ch == 'O' ||
                ch == 'U') && ch >= 65 && ch <= 90;
    }
}
```

```
static int totalConsonants(String str)
{
    int count = 0;
    for (int i = 0; i < str.length(); i++)

        // To check is character is Consonant
        if (isConsonant(str.charAt(i)))
            ++count;
    return count;
}

// Driver code
public static void main(String args[])
{
    String str = "abc de";
    System.out.println( totalConsonants(str));
}

// This code is contributed by Nikita Tiwari.
```

Python3

```
# Iterative Python3 program to count
# total number of consonants

# Function to check for consonant
def isConsonant(ch):

    # To handle lower case
    ch = ch.upper()

    return not (ch == 'A' or ch == 'E' or
                ch == 'I' or ch == 'O' or
                ch == 'U') and ord(ch) >= 65 and ord(ch) <= 90

def totalConsonants(string):

    count = 0

    for i in range(len(string)):

        # To check is character is Consonant
        if (isConsonant(string[i])):
            count += 1

    return count
```

```
# Driver code
string = "abc de"
print(totalConsonants(string))
```

This code id contributed by Ansu Kumari.

C#

```
// Iterative C# program to count
// total number of consonants
using System;

class GFG {

    // Function to check for consonant
    static bool isConsonant(char ch)
    {
        // To handle lower case
        ch = Char.ToUpper(ch);

        return !(ch == 'A' || ch == 'E' ||
                ch == 'I' || ch == 'O' ||
                ch == 'U') && ch >= 65 && ch <= 90;
    }

    static int totalConsonants(String str)
    {
        int count = 0;
        for (int i = 0; i < str.Length; i++)

            // To check is character is Consonant
            if (isConsonant(str[i]))
                ++count;
        return count;
    }

    // Driver code
    public static void Main()
    {
        String str = "abc de";
        Console.Write( totalConsonants(str));
    }
}

// This code is contributed by nitin mittal.
```

Output:

3

2. Recursive Method

C++

```
// Recursive CPP program to count total number
// of consonants
#include <iostream>
using namespace std;

// Function to check for consonant
bool isConsonant(char ch)
{
    // To handle lower case
    ch = toupper(ch);

    return !(ch == 'A' || ch == 'E' ||
            ch == 'I' || ch == 'O' ||
            ch == 'U') && ch >= 65 && ch <= 90;
}

// to count total number of consonants from
// 0 to n-1
int totalConsonants(string str, int n)
{
    if (n == 1)
        return isConsonant(str[0]);

    return totalConsonants(str, n - 1) +
           isConsonant(str[n-1]);
}

// Driver code
int main()
{
    string str = "abc de";
    cout << totalConsonants(str, str.length());
    return 0;
}
```

Python3

```
# Recursive Python3 program to count
# total number of consonants

# Function to check for consonant
```



```
def isConsonant(ch):

    # To handle lower case
    ch = ch.upper()

    return not (ch == 'A' or ch == 'E' or
                ch == 'I' or ch == 'O' or
                ch == 'U') and ord(ch) >= 65 and ord(ch) <= 90

# To count total number of
# consonants from 0 to n-1
def totalConsonants(string, n):

    if n == 1:
        return isConsonant(string[0])

    return totalConsonants(string, n - 1) + isConsonant(string[n-1])

# Driver code
string = "abc de"
print(totalConsonants(string, len(string)))
```

This code is contributed by Ansu Kuamri.

Output :

3

Illustration of recursive method:

Total number of Consonant in String using Recursion

a b c d e

$\text{tConsonant}(\text{"abc de"}) = \text{tConsonant}(\text{"abc d"}) + \text{isConsonant}(\text{'e'})$

a b c d

$\text{tConsonant}(\text{"abc d"}) = \text{tConsonant}(\text{"abc"}) + \text{isConsonant}(\text{'d'})$

a b c

$\text{tConsonant}(\text{"abc"}) = \text{tConsonant}(\text{"ab"}) + \text{isConsonant}(\text{'c'})$

a b c

$\text{tConsonant}(\text{"abc"}) = \text{tConsonant}(\text{"ab"}) + \text{isConsonant}(\text{'c'})$

a b

$\text{tConsonant}(\text{"ab"}) = \text{tConsonant}(\text{"a"}) + \text{isConsonant}(\text{'b'})$

a

Source

<https://www.geeksforgeeks.org/count-consonants-string-iterative-recursive-methods/>

Chapter 19

Count number of ways to partition a set into k subsets

Count number of ways to partition a set into k subsets - GeeksforGeeks

Given two numbers n and k where n represents number of elements in a set, find number of ways to partition the set into k subsets.

Example:

Input: n = 3, k = 2

Output: 3

Explanation: Let the set be {1, 2, 3}, we can partition it into 2 subsets in following ways
{{1,2}, {3}}, {{1}, {2,3}}, {{1,3}, {2}}

Input: n = 3, k = 1

Output: 1

Explanation: There is only one way {{1, 2, 3}}

We strongly recommend you to minimize your browser and try this yourself first.

Let $S(n, k)$ be total number of partitions of n elements into k sets. Value of $S(n, k)$ can be defined recursively as,

$$S(n, k) = k \cdot S(n-1, k) + S(n-1, k-1)$$

$S(n, k)$ is called [Stirling numbers of the second kind](#)

How does above recursive formula work?

When we add a (n+1)'th element to k partitions, there are two possibilities.

- 1) It is added as a single element set to existing partitions, i.e, $S(n, k-1)$
- 2) It is added to all sets of every partition, i.e., $k*S(n, k)$

Therefore $S(n+1, k) = k*S(n, k) + S(n, k-1)$ which means $S(n, k) = k*S(n-1, k) + S(n-1, k-1)$

Below is recursive solution based on above formula.

C++

```
// A C++ program to count number of partitions
// of a set with n elements into k subsets
#include<iostream>
using namespace std;

// Returns count of different partitions of n
// elements in k subsets
int countP(int n, int k)
{
    // Base cases
    if (n == 0 || k == 0 || k > n)
        return 0;
    if (k == 1 || k == n)
        return 1;

    // S(n+1, k) = k*S(n, k) + S(n, k-1)
    return k*countP(n-1, k) + countP(n-1, k-1);
}

// Driver program
int main()
{
    cout << countP(3, 2);
    return 0;
}
```

Java

```
// Java program to count number
// of partitions of a set with
// n elements into k subsets
import java.io.*;

class GFG
{
    // Returns count of different
    // partitions of n elements in
```

```
// k subsets
public static int countP(int n, int k)
{
    // Base cases
    if (n == 0 || k == 0 || k > n)
        return 0;
    if (k == 1 || k == n)
        return 1;

    //  $S(n+1, k) = k*S(n, k) + S(n, k-1)$ 
    return (k * countP(n - 1, k)
        + countP(n - 1, k - 1));
}

// Driver program
public static void main(String args[])
{
    System.out.println(countP(3, 2));
}
}
```

//This code is contributed by Anshika Goyal.

C#

```
// C# program to count number
// of partitions of a set with
// n elements into k subsets
using System;

class GFG {

    // Returns count of different
    // partitions of n elements in
    // k subsets
    public static int countP(int n, int k)
    {

        // Base cases
        if (n == 0 || k == 0 || k > n)
            return 0;
        if (k == 1 || k == n)
            return 1;

        //  $S(n+1, k) = k*S(n, k) + S(n, k-1)$ 
        return (k * countP(n - 1, k)
            + countP(n - 1, k - 1));
    }
}
```

```
    }

    // Driver program
    public static void Main()
    {
        Console.WriteLine(countP(3, 2));
    }
}

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

PHP

```
<?php
// A PHP program to count
// number of partitions of
// a set with n elements
// into k subsets

// Returns count of different
// partitions of n elements
// in k subsets
function countP($n, $k)
{
    // Base cases
    if ($n == 0 || $k == 0 || $k > $n)
        return 0;
    if ($k == 1 || $k == $n)
        return 1;

    // S(n+1, k) = k*S(n, k)
    // + S(n, k-1)
    return $k * countP($n - 1, $k) +
        countP($n - 1, $k - 1);
}

// Driver Code
echo countP(3, 2);

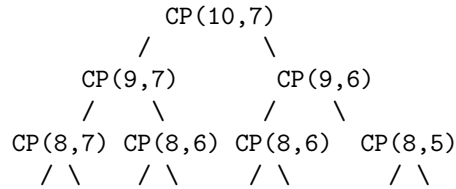
// This code is contributed by aj_36
?>
```

Output:

3

The time complexity of above recursive solution is exponential. The solution can be optimized as there are overlapping subproblems. For example, below is recursion tree of countP(10,7). The subproblem countP(8,6) or CP(8,6) is called multiple times.

CP() represents countP()



Partial Recursion Tree for countP(10, 7)
to highlight overlapping subproblems.

So this problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical [Dynamic Programming\(DP\) problems](#), recomputations of same subproblems can be avoided by constructing a temporary array `dp[][]` in bottom up manner using the above recursive formula.

Below is C++ implementation of Dynamic Programming Solution.

```

// A Dynamic Programming based C++ program to count
// number of partitions of a set with n elements
// into k subsets
#include<iostream>
using namespace std;

// Returns count of different partitions of n
// elements in k subsets
int countP(int n, int k)
{
    // Table to store results of subproblems
    int dp[n+1][k+1];

    // Base cases
    for (int i=0; i<=n; i++)
        dp[i][0] = 0;
    for (int i=0; i<=k; i++)
        dp[0][i] = 0;

    // Fill rest of the entries in dp[][]
    // in bottom up manner
    for (int i=1; i<=n; i++)
        for (int j=1; j<=i; j++)
            if (j == 1 || i == j)
                dp[i][j] = 1;
            else
                dp[i][j] = j*dp[i-1][j] + dp[i-1][j-1];
}

```



```
    return dp[n][k];
}

// Driver program
int main()
{
    cout << countP(5, 2);
    return 0;
}
```

Output:

15

Time Complexity: $O(n \times k)$

Auxiliary Space: $O(n \times k)$

Similar Article: [Bell Numbers](#)

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.

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

Source

<https://www.geeksforgeeks.org/count-number-of-ways-to-partition-a-set-into-k-subsets/>

Chapter 20

Count occurrences of a substring recursively

Count occurrences of a substring recursively - GeeksforGeeks

Given two strings str1 and str2, the task is to count the number of times “str2” occurs in “str1” using recursion.

Examples:

Input : str1 = "geeksforgeeks", str2 = "geek"
Output : 2

Input: kanekihiishishi
Output: 3

Suppose the problem has n parts, divide the problem in such a way that considers n-1 parts already done after which the operation to be performed is limited to only one part. Thereby, dividing the recursion approach into two cases i.e. base case and the recursive case.

In this particular problem, the base case involves the fact that if the length of str1 is less than that of str2.

Now, talking about the recursive case, compare first substring of str1 with str2 and recur for remaining str1.

C++

```
// Recursive C++ program for counting number of substrings
#include <iostream>
#include <string>
using namespace std;
```

```
// Recursive function to count
// the number of occurrences of "hi" in str.
int countSubstrig(string str1, string str2)
{
    int n1 = str1.length();
    int n2 = str2.length();

    // Base Case
    if (n1 == 0 || n1 < n2)
        return 0;

    // Recursive Case
    // Checking if the first substring matches
    if (str1.substr(0, n2).compare(str2) == 0)
        return countSubstrig(str1.substr(n2-1), str2) + 1;

    // Otherwise, return the count from
    // the remaining index
    return countSubstrig(str1.substr(n2-1), str2);
}

// Driver function
int main()
{
    string str1 = "geeksforgeeks", str2 = "geeks";
    cout << countSubstrig(str1, str2) << endl;

    str1 = "hikakashi", str2 = "hi";
    cout << countSubstrig(str1, str2) << endl;
    return 0;
}
```

Java

```
// Recursive Java program for
// counting number of substrings
class GFG
{
    // Recursive function to
    // count the number of
    // occurrences of "hi" in str.
    static int countSubstrig(String str1,
                              String str2)
    {
        int n1 = str1.length();
        int n2 = str2.length();
```

```
// Base Case
if (n1 == 0 || n1 < n2)
    return 0;

// Recursive Case
// Checking if the first
// substring matches
if (str1.substring(0, n2).equals(str2))
    return countSubstrig(str1.substring(n2 - 1),
                        str2) + 1;

// Otherwise, return the count
// from the remaining index
return countSubstrig(str1.substring(n2 - 1),
                    str2);
}

// Driver Code
public static void main(String args[])
{
    String str1 = "geeksforgeeks",
           str2 = "geeks";
    System.out.println(countSubstrig(str1,
                                    str2));

    str1 = "hikakashi";
    str2 = "hi";
    System.out.println(countSubstrig(str1,
                                    str2));
}
}
```

// This code is contributed
// by Arnab Kundu

C#

```
// Recursive C# program for
// counting number of substrings
using System;
class GFG
{
    // Recursive function to
    // count the number of
    // occurrences of "hi" in str.
    static int countSubstrig(String str1,
```

```
        String str2)
{
    int n1 = str1.Length;
    int n2 = str2.Length;

    // Base Case
    if (n1 == 0 || n1 < n2)
        return 0;

    // Recursive Case
    // Checking if the first
    // substring matches
    if (str1.Substring(0, n2).Equals(str2))
        return countSubstrig(str1.Substring(n2 - 1),
                               str2) + 1;

    // Otherwise, return the
    // count from the remaining
    // index
    return countSubstrig(str1.Substring(n2 - 1),
                          str2);
}

// Driver Code
public static void Main()
{
    string str1 = "geeksforgeeks",
           str2 = "geeks";
    Console.Write(countSubstrig(str1,
                                str2));

    Console.Write("\n");

    str1 = "hikakashi";
    str2 = "hi";
    Console.Write(countSubstrig(str1,
                                str2));

}
}
```

// This code is contributed
// by Smita

Output:

2
2

Improved By : [andrew1234](#), [Smitha Dinesh Semwal](#)

Source

<https://www.geeksforgeeks.org/count-occurrences-of-a-substring-recursively/>

Chapter 21

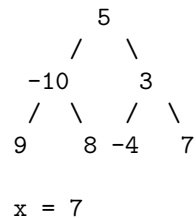
Count subtrees that sum up to a given value x

Count subtrees that sum up to a given value x - GeeksforGeeks

Given a binary tree containing **n** nodes. The problem is to count subtress having total node's data sum equal to a given value **x**.

Examples:

Input :

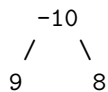


Output : 2

There are 2 subtrees with sum 7.

1st one is leaf node:
7.

2nd one is:



Source: [Microsoft Interview Experience | Set 157](#).

Algorithm:

```
countSubtreesWithSumX(root, count, x)
    if !root then
        return 0

    ls = countSubtreesWithSumX(root->left, count, x)
    rs = countSubtreesWithSumX(root->right, count, x)
    sum = ls + rs + root->data

    if sum == x then
        count++
    return sum

countSubtreesWithSumXUtil(root, x)
    if !root then
        return 0

    Initialize count = 0
    ls = countSubtreesWithSumX(root->left, count, x)
    rs = countSubtreesWithSumX(root->right, count, x)

    if (ls + rs + root->data) == x
        count++
    return count
```

C++

```
// C++ implementation to count subtress that
// sum up to a given value x
#include <bits/stdc++.h>

using namespace std;

// structure of a node of binary tree
struct Node {
    int data;
    Node *left, *right;
};

// function to get a new node
Node* getNode(int data)
{
    // allocate space
    Node* newNode = (Node*)malloc(sizeof(Node));
```



```
// put in the data
newNode->data = data;
newNode->left = newNode->right = NULL;
return newNode;
}

// function to count subtree that
// sum up to a given value x
int countSubtreesWithSumX(Node* root,
                           int& count, int x)
{
    // if tree is empty
    if (!root)
        return 0;

    // sum of nodes in the left subtree
    int ls = countSubtreesWithSumX(root->left, count, x);

    // sum of nodes in the right subtree
    int rs = countSubtreesWithSumX(root->right, count, x);

    // sum of nodes in the subtree rooted
    // with 'root->data'
    int sum = ls + rs + root->data;

    // if true
    if (sum == x)
        count++;

    // return subtree's nodes sum
    return sum;
}

// utility function to count subtree that
// sum up to a given value x
int countSubtreesWithSumXUtil(Node* root, int x)
{
    // if tree is empty
    if (!root)
        return 0;

    int count = 0;

    // sum of nodes in the left subtree
    int ls = countSubtreesWithSumX(root->left, count, x);

    // sum of nodes in the right subtree
```

```

    int rs = countSubtreesWithSumX(root->right, count, x);

    // if tree's nodes sum == x
    if ((ls + rs + root->data) == x)
        count++;

    // required count of subtrees
    return count;
}

// Driver program to test above
int main()
{
    /* binary tree creation
        5
       / \
      -10  3
     / \ / \
    9  8 -4 7
    */
    Node* root = getNode(5);
    root->left = getNode(-10);
    root->right = getNode(3);
    root->left->left = getNode(9);
    root->left->right = getNode(8);
    root->right->left = getNode(-4);
    root->right->right = getNode(7);

    int x = 7;

    cout << "Count = "
         << countSubtreesWithSumXUtil(root, x);

    return 0;
}

```

Java

```

// Java program to find if
// there is a subtree with
// given sum
import java.util.*;
class GFG
{
    // structure of a node
    // of binary tree
    static class Node
    {

```

```
int data;
Node left, right;
}

static class INT
{
    int v;
    INT(int a)
    {
        v = a;
    }
}

// function to get a new node
static Node getNode(int data)
{
    // allocate space
    Node newNode = new Node();

    // put in the data
    newNode.data = data;
    newNode.left = newNode.right = null;
    return newNode;
}

// function to count subtree that
// sum up to a given value x
static int countSubtreesWithSumX(Node root,
INT count, int x)
{
    // if tree is empty
    if (root == null)
        return 0;

    // sum of nodes in the left subtree
    int ls = countSubtreesWithSumX(root.left,
count, x);

    // sum of nodes in the right subtree
    int rs = countSubtreesWithSumX(root.right,
count, x);

    // sum of nodes in the subtree
    // rooted with 'root.data'
    int sum = ls + rs + root.data;

    // if true
    if (sum == x)
        count.v++;

    // return subtree's nodes sum
    return sum;
}
```

```
// utility function to
// count subtree that
// sum up to a given value x
static int countSubtreesWithSumXUtil(Node root,
int x)
{
// if tree is empty
if (root == null)
return 0;

INT count = new INT(0);

// sum of nodes in the left subtree
int ls = countSubtreesWithSumX(root.left,
count, x);

// sum of nodes in the right subtree
int rs = countSubtreesWithSumX(root.right,
count, x);

// if tree's nodes sum == x
if ((ls + rs + root.data) == x)
count.v++;

// required count of subtrees
return count.v;
}

// Driver Code
public static void main(String args[])
{
/* binary tree creation
5
/ \
-10 3
/ \ / \
9 8 -4 7
*/
Node root = getNode(5);
root.left = getNode(-10);
root.right = getNode(3);
root.left.left = getNode(9);
root.left.right = getNode(8);
root.right.left = getNode(-4);
root.right.right = getNode(7);

int x = 7;

System.out.println("Count = " +
countSubtreesWithSumXUtil(root, x));
}
}
```

```
// This code is contributed  
// by Arnab Kundu
```

Output:

Count = 2

Time Complexity: $O(n)$.

Improved By : [andrew1234](#)

Source

<https://www.geeksforgeeks.org/count-subtrees-sum-given-value-x/>

Chapter 22

Count ways to express a number as sum of powers

Count ways to express a number as sum of powers - GeeksforGeeks

Given two integers x and n, we need to find number of ways to express x as sum of n-th powers of unique natural numbers. It is given that $1 \leq n \leq 20$.

Examples:

```
Input   : x = 100
          n = 2
Output  : 3
Explanation: There are three ways to
express 100 as sum of natural numbers
raised to power 2.
100 = 10^2 = 8^2+6^2 = 1^2+3^2+4^2+5^2+7^2
```

```
Input   : x = 100
          n = 3
Output  : 1
Explanation : The only combination is,
1^3 + 2^3 + 3^3 + 4^3
```

We use recursion to solve the problem. We first check one by one that the number is included in summation or not.

C++

```
// C++ program to count number of ways
// to express x as sum of n-th power
```

```
// of unique natural numbers.
#include <bits/stdc++.h>
using namespace std;

// num is current num.
int countWaysUtil(int x, int n, int num)
{
    // Base cases
    int val = (x - pow(num, n));
    if (val == 0)
        return 1;
    if (val < 0)
        return 0;

    // Consider two possibilities, num is
    // included and num is not included.
    return countWaysUtil(val, n, num + 1) +
           countWaysUtil(x, n, num + 1);
}

// Returns number of ways to express
// x as sum of n-th power of two.
int countWays(int x, int n)
{
    return countWaysUtil(x, n, 1);
}

// Driver code
int main()
{
    int x = 100, n = 2;
    cout << countWays(x, n);
    return 0;
}
```

Java

```
// Java program to count number of ways
// to express x as sum of n-th power
// of unique natural numbers.
public class GFG {

    // num is current num.
    static int countWaysUtil(int x, int n, int num)
    {
        // Base cases
        int val = (int) (x - Math.pow(num, n));
        if (val == 0)
```

```
        return 1;
    if (val < 0)
        return 0;

    // Consider two possibilities, num is
    // included and num is not included.
    return countWaysUtil(val, n, num + 1) +
           countWaysUtil(x, n, num + 1);
}

// Returns number of ways to express
// x as sum of n-th power of two.
static int countWays(int x, int n)
{
    return countWaysUtil(x, n, 1);
}

// Driver code
public static void main(String args[])
{
    int x = 100, n = 2;
    System.out.println(countWays(x, n));
}
// This code is contributed by Sumit Ghosh
```

Python3

```
# Python program to count number of ways
# to express x as sum of n-th power
# of unique natural numbers.

# num is current num.
def countWaysUtil(x,n,num):

    # Base cases
    val = (x - pow(num, n))
    if (val == 0):
        return 1
    if (val < 0):
        return 0

    # Consider two possibilities, num is
    # included and num is not included.
    return countWaysUtil(val, n, num + 1) + \
           countWaysUtil(x, n, num + 1)
```



```
# Returns number of ways to express
# x as sum of n-th power of two.
def countWays(x,n):
    return countWaysUtil(x, n, 1)
```

```
# Driver code
x = 100
n = 2
```

```
print(countWays(x, n))
```

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

C#

```
// C# program to count number of ways
// to express x as sum of n-th power
// of unique natural numbers.
using System;

public class GFG {

    // num is current num.
    static int countWaysUtil(int x,
                             int n, int num)
    {

        // Base cases
        int val = (int) (x - Math.Pow(num, n));
        if (val == 0)
            return 1;
        if (val < 0)
            return 0;

        // Consider two possibilities,
        // num is included and num is
        // not included.
        return countWaysUtil(val, n, num + 1)
            + countWaysUtil(x, n, num + 1);
    }

    // Returns number of ways to express
    // x as sum of n-th power of two.
    static int countWays(int x, int n)
    {
        return countWaysUtil(x, n, 1);
    }
}
```

```
    }

    // Driver code
    public static void Main()
    {
        int x = 100, n = 2;

        Console.WriteLine(countWays(x, n));
    }
}

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

PHP

```
<?php
// PHP program to count number of ways
// to express x as sum of n-th power
// of unique natural numbers.

// num is current num.
function countWaysUtil($x, $n, $num)
{
    // Base cases
    $val = ($x - pow($num, $n));
    if ($val == 0)
        return 1;
    if ($val < 0)
        return 0;

    // Consider two possibilities, num is
    // included and num is not included.
    return (countWaysUtil($val, $n, $num + 1) +
            countWaysUtil($x, $n, $num + 1));
}

// Returns number of ways to express
// x as sum of n-th power of two.
function countWays($x, $n)
{
    return countWaysUtil($x, $n, 1);
}

// Driver code
$x = 100; $n = 2;
echo(countWays($x, $n));
```

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

Output:

3

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

Source

<https://www.geeksforgeeks.org/count-ways-express-number-sum-powers/>

Chapter 23

Decimal to binary number using recursion

Decimal to binary number using recursion - GeeksforGeeks

Given a decimal number as input, we need to write a program to convert the given decimal number into equivalent binary number.

Examples :

Input : 7
Output : 111

Input : 10
Output : 1010

We have discussed one iterative solution in below post.

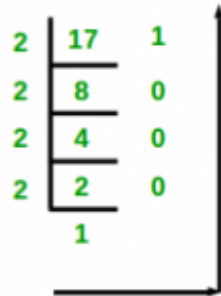
[Program for Decimal to Binary Conversion](#)

Below is Recursive solution

```
findBinary(decimal)
    if (decimal == 0)
        binary = 0
    else
        binary = decimal % 2 + 10 * (findBinary(decimal / 2))
```

.

Decimal number : 17



Binary number: 10001

Step by step process for better understanding of how the algorithm works

Let decimal number be 10.

Step 1-> $10 \% 2$ which is equal-too $0 + 10 * (10/2) \% 2$

Step 2-> $5 \% 2$ which is equal-too $1 + 10 * (5 / 2) \% 2$

Step 3-> $2 \% 2$ which is equal-too $0 + 10 * (2 / 2) \% 2$

Step 4-> $1 \% 2$ which is equal-too $1 + 10 * (1 / 2) \% 2$

C

```
// C/C++ program for decimal to binary
// conversion using recursion
#include <stdio.h>

// Decimal to binary conversion
// using recursion
int find(int decimal_number)
{
    if (decimal_number == 0)
        return 0;
    else
        return (decimal_number % 2 + 10 *
                find(decimal_number / 2));
}

// Driver code
int main()
{
    int decimal_number = 10;
    printf("%d", find(decimal_number));
    return 0;
}
```

Java

```
// Java program for decimal to binary
// conversion using recursion
import java.io.*;

class GFG
{
    // Decimal to binary conversion
    // using recursion
    static int find(int decimal_number)
    {
        if (decimal_number == 0)
            return 0;

        else

            return (decimal_number % 2 + 10 *
                    find(decimal_number / 2));
    }

    // Driver Code
    public static void main(String args[])
    {
        int decimal_number = 10;
        System.out.println(find(decimal_number));
    }
}

// This code is contributed by Nikita Tiwari
```

Python3

```
# Python3 code for decimal to binary
# conversion using recursion

# Decimal to binary conversion
# using recursion
def find( decimal_number ):
    if decimal_number == 0:
        return 0
    else:
        return (decimal_number % 2 + 10 *
                find(int(decimal_number / 2)))
```

```
# Driver Code
decimal_number = 10
print(find(decimal_number))
```

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

C#

```
// C# program for decimal to binary
// conversion using recursion
using System;

class GFG
{
    // Decimal to binary conversion
    // using recursion
    static int find(int decimal_number)
    {
        if (decimal_number == 0)
            return 0;

        else

            return (decimal_number % 2 + 10 *
                    find(decimal_number / 2));
    }

    // Driver Code
    public static void Main()
    {
        int decimal_number = 10;

        Console.WriteLine(find(decimal_number));
    }
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP program for decimal to binary
// conversion using recursion
```

```
// Decimal to binary
// conversion using recursion
function find($decimal_number)
{
    if ($decimal_number == 0)
        return 0;
    else
        return ($decimal_number % 2 + 10 *
                find($decimal_number / 2));
}

// Driver Code
$decimal_number = 10;
echo(find($decimal_number));

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

Output :

1010

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/decimal-binary-number-using-recursion/>

Chapter 24

Decode a string recursively encoded as count followed by substring

Decode a string recursively encoded as count followed by substring - GeeksforGeeks

An encoded string (s) is given, the task is to decode it. The pattern in which the strings are encoded is as follows.

```
<count>[sub_str] ==> The substring 'sub_str'  
                        appears count times.
```

Examples:

```
Input : str[] = "1[b]"  
Output : b
```

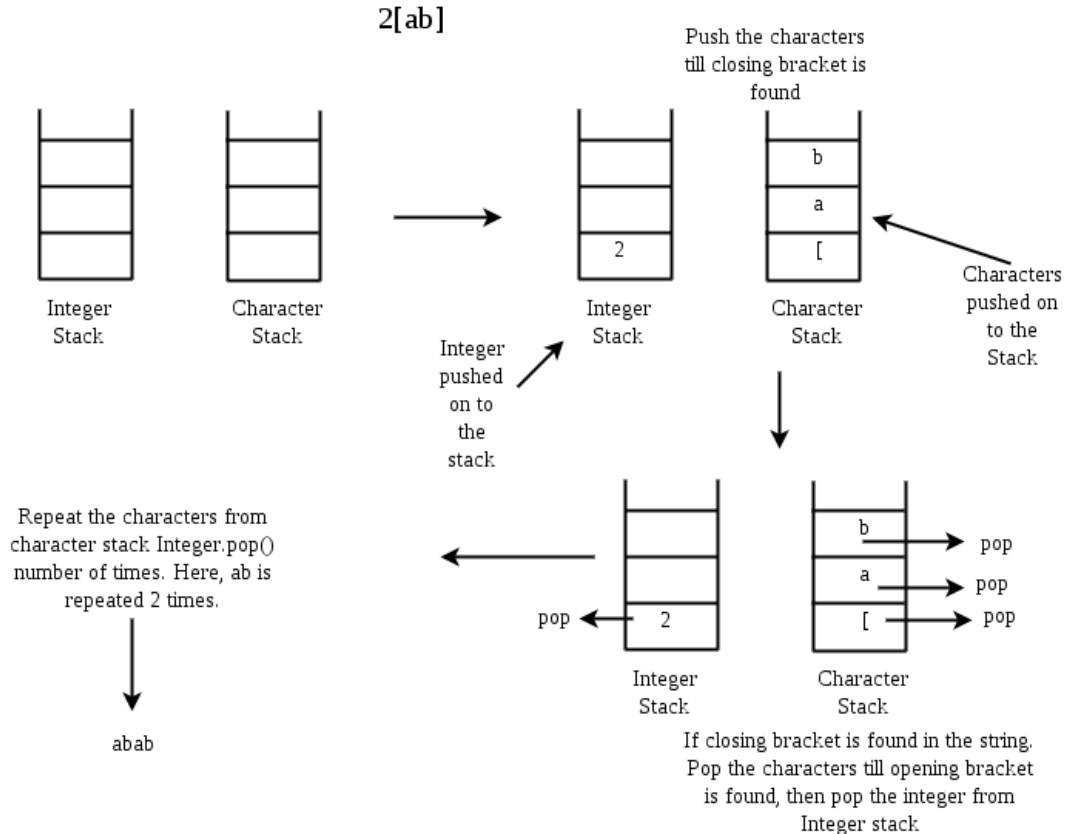
```
Input : str[] = "2[ab]"  
Output : abab
```

```
Input : str[] = "2[a2[b]]"  
Output : abbabb
```

```
Input : str[] = "3[b2[ca]]"  
Output : bcacabacabcaca
```

The idea is to use two stacks, one for integers and another for characters. Now, traverse the string,

1. Whenever we encounter any number, push it into the integer stack and in case of any alphabet (a to z) or open bracket ('['), push it onto the character stack.
2. Whenever any close bracket (']') is encountered pop the character from the character stack until open bracket ('[') is not found in the character stack. Also, pop the top element from the integer stack, say n. Now make a string repeating the popped character n number of times. Now, push all character of the string in the stack.



Below is implementation of this approach:

C++

```
// C++ program to decode a string recursively
// encoded as count followed substring
#include<bits/stdc++.h>
using namespace std;

// Returns decoded string for 'str'
string decode(string str)
{
    stack<int> integerstack;
```

```
stack<char> stringstack;
string temp = "", result = "";

// Traversing the string
for (int i = 0; i < str.length(); i++)
{
    int count = 0;

    // If number, convert it into number
    // and push it into integerstack.
    if (str[i] >= '0' && str[i] <='9')
    {
        while (str[i] >= '0' && str[i] <= '9')
        {
            count = count * 10 + str[i] - '0';
            i++;
        }

        i--;
        integerstack.push(count);
    }

    // If closing bracket ']', pop element until
    // '[' opening bracket is not found in the
    // character stack.
    else if (str[i] == ']')
    {
        temp = "";
        count = 0;

        if (! integerstack.empty())
        {
            count = integerstack.top();
            integerstack.pop();
        }

        while (! stringstack.empty() && stringstack.top() != '[' )
        {
            temp = stringstack.top() + temp;
            stringstack.pop();
        }

        if (! stringstack.empty() && stringstack.top() == '[')
            stringstack.pop();

        // Repeating the popped string 'temo' count
        // number of times.
        for (int j = 0; j < count; j++)
```

```
        result = result + temp;

        // Push it in the character stack.
        for (int j = 0; j < result.length(); j++)
            stringstack.push(result[j]);

        result = "";
    }

    // If '[' opening bracket, push it into character stack.
    else if (str[i] == '[')
    {
        if (str[i-1] >= '0' && str[i-1] <= '9')
            stringstack.push(str[i]);

        else
        {
            stringstack.push(str[i]);
            integerstack.push(1);
        }
    }

    else
        stringstack.push(str[i]);
}

// Pop all the elmenet, make a string and return.
while (! stringstack.empty())
{
    result = stringstack.top() + result;
    stringstack.pop();
}

return result;
}

// Driven Program
int main()
{
    string str = "3[b2[ca]]";
    cout << decode(str) << endl;
    return 0;
}
```

Java

```
// Java program to decode a string recursively
// encoded as count followed substring
```

```
import java.util.Stack;

class Test
{
    // Returns decoded string for 'str'
    static String decode(String str)
    {
        Stack<Integer> integerstack = new Stack<>();
        Stack<Character> stringstack = new Stack<>();
        String temp = "", result = "";

        // Traversing the string
        for (int i = 0; i < str.length(); i++)
        {
            int count = 0;

            // If number, convert it into number
            // and push it into integerstack.
            if (Character.isDigit(str.charAt(i)))
            {
                while (Character.isDigit(str.charAt(i)))
                {
                    count = count * 10 + str.charAt(i) - '0';
                    i++;
                }

                i--;
                integerstack.push(count);
            }

            // If closing bracket ']', pop element until
            // '[' opening bracket is not found in the
            // character stack.
            else if (str.charAt(i) == ']')
            {
                temp = "";
                count = 0;

                if (!integerstack.isEmpty())
                {
                    count = integerstack.peek();
                    integerstack.pop();
                }

                while (!stringstack.isEmpty() && stringstack.peek() != '[' )
                {
                    temp = stringstack.peek() + temp;
                }
            }
        }
    }
}
```

```
        stringstack.pop();
    }

    if (!stringstack.empty() && stringstack.peek() == '[')
        stringstack.pop();

    // Repeating the popped string 'temo' count
    // number of times.
    for (int j = 0; j < count; j++)
        result = result + temp;

    // Push it in the character stack.
    for (int j = 0; j < result.length(); j++)
        stringstack.push(result.charAt(j));

    result = "";
}

// If '[' opening bracket, push it into character stack.
else if (str.charAt(i) == '[')
{
    if (Character.isDigit(str.charAt(i-1)))
        stringstack.push(str.charAt(i));

    else
    {
        stringstack.push(str.charAt(i));
        integerstack.push(1);
    }
}

else
    stringstack.push(str.charAt(i));
}

// Pop all the elmenet, make a string and return.
while (!stringstack.isEmpty())
{
    result = stringstack.peek() + result;
    stringstack.pop();
}

return result;
}

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

```
        String str = "3[b2[ca]]";  
        System.out.println(decode(str));  
    }  
}
```

Output:

bcacabcacabcaca

Source

<https://www.geeksforgeeks.org/decode-string-recursively-encoded-count-followed-substring/>

Chapter 25

Delete a linked list using recursion

Delete a linked list using recursion - GeeksforGeeks

Delete the given linked list using recursion

Method

- 1) If head equal to NULL then linked list is empty, we simply return.
- 2) Recursively delete linked list after head node.
- 3) Delete head node.

```
// C++ program to recursively delete a linked list
#include <bits/stdc++.h>

/* Link list node */
struct Node {
    int data;
    struct Node* next;
};

/* Recursive Function to delete the entire linked list */
void deleteList(struct Node* head)
{
    if (head == NULL)
        return;
    deleteList(head->next);
    free(head);
}

/* Given a reference (pointer to pointer) to
the head of a list and an int, push a new
node on the front of the list. */
```



```
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node = new Node;
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

/* Driver program to test count function*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;

    /* Use push() to construct below list
    1->12->1->4->1 */
    push(&head, 1);
    push(&head, 4);
    push(&head, 1);
    push(&head, 12);
    push(&head, 1);
    printf("\n Deleting linked list");
    deleteList(head);
    printf("\nLinked list deleted");
    return 0;
}
```

Output:

```
Deleting linked list
Linked list deleted
```

Source

<https://www.geeksforgeeks.org/delete-linked-list-using-recursion/>

Chapter 26

Delete middle element of a stack

Delete middle element of a stack - GeeksforGeeks

Given a stack with push(), pop(), empty() operations, delete middle of it without using any additional data structure.

Input : Stack[] = [1, 2, 3, 4, 5]
Output : Stack[] = [1, 2, 4, 5]

Input : Stack[] = [1, 2, 3, 4, 5, 6]
Output : Stack[] = [1, 2, 4, 5, 6]

The idea is to use recursive calls. We first remove all items one by one, then we recur. After recursive calls, we push all items back except the middle item.

C++

```
// C++ code to delete middle of a stack
// without using additional data structure.
#include<bits/stdc++.h>
using namespace std;

// Deletes middle of stack of size
// n. Curr is current item number
void deleteMid(stack<char> &st, int n,
               int curr=0)
{
    // If stack is empty or all items
    // are traversed
    if (st.empty() || curr == n)
        return;
```

```
// Remove current item
int x = st.top();
st.pop();

// Remove other items
deleteMid(st, n, curr+1);

// Put all items back except middle
if (curr != n/2)
    st.push(x);
}

//Driver function to test above functions
int main()
{
    stack<char> st;

    //push elements into the stack
    st.push('1');
    st.push('2');
    st.push('3');
    st.push('4');
    st.push('5');
    st.push('6');
    st.push('7');

    deleteMid(st, st.size());

    // Printing stack after deletion
    // of middle.
    while (!st.empty())
    {
        char p=st.top();
        st.pop();
        cout << p << " ";
    }
    return 0;
}
```

Java

```
// Java code to delete middle of a stack
// without using additional data structure.
import java.io.*;
import java.util.*;

public class GFG {
```

```
// Deletes middle of stack of size
// n. Curr is current item number
static void deleteMid(Stack<Character> st,
                     int n, int curr)
{
    // If stack is empty or all items
    // are traversed
    if (st.empty() || curr == n)
        return;

    // Remove current item
    char x = st.pop();

    // Remove other items
    deleteMid(st, n, curr+1);

    // Put all items back except middle
    if (curr != n/2)
        st.push(x);
}

// Driver function to test above functions
public static void main(String args[])
{
    Stack<Character> st =
        new Stack<Character>();

    // push elements into the stack
    st.push('1');
    st.push('2');
    st.push('3');
    st.push('4');
    st.push('5');
    st.push('6');
    st.push('7');

    deleteMid(st, st.size(), 0);

    // Printing stack after deletion
    // of middle.
    while (!st.empty())
    {
        char p=st.pop();
        System.out.print(p + " ");
    }
}
```

```
}
```

```
// This code is contributed by  
// Manish Shaw (manishshaw1)
```

Python3

```
# Python3 code to delete middle of a stack  
# without using additional data structure.
```

```
# Deletes middle of stack of size  
# n. Curr is current item number  
class Stack:  
    def __init__(self):  
        self.items = []  
  
    def isEmpty(self):  
        return self.items == []  
  
    def push(self, item):  
        self.items.append(item)  
  
    def pop(self):  
        return self.items.pop()  
  
    def peek(self):  
        return self.items[len(self.items)-1]  
  
    def size(self):  
        return len(self.items)  
  
def deleteMid(st, n, curr) :  
  
    # If stack is empty or all items  
    # are traversed  
    if (st.isEmpty() or curr == n) :  
        return  
  
    # Remove current item  
    x = st.peak()  
    st.pop()  
  
    # Remove other items  
    deleteMid(st, n, curr+1)  
  
    # Put all items back except middle  
    if (curr != int(n/2)) :  
        st.push(x)
```

```
# Driver function to test above functions
st = Stack()

# push elements into the stack
st.push('1')
st.push('2')
st.push('3')
st.push('4')
st.push('5')
st.push('6')
st.push('7')

deleteMid(st, st.size(), 0)

# Printing stack after deletion
# of middle.
while (st.isEmpty() == False) :
    p = st.peek()
    st.pop()
    print (str(p) + " ", end="")

# This code is contributed by
# Manish Shaw (manishshaw1)
```

C#

```
// C# code to delete middle of a stack
// without using additional data structure.
using System;
using System.Collections.Generic;

class GFG {

    // Deletes middle of stack of size
    // n. Curr is current item number
    static void deleteMid(Stack<char> st,
                           int n,
                           int curr = 0)
    {

        // If stack is empty or all
        // items are traversed
        if (st.Count == 0 || curr == n)
            return;

        // Remove current item
        char x = st.Peek();
```

```
        st.Pop();

        // Remove other items
        deleteMid(st, n, curr+1);

        // Put all items
        // back except middle
        if (curr != n / 2)
            st.Push(x);
    }

    // Driver Code
    public static void Main()
    {
        Stack<char> st = new Stack<char>();

        // push elements into the stack
        st.Push('1');
        st.Push('2');
        st.Push('3');
        st.Push('4');
        st.Push('5');
        st.Push('6');
        st.Push('7');

        deleteMid(st, st.Count);

        // Printing stack after
        // deletion of middle.
        while (st.Count != 0)
        {
            char p=st.Peek();
            st.Pop();
            Console.Write(p + " ");
        }
    }
}

// This code is contributed by
// Manish Shaw (manishshaw1)
```

Output:

7 6 5 3 2 1

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/delete-middle-element-stack/>

Chapter 27

Diagonal Sum of a Binary Tree

Diagonal Sum of a Binary Tree - GeeksforGeeks

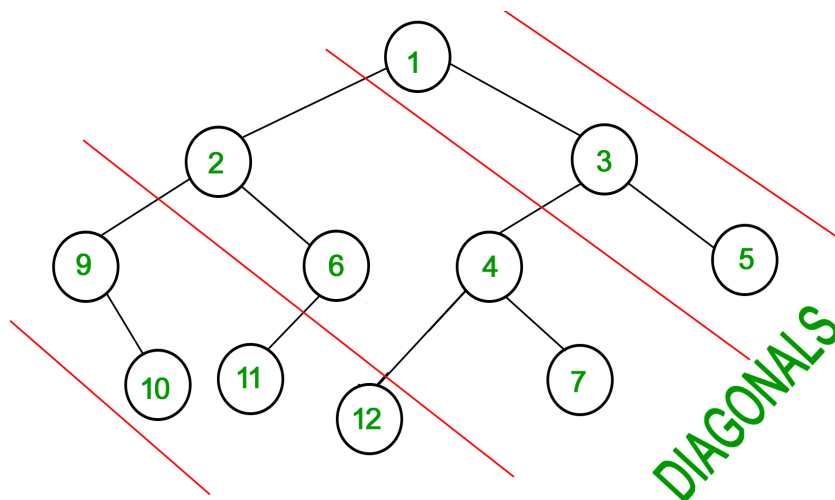
Consider lines of slope -1 passing between nodes (dotted lines in below diagram). Diagonal sum in a binary tree is sum of all node's data lying between these lines. Given a Binary Tree, print all diagonal sums.

For the following input tree, output should be 9, 19, 42.

9 is sum of 1, 3 and 5.

19 is sum of 2, 6, 4 and 7.

42 is sum of 9, 10, 11 and 12.



Algorithm:

The idea is to keep track of vertical distance from top diagonal passing through root. We increment the vertical distance we go down to next diagonal.

1. Add root with vertical distance as 0 to the queue.
2. Process the sum of all right child and right of right child and so on.
3. Add left child current node into the queue for later processing. The vertical distance of

left child is vertical distance of current node plus 1.

4. Keep doing 2nd, 3rd and 4th step till the queue is empty.

Following is the implementation of above idea.

C++

```
// C++ Program to find diagonal
// sum in a Binary Tree
#include <iostream>
#include <stdlib.h>
#include <map>
using namespace std;

struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* newNode(int data)
{
    struct Node* Node =
        (struct Node*)malloc(sizeof(struct Node));

    Node->data = data;
    Node->left = Node->right = NULL;

    return Node;
}

// root - root of the binary tree
// vd - vertical distance diagonally
// diagonalSum - map to store Diagonal
// Sum(Passed by Reference)
void diagonalSumUtil(struct Node* root,
                    int vd, map<int, int> &diagonalSum)
{
    if(!root)
        return;

    diagonalSum[vd] += root->data;

    // increase the vertical distance if left child
    diagonalSumUtil(root->left, vd + 1, diagonalSum);

    // vertical distance remains same for right child
    diagonalSumUtil(root->right, vd, diagonalSum);
}
```

```
}

// Function to calculate diagonal
// sum of given binary tree
void diagonalSum(struct Node* root)
{
    // create a map to store Diagonal Sum
    map<int, int> diagonalSum;

    diagonalSumUtil(root, 0, diagonalSum);

    map<int, int>::iterator it;
    cout << "Diagonal sum in a binary tree is - ";

    for(it = diagonalSum.begin();
        it != diagonalSum.end(); ++it)
    {
        cout << it->second << " ";
    }
}

// Driver code
int main()
{
    struct Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(9);
    root->left->right = newNode(6);
    root->right->left = newNode(4);
    root->right->right = newNode(5);
    root->right->left->right = newNode(7);
    root->right->left->left = newNode(12);
    root->left->right->left = newNode(11);
    root->left->left->right = newNode(10);

    diagonalSum(root);

    return 0;
}

// This code is contributed by Aditya Goel
```

Java

```
// Java Program to find diagonal sum in a Binary Tree
import java.util.*;
```

```
import java.util.Map.Entry;

//Tree node
class TreeNode
{
    int data; //node data
    int vd; //vertical distance diagonally
    TreeNode left, right; //left and right child's reference

    // Tree node constructor
    public TreeNode(int data)
    {
        this.data = data;
        vd = Integer.MAX_VALUE;
        left = right = null;
    }
}

// Tree class
class Tree
{
    TreeNode root; //Tree root

    // Tree constructor
    public Tree(TreeNode root) { this.root = root; }

    // Diagonal sum method
    public void diagonalSum()
    {
        // Queue which stores tree nodes
        Queue<TreeNode> queue = new LinkedList<TreeNode>();

        // Map to store sum of node's data lying diagonally
        Map<Integer, Integer> map = new TreeMap<>();

        // Assign the root's vertical distance as 0.
        root.vd = 0;

        // Add root node to the queue
        queue.add(root);

        // Loop while the queue is not empty
        while (!queue.isEmpty())
        {
            // Remove the front tree node from queue.
            TreeNode curr = queue.remove();

            // Get the vertical distance of the dequeued node.
```

```
        int vd = curr.vd;

        // Sum over this node's right-child, right-of-right-child
        // and so on
        while (curr != null)
        {
            int prevSum = (map.get(vd) == null)? 0: map.get(vd);
            map.put(vd, prevSum + curr.data);

            // If for any node the left child is not null add
            // it to the queue for future processing.
            if (curr.left != null)
            {
                curr.left.vd = vd+1;
                queue.add(curr.left);
            }

            // Move to the current node's right child.
            curr = curr.right;
        }
    }

    // Make an entry set from map.
    Set<Entry<Integer, Integer>> set = map.entrySet();

    // Make an iterator
    Iterator<Entry<Integer, Integer>> iterator = set.iterator();

    // Traverse the map elements using the iterator.
    System.out.print("Diagonal sum in a binary tree is - ");
    while (iterator.hasNext())
    {
        Map.Entry<Integer, Integer> me = iterator.next();

        System.out.print(me.getValue()+" ");
    }
}

//Driver class
public class DiagonalSum
{
    public static void main(String[] args)
    {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(9);
    }
}
```

```
        root.left.right = new TreeNode(6);
        root.right.left = new TreeNode(4);
        root.right.right = new TreeNode(5);
        root.right.left.left = new TreeNode(12);
        root.right.left.right = new TreeNode(7);
        root.left.right.left = new TreeNode(11);
        root.left.left.right = new TreeNode(10);
        Tree tree = new Tree(root);
        tree.diagonalSum();
    }
}
```

Output:

Diagonal sum in a binary tree is - 9 19 42

Exercise:

This problem was for diagonals from top to bottom and slope -1. Try the same problem for slope +1.

This article is contributed by **Kumar Gautam**. 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/diagonal-sum-binary-tree/>

Chapter 28

Equal sum array partition excluding a given element

Equal sum array partition excluding a given element - GeeksforGeeks

Given an array `arr[]` and an index in it. Find whether the array `arr[]` can be partitioned into two disjoint sets such that sum of both the sets is equal and none of the sets includes `arr[index]`

Examples :

```
Input : arr[] = {2, 1, 3, 4},
        index = 2
Output : No
We need to exclude arr[2] which is 3.
Possible sets are :
Set 1: (2, 1), Set 2: 4, sum = 3 4
Set 1: 2, Set 2: (4, 1), sum = 2 5
Set 1: 1, Set 2: (4, 2), sum = 1 6
Neither of the sums are equal.
```

```
Input : arr[] = {2, 5, 1, 4, 0},
        index = 4
Output : Yes
Set 1 : (2, 4), sum = 6
Set 2 : (5, 1), sum = 6
```

Approach: This problem is a variation of [partition problem](#) with an additional constraint that index cannot be included in neither of the partitioned sets of array.

First find sum `S` of array excluding the index-th element. If the sum is even then array can be partitioned otherwise not. If the sum is even then define two variables `set1Sum` and `set2Sum` to store the sum of two sets.

It can be determined recursively that whether set1Sum is equal to set2Sum. Start from position 0 and traverse the array recursively. At every array position, there are two choices: either include current array element in set 1 or in set 2. Recursively call for both the conditions, by including current element in set 1 first then in set 2. If current position is the index to be excluded then recursively call for next position without updating any sum. When entire array is traversed then check for equality of both sets sum. If the sums are equal then result is found otherwise backtrack and check for other possibilities.

Implementation:

C++

```
// C++ program to determine whether an array can be
// partitioned into two equal sum sets when an index
// is always excluded from both sets.
#include <bits/stdc++.h>
using namespace std;

// Utility function to partition array into two sets
// and check whether sum of both sets are equal or not.
bool isSubsetSumPoss(int arr[], int n, int set1Sum,
                    int set2Sum, int index, int pos)
{
    // If the entire array is traversed, then check
    // whether sum of both the sets are equal or not.
    if (pos == n)
        return (set1Sum == set2Sum);

    // If current position is the index to be excluded
    // then call the function for next position without
    // updating any sum.
    if (pos == index)
        isSubsetSumPoss(arr, n, set1Sum,
                        set2Sum, index, pos + 1);

    // Each element can be included either in
    // set 1 or in set 2. Call function for
    // both the cases.
    return isSubsetSumPoss(arr, n, set1Sum + arr[pos],
                          set2Sum, index, pos + 1)
        || isSubsetSumPoss(arr, n, set1Sum, set2Sum +
                          arr[pos], index, pos + 1);
}

// Function that calls the main utility
// function and returns whether array can
// be partitioned into two sets or not.
```



```
bool canPartition(int arr[], int n, int index)
{
    // Calculate sum of entire array
    // excluding position index.
    int sum = 0;

    for (int i = 0; i < n; i++) {
        if (i == index)
            continue;
        sum += arr[i];
    }

    // If sum is not even then array
    // cannot be partitioned into two
    // equal sum sets.
    if (sum % 2 != 0)
        return false;

    // If sum is even call utility function.
    return isSubsetSumPoss(arr, n, 0, 0,
                           index, 0);
}

int main()
{
    int arr[] = { 2, 5, 1, 4, 0 };
    int index = 4;
    int n = sizeof(arr) / sizeof(arr[0]);

    if (canPartition(arr, n, index))
        cout << "Yes";
    else
        cout << "No";
    return 0;
}
```

Java

```
// Java program to determine whether an array
// can be partitioned into two equal sum
// sets when an index is always excluded
// from both sets.
import java.io.*;
import java.util.*;

public class GFG {
```

```
// Utility function to partition array
// into two sets and check whether sum
// of both sets are equal or not.
static boolean isSubsetSumPoss(int []arr,
    int n, int set1Sum, int set2Sum,
    int index, int pos)
{

    // If the entire array is traversed,
    // then check whether sum of both
    // the sets are equal or not.
    if (pos == n)
        return (set1Sum == set2Sum);

    // If current position is the index
    // to be excluded then call the
    // function for next position without
    // updating any sum.
    if (pos == index)
        isSubsetSumPoss(arr, n, set1Sum,
            set2Sum, index, pos + 1);

    // Each element can be included
    // either in set 1 or in set 2.
    // Call function for both the cases.
    return isSubsetSumPoss(arr, n, set1Sum
        + arr[pos], set2Sum, index, pos + 1)
        || isSubsetSumPoss(arr, n, set1Sum,
            set2Sum + arr[pos], index, pos + 1);
}

// Function that calls the main utility
// function and returns whether array can
// be partitioned into two sets or not.
static boolean canPartition(int []arr, int n,
    int index)
{

    // Calculate sum of entire array
    // excluding position index.
    int sum = 0;

    for (int i = 0; i < n; i++) {
        if (i == index)
            continue;
        sum += arr[i];
    }
}
```

```
// If sum is not even then array
// cannot be partitioned into two
// equal sum sets.
if (sum % 2 != 0)
    return false;

// If sum is even call utility function.
return isSubsetSumPoss(arr, n, 0, 0,
                        index, 0);
}

// Driver code
public static void main(String args[])
{
    int []arr = { 2, 5, 1, 4, 0 };
    int index = 4;
    int n = arr.length;

    if (canPartition(arr, n, index))
        System.out.print("Yes");
    else
        System.out.print("No");
}

// This code is contributed by Manish Shaw
// (manishshaw1)
```

Python3

```
# Python3 program to determine whether an array can be
# partitioned into two equal sum sets when an index
# is always excluded from both sets.

# Utility function to partition array into two sets
# and check whether sum of both sets are equal or not.
def isSubsetSumPoss(arr, n, set1Sum, set2Sum, index, pos) :

    # If the entire array is traversed, then check
    # whether sum of both the sets are equal or not.
    if (pos == n) :
        return (set1Sum == set2Sum)

    # If current position is the index to be excluded
    # then call the function for next position without
    # updating any sum.
    if (pos == index) :
        isSubsetSumPoss(arr, n, set1Sum, set2Sum,
```

```
                                index, pos + 1)

# Each element can be included either in
# set 1 or in set 2. Call function for
# both the cases.
return (isSubsetSumPoss(arr, n, set1Sum + arr[pos],
                        set2Sum, index, pos + 1)
        or isSubsetSumPoss(arr, n, set1Sum,
                        set2Sum + arr[pos], index, pos + 1))

# Function that calls the main utility
# function and returns whether array can
# be partitioned into two sets or not.
def canPartition(arr, n, index) :

    # Calculate sum of entire array
    # excluding position index.
    sum = 0

    for i in range (0, n) :
        if (i == index) :
            continue
        sum += arr[i]

    # If sum is not even then array
    # cannot be partitioned into two
    # equal sum sets.
    if (sum % 2 != 0) :
        return false

    # If sum is even call utility function.
    return isSubsetSumPoss(arr, n, 0, 0, index, 0)

# Driver Code
arr = [ 2, 5, 1, 4, 0 ]
index = 4
n = len(arr)

if (canPartition(arr, n, index)) :
    print ("Yes")
else :
    print ("No")

# This code is contributed by Manish Shaw
# (manishshaw1)
```

C#

```
// C# program to determine whether an array
// can be partitioned into two equal sum
// sets when an index is always excluded
// from both sets.
using System;
using System.Collections.Generic;
using System.Linq;
using System.Collections;

class GFG {

    // Utility function to partition array
    // into two sets and check whether sum
    // of both sets are equal or not.
    static bool isSubsetSumPoss(int []arr,
        int n, int set1Sum, int set2Sum,
        int index, int pos)
    {

        // If the entire array is traversed,
        // then check whether sum of both
        // the sets are equal or not.
        if (pos == n)
            return (set1Sum == set2Sum);

        // If current position is the index
        // to be excluded then call the
        // function for next position without
        // updating any sum.
        if (pos == index)
            isSubsetSumPoss(arr, n, set1Sum,
                set2Sum, index, pos + 1);

        // Each element can be included
        // either in set 1 or in set 2.
        // Call function for both the cases.
        return isSubsetSumPoss(arr, n, set1Sum
            + arr[pos], set2Sum, index, pos + 1)
            || isSubsetSumPoss(arr, n, set1Sum,
                set2Sum + arr[pos], index, pos + 1);
    }

    // Function that calls the main utility
    // function and returns whether array can
    // be partitioned into two sets or not.
    static bool canPartition(int []arr, int n,
        int index)
    {
```

```
// Calculate sum of entire array
// excluding position index.
int sum = 0;

for (int i = 0; i < n; i++) {
    if (i == index)
        continue;
    sum += arr[i];
}

// If sum is not even then array
// cannot be partitioned into two
// equal sum sets.
if (sum % 2 != 0)
    return false;

// If sum is even call utility function.
return isSubsetSumPoss(arr, n, 0, 0,
                        index, 0);
}

// Driver code
public static void Main()
{
    int []arr = { 2, 5, 1, 4, 0 };
    int index = 4;
    int n = arr.Length;

    if (canPartition(arr, n, index))
        Console.WriteLine("Yes");
    else
        Console.WriteLine("No");
}

// This code is contributed by Manish Shaw
// (manishshaw1)
```

PHP

```
<?php
// PHP program to determine whether
// an array can be partitioned into
// two equal sum sets when an index
// is always excluded from both sets.

// Utility function to partition array
```

```
// into two sets and check whether sum
// of both sets are equal or not.
function isSubsetSumPoss($arr, $n, $set1Sum,
                        $set2Sum, $index, $pos)
{
    // If the entire array is traversed,
    // then check whether sum of both
    // the sets are equal or not.
    if ($pos == $n)
        return ($set1Sum == $set2Sum);

    // If current position is the index
    // to be excluded then call the
    // function for next position without
    // updating any sum.
    if ($pos == $index)
        isSubsetSumPoss($arr, $n, $set1Sum,
                        $set2Sum, $index,
                        $pos + 1);

    // Each element can be included
    // either in set 1 or in set 2.
    // Call function for both the cases.
    return isSubsetSumPoss($arr, $n, $set1Sum +
                            $arr[$pos], $set2Sum,
                            $index, $pos + 1) ||
        isSubsetSumPoss($arr, $n, $set1Sum,
                        $set2Sum + $arr[$pos],
                        $index, $pos + 1);
}

// Function that calls the main utility
// function and returns whether array can
// be partitioned into two sets or not.
function canPartition($arr, $n, $index)
{
    // Calculate sum of entire array
    // excluding position index.
    $sum = 0;

    for ($i = 0; $i < $n; $i++)
    {
        if ($i == $index)
            continue;
        $sum += $arr[$i];
    }
}
```

```
// If sum is not even then array
// cannot be partitioned into two
// equal sum sets.
if ($sum % 2 != 0)
    return false;

// If sum is even call
// utility function.
return isSubsetSumPoss($arr, $n, 0,
                      0, $index, 0);
}

// Driver Code
$arr = array( 2, 5, 1, 4, 0 );
$index = 4;
$n = count($arr);

if (canPartition($arr, $n, $index))
    echo ("Yes");
else
    echo ("No");

// This code is contributed by
// Manish Shaw (manishshaw1)
?>
```

Output :

Yes

Time Complexity : Exponential $O(2^n)$

Exercise : Try to solve this problem iteratively.

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/equal-sum-array-partition-excluding-given-element/>

Chapter 29

Euler Tour | Subtree Sum using Segment Tree

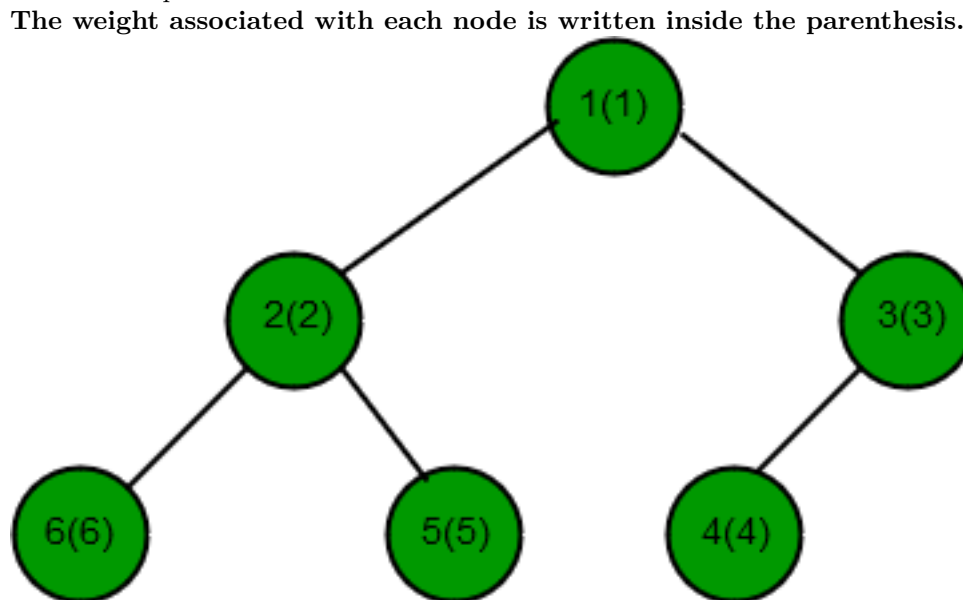
Euler Tour | Subtree Sum using Segment Tree - GeeksforGeeks

Euler tour tree (ETT) is a method for representing a rooted tree as a number sequence. When traversing the tree using [Depth for search\(DFS\)](#), insert each node in a vector twice, once while entered it and next after visiting all its children. This method is very useful for solving subtree problems and one such problem is **Subtree Sum**.

Prerequisite : [Segment Tree\(Sum of given range\)](#)

Naive Approach :

Consider a rooted tree with 6 vertices connected as given in the below diagram. Apply DFS for different queries.

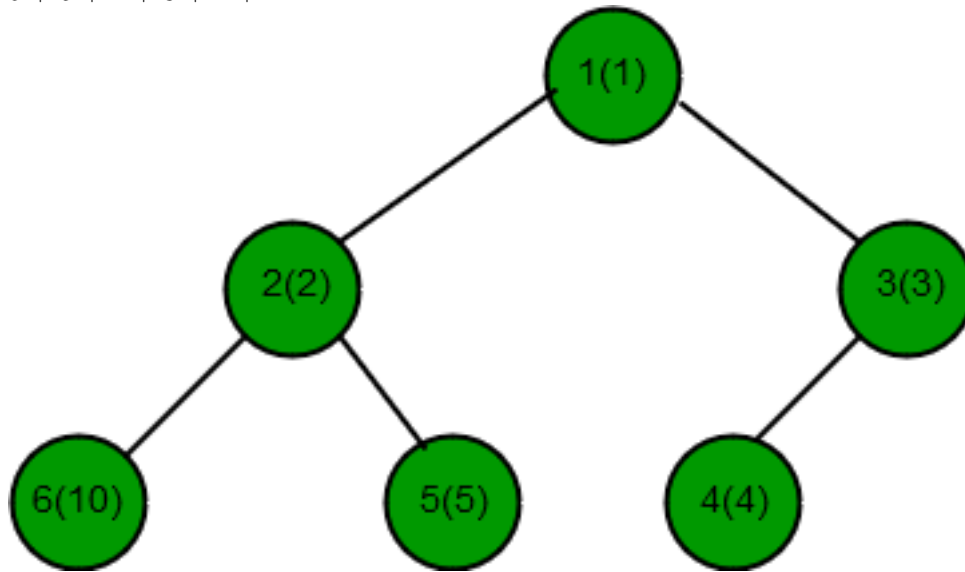


Queries :

1. Sum of all the subtrees of node 1.
2. Update the value of node 6 to 10.
3. Sum of all the subtrees of node 2.

Answers :

1. $6 + 5 + 4 + 3 + 2 + 1 = 21$



2.

3. $10 + 5 + 2 = 17$

Time Complexity Analysis :

Such queries can be performed using depth for search(dfs) in $O(n)$ time complexity.

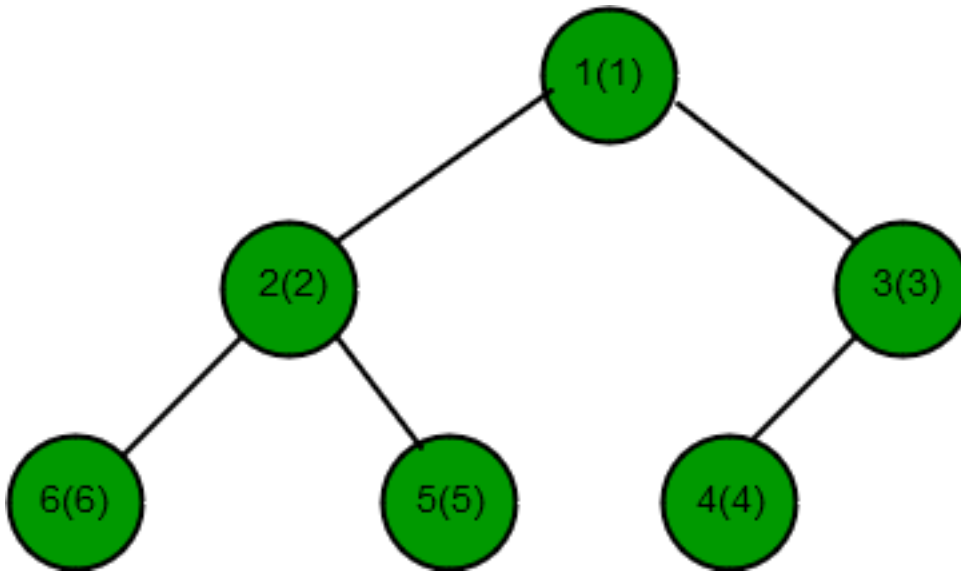
Efficient Approach :

The time complexity for such queries can be reduced to $O(\log(n))$ time by converting the rooted tree into segment tree using Euler tour technique. So, When the number of queries are q , the total complexity becomes $O(q * 5 \log(n))$.

Euler Tour :

In Euler tour Technique, each vertex is added to the vector twice, while descending into it and while leaving it.

Let us understand with the help of previous example :



On performing depth for search(DFS) using euler tour technique on the given rooted tree, the vector so formed is :

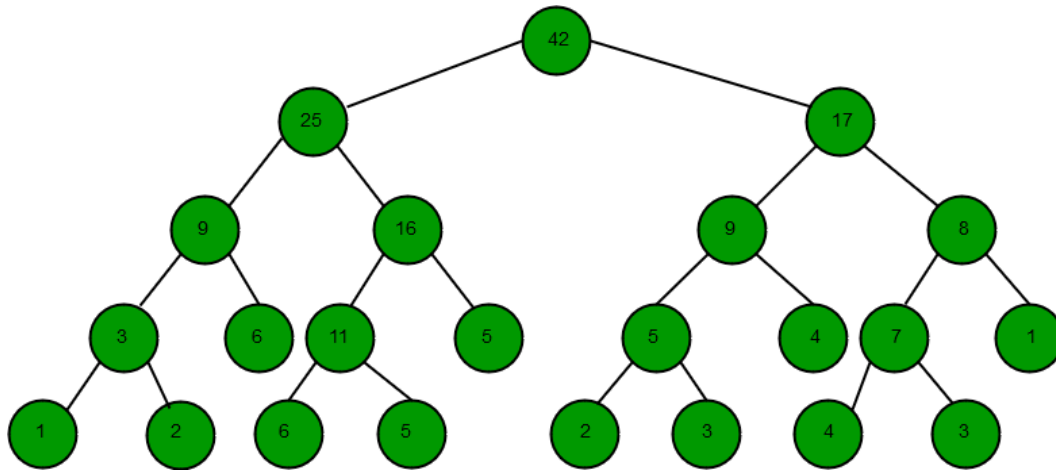
```
s[]={1, 2, 6, 6, 5, 5, 2, 3, 4, 4, 3, 1}
```

```
// DFS function to traverse the tree
int dfs(int root)
{
    s.push_back(root);
    if (v[root].size() == 0)
        return root;

    for (int i = 0; i < v[root].size(); i++) {
        int temp = dfs(v[root][i]);
        s.push_back(temp);
    }
    return root;
}
```

Now, use vector `s[]` to **Create Segment Tree**.

Below is the representation of segment tree of vector `s[]`.



For the output and update query, store the entry time and exit time(which serve as index range) for each node of the rooted tree.

`s[]={1, 2, 6, 6, 5, 5, 2, 3, 4, 4, 3, 1}`

Node	Entry time	Exit time
1	1	12
2	2	7
3	8	11
4	9	10
5	5	6
6	3	4

Query of type 1 :

Find the range sum on segment tree for output query where range is exit time and entry time of the rooted tree node. Deduce that the answer is always twice the expected answer because each node is added twice in segment tree. So reduce the answer by half.

Query of type 2 :

For update query, update the leaf node of segment tree at the entry time and exit time of the rooted tree node.

Below is the implementation of above approach :

```
// C++ program for implementation of
// Euler Tour | Subtree Sum.
#include <bits/stdc++.h>
using namespace std;

vector<int> v[1001];
vector<int> s;
int seg[1001] = { 0 };
```

```
// Value/Weight of each node of tree,
// value of 0th(no such node) node is 0.
int ar[] = { 0, 1, 2, 3, 4, 5, 6 };

int vertices = 6;
int edges = 5;

// A recursive function that constructs
// Segment Tree for array ar[] = { }.
// 'pos' is index of current node
// in segment tree seg[].
int segment(int low, int high, int pos)
{
    if (high == low) {
        seg[pos] = ar[s[low]];
    }
    else {
        int mid = (low + high) / 2;
        segment(low, mid, 2 * pos);
        segment(mid + 1, high, 2 * pos + 1);
        seg[pos] = seg[2 * pos] + seg[2 * pos + 1];
    }
}

/* Return sum of elements in range
   from index l to r . It uses the
   seg[] array created using segment()
   function. 'pos' is index of current
   node in segment tree seg[].
*/
int query(int node, int start,
          int end, int l, int r)
{
    if (r < start || end < l) {
        return 0;
    }

    if (l <= start && end <= r) {
        return seg[node];
    }

    int mid = (start + end) / 2;
    int p1 = query(2 * node, start,
                  mid, l, r);
    int p2 = query(2 * node + 1, mid + 1,
                  end, l, r);

    return (p1 + p2);
}
```

```
}

/* A recursive function to update the
   nodes which have the given index in
   their range. The following are
   parameters pos --> index of current
   node in segment tree seg[]. idx -->
   index of the element to be updated.
   This index is in input array.
   val --> Value to be change at node idx
*/
int update(int pos, int low, int high,
           int idx, int val)
{
    if (low == high) {
        seg[pos] = val;
    }
    else {
        int mid = (low + high) / 2;

        if (low <= idx && idx <= mid) {
            update(2 * pos, low, mid,
                  idx, val);
        }
        else {
            update(2 * pos + 1, mid + 1,
                  high, idx, val);
        }

        seg[pos] = seg[2 * pos] + seg[2 * pos + 1];
    }
}

/* A recursive function to form array
   ar[] from a directed tree .
*/
int dfs(int root)
{
    // pushing each node in vector s
    s.push_back(root);
    if (v[root].size() == 0)
        return root;

    for (int i = 0; i < v[root].size(); i++) {
        int temp = dfs(v[root][i]);
        s.push_back(temp);
    }
    return root;
}
```

```
}

// Driver program to test above functions
int main()
{
    // Edges between the nodes
    v[1].push_back(2);
    v[1].push_back(3);
    v[2].push_back(6);
    v[2].push_back(5);
    v[3].push_back(4);

    // Calling dfs function.
    int temp = dfs(1);
    s.push_back(temp);

    // Storing entry time and exit
    // time of each node
    vector<pair<int, int> > p;

    for (int i = 0; i <= vertices; i++)
        p.push_back(make_pair(0, 0));

    for (int i = 0; i < s.size(); i++) {
        if (p[s[i]].first == 0)
            p[s[i]].first = i + 1;
        else
            p[s[i]].second = i + 1;
    }

    // Build segment tree from array ar[].
    segment(0, s.size() - 1, 1);

    // query of type 1 return the
    // sum of subtree at node 1.
    int node = 1;
    int e = p[node].first;
    int f = p[node].second;

    int ans = query(1, 1, s.size(), e, f);

    // print the sum of subtree
    cout << "Subtree sum of node " << node << " is : " << (ans / 2) << endl;

    // query of type 2 return update
    // the subtree at node 6.
    int val = 10;
    node = 6;
```

```
e = p[node].first;
f = p[node].second;
update(1, 1, s.size(), e, val);
update(1, 1, s.size(), f, val);

// query of type 1 return the
// sum of subtree at node 2.
node = 2;

e = p[node].first;
f = p[node].second;

ans = query(1, 1, s.size(), e, f);

// print the sum of subtree
cout << "Subtree sum of node " << node << " is : " << (ans / 2) << endl;

return 0;
}
```

Output:

```
Subtree sum of node 1 is : 21
Subtree sum of node 2 is : 17
```

Time Complexity : $O(q \cdot \log(n))$

Source

<https://www.geeksforgeeks.org/euler-tour-subtree-sum-using-segment-tree/>

Chapter 30

Find Nth term (A matrix exponentiation example)

Find Nth term (A matrix exponentiation example) - GeeksforGeeks

We are given a recursive function that describes Nth terms in form of other terms. In this article we have taken specific example.

$$f(n) = f(n-1) + f(n-2) + f(n-3) + f(n-4) + f(n-5) + f(n-6) + f(n-7) + f(n-8) + f(n-9) + f(n-10) + f(n-11) + f(n-12) + f(n-13) + f(n-14) + f(n-15) + f(n-16) + f(n-17) + f(n-18) + f(n-19) + f(n-20)$$

Now you are given n, and you have to find out nth term using above formula.

Examples:

Input : n = 2

Output : 5

Input : n = 3

Output : 13

Prerequisite :

Basic Approach: This problem can be solved by simply just iterating over the n terms. Every time you find a term, using this term find next one and so on. But time complexity of this problem is of order $O(n)$.

Optimized Approach

All such problem where a term is a function of other terms in linear fashion. Then these can be solved using Matrix (Please refer : [Matrix Exponentiation](#)). First we make transformation matrix and then just use matrix exponentiation to find Nth term.

Step by Step method includes:

Step 1. Determine k the number of terms on which T(i) depends.

In our example T(i) depends on two terms.so, $k = 2$

Step 2. Determine initial values

As in this article T0=1, T1=1 are given.

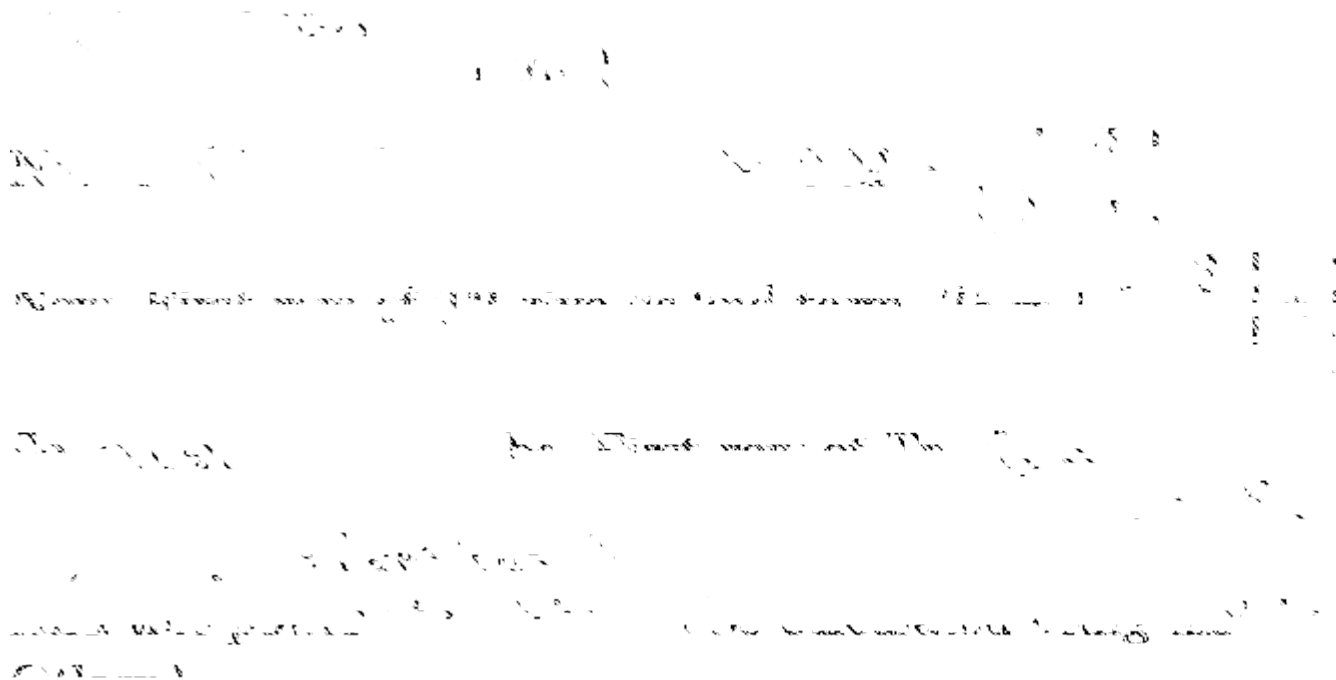
Step 3. Determine TM, the transformation matrix.

This is the most important step in solving recurrence relation. In this step, we have to make matrix of dimension $k \times k$.

Such that

$T(i) = TM * (\text{initial value vector})$

Here **initial value vector** is vector that contains initial value.we name this vector as **initial**.



Below is C++ program to implement above approach

```
// CPP program to find n-th term of a recursive
// function using matrix exponentiation.
#include <bits/stdc++.h>
using namespace std;
#define MOD 1000000009

#define ll long long int

ll power(ll n)
```

```

{
    if (n <= 1)
        return 1;

    // This power function returns first row of
    // {Transformation Matrix}n-1*Initial Vector
    n--;

    // This is an identity matrix.
    ll res[2][2] = { 1, 0, 0, 1 };

    // this is Transformation matrix.
    ll tMat[2][2] = { 2, 3, 1, 0 };

    // Matrix exponentiation to calculate power of {tMat}n-1
    // store res in "res" matrix.
    while (n) {

        if (n & 1) {
            ll tmp[2][2];
            tmp[0][0] = (res[0][0] * tMat[0][0] +
                        res[0][1] * tMat[1][0]) % MOD;
            tmp[0][1] = (res[0][0] * tMat[0][1] +
                        res[0][1] * tMat[1][1]) % MOD;
            tmp[1][0] = (res[1][0] * tMat[0][0] +
                        res[1][1] * tMat[1][0]) % MOD;
            tmp[1][1] = (res[1][0] * tMat[0][1] +
                        res[1][1] * tMat[1][1]) % MOD;
            res[0][0] = tmp[0][0];
            res[0][1] = tmp[0][1];
            res[1][0] = tmp[1][0];
            res[1][1] = tmp[1][1];
        }
        n = n / 2;
        ll tmp[2][2];
        tmp[0][0] = (tMat[0][0] * tMat[0][0] +
                    tMat[0][1] * tMat[1][0]) % MOD;
        tmp[0][1] = (tMat[0][0] * tMat[0][1] +
                    tMat[0][1] * tMat[1][1]) % MOD;
        tmp[1][0] = (tMat[1][0] * tMat[0][0] +
                    tMat[1][1] * tMat[1][0]) % MOD;
        tmp[1][1] = (tMat[1][0] * tMat[0][1] +
                    tMat[1][1] * tMat[1][1]) % MOD;
        tMat[0][0] = tmp[0][0];
        tMat[0][1] = tmp[0][1];
        tMat[1][0] = tmp[1][0];
        tMat[1][1] = tmp[1][1];
    }
}

```

```
// res store {Transformation matrix}^n-1
// hence will be first row of res*Initial Vector.
return (res[0][0] * 1 + res[0][1] * 1) % MOD;
}

// Driver code
int main()
{
    ll n = 3;
    cout << power(n);
    return 0;
}
```

Output:

13

Time Complexity : $O(\log n)$

The same idea is used to [find n-th Fibonacci number in \$O\(\log n\)\$](https://www.geeksforgeeks.org/find-nth-fibonacci-number-in-O-log-n/)

Source

<https://www.geeksforgeeks.org/find-nth-term-a-matrix-exponentiation-example/>

Chapter 31

Find all even length binary sequences with same sum of first and second half bits

Find all even length binary sequences with same sum of first and second half bits - Geeks-forGeeks

Given a number n , find all binary sequences of length $2n$ such that sum of first n bits is same as sum of last n bits.

Examples:

Input: $N = 2$

Output:

0101 1111 1001 0110 0000 1010

Input: $N = 3$

Output:

011011 001001 011101 010001 101011 111111
110011 101101 100001 110101 001010 011110
010010 001100 000000 010100 101110 100010
110110 100100

The idea is to fix first and last bits and then recur for remaining $2^{*(n-1)}$ bits. There are four possibilities when we fix first and last bits –

1. First and last bits are 1, remaining $n - 1$ bits on both sides should also have the same sum.
2. First and last bits are 0, remaining $n - 1$ bits on both sides should also have the same sum.

3. First bit is 1 and last bit is 0, sum of remaining $n - 1$ bits on left side should be 1 less than the sum $n-1$ bits on right side.
4. First bit is 0 and last bit is 1, sum of remaining $n - 1$ bits on left side should be 1 more than the sum $n-1$ bits on right side.

Below is implementation of above idea –

C++

```
// C++ program to print even length binary sequences
// whose sum of first and second half bits is same
#include <bits/stdc++.h>
using namespace std;

// Function to print even length binary sequences
// whose sum of first and second half bits is same

// diff --> difference between sums of first n bits
// and last n bits
// out --> output array
// start --> starting index
// end --> ending index
void findAllSequences(int diff, char* out, int start, int end)
{
    // We can't cover difference of more than n with 2n bits
    if (abs(diff) > (end - start + 1) / 2)
        return;

    // if all bits are filled
    if (start > end)
    {
        // if sum of first n bits and last n bits are same
        if (diff == 0)
            cout << out << " ";
        return;
    }

    // fill first bit as 0 and last bit as 1
    out[start] = '0', out[end] = '1';
    findAllSequences(diff + 1, out, start + 1, end - 1);

    // fill first and last bits as 1
    out[start] = out[end] = '1';
    findAllSequences(diff, out, start + 1, end - 1);

    // fill first and last bits as 0
    out[start] = out[end] = '0';
    findAllSequences(diff, out, start + 1, end - 1);
}
```

```
// fill first bit as 1 and last bit as 0
out[start] = '1', out[end] = '0';
findAllSequences(diff - 1, out, start + 1, end - 1);
}

// Driver program
int main()
{
    // input number
    int n = 2;

    // allocate string containing 2*n characters
    char out[2 * n + 1];

    // null terminate output array
    out[2 * n] = '\0';

    findAllSequences(0, out, 0, 2*n - 1);

    return 0;
}
```

Java

```
// Java program to print even length binary
// sequences whose sum of first and second
// half bits is same
import java.io.*;
import java.util.*;

class GFG
{
    // Function to print even length binary sequences
    // whose sum of first and second half bits is same

    // diff --> difference between sums of first n bits
    // and last n bits
    // out --> output array
    // start --> starting index
    // end --> ending index
    static void findAllSequences(int diff, char out[],
                                int start, int end)
    {
        // We can't cover difference of more
        // than n with 2n bits
        if (Math.abs(diff) > (end - start + 1) / 2)
            return;
    }
}
```

```
// if all bits are filled
if (start > end)
{
    // if sum of first n bits and
    // last n bits are same
    if (diff == 0)
    {
        System.out.print(out);
        System.out.print(" ");
    }
    return;
}

// fill first bit as 0 and last bit as 1
out[start] = '0';
out[end] = '1';
findAllSequences(diff + 1, out, start + 1, end - 1);

// fill first and last bits as 1
out[start] = out[end] = '1';
findAllSequences(diff, out, start + 1, end - 1);

// fill first and last bits as 0
out[start] = out[end] = '0';
findAllSequences(diff, out, start + 1, end - 1);

// fill first bit as 1 and last bit as 0
out[start] = '1';
out[end] = '0';
findAllSequences(diff - 1, out, start + 1, end - 1);
}

// Driver program
public static void main (String[] args)
{
    // input number
    int n = 2;

    // allocate string containing 2*n characters
    char[] out = new char[2 * n + 1];

    // null terminate output array
    out[2 * n] = '\0';

    findAllSequences(0, out, 0, 2*n - 1);
}
}
```


// This code is contributed by Pramod Kumar

C#

```
// C# program to print even length binary
// sequences whose sum of first and second
// half bits is same
using System;

class GFG {

    // Function to print even length binary
    // sequences whose sum of first and
    // second half bits is same

    // diff --> difference between sums of
    // first n bits
    // and last n bits
    // out --> output array
    // start --> starting index
    // end --> ending index
    static void findAllSequences(int diff,
                                char []outt, int start, int end)
    {

        // We can't cover difference of
        // more than n with 2n bits
        if (Math.Abs(diff) > (end - start
                               + 1) / 2)
            return;

        // if all bits are filled
        if (start > end)
        {

            // if sum of first n bits and
            // last n bits are same
            if (diff == 0)
            {
                Console.Write(outt);
                Console.Write(" ");
            }
            return;
        }

        // fill first bit as 0 and last bit
        // as 1
```

```
    outt[start] = '0';
    outt[end] = '1';
    findAllSequences(diff + 1, outt,
                     start + 1, end - 1);

    // fill first and last bits as 1
    outt[start] = outt[end] = '1';
    findAllSequences(diff, outt,
                     start + 1, end - 1);

    // fill first and last bits as 0
    outt[start] = outt[end] = '0';
    findAllSequences(diff, outt,
                     start + 1, end - 1);

    // fill first bit as 1 and last
    // bit as 0
    outt[start] = '1';
    outt[end] = '0';
    findAllSequences(diff - 1, outt,
                     start + 1, end - 1);
}

// Driver program
public static void Main ()
{
    // input number
    int n = 2;

    // allocate string containing 2*n
    // characters
    char []outt = new char[2 * n + 1];

    // null terminate output array
    outt[2 * n] = '\0';

    findAllSequences(0, outt, 0, 2*n - 1);
}

// This code is contributed by nitin mittal.
```

Output:

0101 1111 1001 0110 0000 1010

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/find-all-even-length-binary-sequences-with-same-sum-of-first-and-second-half-bits/>

Chapter 32

Find m-th summation of first n natural numbers.

Find m-th summation of first n natural numbers. - GeeksforGeeks
m-th summation of first n natural numbers is defined as following.

```
If m > 1
    SUM(n, m) = SUM(SUM(n, m - 1), 1)
Else
    SUM(n, 1) = Sum of first n natural numbers.
```

We are given m and n, we need to find SUM(n, m).

Examples:

```
Input   : n = 4, m = 1
Output  : SUM(4, 1) = 10
Explanation : 1 + 2 + 3 + 4 = 10
```

```
Input   : n = 3, m = 2
Output  : SUM(3, 2) = 21
Explanation : SUM(3, 2)
              = SUM(SUM(3, 1), 1)
              = SUM(6, 1)
              = 21
```

Naive Approach : We can solve this problem using two nested loop, where outer loop iterate for m and inner loop iterate for n. After completion of a single outer iteration, we should update n as whole of inner loop got executed and value of n must be changed then. Time complexity should be $O(n*m)$.

```
for (int i = 1; i <= m; i++)
{
    sum = 0;
    for (int j = 1; j <= n; j++)
        sum += j;
    n = sum; // update n
}
```

Efficient Approach :

We can use direct formula for sum of first n numbers to reduce time.

We can also use recursion. In this approach $m = 1$ will be our base condition and for any intermediate step $SUM(n, m)$, we will call $SUM(SUM(n, m-1), 1)$ and for a single step $SUM(n, 1) = n * (n + 1) / 2$ will be used. This will reduce our time complexity to $O(m)$.

```
int SUM (int n, int m)
{
    if (m == 1)
        return (n * (n + 1) / 2);
    int sum = SUM(n, m-1);
    return (sum * (sum + 1) / 2);
}
```

Below is the implementation of above idea :

C++

```
// CPP program to find m-th summation
#include <bits/stdc++.h>
using namespace std;

// Function to return mth summation
int SUM(int n, int m)
{
    // base case
    if (m == 1)
        return (n * (n + 1) / 2);

    int sum = SUM(n, m-1);
    return (sum * (sum + 1) / 2);
}

// driver program
int main()
{
    int n = 5;
    int m = 3;
```

```
    cout << "SUM(" << n << ", " << m
        << "): " << SUM(n, m);
    return 0;
}
```

Java

```
// Java program to find m-th summation.
class GFG {

    // Function to return mth summation
    static int SUM(int n, int m) {

        // base case
        if (m == 1)
            return (n * (n + 1) / 2);

        int sum = SUM(n, m - 1);

        return (sum * (sum + 1) / 2);
    }

    // Driver code
    public static void main(String[] args) {

        int n = 5;
        int m = 3;

        System.out.println("SUM(" + n + ", "
            + m + "): " + SUM(n, m));
    }
}

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

Python3

```
# Python3 program to find m-th summation

# Function to return mth summation
def SUM(n, m):

    # base case
    if (m == 1):
        return (n * (n + 1) / 2)

    sum = SUM(n, m-1)
```

```
return int(sum * (sum + 1) / 2)
```

```
# driver program
n = 5
m = 3
print("SUM(", n, ", ", m, "):", SUM(n, m))

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

C#

```
// C# program to find m-th summation.
using System;

class GFG
{
    // Function to return mth summation
    static int SUM(int n, int m)
    {
        // base case
        if (m == 1)
            return (n * (n + 1) / 2);

        int sum = SUM(n, m - 1);

        return (sum * (sum + 1) / 2);
    }

    // Driver Code
    public static void Main()
    {
        int n = 5;
        int m = 3;

        Console.WriteLine("SUM(" + n + ", "
            + m + "): " + SUM(n, m));
    }
}

// This code is contributed by Nitin Mittal.
```

PHP

```
<?php
```

```
// PHP program to find m-th summation

// Function to return
// mth summation
function SUM($n, $m)
{
    // base case
    if ($m == 1)
        return ($n * ($n + 1) / 2);

    $sum = SUM($n, $m - 1);
    return ($sum * ($sum + 1) / 2);
}

// Driver Code
$n = 5;
$m = 3;
echo "SUM(" , $n , ", " , $m ,
    "): " , SUM($n, $m);

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

Output:

SUM(5, 3): 7260

Improved By : [nitin mittal](#), [vt_m](#)

Source

<https://www.geeksforgeeks.org/find-mth-summation-first-n-natural-numbers/>

Chapter 33

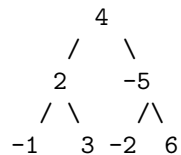
Find maximum level product in Binary Tree

Find maximum level product in Binary Tree - GeeksforGeeks

Given a Binary Tree having positive and negative nodes, the task is to find maximum product level in it.

Examples:

Input :



Output: 36

Explanation :

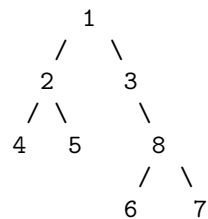
Product of all nodes of 0'th level is 4

Product of all nodes of 1'th level is -10

Product of all nodes of 0'th level is 36

Hence maximum product is 6

Input :



Output : 160

Explanation :

Product of all nodes of 0'th level is 1

Product of all nodes of 1'th level is 6
Product of all nodes of 0'th level is 160
Product of all nodes of 0'th level is 42
Hence maximum product is 160

Prerequisites: [Maximum Width of a Binary Tree](#)

Approach : The idea is to do level order traversal of tree. While doing traversal, process nodes of different level separately. For every level being processed, compute product of nodes in the level and keep track of maximum product.

```
// A queue based C++ program to find maximum product
// of a level in Binary Tree
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to left child
and a pointer to right child */
struct Node {
    int data;
    struct Node *left, *right;
};

// Function to find the maximum product of a level in tree
// using level order traversal
int maxLevelProduct(struct Node* root)
{
    // Base case
    if (root == NULL)
        return 0;

    // Initialize result
    int result = root->data;

    // Do Level order traversal keeping track of number
    // of nodes at every level.
    queue<Node*> q;
    q.push(root);
    while (!q.empty()) {

        // Get the size of queue when the level order
        // traversal for one level finishes
        int count = q.size();

        // Iterate for all the nodes in the queue currently
        int product = 1;
        while (count-->0) {
```

```
        // Dequeue an node from queue
        Node* temp = q.front();
        q.pop();

        // Multiply this node's value to current product.
        product = product * temp->data;

        // Enqueue left and right children of
        // dequeued node
        if (temp->left != NULL)
            q.push(temp->left);
        if (temp->right != NULL)
            q.push(temp->right);
    }

    // Update the maximum node count value
    result = max(product, result);
}

return result;
}
```

```
/* Helper function that allocates a new node with the
given data and NULL left and right pointers. */
```

```
struct Node* newNode(int data)
{
    struct Node* node = new Node;
    node->data = data;
    node->left = node->right = NULL;
    return (node);
}
```

```
// Driver code
```

```
int main()
{
    struct Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->right = newNode(8);
    root->right->right->left = newNode(6);
    root->right->right->right = newNode(7);
```

```
/* Constructed Binary tree is:
```

```
    1
   / \
  2   3
```

```

      / \  \
     4  5  8
      \ \
       6  7 */
cout << "Maximum level product is "
    << maxLevelProduct(root) << endl;
return 0;
}
```

Output :

Maximum level product is 160

Time Complexity : $O(n)$

Auxiliary Space : $O(n)$

Source

<https://www.geeksforgeeks.org/find-maximum-level-product-binary-tree/>

Chapter 34

Find middle of singly linked list Recursively

Find middle of singly linked list Recursively - GeeksforGeeks

Given a singly linked list and the task is to find middle of linked list.

Examples:

Input : 1->2->3->4->5

Output : 3

Input : 1->2->3->4->5->6

Output : 4

We have already discussed [Iterative Solution](#). In this post iterative solution is discussed. Count total number of nodes in the list in recursive manner and do half of this, suppose this value is n. Then rolling back through recursion decrement n by one for each call. Return the node where n is zero.

```
// C++ program for Recursive approach to find
// middle of singly linked list
#include <iostream>
using namespace std;

// Tree Node Structure
struct Node
{
    int data;
    struct Node* next;
};
```

```
// Create new Node
Node* newLNode(int data)
{
    Node* temp = new Node;
    temp->data = data;
    temp->next = NULL;
    return temp;
}

// Function for finding midpoint recursively
void midpoint_util(Node* head, int* n, Node** mid)
{
    // If we reached end of linked list
    if (head == NULL)
    {
        *n = (*n) / 2;
        return;
    }

    *n = *n + 1;

    midpoint_util(head->next, n, mid);

    // Rolling back, decrement n by one
    *n = *n - 1;
    if (*n == 0)
    {
        // Final answer
        *mid = head;
    }
}

Node* midpoint(Node* head)
{
    Node* mid = NULL;
    int n = 1;
    midpoint_util(head, &n, &mid);
    return mid;
}

int main()
{
    Node* head = newLNode(1);
    head->next = newLNode(2);
    head->next->next = newLNode(3);
    head->next->next->next = newLNode(4);
}
```

```
    head->next->next->next->next = newLNode(5);  
    Node* result = midpoint(head);  
    cout << result->data << endl;  
    return 0;  
}
```

Output:

3

Source

<https://www.geeksforgeeks.org/find-middle-singly-linked-list-recursively/>

Chapter 35

Find n-th lexicographically permutation of a string | Set 2

Find n-th lexicographically permutation of a string | Set 2 - GeeksforGeeks

Given a string of length m containing lowercase alphabets only. We need to find the n-th permutation of string lexicographically.

Examples:

```
Input : str[] = "abc", n = 3
Output : Result = "bac"
Explanation : All possible permutation
in sorted order: abc, acb, bac, bca,
cab, cba
```

```
Input : str[] = "aba", n = 2
Output : Result = "aba"
Explanation : All possible permutation
in sorted order: aab, aba, baa
```

We have discussed how to [find lexicographically n-th permutation using STL](#). Time complexity of previous approach is $O(n * n!)$ which is quite high.

Here we use some Mathematical concept for solving this problem.

The idea is based on below facts.

- The total number of permutation of a string formed by N characters (all distinct) is $N!$
- The Total number of permutation of a string formed by N characters (where the frequency of character C1 is M1, C2 is M2... and so the frequency of character Ck is Mk) is $N! / (M1! * M2! * \dots * Mk!)$.

- The total number of permutation of a string formed by N characters (all distinct) after fixing the first character is $(N-1)!$

We first count frequencies of all characters in an array `freq[]`. Now from the first smallest character present in the string (smallest index i such that `freq[i] > 0`), we compute the number of maximum permutation possible after setting that particular i -th character as the first character. If this sum value is more than given n , then we set that character as the first result output character, decrement `freq[i]`, and continue same for remaining $n-1$ characters. On the other hand, if the count is less than the required n , iterate for the next character in the frequency table and update the count over and over again until we find a character that produces a count greater than the required n .

Time Complexity: $O(n)$ i.e. order of string length

C++

```
// C++ program to print n-th permutation
#include <bits/stdc++.h>
using namespace std;

#define ll long long int

const int MAX_CHAR = 26;
const int MAX_FACT = 20;
ll fact[MAX_FACT];

// utility for calculating factorials
void precomputeFactorials()
{
    fact[0] = 1;
    for (int i = 1; i < MAX_FACT; i++)
        fact[i] = fact[i - 1] * i;
}

// function for nth permutation
void nPermute(char str[], int n)
{
    precomputeFactorials();

    // length of given string
    int len = strlen(str);

    // Count frequencies of all
    // characters
    int freq[MAX_CHAR] = { 0 };
    for (int i = 0; i < len; i++)
        freq[str[i] - 'a']++;

    // out string for output string
```

```
char out[MAX_CHAR];

// iterate till sum equals n
int sum = 0;
int k = 0;

// We update both n and sum in this
// loop.
while (sum != n) {

    sum = 0;
    // check for characters present in freq[]
    for (int i = 0; i < MAX_CHAR; i++) {
        if (freq[i] == 0)
            continue;

        // Remove character
        freq[i]--;

        // calculate sum after fixing
        // a particular char
        int xsum = fact[len - 1 - k];
        for (int j = 0; j < MAX_CHAR; j++)
            xsum /= fact[freq[j]];
        sum += xsum;

        // if sum > n fix that char as
        // present char and update sum
        // and required nth after fixing
        // char at that position
        if (sum >= n) {
            out[k++] = i + 'a';
            n -= (sum - xsum);
            break;
        }

        // if sum < n, add character back
        if (sum < n)
            freq[i]++;
    }
}

// if sum == n means this char will provide its
// greatest permutation as nth permutation
for (int i=MAX_CHAR-1; k < len && i >= 0; i--)
    if (freq[i]) {
        out[k++] = i + 'a';
        freq[i]--;
    }
```

```
    }

    // append string termination
    // character and print result
    out[k] = '\\0';
    cout << out;
}

// Driver program
int main()
{
    int n = 2;
    char str[] = "geeksquiz";

    nPermute(str, n);
    return 0;
}
```

Java

```
// Java program to print n-th permutation
public class PermuteString
{
    final static int MAX_CHAR = 26;
    final static int MAX_FACT = 20;
    static long fact[] = new long[MAX_FACT];

    // utility for calculating factorial
    static void precomputeFactorials()
    {
        fact[0] = 1;
        for (int i = 1; i < MAX_FACT; i++)
            fact[i] = fact[i - 1] * i;
    }

    // function for nth permutation
    static void nPermute(String str, int n)
    {
        precomputeFactorials();

        // length of given string
        int len = str.length();

        // Count frequencies of all
        // characters
        int freq[] = new int[MAX_CHAR];
        for (int i = 0; i < len; i++)
            freq[str.charAt(i) - 'a']++;
    }
}
```

```
// out string for output string
String out = "";

// iterate till sum equals n
int sum = 10;
int k = 0;

// We update both n and sum in this
// loop.
while (sum >= n) {

    // check for characters present in freq[]
    for (int i = 0; i < MAX_CHAR; i++) {
        if (freq[i] == 0)
            continue;

        // Remove character
        freq[i]--;

        // calculate sum after fixing
        // a particular char
        sum = 0;
        int xsum = (int) fact[len - 1 - k];
        for (int j = 0; j < MAX_CHAR; j++)
            xsum /= fact[freq[j]];
        sum += xsum;

        // if sum > n fix that char as
        // present char and update sum
        // and required nth after fixing
        // char at that position
        if (sum >= n) {
            out += (char)(i + 'a');
            k++;
            n -= (sum - xsum);
            break;
        }

        // if sum < n, add character back
        if (sum < n)
            freq[i]++;
    }

    // if sum == n means this char will provide its
    // greatest permutation as nth permutation
    for (int i = MAX_CHAR - 1; k < len && i >= 0; i--)
```

```
        if (freq[i] != 0) {
            out += (char)(i + 'a');
            freq[i++]--;
        }

        // append string termination
        // character and print result
        System.out.println(out);
    }

    // Driver program to test above method
    public static void main(String[] args) {

        // TODO Auto-generated method stub
        int n = 2;
        String str = "geeksquiz";

        nPermute(str, n);
    }
}
// This code is contributed by Sumit Ghosh
```

C#

```
// C# program to print n-th permutation
using System;

public class GFG {

    static int MAX_CHAR = 26;
    static int MAX_FACT = 20;
    static long []fact = new long[MAX_FACT];

    // utility for calculating factorial
    static void precomputeFactorials()
    {
        fact[0] = 1;
        for (int i = 1; i < MAX_FACT; i++)
            fact[i] = fact[i - 1] * i;
    }

    // function for nth permutation
    static void nPermute(String str, int n)
    {
        precomputeFactorials();

        // length of given string
        int len = str.Length;
```

```
// Count frequencies of all
// characters
int []freq = new int[MAX_CHAR];

for (int i = 0; i < len; i++)
    freq[str[i] - 'a']++;

// out string for output string
string ou = "";

// iterate till sum equals n
int sum = 10;
int k = 0;

// We update both n and sum in this
// loop.
while (sum >= n) {

    // check for characters present in freq[]
    for (int i = 0; i < MAX_CHAR; i++) {
        if (freq[i] == 0)
            continue;

        // Remove character
        freq[i]--;

        // calculate sum after fixing
        // a particular char
        sum = 0;
        int xsum = (int) fact[len - 1 - k];

        for (int j = 0; j < MAX_CHAR; j++)
            xsum /= (int)(fact[freq[j]]);

        sum += xsum;

        // if sum > n fix that char as
        // present char and update sum
        // and required nth after fixing
        // char at that position
        if (sum >= n) {
            ou += (char)(i + 'a');
            k++;
            n -= (sum - xsum);
            break;
        }
    }
}
```

```
        // if sum < n, add character back
        if (sum < n)
            freq[i]++;
    }
}

// if sum == n means this char will provide its
// greatest permutation as nth permutation
for (int i = MAX_CHAR - 1; k < len && i >= 0; i--)
    if (freq[i] != 0) {
        ou += (char)(i + 'a');
        freq[i]--;
    }

// append string termination
// character and print result
Console.WriteLine(ou);
}

// Driver program to test above method
public static void Main() {

    // TODO Auto-generated method stub
    int n = 2;
    String str = "geeksquiz";

    nPermute(str, n);
}

// This code is contributed by nitin mittal.
```

Output:

eegikqszu

Improved By : [nitin mittal](#), [pvsdileep](#)

Source

<https://www.geeksforgeeks.org/find-n-th-lexicographically-permutation-string-set-2/>

Chapter 36

Find n-th node in Postorder traversal of a Binary Tree

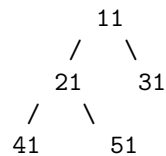
Find n-th node in Postorder traversal of a Binary Tree - GeeksforGeeks

Given a Binary tree and a number N, write a program to find the N-th node in the Postorder traversal of the given Binary tree.

Prerequisite: [Tree Traversal](#)

Examples:

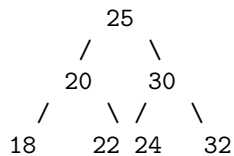
Input : N = 4



Output : 31

Explanation: Postorder Traversal of given Binary Tree is 41 51 21 31 11, so 4th node will be 31.

Input : N = 5



Output : 32

The idea to solve this problem is to do [postorder traversal](#) of the given binary tree and keep track of the count of nodes visited while traversing the tree and print the current node when the count becomes equal to N.

Below is the implementation of the above approach:

```
// C++ program to find n-th node of
// Postorder Traversal of Binary Tree
#include <bits/stdc++.h>
using namespace std;

// node of tree
struct Node {
    int data;
    Node *left, *right;
};

// function to create a new node
struct Node* createNode(int item)
{
    Node* temp = new Node;
    temp->data = item;
    temp->left = NULL;
    temp->right = NULL;

    return temp;
}

// function to find the N-th node in the postorder
// traversal of a given binary tree
void NthPostordernode(struct Node* root, int N)
{
    static int flag = 0;

    if (root == NULL)
        return;

    if (flag <= N) {

        // left recursion
        NthPostordernode(root->left, N);

        // right recursion
        NthPostordernode(root->right, N);

        flag++;

        // prints the n-th node of preorder traversal
        if (flag == N)
            cout << root->data;
    }
}
```

```
// driver code
int main()
{
    struct Node* root = createNode(25);
    root->left = createNode(20);
    root->right = createNode(30);
    root->left->left = createNode(18);
    root->left->right = createNode(22);
    root->right->left = createNode(24);
    root->right->right = createNode(32);

    int N = 6;

    // prints n-th node found
    NthPostordernode(root, N);

    return 0;
}
```

Output:

30

Time Complexity: $O(n)$, where n is the number of nodes in the given binary tree.

Auxiliary Space: $O(1)$

Source

<https://www.geeksforgeeks.org/find-n-th-node-in-postorder-traversal-of-a-binary-tree/>

Chapter 37

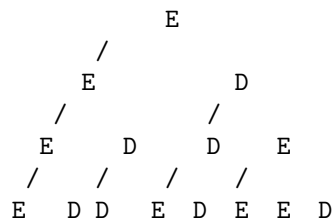
Find profession in a special family

Find profession in a special family - GeeksforGeeks

Consider a special family of Engineers and Doctors with following rules :

1. Everybody has two children.
2. First child of an Engineer is an Engineer and second child is a Doctor.
3. First child of an Doctor is Doctor and second child is an Engineer.
4. All generations of Doctors and Engineers start with Engineer.

We can represent the situation using below diagram:



Given level and position of a person in above ancestor tree, find the profession of the person.

Examples :

Input : level = 4, pos = 2
Output : Doctor

Input : level = 3, pos = 4
Output : Engineer

Method 1 (Recursive)

The idea is based on the fact that profession of a person depends on following two.

1. Profession of parent.
2. Position of node : If position of a node is odd, then its profession is same as its parent. Else profession is different from its parent.

We recursively find the profession of parent, then use point 2 above to find the profession of current node.

Below is implementation of above idea.

C++

```
// C++ program to find profession of a person at
// given level and position.
#include<bits/stdc++.h>
using namespace std;

// Returns 'e' if profession of node at given level
// and position is engineer. Else doctor. The function
// assumes that given position and level have valid values.
char findProffesion(int level, int pos)
{
    // Base case
    if (level == 1)
        return 'e';

    // Recursively find parent's profession. If parent
    // is a doctar, this node will be a doctal if it is
    // at odd position and an engineer if at even position
    if (findProffesion(level-1, (pos+1)/2) == 'd')
        return (pos%2)? 'd' : 'e';

    // If parent is an engineer, then current node will be
    // an enginner if at add position and doctor if even
    // position.
    return (pos%2)? 'e' : 'd';
}

// Driver code
int main(void)
{
    int level = 4, pos = 2;
    (findProffesion(level, pos) == 'e')? cout << "Engineer"
                                         : cout << "Doctor" ;

    return 0;
}
```

Java

```
// Java program to find
// profession of a person
// at given level and position
import java.io.*;

class GFG
{
    // Returns 'e' if profession
    // of node at given level
    // and position is engineer.
    // Else doctor. The function
    // assumes that given position
    // and level have valid values.
    static char findProffesion(int level,
                                int pos)
    {
        // Base case
        if (level == 1)
            return 'e';

        // Recursively find parent's
        // profession. If parent
        // is a doctar, this node
        // will be a doctal if it
        // is at odd position and an
        // engineer if at even position
        if (findProffesion(level - 1,
                            (pos + 1) / 2) == 'd')
            return (pos % 2 > 0) ?
                    'd' : 'e';

        // If parent is an engineer,
        // then current node will be
        // an enginner if at add
        // position and doctor if even
        // position.
        return (pos % 2 > 0) ?
                'e' : 'd';
    }

    // Driver code
    public static void main (String[] args)
    {
        int level = 4, pos = 2;
        if(findProffesion(level,
```

```
        pos) == 'e')
    System.out.println("Engineer");
    else
        System.out.println("Doctor");
}
}

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

C#

```
// C# program to find
// profession of a person
// at given level and position
using System;

class GFG
{
    // Returns 'e' if profession
    // of node at given level
    // and position is engineer.
    // Else doctor. The function
    // assumes that given position
    // and level have valid values.
    static char findProffesion(int level,
                                int pos)
    {
        // Base case
        if (level == 1)
            return 'e';

        // Recursively find parent's
        // profession. If parent
        // is a doctar, this node
        // will be a doctal if it
        // is at odd position and an
        // engineer if at even position
        if (findProffesion(level - 1,
                            (pos + 1) / 2) == 'd')
            return (pos % 2 > 0) ?
                'd' : 'e';

        // If parent is an engineer,
        // then current node will be
        // an enginner if at add
        // position and doctor if even
    }
}
```

```
// position.
return (pos % 2 > 0) ?
        'e' : 'd';
}

// Driver code
public static void Main ()
{
    int level = 4, pos = 2;
    if(findProffesion(level,
        pos) == 'e')
        Console.WriteLine("Engineer");
    else
        Console.WriteLine("Doctor");
}
}
```

// This code is contributed
// by anuj_67.

PHP

```
<?php
// PHP program to find profession
// of a person at given level
// and position.

// Returns 'e' if profession of
// node at given level and position
// is engineer. Else doctor. The
// function assumes that given
// position and level have valid values.
function findProffesion($level, $pos)
{
    // Base case
    if ($level == 1)
        return 'e';

    // Recursively find parent's
    // profession. If parent is
    // a Doctor, this node will
    // be a doctor if it is at
    // odd position and an engineer
    // if at even position
    if (findProffesion($level - 1,
        ($pos + 1) / 2) == 'd')
        return ($pos % 2) ? 'd' : 'e';
}
```

```
// If parent is an engineer, then
// current node will be an enginner
// if at odd position and doctor
// if even position.
return ($pos % 2) ? 'e' : 'd';
}

// Driver code
$level = 4; $pos = 2;
if((findProffesion($level,
                    $pos) == 'e') == true)
    echo "Engineer";
else
    echo "Doctor" ;

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

Output :

Doctor

Method 2 (Using Bitwise Operators)

```
Level 1: E
Level 2: ED
Level 3: EDDE
Level 4: EDDEDEED
Level 5: EDDEDEEDDEEDDEDE
```

Level input isn't necessary (if we ignore max position limit) because first elements are same.

The result is based on count of 1's in binary representation of position minus one. If count of 1's is even then result is Engineer, else then Doctor.

And of course position limit is $2^{(\text{Level}-1)}$

C++

```
// C++ program to find profession of a person at
// given level and position.
#include<bits/stdc++.h>
using namespace std;
```



```
/* Function to get no of set bits in binary
   representation of passed binary no. */
int countSetBits(int n)
{
    int count = 0;
    while (n)
    {
        n &= (n-1) ;
        count++;
    }
    return count;
}

// Returns 'e' if profession of node at given level
// and position is engineer. Else doctor. The function
// assumes that given position and level have valid values.
char findProffesion(int level, int pos)
{
    // Count set bits in 'pos-1'
    int c = countSetBits(pos-1);

    // If set bit count is odd, then doctor, else engineer
    return (c%2)? 'd' : 'e';
}

// Driver code
int main(void)
{
    int level = 3, pos = 4;
    (findProffesion(level, pos) == 'e')? cout << "Engineer"
                                         : cout << "Doctor" ;

    return 0;
}
```

PHP

```
<?php
// PHP program to find profession
// of a person at given level and position.

// Function to get no of set
// bits in binary representation
// of passed binary no.
function countSetBits($n)
{
    $count = 0;
    while ($n)
    {
```

```
        $n &= ($n - 1) ;
        $count++;
    }
    return $count;
}

// Returns 'e' if profession of
// node at given level and position
// is engineer. Else doctor. The
// function assumes that given
// position and level have valid values.
function findProffesion($level, $pos)
{
    // Count set bits in 'pos-1'
    $c = countSetBits($pos - 1);

    // If set bit count is odd,
    // then doctor, else engineer
    return ($c % 2) ? 'd' : 'e';
}

// Driver Code
$level = 3;
$pos = 4;
if((findProffesion($level,
                    $pos) == 'e') == true)
    echo "Engineer \n";
else
    echo "Doctor \n";

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

Output :

Engineer

Thanks to Furkan Uslu for suggesting this method.

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

Source

<https://www.geeksforgeeks.org/find-profession-in-a-hypothetical-special-situation/>

Chapter 38

Find ways an Integer can be expressed as sum of n-th power of unique natural numbers

Find ways an Integer can be expressed as sum of n-th power of unique natural numbers - GeeksforGeeks

Given two numbers x and n, find number of ways x can be expressed as sum of n-th power of unique natural numbers.

Examples :

Input : x = 10, n = 2

Output : 1

Explanation: $10 = 1^2 + 3^2$,

Hence total 1 possibility

Input : x = 100, n = 2

Output : 3

Explanation: $100 = 10^2$

OR $6^2 + 8^2$

OR $1^2 + 3^2 + 4^2 + 5^2 + 7^2$

Hence total 3 possibilities

The idea is simple. We iterate through all number starting from 1. For every number, we recursively try all greater numbers and if we are able to find sum, we increment result.

C++

```
// C++ program to count number of ways any
// given integer x can be expressed as n-th
```

```
// power of unique natural numbers.
#include<bits/stdc++.h>
using namespace std;

// Function to calculate and return the
// power of any given number
int power(int num, unsigned int n)
{
    if (n == 0)
        return 1;
    else if (n%2 == 0)
        return power(num, n/2)*power(num, n/2);
    else
        return num*power(num, n/2)*power(num, n/2);
}

// Function to check power representations recursively
int checkRecursive(int x, int n, int curr_num = 1,
                  int curr_sum = 0)
{
    // Initialize number of ways to express
    // x as n-th powers of different natural
    // numbers
    int results = 0;

    // Calling power of 'i' raised to 'n'
    int p = power(curr_num, n);
    while (p + curr_sum < x)
    {
        // Recursively check all greater values of i
        results += checkRecursive(x, n, curr_num+1,
                                p+curr_sum);

        curr_num++;
        p = power(curr_num, n);
    }

    // If sum of powers is equal to x
    // then increase the value of result.
    if (p + curr_sum == x)
        results++;

    // Return the final result
    return results;
}

// Driver Code.
int main()
{
```

```
int x = 10, n = 2;
cout << checkRecursive(x, n);
return 0;
}
```

PHP

```
<?php
// PHP program to count
// number of ways any
// given integer x can
// be expressed as n-th
// power of unique
// natural numbers.

// Function to calculate and return
// the power of any given number
function power($num, $n)
{
    if ($n == 0)
        return 1;
    else if ($n % 2 == 0)
        return power($num, (int)($n / 2)) *
            power($num, (int)($n / 2));
    else
        return $num * power($num, (int)($n / 2)) *
            power($num, (int)($n / 2));
}

// Function to check power
// representations recursively
function checkRecursive($x, $n,
                        $curr_num = 1,
                        $curr_sum = 0)
{
    // Initialize number of
    // ways to express
    // x as n-th powers
    // of different natural
    // numbers
    $results = 0;

    // Calling power of 'i'
    // raised to 'n'
    $p = power($curr_num, $n);
    while ($p + $curr_sum < $x)
```

```
{

    // Recursively check all
    // greater values of i
    $results += checkRecursive($x, $n,
                               $curr_num + 1,
                               $p + $curr_sum);

    $curr_num++;
    $p = power($curr_num, $n);
}

// If sum of powers
// is equal to x
// then increase the
// value of result.
if ($p + $curr_sum == $x)
    $results++;

// Return the final result
return $results;
}

// Driver Code.
$x = 10; $n = 2;
echo(checkRecursive($x, $n));

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

Output:

1

Alternate Solution :

Below is an alternate simpler solution provided by **Shivam Kanodia**.

Java

```
// Java program to find number of ways to express a
// number as sum of n-th powers of numbers.
import java.io.*;
import java.util.*;

public class Solution {

    static int res = 0;
```

```
static int checkRecursive(int num, int x, int k, int n)
{
    if (x == 0)
        res++;

    int r = (int)Math.floor(Math.pow(num, 1.0 / n));

    for (int i = k + 1; i <= r; i++) {
        int a = x - (int)Math.pow(i, n);
        if (a >= 0)
            checkRecursive(num,
                x - (int)Math.pow(i, n), i, n);
    }
    return res;
}

// Wrapper over checkRecursive()
static int check(int x, int n)
{
    return checkRecursive(x, x, 0, n);
}

public static void main(String[] args)
{
    System.out.println(check(10, 2));
}
}
```

C#

```
// C# program to find number of
// ways to express a number as sum
// of n-th powers of numbers.
using System;

class Solution {

    static int res = 0;
    static int checkRecursive(int num, int x,
                               int k, int n)
    {
        if (x == 0)
            res++;

        int r = (int)Math.Floor(Math.Pow(num, 1.0 / n));

        for (int i = k + 1; i <= r; i++)
        {
```



```
        int a = x - (int)Math.Pow(i, n);
        if (a >= 0)
            checkRecursive(num, x -
                           (int)Math.Pow(i, n), i, n);
    }
    return res;
}

// Wrapper over checkRecursive()
static int check(int x, int n)
{
    return checkRecursive(x, x, 0, n);
}

// Driver code
public static void Main()
{
    Console.WriteLine(check(10, 2));
}

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

PHP

```
<?php
// PHP program to find number
// of ways to express a number
// as sum of n-th powers of numbers.
$res = 0;

function checkRecursive($num, $x,
                       $k, $n)
{
    global $res;
    if ($x == 0)
        $res++;

    $r = (int)floor(pow($num,
                      1.0 / $n));

    for ($i = $k + 1;
        $i <= $r; $i++)
    {
        $a = $x - (int)pow($i, $n);
        if ($a >= 0)
            checkRecursive($num, $x -
                          (int)pow($i, $n),
```

```
        $i, $n);
    }
    return $res;
}

// Wrapper over
// checkRecursive()
function check($x, $n)
{
    return checkRecursive($x, $x,
                          0, $n);
}

// Driver Code
echo (check(10, 2));

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

Output:

1

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

Source

<https://www.geeksforgeeks.org/find-ways-integer-can-expressed-sum-n-th-power-unique-natural-numbers/>

Chapter 39

First uppercase letter in a string (Iterative and Recursive)

First uppercase letter in a string (Iterative and Recursive) - GeeksforGeeks

Given a string find its first uppercase letter

Examples :

Input : geeksforgeeKs
Output : K

Input : geekS
Output : S

Method 1: linear search

Using linear search, find the first character which is capital

C++

```
// C++ program to find the first
// uppercase letter using linear search
#include <bits/stdc++.h>
using namespace std;

// Function to find string which has
// first character of each word.
char first(string str)
{
    for (int i = 0; i < str.length(); i++)
        if (isupper(str[i]))
```

```
        return str[i];
    return 0;
}

// Driver code
int main()
{
    string str = "geeksforGeeKS";
    char res = first(str);
    if (res == 0)
        cout << "No uppercase letter";
    else
        cout << res << "\n";
    return 0;
}
```

Java

```
// Java program to find the first
// uppercase letter using linear search
import java.io.*;
import java.util.*;

class GFG {

    // Function to find string which has
    // first character of each word.
    static char first(String str)
    {
        for (int i = 0; i < str.length(); i++)
            if (Character.isUpperCase(str.charAt(i)))
                return str.charAt(i);
        return 0;
    }

    // Driver program
    public static void main(String args[])
    {
        String str = "geeksforGeeKS";
        char res = first(str);
        if (res == 0)
            System.out.println("No uppercase letter");
        else
            System.out.println(res);
    }
}
```

```
// This code is contributed
// by Nikita Tiwari.
```

Python3

```
# Python3 program to find the first
# uppercase letter using linear search

# Function to find string which has
# first character of each word.
def first(str) :

    for i in range(0, len(str)) :

        if (str[i].istitle()) :
            return str[i]

    return 0

# Driver code
str = "geeksforGeeKS"
res = first(str)

if (res == 0) :
    print("No uppercase letter")

else :
    print(res)

# This code is contributed by Nikita Tiwari
```

C#

```
// C# program to find the first uppercase
// letter using linear search
using System;

class GFG {

    // Function to find string which has
    // first character of each word.
    static char first(string str)
    {
        for (int i = 0; i < str.Length; i++)
            if (char.IsUpper(str[i]) )
```

```
        return str[i];
    return '0';
}

// Driver function
public static void Main()
{
    string str = "geeksforGeeKS";
    char res = first(str);
    if (res == '0')
        Console.WriteLine("No uppercase"
                           + " letter");
    else
        Console.WriteLine(res);
}
}

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP program to find the first
// uppercase letter using linear search

// Function to find string which has
// first character of each word.
function first($str)
{
    for ($i = 0; $i < strlen($str); $i++)
        if (ctype_upper($str[$i]))
        {
            return $str[$i];
        }
    return 0;
}

// Driver code
$str = "geeksforGeeKS";
$res = first($str);

if (ord($res) == ord(0) )
    echo "No uppercase letter";
else
    echo $res . "\n";

// This code is contributed by Sam007
```

?>

Output:

G

Method 2 (Using recursion)

Recursively traverse the string and if any uppercase is found return that character

C++

```
// C++ program to find the
// first uppercase letter.
#include <bits/stdc++.h>
using namespace std;

// Function to find string which has
// first character of each word.
char first(string str, int i=0)
{
    if (str[i] == '\0')
        return 0;
    if (isupper(str[i]))
        return str[i];
    return first(str, i+1);
}

// Driver code
int main()
{
    string str = "geeksforGeeKS";
    char res = first(str);
    if (res == 0)
        cout << "No uppercase letter";
    else
        cout << res << "\n";
    return 0;
}
```

Java

```
// Java program to find the
// first uppercase letter.
import java.io.*;

class GFG {
```

```
// Function to find string which has
// first character of each word.
static char first(String str, int i)
{
    if (str.charAt(i) == '\0')
        return 0;
    if (Character.isUpperCase(str.charAt(i)))
        return str.charAt(i);
    return first(str, i + 1);
}

// Driver code
public static void main(String args[])
{
    String str = "geeksforGeeKS";
    char res = first(str,0);
    if (res == 0)
        System.out.println("No uppercase letter");
    else
        System.out.println (res );
}

// This code is contributed
// by Nikita Tiwari.
```

Python 3

```
# Python 3 program to find the
# first uppercase letter.
#include <bits/stdc++.h>

# Function to find string which has
# first character of each word.
def first(str, i):

    if (str[i] == '\0'):
        return 0
    if (str[i].isupper()):
        return str[i]
    return first(str, i+1)

# Driver code
str = "geeksforGeeKS"
res = first(str,0)
if (res == 0):
    print("No uppercase letter")
```



```
else:
    print(res)
```

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

C#

```
// C# program to find the
// first uppercase letter.
using System;

class GFG
{
    // Function to find string
    // which has first character
    // of each word.
    static char first(string str, int i)
    {
        if (str[i] == '\0')
            return '0';
        if (char.IsUpper(str[i]))
            return (str[i]);
        return first(str, i + 1);
    }

    // Driver code
    static public void Main ()
    {
        string str = "geeksforGeeKS";
        char res = first(str, 0);
        if (res == 0)
            Console.WriteLine("No uppercase letter");
        else
            Console.WriteLine(res );
    }
}

// This code is contributed by Anuj_67.
```

PHP

```
<?php
//PHP program to find the
// first uppercase letter.
```

```
// Function to find string
// which has first character
// of each word.

function first($str, $i = 0)
{
    if ($str[$i] == '\0')
        return 0;
    if (ctype_upper($str[$i]))
        return $str[$i];
    return first($str, $i+1);
}

// Driver code
$str = "geeksforGeeKS";
$res = first($str);

if (ord($res) == ord(0))
    echo "No uppercase letter";
else
    echo $res , "\n";

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

Output :

G

Improved By : [Sam007](#), [vt_m](#), [jit_t](#), [Smitha Dinesh Semwal](#)

Source

<https://www.geeksforgeeks.org/first-uppercase-letter-in-a-string-iterative-and-recursive/>

Chapter 40

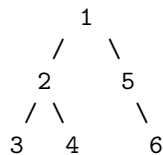
Flatten a binary tree into linked list

Flatten a binary tree into linked list - GeeksforGeeks

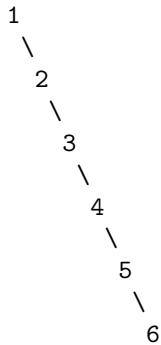
Given a binary tree, flatten it into linked list in-place. Usage of auxiliary data structure is not allowed. After flattening, left of each node should point to NULL and right should contain next node in level order.

Examples:

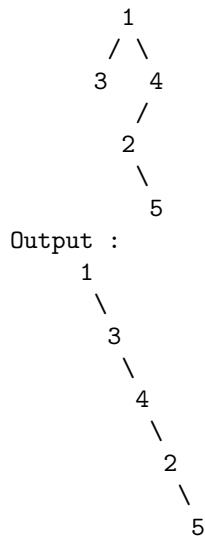
Input :



Output :



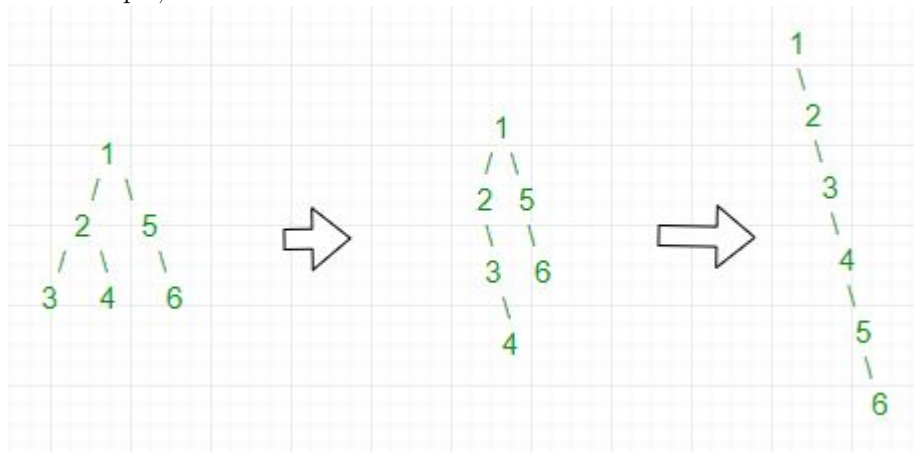
Input :



Simple Approach: A simple solution is to use [Level Order Traversal using Queue](#). In level order traversal, keep track of previous node. Make current node as right child of previous and left of previous node as NULL. This solution requires queue, but question asks to solve without additional data structure.

Efficient Without Additional Data Structure Recursively look for the node with no grandchildren and both left and right child in the left sub-tree. Then store node->right in temp and make node->right=node->left. Insert temp in first node NULL on right of node by node=node->right. Repeat until it is converted to linked list.

For Example,



```

/* Program to flatten a given Binary
Tree into linked list */
#include <iostream>
using namespace std;

```

```
struct Node {
    int key;
    Node *left, *right;
};

/* utility that allocates a new Node
   with the given key */
Node* newNode(int key)
{
    Node* node = new Node;
    node->key = key;
    node->left = node->right = NULL;
    return (node);
}

// Function to convert binary tree into
// linked list by altering the right node
// and making left node point to NULL
void flatten(struct Node* root)
{
    // base condition- return if root is NULL
    // or if it is a leaf node
    if (root == NULL || root->left == NULL &&
        root->right == NULL) {
        return;
    }

    // if root->left exists then we have
    // to make it root->right
    if (root->left != NULL) {

        // move left recursively
        flatten(root->left);

        // store the node root->right
        struct Node* tmpRight = root->right;
        root->right = root->left;
        root->left = NULL;

        // find the position to insert
        // the stored value
        struct Node* t = root->right;
        while (t->right != NULL) {
            t = t->right;
        }

        // insert the stored value
        t->right = tmpRight;
    }
}
```

```
    }

    // now call the same function
    // for root->right
    flatten(root->right);
}

// To find the inorder traversal
void inorder(struct Node* root)
{
    // base condition
    if (root == NULL)
        return;
    inorder(root->left);
    cout << root->key << " ";
    inorder(root->right);
}

/* Driver program to test above functions*/
int main()
{
    /*      1
       /   \
      2     5
     /\    \
    3  4    6 */
    Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(5);
    root->left->left = newNode(3);
    root->left->right = newNode(4);
    root->right->right = newNode(6);

    flatten(root);

    cout << "The Inorder traversal after "
          "flattening binary tree ";
    inorder(root);
    return 0;
}
```

Output:

```
The Inorder traversal after flattening
binary tree 1 2 3 4 5 6
```

Source

<https://www.geeksforgeeks.org/flatten-a-binary-tree-into-linked-list/>

Chapter 41

Flood fill Algorithm – how to implement fill() in paint?

Flood fill Algorithm - how to implement fill() in paint? - GeeksforGeeks

In MS-Paint, when we take the brush to a pixel and click, the color of the region of that pixel is replaced with a new selected color. Following is the problem statement to do this task.

Given a 2D screen, location of a pixel in the screen and a color, replace color of the given pixel and all adjacent same colored pixels with the given color.

Example:

Input:

```
screen[M][N] = {{1, 1, 1, 1, 1, 1, 1, 1},
                 {1, 1, 1, 1, 1, 1, 0, 0},
                 {1, 0, 0, 1, 1, 0, 1, 1},
                 {1, 2, 2, 2, 2, 0, 1, 0},
                 {1, 1, 1, 2, 2, 0, 1, 0},
                 {1, 1, 1, 2, 2, 2, 2, 0},
                 {1, 1, 1, 1, 1, 2, 1, 1},
                 {1, 1, 1, 1, 1, 2, 2, 1},
                 };
x = 4, y = 4, newColor = 3
```

The values in the given 2D screen indicate colors of the pixels.

x and y are coordinates of the brush, newColor is the color that should replace the previous color on screen[x][y] and all surrounding pixels with same color.

Output:

Screen should be changed to following.

```
screen[M][N] = {{1, 1, 1, 1, 1, 1, 1, 1},
```



```
{1, 1, 1, 1, 1, 1, 0, 0},
{1, 0, 0, 1, 1, 0, 1, 1},
{1, 3, 3, 3, 3, 0, 1, 0},
{1, 1, 1, 3, 3, 0, 1, 0},
{1, 1, 1, 3, 3, 3, 3, 0},
{1, 1, 1, 1, 1, 3, 1, 1},
{1, 1, 1, 1, 1, 3, 3, 1},
};
```

Flood Fill Algorithm:

The idea is simple, we first replace the color of current pixel, then recur for 4 surrounding points. The following is detailed algorithm.

```
// A recursive function to replace previous color 'prevC' at '(x, y)'
// and all surrounding pixels of (x, y) with new color 'newC' and
floodFill(screen[M][N], x, y, prevC, newC)
1) If x or y is outside the screen, then return.
2) If color of screen[x][y] is not same as prevC, then return
3) Recur for north, south, east and west.
    floodFillUtil(screen, x+1, y, prevC, newC);
    floodFillUtil(screen, x-1, y, prevC, newC);
    floodFillUtil(screen, x, y+1, prevC, newC);
    floodFillUtil(screen, x, y-1, prevC, newC);
```

The following is C++ implementation of above algorithm.

```
// A C++ program to implement flood fill algorithm
#include<iostream>
using namespace std;

// Dimensions of paint screen
#define M 8
#define N 8

// A recursive function to replace previous color 'prevC' at '(x, y)'
// and all surrounding pixels of (x, y) with new color 'newC' and
void floodFillUtil(int screen[][N], int x, int y, int prevC, int newC)
{
    // Base cases
    if (x < 0 || x >= M || y < 0 || y >= N)
        return;
    if (screen[x][y] != prevC)
        return;

    // Replace the color at (x, y)
    screen[x][y] = newC;
```

```

    // Recur for north, east, south and west
    floodFillUtil(screen, x+1, y, prevC, newC);
    floodFillUtil(screen, x-1, y, prevC, newC);
    floodFillUtil(screen, x, y+1, prevC, newC);
    floodFillUtil(screen, x, y-1, prevC, newC);
}

// It mainly finds the previous color on (x, y) and
// calls floodFillUtil()
void floodFill(int screen[][N], int x, int y, int newC)
{
    int prevC = screen[x][y];
    floodFillUtil(screen, x, y, prevC, newC);
}

// Driver program to test above function
int main()
{
    int screen[M][N] = {{1, 1, 1, 1, 1, 1, 1, 1},
                        {1, 1, 1, 1, 1, 1, 0, 0},
                        {1, 0, 0, 1, 1, 0, 1, 1},
                        {1, 2, 2, 2, 2, 0, 1, 0},
                        {1, 1, 1, 2, 2, 0, 1, 0},
                        {1, 1, 1, 2, 2, 2, 2, 0},
                        {1, 1, 1, 1, 1, 2, 1, 1},
                        {1, 1, 1, 1, 1, 2, 2, 1},
                        };

    int x = 4, y = 4, newC = 3;
    floodFill(screen, x, y, newC);

    cout << "Updated screen after call to floodFill: n";
    for (int i=0; i<M; i++)
    {
        for (int j=0; j<N; j++)
            cout << screen[i][j] << " ";
        cout << endl;
    }
}

```

Output:

```

Updated screen after call to floodFill:
1 1 1 1 1 1 1 1
1 1 1 1 1 1 0 0
1 0 0 1 1 0 1 1
1 3 3 3 3 0 1 0
1 1 1 3 3 0 1 0

```

```
1 1 1 3 3 3 3 0
1 1 1 1 1 3 1 1
1 1 1 1 1 3 3 1
```

References:

http://en.wikipedia.org/wiki/Flood_fill

This article is contributed by **Anmol**. 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/flood-fill-algorithm-implement-fill-paint/>

Chapter 42

Function to copy string (Iterative and Recursive)

Function to copy string (Iterative and Recursive) - GeeksforGeeks

Given two strings, copy one string to other using recursion. We basically need to write our own recursive version of [strcpy in C/C++](#)

Examples:

```
Input : s1 = "hello"
        s2 = "geeksforgeeks"
Output : s2 = "hello"
```

```
Input : s1 = "geeksforgeeks"
        s2 = ""
Output : s2 = "geeksforgeeks"
```

Iterative :

Copy every character from s1 to s2 starting from index = 0 and in each call increase the index by 1 until s1 doesn't reach to end;

```
// Iterative CPP Program to copy one String
// to another.
#include <bits/stdc++.h>
using namespace std;

// Function to copy one string to other
// assuming that other string has enough
// space.
void myCopy(char s1[], char s2[])
```

```
{
    int i = 0;
    for (i=0; s1[i] != '\0'; i++)
        s2[i] = s1[i];
    s2[i] = '\0';
}

// Driver function
int main()
{
    char s1[100] = "GEEKSFORGEEKS";
    char s2[100] = "";
    myCopy(s1, s2);
    cout << s2;
    return 0;
}
```

Output:

GEEKSFORGEEKS

Recursive :

Copy every character from s1 to s2 starting from index = 0 and in each call increase the index by 1 until s1 doesn't reach to end;

```
// CPP Program to copy one String to
// another using Recursion
#include <bits/stdc++.h>
using namespace std;

// Function to copy one string in to other
// using recursion
void myCopy(char s1[], char s2[], int index = 0)
{
    // copying each character from s1 to s2
    s2[index] = s1[index];

    // if string reach to end then stop
    if (s1[index] == '\0')
        return;

    // increase character index by one
    myCopy(s1, s2, index + 1);
}

// Driver function
```

```
int main()
{
    char s1[100] = "GEEKSFORGEEKS";
    char s2[100] = "";
    myCopy(s1, s2);
    cout << s2;
    return 0;
}
```

Output:

GEEKSFORGEEKS

Source

<https://www.geeksforgeeks.org/function-copy-string-iterative-recursive/>

Chapter 43

Generate all binary strings without consecutive 1's

Generate all binary strings without consecutive 1's - GeeksforGeeks

Given a integer K. Task is Print All binary string of size K (Given number).

Examples:

Input : K = 3

Output : 000 , 001 , 010 , 100 , 101

Input : K = 4

Output : 0000 0001 0010 0100 0101 1000 1001 1010

Idea behind that is IF string ends with '1' then we put only '0' at the end. IF string ends with '0' then we put both '0' and '1' at the end of string for generating new string.

Below is algorithm

K : size of string

First We Generate All string starts with '0'

initialize n = 1 .

GenerateAllString (K , Str , n)

- a. IF n == K
 PRINT str.
- b. IF previous character is '1' :: str[n-1] == '1'
 put str[n] = '1'
 GenerateAllString (K , str , n+1)
- c. IF previous character is '0' :: str[n-1] == '0'
 First We Put zero at end and call function

```
PUT  str[n] = '0'
      GenerateAllString ( K , str , n+1 )
PUT  str[n] = '1'
      GenerateAllString ( K , str , n+1 )
```

Second Generate all binary string starts with '1'
DO THE SAME PROCESS

Below is recursive C++ implementation.

```
// C++ program to Generate all binary string without
// consecutive 1's of size K
#include<bits/stdc++.h>
using namespace std ;

// A utility function generate all string without
// consecutive 1's of size K
void generateAllStringsUtil(int K, char str[], int n)
{
    // print binary string without consecutive 1's
    if (n == K)
    {
        // terminate binary string
        str[n] = '\0' ;
        cout << str << " ";
        return ;
    }

    // if previous character is '1' then we put
    // only 0 at end of string
    //example str = "01" then new string be "000"
    if (str[n-1] == '1')
    {
        str[n] = '0';
        generateAllStringsUtil (K , str , n+1);
    }

    // if previous character is '0' than we put
    // both '1' and '0' at end of string
    // example str = "00" then new string "001" and "000"
    if (str[n-1] == '0')
    {
        str[n] = '0';
        generateAllStringsUtil(K, str, n+1);
        str[n] = '1';
        generateAllStringsUtil(K, str, n+1) ;
    }
}
```



```
// function generate all binary string without
// consecutive 1's
void generateAllStrings(int K )
{
    // Base case
    if (K <= 0)
        return ;

    // One by one stores every binary string of length K
    char str[K];

    // Generate all Binary string starts with '0'
    str[0] = '0' ;
    generateAllStringsUtil ( K , str , 1 ) ;

    // Generate all Binary string starts with '1'
    str[0] = '1' ;
    generateAllStringsUtil ( K , str , 1 );
}

// Driver program to test above function
int main()
{
    int K = 3;
    generateAllStrings (K) ;
    return 0;
}
```

Output:

000 001 010 100 101

Source

<https://www.geeksforgeeks.org/generate-binary-strings-without-consecutive-1s/>

Chapter 44

Generate all passwords from given character set

Generate all passwords from given character set - GeeksforGeeks

Given a set of characters generate all possible passwords from them. This means we should generate all possible permutations of words using the given characters, with repetitions and also upto a given length.

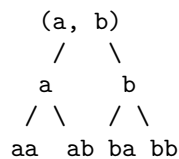
Examples:

Input : arr[] = {a, b},
 len = 2.

Output :
a b aa ab ba bb

The solution is to use recursion on the given character array. The idea is to pass all possible lengths and an empty string initially to a helper function. In the helper function we keep appending all the characters one by one to the current string and recur to fill the remaining string till the desired length is reached.

It can be better visualized using the below recursion tree:



Following is the implementation of the above method.

C++

```
// C++ program to generate all passwords for given characters
#include <bits/stdc++.h>
using namespace std;

// int cnt;

// Recursive helper function, adds/removes characters
// until len is reached
void generate(char* arr, int i, string s, int len)
{
    // base case
    if (i == 0) // when len has been reached
    {
        // print it out
        cout << s << "\n";
        // cnt++;
        return;
    }

    // iterate through the array
    for (int j = 0; j < len; j++) {

        // Create new string with next character
        // Call generate again until string has
        // reached its len
        string appended = s + arr[j];
        generate(arr, i - 1, appended, len);
    }

    return;
}

// function to generate all possible passwords
void crack(char* arr, int len)
{
    // call for all required lengths
    for (int i = 1; i <= len; i++) {
        generate(arr, i, "", len);
    }
}

// driver function
int main()
{
    char arr[] = { 'a', 'b', 'c' };
}
```

```
int len = sizeof(arr) / sizeof(arr[0]);
crack(arr, len);

//cout << cnt << endl;
return 0;
}
// This code is contributed by Satish Srinivas.
```

Python 3

```
# Python3 program to
# generate all passwords
# for given characters

# Recursive helper function,
# adds/removes characters
# until len is reached
def generate(arr, i, s, len):

    # base case
    if (i == 0): # when len has
                  # been reached

        # print it out
        print(s)
        return

    # iterate through the array
    for j in range(0, len):

        # Create new string with
        # next character Call
        # generate again until
        # string has reached its len
        appended = s + arr[j]
        generate(arr, i - 1, appended, len)

    return

# function to generate
# all possible passwords
def crack(arr, len):

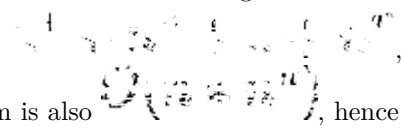
    # call for all required lengths
    for i in range(1, len + 1):
        generate(arr, i, "", len)

# Driver Code
```

```
arr = ['a', 'b', 'c' ]  
len = len(arr)  
crack(arr, len)  
  
# This code is contributed by Smita.
```

Output:

```
a  
b  
c  
aa  
ab  
ac  
ba  
bb  
bc  
ca  
cb  
cc  
aaa  
aab  
aac  
aba  
abb  
abc  
aca  
acb  
acc  
baa  
bab  
bac  
bba  
bbb  
bbc  
bca  
bcb  
bcc  
caa  
cab  
cac  
cba  
cbb  
cbc  
cca  
ccb  
ccc
```

If we want to see the count of the words, we can uncomment the lines having cnt variable in the code. We can observe that it comes out to be , where $n = \text{len}$. Thus the time complexity of the program is also exponential. We can also check for a particular password while generating and break the loop and return if it is found. We can also include other symbols to be generated and if needed remove duplicates by preprocessing the input using a HashTable.

Improved By : [Smitha Dinesh Semwal](#)

Source

<https://www.geeksforgeeks.org/generate-passwords-given-character-set/>

Chapter 45

Generate all possible sorted arrays from alternate elements of two given sorted arrays

Generate all possible sorted arrays from alternate elements of two given sorted arrays - GeeksforGeeks

Given two sorted arrays A and B, generate all possible arrays such that first element is taken from A then from B then from A and so on in increasing order till the arrays exhausted. The generated arrays should end with an element from B.

For Example

A = {10, 15, 25}

B = {1, 5, 20, 30}

The resulting arrays are:

```
10 20
10 20 25 30
10 30
15 20
15 20 25 30
15 30
25 30
```

We strongly recommend you to minimize your browser and try this yourself first.

The idea is to use recursion. In the recursive function, a flag is passed to indicate whether current element in output should be taken from 'A' or 'B'. Below is C++ implementation.

C

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

void printArr(int arr[], int n);

/* Function to generates and prints all sorted arrays from alternate elements of
'A[i..m-1]' and 'B[j..n-1]'
If 'flag' is true, then current element is to be included from A otherwise
from B.
'len' is the index in output array C[]. We print output array each time
before including a character from A only if length of output array is
greater than 0. We try than all possible combinations */
void generateUtil(int A[], int B[], int C[], int i, int j, int m, int n,
                  int len, bool flag)
{
    if (flag) // Include valid element from A
    {
        // Print output if there is at least one 'B' in output array 'C'
        if (len)
            printArr(C, len+1);

        // Recur for all elements of A after current index
        for (int k = i; k < m; k++)
        {
            if (!len)
            {
                /* this block works for the very first call to include
                the first element in the output array */
                C[len] = A[k];

                // don't increment len as B is included yet
                generateUtil(A, B, C, k+1, j, m, n, len, !flag);
            }
            else /* include valid element from A and recur */
            {
                if (A[k] > C[len])
                {
                    C[len+1] = A[k];
                    generateUtil(A, B, C, k+1, j, m, n, len+1, !flag);
                }
            }
        }
    }
    else /* Include valid element from B and recur */
    {
        for (int l = j; l < n; l++)
        {
            if (B[l] > C[len])
```



```
        {
            C[len+1] = B[l];
            generateUtil(A, B, C, i, l+1, m, n, len+1, !flag);
        }
    }
}
```

```
/* Wrapper function */
void generate(int A[], int B[], int m, int n)
{
    int C[m+n];    /* output array */
    generateUtil(A, B, C, 0, 0, m, n, 0, true);
}
```

```
// A utility function to print an array
void printArr(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}
```

```
// Driver program
int main()
{
    int A[] = {10, 15, 25};
    int B[] = {5, 20, 30};
    int n = sizeof(A)/sizeof(A[0]);
    int m = sizeof(B)/sizeof(B[0]);
    generate(A, B, n, m);
    return 0;
}
```

Java

```
class GenerateArrays {

    /* Function to generates and prints all sorted arrays from alternate
       elements of 'A[i..m-1]' and 'B[j..n-1]'
       If 'flag' is true, then current element is to be included from A
       otherwise from B.
       'len' is the index in output array C[]. We print output array
       each time before including a character from A only if length of
       output array is greater than 0. We try than all possible
       combinations */
    void generateUtil(int A[], int B[], int C[], int i, int j, int m, int n,
        int len, boolean flag)
```

```
{
    if (flag) // Include valid element from A
    {
        // Print output if there is at least one 'B' in output array 'C'
        if (len != 0)
            printArr(C, len + 1);

        // Recur for all elements of A after current index
        for (int k = i; k < m; k++)
        {
            if (len == 0)
            {
                /* this block works for the very first call to include
                the first element in the output array */
                C[len] = A[k];

                // don't increment len as B is included yet
                generateUtil(A, B, C, k + 1, j, m, n, len, !flag);
            }

            /* include valid element from A and recur */
            else if (A[k] > C[len])
            {
                C[len + 1] = A[k];
                generateUtil(A, B, C, k + 1, j, m, n, len + 1, !flag);
            }
        }
    }

    /* Include valid element from B and recur */
    else
    {
        for (int l = j; l < n; l++)
        {
            if (B[l] > C[len])
            {
                C[len + 1] = B[l];
                generateUtil(A, B, C, i, l + 1, m, n, len + 1, !flag);
            }
        }
    }
}

/* Wrapper function */
void generate(int A[], int B[], int m, int n)
{
    int C[] = new int[m + n];
```

```
        /* output array */
        generateUtil(A, B, C, 0, 0, m, n, 0, true);
    }

    // A utility function to print an array
    void printArr(int arr[], int n)
    {
        for (int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        System.out.println("");
    }

    public static void main(String[] args)
    {
        GenerateArrays generate = new GenerateArrays();
        int A[] = {10, 15, 25};
        int B[] = {5, 20, 30};
        int n = A.length;
        int m = B.length;
        generate.generate(A, B, n, m);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Python3

```
# A utility function to print an array
def printArr(arr,n):

    for i in range(n):
        print(arr[i] , " ",end="")
    print()

''' Function to generates and prints all
sorted arrays from alternate elements of
'A[i..m-1]' and 'B[j..n-1]'
If 'flag' is true, then current element
is to be included from A otherwise
from B.
'len' is the index in output array C[].
We print output array each time
before including a character from A
only if length of output array is
greater than 0. We try than all possible combinations '''
def generateUtil(A,B,C,i,j,m,n,len,flag):

    if (flag): # Include valid element from A
```

```
# Print output if there is at
# least one 'B' in output array 'C'
if (len):
    printArr(C, len+1)

# Recur for all elements of
# A after current index
for k in range(i,m):

    if ( not len):

        ''' this block works for the
            very first call to include
            the first element in the output array '''
        C[len] = A[k]

        # don't increment len
        # as B is included yet
        generateUtil(A, B, C, k+1, j, m, n, len, not flag)

    else:

        # include valid element from A and recur
        if (A[k] > C[len]):

            C[len+1] = A[k]
            generateUtil(A, B, C, k+1, j, m, n, len+1, not flag)

else:

    # Include valid element from B and recur
    for l in range(j,n):

        if (B[l] > C[len]):

            C[len+1] = B[l]
            generateUtil(A, B, C, i, l+1, m, n, len+1, not flag)

# Wrapper function
def generate(A,B,m,n):

    C=[]    #output array
    for i in range(m+n+1):
        C.append(0)
    generateUtil(A, B, C, 0, 0, m, n, 0, True)
```

```
# Driver program

A = [10, 15, 25]
B = [5, 20, 30]
n = len(A)
m = len(B)

generate(A, B, n, m)

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

Output:

```
10 20
10 20 25 30
10 30
15 20
15 20 25 30
15 30
25 30
```

This article is contributed by [Gaurav Ahirwar](#). 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/generate-all-possible-sorted-arrays-from-alternate-elements-of-two-given-arrays/>

Chapter 46

Generating all possible Subsequences using Recursion

Generating all possible Subsequences using Recursion - GeeksforGeeks

Given an array. The task is to generate and print all of the possible subsequences of the given array using recursion.

Examples:

Input : [1, 2, 3]

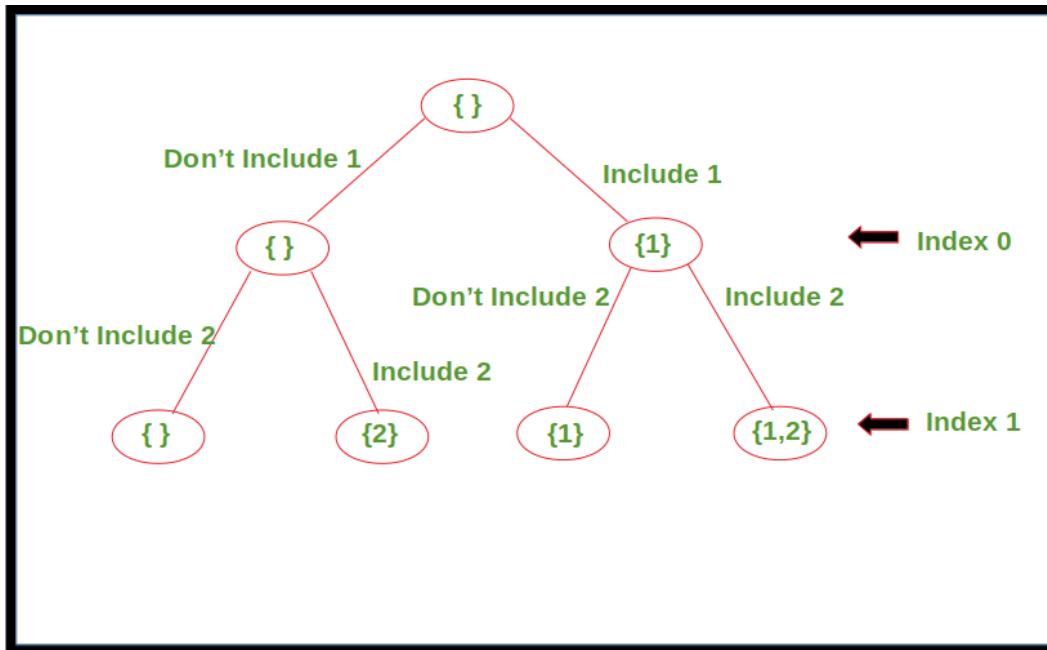
Output : [3], [2], [2, 3], [1], [1, 3], [1, 2], [1, 2, 3]

Input : [1, 2]

Output : [2], [1], [1, 2]

Approach: For every element in the array, there are two choices, either to include it in the subsequence or not include it. Apply this for every element in the array starting from index 0 until we reach the last index. Print the subsequence once the last index is reached.

Below diagram shows the recursion tree for array, `arr[] = {1, 2}`.



Below is the implementation of the above approach.

```

# Python3 code to print all possible
# subsequences for given array using
# recursion

# Recursive function to print all
# possible subsequences for given array
def printSubsequences(arr, index, subarr):

    # Print the subsequence when reach
    # the leaf of recursion tree
    if index == len(arr):

        # Condition to avoid printing
        # empty subsequence
        if len(subarr) != 0:
            print(subarr)

    else:
        # Subsequence without including
        # the element at current index
        printSubsequences(arr, index + 1, subarr)

        # Subsequence including the element
        # at current index
        printSubsequences(arr, index + 1, \

```

```
        subarr+[arr[index]])

    return

arr = [1, 2, 3]

printSubsequences(arr, 0, [])
```

Output:

```
[3]
[2]
[2, 3]
[1]
[1, 3]
[1, 2]
[1, 2, 3]
```

Time Complexity: $O(2^N)$

Source

<https://www.geeksforgeeks.org/generating-all-possible-subsequences-using-recursion/>

Chapter 47

Generating subarrays using recursion

Generating subarrays using recursion - GeeksforGeeks

Given an array, generate all the possible subarrays of the given array using recursion.

Examples:

Input : [1, 2, 3]

Output : [1], [1, 2], [2], [1, 2, 3], [2, 3], [3]

Input : [1, 2]

Output : [1], [1, 2], [2]

We have discussed [iterative program to generate all subarrays](#). In this post, recursive is discussed.

Approach: We use two pointers **start** and **end** to maintain the starting and ending point of the array and follow the steps given below:

- Stop if we have reached the end of the array
- Increment the **end** index if **start** has become greater than **end**
- Print the subarray from index **start** to **end** and increment the starting index

Below is the implementation of the above approach.

```
# Python3 code to print all possible subarrays
# for given array using recursion

# Recursive function to print all possible subarrays
```


Chapter 48

Given a matrix of 'O' and 'X', replace 'O' with 'X' if surrounded by 'X'

Given a matrix of 'O' and 'X', replace 'O' with 'X' if surrounded by 'X' - GeeksforGeeks

Given a matrix where every element is either 'O' or 'X', replace 'O' with 'X' if surrounded by 'X'. A 'O' (or a set of 'O') is considered to be surrounded by 'X' if there are 'X' at locations just below, just above, just left and just right of it.

Examples:

```
Input: mat[M][N] = {{'X', 'O', 'X', 'X', 'X', 'X'},
                    {'X', 'O', 'X', 'X', 'O', 'X'},
                    {'X', 'X', 'X', 'O', 'O', 'X'},
                    {'O', 'X', 'X', 'X', 'X', 'X'},
                    {'X', 'X', 'X', 'O', 'X', 'O'},
                    {'O', 'O', 'X', 'O', 'O', 'O'},
                    };
```

```
Output: mat[M][N] = {{'X', 'O', 'X', 'X', 'X', 'X'},
                    {'X', 'O', 'X', 'X', 'X', 'X'},
                    {'X', 'X', 'X', 'X', 'X', 'X'},
                    {'O', 'X', 'X', 'X', 'X', 'X'},
                    {'X', 'X', 'X', 'O', 'X', 'O'},
                    {'O', 'O', 'X', 'O', 'O', 'O'},
                    };
```

```
Input: mat[M][N] = {{'X', 'X', 'X', 'X'}
                    {'X', 'O', 'X', 'X'}
                    {'X', 'O', 'O', 'X'}}
```

```
{'X', 'O', 'X', 'X'}
{'X', 'X', 'O', 'O'}
};
```

```
Input: mat[M][N] = {{'X', 'X', 'X', 'X'}
{'X', 'X', 'X', 'X'}
{'X', 'X', 'X', 'X'}
{'X', 'X', 'X', 'X'}
{'X', 'X', 'O', 'O'}
};
```

This is mainly an application of [Flood-Fill algorithm](#). The main difference here is that a 'O' is not replaced by 'X' if it lies in region that ends on a boundary. Following are simple steps to do this special flood fill.

- 1) Traverse the given matrix and replace all 'O' with a special character '-'.
- 2) Traverse four edges of given matrix and call `floodFill('-', 'O')` for every '-' on edges. The remaining '-' are the characters that indicate 'O's (in the original matrix) to be replaced by 'X'.
- 3) Traverse the matrix and replace all '-'s with 'X's.

Let us see steps of above algorithm with an example. Let following be the input matrix.

```
mat[M][N] = {{'X', 'O', 'X', 'X', 'X', 'X'},
{'X', 'O', 'X', 'X', 'O', 'X'},
{'X', 'X', 'X', 'O', 'O', 'X'},
{'O', 'X', 'X', 'X', 'X', 'X'},
{'X', 'X', 'X', 'O', 'X', 'O'},
{'O', 'O', 'X', 'O', 'O', 'O'},
};
```

Step 1: Replace all 'O' with '-'.

```
mat[M][N] = {{'X', '-', 'X', 'X', 'X', 'X'},
{'X', '-', 'X', 'X', '-', 'X'},
{'X', 'X', 'X', '-', '-', 'X'},
{'-', 'X', 'X', 'X', 'X', 'X'},
{'X', 'X', 'X', '-', 'X', '-'},
{'-', '-', 'X', '-', '-', '-'},
};
```

Step 2: Call `floodFill('-', 'O')` for all edge elements with value equals to '-'

```
mat[M][N] = {{'X', 'O', 'X', 'X', 'X', 'X'},
              {'X', 'O', 'X', 'X', '-', 'X'},
              {'X', 'X', 'X', '-', '-', 'X'},
              {'O', 'X', 'X', 'X', 'X', 'X'},
              {'X', 'X', 'X', 'O', 'X', 'O'},
              {'O', 'O', 'X', 'O', 'O', 'O'}
            };
```

Step 3: Replace all '-' with 'X'.

```
mat[M][N] = {{'X', 'O', 'X', 'X', 'X', 'X'},
              {'X', 'O', 'X', 'X', 'X', 'X'},
              {'X', 'X', 'X', 'X', 'X', 'X'},
              {'O', 'X', 'X', 'X', 'X', 'X'},
              {'X', 'X', 'X', 'O', 'X', 'O'},
              {'O', 'O', 'X', 'O', 'O', 'O'}
            };
```

The following is implementation of above algorithm.

C++

```
// A C++ program to replace all 'O's with 'X's if surrounded by 'X'
#include<iostream>
using namespace std;

// Size of given matrix is M X N
#define M 6
#define N 6

// A recursive function to replace previous value 'prevV' at '(x, y)'
// and all surrounding values of (x, y) with new value 'newV'.
void floodFillUtil(char mat[][N], int x, int y, char prevV, char newV)
{
    // Base cases
    if (x < 0 || x >= M || y < 0 || y >= N)
        return;
    if (mat[x][y] != prevV)
        return;

    // Replace the color at (x, y)
    mat[x][y] = newV;

    // Recur for north, east, south and west
    floodFillUtil(mat, x+1, y, prevV, newV);
```

```

        floodFillUtil(mat, x-1, y, prevV, newV);
        floodFillUtil(mat, x, y+1, prevV, newV);
        floodFillUtil(mat, x, y-1, prevV, newV);
    }

    // Returns size of maximum size subsquare matrix
    // surrounded by 'X'
    int replaceSurrounded(char mat[][N])
    {
        // Step 1: Replace all 'O' with '-'
        for (int i=0; i<M; i++)
            for (int j=0; j<N; j++)
                if (mat[i][j] == 'O')
                    mat[i][j] = '-';

        // Call floodFill for all '-' lying on edges
        for (int i=0; i<M; i++) // Left side
            if (mat[i][0] == '-')
                floodFillUtil(mat, i, 0, '-', 'O');
        for (int i=0; i<M; i++) // Right side
            if (mat[i][N-1] == '-')
                floodFillUtil(mat, i, N-1, '-', 'O');
        for (int i=0; i<N; i++) // Top side
            if (mat[0][i] == '-')
                floodFillUtil(mat, 0, i, '-', 'O');
        for (int i=0; i<N; i++) // Bottom side
            if (mat[M-1][i] == '-')
                floodFillUtil(mat, M-1, i, '-', 'O');

        // Step 3: Replace all '-' with 'X'
        for (int i=0; i<M; i++)
            for (int j=0; j<N; j++)
                if (mat[i][j] == '-')
                    mat[i][j] = 'X';
    }

    // Driver program to test above function
    int main()
    {
        char mat[][N] = {{'X', 'O', 'X', 'O', 'X', 'X'},
                        {'X', 'O', 'X', 'X', 'O', 'X'},
                        {'X', 'X', 'X', 'O', 'X', 'X'},
                        {'O', 'X', 'X', 'X', 'X', 'X'},
                        {'X', 'X', 'X', 'O', 'X', 'O'},
                        {'O', 'O', 'X', 'O', 'O', 'O'}},
            };
        replaceSurrounded(mat);
    }

```

```
for (int i=0; i<M; i++)
{
    for (int j=0; j<N; j++)
        cout << mat[i][j] << " ";
    cout << endl;
}
return 0;
}
```

Java

```
// A Java program to replace
// all 'O's with 'X's if
// surrounded by 'X'
import java.io.*;

class GFG
{
    static int M = 6;
    static int N = 6;
    static void floodFillUtil(char mat[][], int x,
                               int y, char prevV,
                               char newV)
    {
        // Base cases
        if (x < 0 || x >= M ||
            y < 0 || y >= N)
            return;

        if (mat[x][y] != prevV)
            return;

        // Replace the color at (x, y)
        mat[x][y] = newV;

        // Recur for north,
        // east, south and west
        floodFillUtil(mat, x + 1, y,
                      prevV, newV);
        floodFillUtil(mat, x - 1, y,
                      prevV, newV);
        floodFillUtil(mat, x, y + 1,
                      prevV, newV);
        floodFillUtil(mat, x, y - 1,
                      prevV, newV);
    }
}
```

```
// Returns size of maximum
// size subsquare matrix
// surrounded by 'X'
static void replaceSurrounded(char mat[][])
{

    // Step 1: Replace
    // all 'O' with '-'
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            if (mat[i][j] == 'O')
                mat[i][j] = '-';

    // Call floodFill for
    // all '-' lying on edges
    for (int i = 0; i < M; i++) // Left side
        if (mat[i][0] == '-')
            floodFillUtil(mat, i, 0,
                          '-', 'O');
    for (int i = 0; i < M; i++) // Right side
        if (mat[i][N - 1] == '-')
            floodFillUtil(mat, i, N - 1,
                          '-', 'O');
    for (int i = 0; i < N; i++) // Top side
        if (mat[0][i] == '-')
            floodFillUtil(mat, 0, i,
                          '-', 'O');
    for (int i = 0; i < N; i++) // Bottom side
        if (mat[M - 1][i] == '-')
            floodFillUtil(mat, M - 1,
                          i, '-', 'O');

    // Step 3: Replace
    // all '-' with 'X'
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            if (mat[i][j] == '-')
                mat[i][j] = 'X';
}

// Driver Code
public static void main (String[] args)
{
    char[][] mat = {{'X', 'O', 'X',
                     'O', 'X', 'X'},
                    {'X', 'O', 'X',
                     'X', 'O', 'X'}},
```



```
        {'X', 'X', 'X',
         'O', 'X', 'X'},
        {'O', 'X', 'X',
         'X', 'X', 'X'},
        {'X', 'X', 'X',
         'O', 'X', 'O'},
        {'O', 'O', 'X',
         'O', 'O', 'O'}};

replaceSurrounded(mat);

for (int i = 0; i < M; i++)
{
    for (int j = 0; j < N; j++)
        System.out.print(mat[i][j] + " ");

    System.out.println("");
}
}
```

// This code is contributed
// by shiv_bhakt

C#

```
// A C# program to replace
// all 'O's with 'X's if
// surrounded by 'X'
using System;

class GFG
{
    static int M = 6;
    static int N = 6;
    static void floodFillUtil(char [,]mat, int x,
                              int y, char prevV,
                              char newV)
    {
        // Base cases
        if (x < 0 || x >= M ||
            y < 0 || y >= N)
            return;

        if (mat[x, y] != prevV)
            return;

        // Replace the color at (x, y)
```

```
    mat[x, y] = newV;

    // Recur for north,
    // east, south and west
    floodFillUtil(mat, x + 1, y,
                  prevV, newV);
    floodFillUtil(mat, x - 1, y,
                  prevV, newV);
    floodFillUtil(mat, x, y + 1,
                  prevV, newV);
    floodFillUtil(mat, x, y - 1,
                  prevV, newV);
}

// Returns size of maximum
// size subsquare matrix
// surrounded by 'X'
static void replaceSurrounded(char [,]mat)
{

    // Step 1: Replace
    // all 'O' with '-'
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++)
            if (mat[i, j] == 'O')
                mat[i, j] = '-';

    // Call floodFill for
    // all '-' lying on edges
    for (int i = 0; i < M; i++) // Left side
        if (mat[i, 0] == '-')
            floodFillUtil(mat, i, 0,
                          '-', 'O');
    for (int i = 0; i < M; i++) // Right side
        if (mat[i, N - 1] == '-')
            floodFillUtil(mat, i, N - 1,
                          '-', 'O');
    for (int i = 0; i < N; i++) // Top side
        if (mat[0, i] == '-')
            floodFillUtil(mat, 0, i,
                          '-', 'O');
    for (int i = 0; i < N; i++) // Bottom side
        if (mat[M - 1, i] == '-')
            floodFillUtil(mat, M - 1,
                          i, '-', 'O');

    // Step 3: Replace
    // all '-' with 'X'
```

```
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++)
        if (mat[i, j] == '-')
            mat[i, j] = 'X';
}

// Driver Code
public static void Main ()
{
    char [,]mat = new char[,],
        {{'X', 'O', 'X',
          'O', 'X', 'X'},
         {'X', 'O', 'X',
          'X', 'O', 'X'},
         {'X', 'X', 'X',
          'O', 'X', 'X'},
         {'O', 'X', 'X',
          'X', 'X', 'X'},
         {'X', 'X', 'X',
          'O', 'X', 'O'},
         {'O', 'O', 'X',
          'O', 'O', 'O'}};

    replaceSurrounded(mat);

    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
            Console.Write(mat[i, j] + " ");

        Console.WriteLine("");
    }
}

// This code is contributed
// by shiv_bhakt
```

PHP

```
<?php
// A PHP program to replace all
// 'O's with 'X's if surrounded by 'X'

// Size of given
// matrix is M X N
$M = 6;
$N = 6;
```

```
// A recursive function to replace
// previous value 'prevV' at '(x, y)'
// and all surrounding values of
// (x, y) with new value 'newV'.
function floodFillUtil(&$mat, $x, $y,
                      $prevV, $newV)
{
    // Base cases
    if ($x < 0 || $x >= $GLOBALS['M'] ||
        $y < 0 || $y >= $GLOBALS['N'])
        return;
    if ($mat[$x][$y] != $prevV)
        return;

    // Replace the color at (x, y)
    $mat[$x][$y] = $newV;

    // Recur for north,
    // east, south and west
    floodFillUtil($mat, $x + 1, $y, $prevV, $newV);
    floodFillUtil($mat, $x - 1, $y, $prevV, $newV);
    floodFillUtil($mat, $x, $y + 1, $prevV, $newV);
    floodFillUtil($mat, $x, $y - 1, $prevV, $newV);
}

// Returns size of maximum
// size subsquare matrix
// surrounded by 'X'
function replaceSurrounded(&$mat)
{
    // Step 1: Replace all 'O' with '-'
    for ($i = 0; $i < $GLOBALS['M']; $i++)
        for ($j = 0; $j < $GLOBALS['N']; $j++)
            if ($mat[$i][$j] == 'O')
                $mat[$i][$j] = '-';

    // Call floodFill for all
    // '-' lying on edges
    for ($i = 0;
        $i < $GLOBALS['M']; $i++) // Left side
        if ($mat[$i][0] == '-')
            floodFillUtil($mat, $i, 0, '-', 'O');

    for ($i = 0; $i < $GLOBALS['M']; $i++) // Right side
        if ($mat[$i][$GLOBALS['N'] - 1] == '-')
            floodFillUtil($mat, $i, $GLOBALS['N'] - 1, '-', 'O');
```

```
floodFillUtil($mat, $i,
              $GLOBALS['N'] - 1, '-', 'O');

for ($i = 0; $i < $GLOBALS['N']; $i++) // Top side
    if ($mat[0][$i] == '-')
        floodFillUtil($mat, 0, $i, '-', 'O');

for ($i = 0; $i < $GLOBALS['N']; $i++) // Bottom side
    if ($mat[$GLOBALS['M'] - 1][$i] == '-')
        floodFillUtil($mat, $GLOBALS['M'] - 1,
                      $i, '-', 'O');

// Step 3: Replace all '-' with 'X'
for ($i = 0; $i < $GLOBALS['M']; $i++)
    for ($j = 0; $j < $GLOBALS['N']; $j++)
        if ($mat[$i][$j] == '-')
            $mat[$i][$j] = 'X';
}

// Driver Code
$mat = array(array('X', 'O', 'X', 'O', 'X', 'X'),
             array('X', 'O', 'X', 'X', 'O', 'X'),
             array('X', 'X', 'X', 'O', 'X', 'X'),
             array('O', 'X', 'X', 'X', 'X', 'X'),
             array('X', 'X', 'X', 'O', 'X', 'O'),
             array('O', 'O', 'X', 'O', 'O', 'O'));
replaceSurrounded($mat);

for ($i = 0; $i < $GLOBALS['M']; $i++)
{
    for ($j = 0; $j < $GLOBALS['N']; $j++)
        echo $mat[$i][$j]." ";
    echo "\n";
}

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

Output:

```
X O X O X X
X O X X X X
X X X X X X
O X X X X X
X X X O X O
O O X O O O
```

Time Complexity of the above solution is $O(MN)$. Note that every element of matrix is processed at most three times.

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

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

Source

<https://www.geeksforgeeks.org/given-matrix-o-x-replace-o-x-surrounded-x/>

Chapter 49

Given a string, print all possible palindromic partitions

Given a string, print all possible palindromic partitions - GeeksforGeeks

Given a string, find all possible palindromic partitions of given string.

Example:

```
Input:  nitin
Output: n i t i n
        n iti n
        nitin

Input:  geeks
Output: g e e k s
        g ee k s
```

Note that this problem is different from [Palindrome Partitioning Problem](#), there the task was to find the partitioning with minimum cuts in input string. Here we need to print all possible partitions.

The idea is to go through every substring starting from first character, check if it is palindrome. If yes, then add the substring to solution and recur for remaining part. Below is complete algorithm.

Below is C++ implementation of above idea

C++

```
// C++ program to print all palindromic partitions of a given string
#include<bits/stdc++.h>
using namespace std;

// A utility function to check if str is palindromem
```

```
bool isPalindrome(string str, int low, int high)
{
    while (low < high)
    {
        if (str[low] != str[high])
            return false;
        low++;
        high--;
    }
    return true;
}

// Recursive function to find all palindromic partitions of str[start..n-1]
// allPart --> A vector of vector of strings. Every vector inside it stores
//             a partition
// currPart --> A vector of strings to store current partition
void allPalPartUtil(vector<vector<string> >&allPart, vector<string> &currPart,
                    int start, int n, string str)
{
    // If 'start' has reached len
    if (start >= n)
    {
        allPart.push_back(currPart);
        return;
    }

    // Pick all possible ending points for substrings
    for (int i=start; i<n; i++)
    {
        // If substring str[start..i] is palindrome
        if (isPalindrome(str, start, i))
        {
            // Add the substring to result
            currPart.push_back(str.substr(start, i-start+1));

            // Recur for remaining remaining substring
            allPalPartUtil(allPart, currPart, i+1, n, str);

            // Remove substring str[start..i] from current
            // partition
            currPart.pop_back();
        }
    }
}

// Function to print all possible palindromic partitions of
// str. It mainly creates vectors and calls allPalPartUtil()
void allPalPartitions(string str)
```



```
{
    int n = str.length();

    // To Store all palindromic partitions
    vector<vector<string> > allPart;

    // To store current palindromic partition
    vector<string> currPart;

    // Call recursive function to generate all partiions
    // and store in allPart
    allPalPartUtil(allPart, currPart, 0, n, str);

    // Print all partitions generated by above call
    for (int i=0; i< allPart.size(); i++ )
    {
        for (int j=0; j<allPart[i].size(); j++)
            cout << allPart[i][j] << " ";
        cout << "\n";
    }
}

// Driver program
int main()
{
    string str = "nitin";
    allPalPartitions(str);
    return 0;
}
```

Output:

```
n i t i n
n iti n
nitin
```

This article is contributed by [Ekta Goel](#). 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/given-a-string-print-all-possible-palindromic-partition/>

Chapter 50

Given an array A[] and a number x, check for pair in A[] with sum as x

Given an array A[] and a number x, check for pair in A[] with sum as x - GeeksforGeeks

Write a program that, given an array A[] of n numbers and another number x, determines whether or not there exist two elements in S whose sum is exactly x.

METHOD 1 (Use Sorting)

Algorithm :

```
hasArrayTwoCandidates (A[], ar_size, sum)
1) Sort the array in non-decreasing order.
2) Initialize two index variables to find the candidate
   elements in the sorted array.
   (a) Initialize first to the leftmost index: l = 0
   (b) Initialize second the rightmost index: r = ar_size-1
3) Loop while l < r.
   (a) If (A[l] + A[r] == sum) then return 1
   (b) Else if ( A[l] + A[r] < sum ) then l++
   (c) Else r--
4) No candidates in whole array - return 0
```

Time Complexity: Depends on what sorting algorithm we use. If we use Merge Sort or Heap Sort then $O(n \log n)$ in worst case. If we use Quick Sort then $O(n^2)$ in worst case.

Auxiliary Space : Again, depends on sorting algorithm. For example auxiliary space is $O(n)$ for merge sort and $O(1)$ for Heap Sort.

Example :

Let Array be {1, 4, 45, 6, 10, -8} and sum to find be 16

Sort the array

$A = \{-8, 1, 4, 6, 10, 45\}$

Initialize $l = 0, r = 5$

$A[l] + A[r] \ (-8 + 45) > 16 \Rightarrow$ decrement r . Now $r = 10$

$A[l] + A[r] \ (-8 + 10)$ increment l . Now $l = 1$

$A[l] + A[r] \ (1 + 10)$ increment l . Now $l = 2$

$A[l] + A[r] \ (4 + 10)$ increment l . Now $l = 3$

$A[l] + A[r] \ (6 + 10) == 16 \Rightarrow$ Found candidates (return 1)

Note: If there are more than one pair having the given sum then this algorithm reports only one. Can be easily extended for this though.

Below is the implementation of the above approach.

C

```
// C program to check if given array
// has 2 elements whose sum is equal
// to the given value

#include <stdio.h>
#define bool int

void quickSort(int *, int, int);

bool hasArrayTwoCandidates(int A[], int arr_size, int sum)
{
    int l, r;

    /* Sort the elements */
    quickSort(A, 0, arr_size-1);

    /* Now look for the two candidates in the sorted
       array*/
    l = 0;
    r = arr_size-1;
    while (l < r)
    {
        if(A[l] + A[r] == sum)
            return 1;
        else if(A[l] + A[r] < sum)
            l++;
        else // A[i] + A[j] > sum
            r--;
    }
    return 0;
}
```

```
}

/* FOLLOWING FUNCTIONS ARE ONLY FOR SORTING
   PURPOSE */
void exchange(int *a, int *b)
{
    int temp;
    temp = *a;
    *a   = *b;
    *b   = temp;
}

int partition(int A[], int si, int ei)
{
    int x = A[ei];
    int i = (si - 1);
    int j;

    for (j = si; j <= ei - 1; j++)
    {
        if(A[j] <= x)
        {
            i++;
            exchange(&A[i], &A[j]);
        }
    }
    exchange (&A[i + 1], &A[ei]);
    return (i + 1);
}

/* Implementation of Quick Sort
   A[] --> Array to be sorted
   si --> Starting index
   ei --> Ending index
   */
void quickSort(int A[], int si, int ei)
{
    int pi;    /* Partitioning index */
    if(si < ei)
    {
        pi = partition(A, si, ei);
        quickSort(A, si, pi - 1);
        quickSort(A, pi + 1, ei);
    }
}

/* Driver program to test above function */
int main()
```

```
{
    int A[] = {1, 4, 45, 6, 10, -8};
    int n = 16;
    int arr_size = 6;

    if( hasArrayTwoCandidates(A, arr_size, n))
        printf("Array has two elements with given sum");
    else
        printf("Array doesn't have two elements with given sum");

    getchar();
    return 0;
}
```

C++

```
// C++ program to check if given array
// has 2 elements whose sum is equal
// to the given value

#include <bits/stdc++.h>

using namespace std;

// Function to check if array has 2 elements
// whose sum is equal to the given value
bool hasArrayTwoCandidates(int A[], int arr_size,
                           int sum)
{
    int l, r;

    /* Sort the elements */
    sort(A, A + arr_size);

    /* Now look for the two candidates in
       the sorted array*/
    l = 0;
    r = arr_size - 1;
    while (l < r)
    {
        if(A[l] + A[r] == sum)
            return 1;
        else if(A[l] + A[r] < sum)
            l++;
        else // A[i] + A[j] > sum
            r--;
    }
    return 0;
}
```

```
}

/* Driver program to test above function */
int main()
{
    int A[] = {1, 4, 45, 6, 10, -8};
    int n = 16;
    int arr_size = sizeof(A) / sizeof(A[0]);

    // Function calling
    if(hasArrayTwoCandidates(A, arr_size, n))
        cout << "Array has two elements with given sum";
    else
        cout << "Array doesn't have two elements with given sum";

    return 0;
}
```

Java

```
// Java program to check if given array
// has 2 elements whose sum is equal
// to the given value
import java.util.*;

class GFG
{
    // Function to check if array has 2 elements
    // whose sum is equal to the given value
    static boolean hasArrayTwoCandidates(int A[],
                                         int arr_size, int sum)
    {
        int l, r;

        /* Sort the elements */
        Arrays.sort(A);

        /* Now look for the two candidates
        in the sorted array*/
        l = 0;
        r = arr_size-1;
        while (l < r)
        {
            if(A[l] + A[r] == sum)
                return true;
            else if(A[l] + A[r] < sum)
                l++;
            else // A[i] + A[j] > sum
                r--;
        }
    }
}
```

```
        r--;
    }
    return false;
}

// Driver code
public static void main(String args[])
{
    int A[] = {1, 4, 45, 6, 10, -8};
    int n = 16;
    int arr_size = A.length;

    // Function calling
    if(hasArrayTwoCandidates(A, arr_size, n))
        System.out.println("Array has two " +
                           "elements with given sum");
    else
        System.out.println("Array doesn't have " +
                           "two elements with given sum");

}

}
```

Python

```
# Python program to check for the sum condition to be satisfied

def hasArrayTwoCandidates(A,arr_size,sum):

    # sort the array
    quickSort(A,0,arr_size-1)
    l = 0
    r = arr_size-1

    # traverse the array for the two elements
    while l<r:
        if (A[l] + A[r] == sum):
            return 1
        elif (A[l] + A[r] < sum):
            l += 1
        else:
            r -= 1
    return 0

# Implementation of Quick Sort
# A[] --> Array to be sorted
# si --> Starting index
```

```
# ei --> Ending index
def quickSort(A, si, ei):
    if si < ei:
        pi=partition(A,si,ei)
        quickSort(A,si,pi-1)
        quickSort(A,pi+1,ei)

# Utility function for partitioning the array(used in quick sort)
def partition(A, si, ei):
    x = A[ei]
    i = (si-1)
    for j in range(si,ei):
        if A[j] <= x:
            i += 1

        # This operation is used to swap two variables in python
        A[i], A[j] = A[j], A[i]

    A[i+1], A[ei] = A[ei], A[i+1]

    return i+1

# Driver program to test the functions
A = [1,4,45,6,10,-8]
n = 16
if (hasArrayTwoCandidates(A, len(A), n)):
    print("Array has two elements with the given sum")
else:
    print("Array doesn't have two elements with the given sum")

## This code is contributed by __Devesh Agrawal__
```

C#

```
// C# program to check for pair
// in A[] with sum as x

using System;

class GFG
{
    static bool hasArrayTwoCandidates(int []A,
                                      int arr_size, int sum)
    {
        int l, r;

        /* Sort the elements */
```



```
sort(A, 0, arr_size-1);

/* Now look for the two candidates
in the sorted array*/
l = 0;
r = arr_size-1;
while (l < r)
{
    if(A[l] + A[r] == sum)
        return true;
    else if(A[l] + A[r] < sum)
        l++;
    else // A[i] + A[j] > sum
        r--;
}
return false;
}

/* Below functions are only to sort the
array using QuickSort */

/* This function takes last element as pivot,
places the pivot element at its correct
position in sorted array, and places all
smaller (smaller than pivot) to left of
pivot and all greater elements to right
of pivot */
static int partition(int []arr, int low, int high)
{
    int pivot = arr[high];

    // index of smaller element
    int i = (low-1);
    for (int j = low; j <= high - 1; j++)
    {
        // If current element is smaller
        // than or equal to pivot
        if (arr[j] <= pivot)
        {
            i++;

            // swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
}
```

```
        // swap arr[i+1] and arr[high] (or pivot)
        int temp1 = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp1;

        return i+1;
    }

    /* The main function that
    implements QuickSort()
    arr[] --> Array to be sorted,
    low --> Starting index,
    high --> Ending index */
    static void sort(int []arr, int low, int high)
    {
        if (low < high)
        {
            /* pi is partitioning index, arr[pi]
            is now at right place */
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }

    // Driver code
    public static void Main()
    {
        int []A = {1, 4, 45, 6, 10, -8};
        int n = 16;
        int arr_size = 6;

        if( hasArrayTwoCandidates(A, arr_size, n))
            Console.WriteLine("Array has two elements"+
                             " with given sum");
        else
            Console.WriteLine("Array doesn't have "+
                             "two elements with given sum");
    }
}
```

// This code is contributed by Sam007

PHP

```
<?php
// PHP program to check if given
// array has 2 elements whose sum
// is equal to the given value

// Function to check if array has
// 2 elements whose sum is equal
// to the given value
function hasArrayTwoCandidates($A, $arr_size,
                               $sum)
{
    $l; $r;

    /* Sort the elements */
    //sort($A, A + arr_size);
    sort($A);

    /* Now look for the two candidates
    in the sorted array*/
    $l = 0;
    $r = $arr_size - 1;
    while ($l < $r)
    {
        if($A[$l] + $A[$r] == $sum)
            return 1;
        else if($A[$l] + $A[$r] < $sum)
            $l++;
        else // A[i] + A[j] > sum
            $r--;
    }
    return 0;
}

// Driver Code
$A = array (1, 4, 45, 6, 10, -8);
$n = 16;
$arr_size = sizeof($A);

// Function calling
if(hasArrayTwoCandidates($A, $arr_size, $n))
    echo "Array has two elements " .
        "with given sum";
else
    echo "Array doesn't have two " .
        "elements with given sum";

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

Output :

Array has two elements with the given sum

METHOD 2 (Use Hashing)

This method works in $O(n)$ time.

- 1) Initialize an empty hash table s .
- 2) Do following for each element $A[i]$ in $A[]$
 - (a) If $s[x - A[i]]$ is set then print the pair $(A[i], x - A[i])$
 - (b) Insert $A[i]$ into s .

Below is the implementation of the above approach :

C

```
// C++ program to check if given array
// has 2 elements whose sum is equal
// to the given value

// Works only if range elements is limited
#include <stdio.h>
#define MAX 100000

void printPairs(int arr[], int arr_size, int sum)
{
    int i, temp;
    bool s[MAX] = {0}; /*initialize hash set as 0*/

    for (i = 0; i < arr_size; i++)
    {
        temp = sum - arr[i];
        if (temp >= 0 && s[temp] == 1)
            printf("Pair with given sum %d is (%d, %d) n",
                sum, arr[i], temp);
        s[arr[i]] = 1;
    }
}

/* Driver program to test above function */
int main()
{
    int A[] = {1, 4, 45, 6, 10, 8};
    int n = 16;
    int arr_size = sizeof(A)/sizeof(A[0]);
```

```
    printPairs(A, arr_size, n);

    getchar();
    return 0;
}
```

C++

```
// C++ program to check if given array
// has 2 elements whose sum is equal
// to the given value
#include <bits/stdc++.h>

using namespace std;

void printPairs(int arr[], int arr_size, int sum)
{
    unordered_set<int> s;
    for (int i = 0; i < arr_size; i++)
    {
        int temp = sum - arr[i];

        if (temp >= 0 && s.find(temp) != s.end())
            cout << "Pair with given sum " << sum <<
                " is (" << arr[i] << ", " << temp <<
                ")" << endl;

        s.insert(arr[i]);
    }
}

/* Driver program to test above function */
int main()
{
    int A[] = {1, 4, 45, 6, 10, 8};
    int n = 16;
    int arr_size = sizeof(A)/sizeof(A[0]);

    // Function calling
    printPairs(A, arr_size, n);

    return 0;
}
```

Java

```
// Java implementation using Hashing
```

```
import java.io.*;
import java.util.HashSet;

class PairSum
{
    static void printpairs(int arr[],int sum)
    {
        HashSet<Integer> s = new HashSet<Integer>();
        for (int i=0; i<arr.length; ++i)
        {
            int temp = sum-arr[i];

            // checking for condition
            if (temp>=0 && s.contains(temp))
            {
                System.out.println("Pair with given sum " +
                                   sum + " is (" + arr[i] +
                                   ", "+temp+"");
            }
            s.add(arr[i]);
        }
    }

    // Main to test the above function
    public static void main (String[] args)
    {
        int A[] = {1, 4, 45, 6, 10, 8};
        int n = 16;
        printpairs(A, n);
    }
}

// This article is contributed by Aakash Hasija
```

Python

```
# Python program to find if there are
# two elements with given sum

# function to check for the given sum
# in the array
def printPairs(arr, arr_size, sum):

    # Create an empty hash set
    s = set()

    for i in range(0,arr_size):
        temp = sum-arr[i]
```

```
        if (temp>=0 and temp in s):
            print ("Pair with the given sum is", arr[i], "and", temp)
            s.add(arr[i])

# driver program to check the above function
A = [1,4,45,6,10,8]
n = 16
printPairs(A, len(A), n)

# This code is contributed by __Devesh Agrawal__
```

C#

```
// C# implementation using Hashing
using System;
using System.Collections.Generic;

class GFG
{
    static void printpairs(int []arr,
                          int sum)
    {
        HashSet<int> s = new HashSet<int>();
        for (int i = 0; i < arr.Length; ++i)
        {
            int temp = sum - arr[i];

            // checking for condition
            if (temp >= 0 && s.Contains(temp))
            {
                Console.WriteLine("Pair with given sum " +
                                sum + " is (" + arr[i] +
                                ", " + temp + ")");
            }
            s.Add(arr[i]);
        }
    }
}

// Driver Code
static void Main ()
{
    int []A = new int[]{1, 4, 45,
                        6, 10, 8};

    int n = 16;
    printpairs(A, n);
}
}
```

```
// This code is contributed by  
// Manish Shaw(manishshaw1)
```

Output:

Pair with given sum 16 is (10, 6)

Time Complexity: $O(n)$

Auxiliary Space: $O(n)$ where n is size of array.

If range of numbers include negative numbers then also it works. All we have to do for negative numbers is to make everything positive by adding the absolute value of smallest negative integer to all numbers.

Related Problems:

[Given two unsorted arrays, find all pairs whose sum is x](#)

[Count pairs with given sum](#)[Count all distinct pairs with difference equal to k](#)

Improved By : [jit_t](#), [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/given-an-array-a-and-a-number-x-check-for-pair-in-a-with-sum-as-x/>

Chapter 51

Happy Number

Happy Number - GeeksforGeeks

A number is called happy if it leads to 1 after a sequence of steps where in each step number is replaced by sum of squares of its digit that is if we start with Happy Number and keep replacing it with digits square sum, we reach 1.

Examples :

```
Input: n = 19
Output: True
19 is Happy Number,
 $1^2 + 9^2 = 82$ 
 $8^2 + 2^2 = 68$ 
 $6^2 + 8^2 = 100$ 
 $1^2 + 0^2 + 0^2 = 1$ 
As we reached to 1, 19 is a Happy Number.
```

```
Input: n = 20
Output: False
```

A number will not be a Happy Number when it makes a loop in its sequence that is it touches a number in sequence which already been touched. So to check whether a number is happy or not, we can keep a set, if same number occurs again we flag result as not happy. A simple function on above approach can be written as below –

```
// method return true if n is Happy Number
// numSquareSum method is given in below detailed code snippet
int isHappyNumber(int n)
{
    set<int> st;
    while (1)
    {
```

```
        n = numSquareSum(n);
        if (n == 1)
            return true;
        if (st.find(n) != st.end())
            return false;
        st.insert(n);
    }
}
```

We can solve this problem without using extra space and that technique can be used in some other similar problem also. If we treat every number as a node and replacement by square sum digit as a link, then this problem is same as [finding a loop in a linklist](#) :

So as proposed solution from above link, we will keep two number slow and fast both initialize from given number, slow is replaced one step at a time and fast is replaced two steps at a time. If they meet at 1, then the given number is Happy Number otherwise not.

C++

```
// C/C++ program to check a number is a Happy
// number or not
#include <bits/stdc++.h>
using namespace std;

// Utility method to return sum of square of
// digit of n
int numSquareSum(int n)
{
    int squareSum = 0;
    while (n)
    {
        squareSum += (n % 10) * (n % 10);
        n /= 10;
    }
    return squareSum;
}

// method return true if n is Happy number
bool isHappyNumber(int n)
{
    int slow, fast;

    // initialize slow and fast by n
    slow = fast = n;
    do
    {
        // move slow number by one iteration
        slow = numSquareSum(slow);
```

```
        //    move fast number by two iteration
        fast = numSquareSum(numSquareSum(fast));

    }
    while (slow != fast);

    //    if both number meet at 1, then return true
    return (slow == 1);
}

//    Driver code to test above methods
int main()
{
    int n = 13;
    if (isHappynumber(n))
        cout << n << " is a Happy number\n";
    else
        cout << n << " is not a Happy number\n";
}
```

Java

```
//    Java program to check a number is a Happy
//    number or not

class GFG {

//    Utility method to return sum of square of
//    digit of n
static int numSquareSum(int n)
{
    int squareSum = 0;
    while (n!= 0)
    {
        squareSum += (n % 10) * (n % 10);
        n /= 10;
    }
    return squareSum;
}

//    method return true if n is Happy number
static boolean isHappynumber(int n)
{
    int slow, fast;

    //    initialize slow and fast by n
    slow = fast = n;
```

```
do
{
    // move slow number
    // by one iteration
    slow = numSquareSum(slow);

    // move fast number
    // by two iteration
    fast = numSquareSum(numSquareSum(fast));

}
while (slow != fast);

// if both number meet at 1,
// then return true
return (slow == 1);
}

// Driver code to test above methods
public static void main(String[] args)
{
    int n = 13;
    if (isHappyNumber(n))
        System.out.println(n +
            " is a Happy number");
    else
        System.out.println(n +
            " is not a Happy number");
}
}
```

C#

```
// C# program to check a number
// is a Happy number or not
using System;
class GFG {

// Utility method to return
// sum of square of digit of n
static int numSquareSum(int n)
{
    int squareSum = 0;
    while (n!= 0)
    {
        squareSum += (n % 10) *
                     (n % 10);
        n /= 10;
    }
}
```

```
    }
    return squareSum;
}

// method return true if
// n is Happy number
static bool isHappynumber(int n)
{
    int slow, fast;

    // initialize slow and
    // fast by n
    slow = fast = n;
    do
    {
        // move slow number
        // by one iteration
        slow = numSquareSum(slow);

        // move fast number
        // by two iteration
        fast = numSquareSum(numSquareSum(fast));

    }
    while (slow != fast);

    // if both number meet at 1,
    // then return true
    return (slow == 1);
}

// Driver code
public static void Main()
{
    int n = 13;
    if (isHappynumber(n))
        Console.WriteLine(n +
            " is a Happy number");
    else
        Console.WriteLine(n +
            " is not a Happy number");
}
}

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

PHP

```
<?php
// PHP program to check a number
// is a Happy number or not

// Utility method to return
// sum of square of digit of n
function numSquareSum( $n)
{
    $squareSum = 0;
    while ($n)
    {
        $squareSum += ($n % 10) *
                      ($n % 10);
        $n /= 10;
    }
    return $squareSum;
}

// method return true if
// n is Happy number
function isHappynumber( $n)
{
    $slow; $fast;

    // initialize slow
    // and fast by n
    $slow = $n;
    $fast = $n;
    do
    {
        // move slow number
        // by one iteration
        $slow = numSquareSum($slow);

        // move fast number
        // by two iteration
        $fast = numSquareSum(numSquareSum($fast));

    }
    while ($slow != $fast);

    // if both number meet at 1,
    // then return true
    return ($slow == 1);
}

// Driver Code
$n = 13;
```

```
if (isHappynumber($n))
    echo $n , " is a Happy number\n";
else
    echo n , " is not a Happy number\n";

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

Output :

13 is a Happy Number

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/happy-number/>

Chapter 52

Highest power of 2 less than or equal to given number

Highest power of 2 less than or equal to given number - GeeksforGeeks

Given a number n, find the highest power of 2 that is smaller than or equal to n.

Examples :

Input : n = 10
Output : 8

Input : n = 19
Output : 16

Input : n = 32
Output : 32

A **simple solution** is to start checking from n and keep decrementing until we find a power of 2.

C++

```
// C++ program to find highest power of 2 smaller
// than or equal to n.
#include<bits/stdc++.h>
using namespace std;

int highestPowerof2(int n)
{
    int res = 0;
```



```
    for (int i=n; i>=1; i--)
    {
        // If i is a power of 2
        if ((i & (i-1)) == 0)
        {
            res = i;
            break;
        }
    }
    return res;
}
```

```
// Driver code
int main()
{
    int n = 10;
    cout << highestPowerof2(n);
    return 0;
}
```

Java

```
// Java code to find highest power
// of 2 smaller than or equal to n.
class GFG
{
    static int highestPowerof2(int n)
    {
        int res = 0;
        for (int i = n; i >= 1; i--)
        {
            // If i is a power of 2
            if ((i & (i - 1)) == 0)
            {
                res = i;
                break;
            }
        }
        return res;
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 10;
        System.out.println(highestPowerof2(n));
    }
}
```

// This code is contributed by prerna saini.

C#

```
// C# code to find highest power
// of 2 smaller than or equal to n.
using System;

class GFG
{
    public static int highestPowerof2(int n)
    {
        int res = 0;
        for (int i = n; i >= 1; i--)
        {
            // If i is a power of 2
            if ((i & (i - 1)) == 0)
            {
                res = i;
                break;
            }
        }
        return res;
    }

    // Driver Code
    static public void Main ()
    {
        int n = 10;
        Console.WriteLine(highestPowerof2(n));
    }
}
```

// This code is contributed by ajit

PHP

```
<?php
// PHP program to find highest
// power of 2 smaller than or
// equal to n.
function highestPowerof2($n)
{
    $res = 0;
    for ($i = $n; $i >= 1; $i--)
    {
```

```
// If i is a power of 2
if (($i & ($i - 1)) == 0)
{
    $res = $i;
    break;
}
}
return $res;
}

// Driver code
$n = 10;
echo highestPowerof2($n);

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

Output :

8

Time complexity : $O(n)$. In worst case, the loop runs $\text{floor}(n/2)$ times. The worst case happens when n is of the form $2^x - 1$.

An **efficient solution** is to use bitwise left shift operator to find all powers of 2 starting from 1. For every power check if it is smaller than or equal to n or not. Below is C++ implementation of the idea.

C++

```
// C++ program to find highest power of 2 smaller
// than or equal to n.
#include<bits/stdc++.h>
using namespace std;

int highestPowerof2(unsigned int n)
{
    // Invalid input
    if (n < 1)
        return 0;

    int res = 1;

    // Try all powers starting from 2^1
    for (int i=0; i<8*sizeof(unsigned int); i++)
    {
```

```
        int curr = 1 << i;

        // If current power is more than n, break
        if (curr > n)
            break;

        res = curr;
    }

    return res;
}

// Driver code
int main()
{
    int n = 10;
    cout << highestPowerof2(n);
    return 0;
}
```

Java

```
// Java program to find
// highest power of 2 smaller
// than or equal to n.
import java.io.*;

class GFG
{
    static int highestPowerof2(int n)
    {
        // Invalid input
        if (n < 1)
            return 0;

        int res = 1;

        // Try all powers
        // starting from 2^1
        for (int i = 0; i < 8 ; i++)
        {
            int curr = 1 << i;

            // If current power is
            // more than n, break
            if (curr > n)
                break;
        }
    }
}
```

```
        res = curr;
    }

    return res;
}

// Driver code
public static void main(String[] args)
{
    int n = 10;
    System.out.println(highestPowerof2(n));
}
}
```

// This code is contributed aj_36

C#

```
// C# program to find
// highest power of 2 smaller
// than or equal to n.
using System;

class GFG
{
    static int highestPowerof2(int n)
    {
        // Invalid input
        if (n < 1)
            return 0;

        int res = 1;

        // Try all powers
        // starting from 2^1
        for (int i = 0; i < 8 ; i++)
        {
            int curr = 1 << i;

            // If current power is
            // more than n, break
            if (curr > n)
                break;

            res = curr;
        }

        return res;
    }
}
```

```
}

// Driver code
static public void Main ()
{
    int n = 10;
    Console.WriteLine(highestPowerof2(n));
}
}

// This code is contributed ajit
```

PHP

```
<?php
// PHP program to find highest
// power of 2 smaller
// than or equal to n.

function highestPowerof2($n)
{
    // Invalid input
    if ($n < 1)
        return 0;

    $res = 1;

    // Try all powers starting
    // from 2^1
    for ($i = 0; $i < 8 ; $i++)
    {
        $curr = 1 << $i;

        // If current power is
        // more than n, break
        if ($curr > $n)
            break;

        $res = $curr;
    }

    return $res;
}

// Driver code
$n = 10;
echo highestPowerof2($n);
```

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

Output :

8

A Solution using Log

Thanks to Anshuman Jha for suggesting this solution.

C++

```
// C++ program to find highest power of 2 smaller
// than or equal to n.
#include<bits/stdc++.h>
using namespace std;

int highestPowerof2(int n)
{
    int p = (int)log2(n);
    return (int)pow(2, p);
}

// Driver code
int main()
{
    int n = 10;
    cout << highestPowerof2(n);
    return 0;
}
```

Java

```
// Java program to find
// highest power of 2
// smaller than or equal to n.
import java.io.*;

class GFG
{
    static int highestPowerof2(int n)
    {

        int p = (int)(Math.log(n) /
                      Math.log(2));
        return (int)Math.pow(2, p);
    }
}
```

```
}

// Driver code
public static void main (String[] args)
{
    int n = 10;
    System.out.println(highestPowerof2(n));
}
}

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

C#

```
// C# program to find
// highest power of 2
// smaller than or equal to n.
using System;

class GFG
{
    static int highestPowerof2(int n)
    {
        int p = (int)(Math.Log(n) /
                      Math.Log(2));
        return (int)Math.Pow(2, p);
    }

    // Driver code
    static public void Main ()
    {
        int n = 10;
        Console.WriteLine(highestPowerof2(n));
    }
}

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

PHP

```
<?php
// PHP program to find highest
// power of 2 smaller than or
// equal to n.
function highestPowerof2($n)
```



```
{
    $p = (int)log($n, 2);
    return (int)pow(2, $p);
}

// Driver code
$n = 10;
echo highestPowerof2($n);

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

Output :

8

Application Problem:

Some people are standing in a queue. A selection process follows a rule where people standing on even positions are selected. Of the selected people a queue is formed and again out of these only people on even position are selected. This continues until we are left with one person. Find out the position of that person in the original queue.

Print the position(original queue) of that person who is left.

Examples :

Input: n = 10

Output:8

Explanation :

```
1 2 3 4 5 6 7 8 9 10  ==>Given queue
    2 4 6 8 10
        4 8
            8
```

Input: n = 17

Input: 16

Explanation :

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17  ==>Given queue
    2 4 6 8 10 12 14 16
        4 8 12 16
            8 16
                16
```

Related Article :

[Power of 2 greater than or equal to a given number.](#)

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/highest-power-2-less-equal-given-number/>

Chapter 53

How to print maximum number of A's using given four keys

How to print maximum number of A's using given four keys - GeeksforGeeks

This is a famous interview question asked in [Google](#), [Paytm](#) and many other company interviews.

Below is the problem statement.

Imagine you have a special keyboard with the following keys:

Key 1: Prints 'A' on screen

Key 2: (Ctrl-A): Select screen

Key 3: (Ctrl-C): Copy selection to buffer

Key 4: (Ctrl-V): Print buffer on screen appending it
after what has already been printed.

If you can only press the keyboard for N times (with the above four keys), write a program to produce maximum numbers of A's. That is to say, the input parameter is N (No. of keys that you can press), the output is M (No. of As that you can produce).

Examples:

Input: N = 3

Output: 3

We can at most get 3 A's on screen by pressing following key sequence.

A, A, A

Input: N = 7
Output: 9
We can at most get 9 A's on screen by pressing
following key sequence.
A, A, A, Ctrl A, Ctrl C, Ctrl V, Ctrl V

Input: N = 11
Output: 27
We can at most get 27 A's on screen by pressing
following key sequence.
A, A, A, Ctrl A, Ctrl C, Ctrl V, Ctrl V, Ctrl A,
Ctrl C, Ctrl V, Ctrl V

Below are few important points to note.

- a) For $N < 7$, the output is N itself.
- b) Ctrl V can be used multiple times to print current buffer (See last two examples above). The idea is to compute the optimal string length for N keystrokes by using a simple insight. The sequence of N keystrokes which produces an optimal string length will end with a suffix of Ctrl-A, a Ctrl-C, followed by only Ctrl-V's . (For $N > 6$)

The task is to find out the break=point after which we get the above suffix of keystrokes. Definition of a breakpoint is that instance after which we need to only press Ctrl-A, Ctrl-C once and the only Ctrl-V's afterwards to generate the optimal length. If we loop from N-3 to 1 and choose each of these values for the break-point, and compute that optimal string they would produce. Once the loop ends, we will have the maximum of the optimal lengths for various breakpoints, thereby giving us the optimal length for N keystrokes.

Below is implementation based on above idea.

C

```
/* A recursive C program to print maximum number of A's using
   following four keys */
#include<stdio.h>

// A recursive function that returns the optimal length string
// for N keystrokes
int findoptimal(int N)
{
    // The optimal string length is N when N is smaller than 7
    if (N <= 6)
        return N;

    // Initialize result
    int max = 0;

    // TRY ALL POSSIBLE BREAK-POINTS
    // For any keystroke N, we need to loop from N-3 keystrokes
```

```
// back to 1 keystroke to find a breakpoint 'b' after which we
// will have Ctrl-A, Ctrl-C and then only Ctrl-V all the way.
int b;
for (b=N-3; b>=1; b--)
{
    // If the breakpoint is s at b'th keystroke then
    // the optimal string would have length
    // (n-b-1)*screen[b-1];
    int curr = (N-b-1)*findoptimal(b);
    if (curr > max)
        max = curr;
}
return max;
}

// Driver program
int main()
{
    int N;

    // for the rest of the array we will rely on the previous
    // entries to compute new ones
    for (N=1; N<=20; N++)
        printf("Maximum Number of A's with %d keystrokes is %d\n",
            N, findoptimal(N));
}
```

Java

```
/* A recursive C program to print
maximum number of A's using
following four keys */
import java.io.*;

class GFG {

    // A recursive function that returns
    // the optimal length string for N keystrokes
    static int findoptimal(int N)
    {
        // The optimal string length is N
        // when N is smaller than 7
        if (N <= 6)
            return N;

        // Initialize result
        int max = 0;
    }
}
```

```
// TRY ALL POSSIBLE BREAK-POINTS
// For any keystroke N, we need to
// loop from N-3 keystrokes back to
// 1 keystroke to find a breakpoint
// 'b' after which we will have Ctrl-A,
// Ctrl-C and then only Ctrl-V all the way.
int b;
for (b = N - 3; b >= 1; b--)
{
    // If the breakpoint is s at b'th
    // keystroke then the optimal string
    // would have length
    // (n-b-1)*screen[b-1];
    int curr = (N - b - 1) * findoptimal(b);
    if (curr > max)
        max = curr;
}
return max;
}

// Driver program
public static void main(String[] args)
{
    int N;

    // for the rest of the array we
    // will rely on the previous
    // entries to compute new ones
    for (N = 1; N <= 20; N++)
        System.out.println("Maximum Number of A's with keystrokes is " +
                           N + findoptimal(N));
}

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

C#

```
/* A recursive C# program
to print maximum number
of A's using following
four keys */
using System;

class GFG
{
    // A recursive function that
    // returns the optimal length
```

```
// string for N keystrokes
static int findoptimal(int N)
{
    // The optimal string length
    // is N when N is smaller than 7
    if (N <= 6)
        return N;

    // Initialize result
    int max = 0;

    // TRY ALL POSSIBLE BREAK-POINTS
    // For any keystroke N, we need
    // to loop from N-3 keystrokes
    // back to 1 keystroke to find
    // a breakpoint 'b' after which
    // we will have Ctrl-A, Ctrl-C
    // and then only Ctrl-V all the way.
    int b;
    for (b = N - 3; b >= 1; b--)
    {
        // If the breakpoint is s
        // at b'th keystroke then
        // the optimal string would
        // have length (n-b-1)*screen[b-1];
        int curr = (N - b - 1) *
            findoptimal(b);
        if (curr > max)
            max = curr;
    }
    return max;
}

// Driver code
static void Main()
{
    int N;

    // for the rest of the array
    // we will rely on the
    // previous entries to compute
    // new ones
    for (N = 1; N <= 20; N++)
        Console.WriteLine("Maximum Number of A's with " +
            N + " keystrokes is " +
            findoptimal(N));
}
}
```

```
// This code is contributed by Sam007
```

Output:

```
Maximum Number of A's with 1 keystrokes is 1
Maximum Number of A's with 2 keystrokes is 2
Maximum Number of A's with 3 keystrokes is 3
Maximum Number of A's with 4 keystrokes is 4
Maximum Number of A's with 5 keystrokes is 5
Maximum Number of A's with 6 keystrokes is 6
Maximum Number of A's with 7 keystrokes is 9
Maximum Number of A's with 8 keystrokes is 12
Maximum Number of A's with 9 keystrokes is 16
Maximum Number of A's with 10 keystrokes is 20
Maximum Number of A's with 11 keystrokes is 27
Maximum Number of A's with 12 keystrokes is 36
Maximum Number of A's with 13 keystrokes is 48
Maximum Number of A's with 14 keystrokes is 64
Maximum Number of A's with 15 keystrokes is 81
Maximum Number of A's with 16 keystrokes is 108
Maximum Number of A's with 17 keystrokes is 144
Maximum Number of A's with 18 keystrokes is 192
Maximum Number of A's with 19 keystrokes is 256
Maximum Number of A's with 20 keystrokes is 324
```

The above function computes the same subproblems again and again. Recomputations of same subproblems can be avoided by storing the solutions to subproblems and solving problems in bottom up manner.

Below is Dynamic Programming based C implementation where an auxiliary array `screen[N]` is used to store result of subproblems.

C

```
/* A Dynamic Programming based C program to find maximum number of A's
   that can be printed using four keys */
#include<stdio.h>

// this function returns the optimal length string for N keystrokes
int findoptimal(int N)
{
    // The optimal string length is N when N is smaller than 7
    if (N <= 6)
        return N;
```



```
// An array to store result of subproblems
int screen[N];

int b; // To pick a breakpoint

// Initializing the optimal lengths array for upto 6 input
// strokes.
int n;
for (n=1; n<=6; n++)
    screen[n-1] = n;

// Solve all subproblems in bottom manner
for (n=7; n<=N; n++)
{
    // Initialize length of optimal string for n keystrokes
    screen[n-1] = 0;

    // For any keystroke n, we need to loop from n-3 keystrokes
    // back to 1 keystroke to find a breakpoint 'b' after which we
    // will have ctrl-a, ctrl-c and then only ctrl-v all the way.
    for (b=n-3; b>=1; b--)
    {
        // if the breakpoint is at b'th keystroke then
        // the optimal string would have length
        // (n-b-1)*screen[b-1];
        int curr = (n-b-1)*screen[b-1];
        if (curr > screen[n-1])
            screen[n-1] = curr;
    }
}

return screen[N-1];
}

// Driver program
int main()
{
    int N;

    // for the rest of the array we will rely on the previous
    // entries to compute new ones
    for (N=1; N<=20; N++)
        printf("Maximum Number of A's with %d keystrokes is %d\n",
            N, findoptimal(N));
}
```

Java

```
/* A Dynamic Programming based C
program to find maximum number
of A's that can be printed using
four keys */
import java.io.*;

class GFG {

    // this function returns the optimal
    // length string for N keystrokes
    static int findoptimal(int N)
    {
        // The optimal string length is N
        // when N is smaller than 7
        if (N <= 6)
            return N;

        // An array to store result
        // of subproblems
        int screen[] = new int[N];

        int b; // To pick a breakpoint

        // Initializing the optimal lengths
        // array for upto 6 input strokes
        int n;
        for (n = 1; n <= 6; n++)
            screen[n - 1] = n;

        // Solve all subproblems in bottom manner
        for (n = 7; n <= N; n++)
        {
            // Initialize length of optimal
            // string for n keystrokes
            screen[n - 1] = 0;

            // For any keystroke n, we need
            // to loop from n-3 keystrokes
            // back to 1 keystroke to find
            // a breakpoint 'b' after which we
            // will have ctrl-a, ctrl-c and
            // then only ctrl-v all the way.
            for (b = n - 3; b >= 1; b--)
            {
                // if the breakpoint is
                // at b'th keystroke then
                // the optimal string would
                // have length
```

```
        // (n-b-1)*screen[b-1];
        int curr = (n - b - 1) * screen[b - 1];
        if (curr > screen[n - 1])
            screen[n - 1] = curr;
    }
}

return screen[N - 1];
}

// Driver program
public static void main(String [] args)
{
    int N;

    // for the rest of the array we will rely on the previous
    // entries to compute new ones
    for (N = 1; N <= 20; N++)
        System.out.println("Maximum Number of A's with keystrokes is "+
                           N + findoptimal(N));
}

}
```

// This article is contributed by vt_m.

Output:

```
Maximum Number of A's with 1 keystrokes is 1
Maximum Number of A's with 2 keystrokes is 2
Maximum Number of A's with 3 keystrokes is 3
Maximum Number of A's with 4 keystrokes is 4
Maximum Number of A's with 5 keystrokes is 5
Maximum Number of A's with 6 keystrokes is 6
Maximum Number of A's with 7 keystrokes is 9
Maximum Number of A's with 8 keystrokes is 12
Maximum Number of A's with 9 keystrokes is 16
Maximum Number of A's with 10 keystrokes is 20
Maximum Number of A's with 11 keystrokes is 27
Maximum Number of A's with 12 keystrokes is 36
Maximum Number of A's with 13 keystrokes is 48
Maximum Number of A's with 14 keystrokes is 64
Maximum Number of A's with 15 keystrokes is 81
Maximum Number of A's with 16 keystrokes is 108
Maximum Number of A's with 17 keystrokes is 144
Maximum Number of A's with 18 keystrokes is 192
Maximum Number of A's with 19 keystrokes is 256
```

Maximum Number of A's with 20 keystrokes is 324

Thanks to **Gaurav Saxena** for providing the above approach to solve this problem.

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/how-to-print-maximum-number-of-a-using-given-four-keys/>

Chapter 54

How will you print numbers from 1 to 100 without using loop?

How will you print numbers from 1 to 100 without using loop? - GeeksforGeeks

If we take a look at this problem carefully, we can see that the idea of “loop” is to track some counter value e.g. “i=0” till “i <= 100”. So if we aren’t allowed to use loop, how else can be track something in C language!

Well, one possibility is the use of ‘recursion’ provided we use the terminating condition carefully. Here is a solution that prints numbers using recursion.

C

```
#include <stdio.h>

// Prints numbers from 1 to n
void printNos(unsigned int n)
{
    if(n > 0)
    {
        printNos(n - 1);
        printf("%d ", n);
    }
    return;
}

// Driver code
```

```
int main()
{
    printNos(100);
    getchar();
    return 0;
}
```

Java

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

class GFG
{
    // Prints numbers from 1 to n
    static void printNos(int n)
    {
        if(n > 0)
        {
            printNos(n - 1);
            System.out.print(n + " ");
        }
        return;
    }

    // Driver Code
    public static void main(String[] args)
    {
        printNos(100);
    }
}

// This code is contributed by Manish_100
```

Python3

```
# Python3 program to Print
# numbers from 1 to n

def printNos(n):
    if n > 0:
        printNos(n - 1)
        print(n, end = ' ')
```

```
# Driver code
printNos(100)
```

This code is contributed by Smitha Dinesh Semwal

C#

```
// C# code for print numbers from
// 1 to 100 without using loop
using System;
```

```
class GFG
{
    // Prints numbers from 1 to n
    static void printNos(int n)
    {
        if(n > 0)
        {
            printNos(n - 1);
            Console.Write(n + " ");
        }
        return;
    }
}
```

```
// Driver Code
public static void Main()
{
    printNos(100);
}
}
```

// This code is contributed by Ajit

PHP

```
<?php
// PHP program print numbers
// from 1 to 100 without
// using loop

// Prints numbers from 1 to n
function printNos($n)
{
    if($n > 0)
    {
        printNos($n - 1);
    }
}
```

```
        echo $n, " ";
    }
    return;
}

// Driver code
printNos(100);

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

Output :

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51
52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99
100
```

Time Complexity : $O(n)$

Now try writing a program that does the same but without any “if” construct.

Hint — use some operator which can be used instead of “if”.

Please note that recursion technique is good but every call to the function creates one “stack-frame” in program stack. So if there’s constraint to the limited memory and we need to print large set of numbers, “recursion” might not be a good idea. So what could be the other alternative?

Another alternative is “goto” statement. Though use of “goto” is not suggestible as a general programming practice as “goto” statement changes the normal program execution sequence yet in some cases, use of “goto” is the best working solution.

So please give a try printing numbers from 1 to 100 with “goto” statement. You can use [GfG IDE!](#)

[Print 1 to 100 in C++, without loop and recursion](#)

Improved By : [Manish_100](#), [vt_m](#), [jit_t](#)

Source

<https://www.geeksforgeeks.org/how-will-you-print-numbers-from-1-to-200-without-using-loop/>

Chapter 55

How will you print numbers from 1 to 100 without using loop? | Set-2

How will you print numbers from 1 to 100 without using loop? | Set-2 - GeeksforGeeks

If we take a look at this problem carefully, we can see that the idea of “loop” is to track some counter value e.g. “i=0” till “i <= 100”. So if we aren’t allowed to use loop, how else can be track something in C language!

It can be done in many ways to print numbers using any looping conditions such as for(), while(), do while(). But the same can be done without using loops (using recursive functions, goto statement).

Printing numbers from 1 to 100 using recursive functions has already been discussed in [Set-1](#). In this post, other two methods have been discussed:

1. Using goto statement:

```
#include <stdio.h>

int main()
{
    int i = 0;
begin:
    i = i + 1;
    printf("%d ", i);

    if (i < 100)
        goto begin;
    return 0;
}
```

Output:

1 2 3 4 . . . 97 98 99 100

2. Using recursive main function:

```
#include <stdio.h>

int main()
{
    static int i = 1;
    if (i <= 100) {
        printf("%d ", i++);
        main();
    }
    return 0;
}
```

Output:

1 2 3 4 . . . 97 98 99 100

Source

<https://www.geeksforgeeks.org/program-print-upto-100-without-using-loops/>

Chapter 56

Identify all Grand-Parent Nodes of each Node in a Map

Identify all Grand-Parent Nodes of each Node in a Map - GeeksforGeeks

Given in input that has relationships between a person and their children, for all the people in this data, identify grandparents of all the people in the input.

Input: A map of all the people and their children
`Map[String, Set[String]]`

Output: A map of all people and their grandparents
`Map[String, Set[String]]`

Example:

Input
`Map(A -> Set(B,C), B -> Set(D, C), C -> Set(E))`

Output:
`Map(D -> Set(A), C -> Set(A), E -> Set(A, B))`

We strongly recommend you to minimize your browser and try this yourself first

Here we have iterate over each and every node in Map and find out the grand-parent of each child.

The idea is to use Recursion. But, it can be solved without Recursion too.

Lets see the Without Recursion solution first:

Solution 1 (Without Recursion):

Here we have to use Mutable Scala Map. Below is the Scala Code.

```
val input = Map("A" -> Set("B","C"), "B" -> Set("D", "C"), "C" -> Set("E"))
val output: scala.collection.mutable.Map[String, Set[String]]
  = scala.collection.mutable.Map()

input.map(node => node._2.map(child =>
  input.get(child).map(grandchildren =>
    grandchildren.map{grandchild =>
      if(output.keys.exists(_ == grandchild)) {
        output.put(grandchild, output.get(grandchild).get ++ Set(node._1))
      } else {
        output.put(grandchild, Set(node._1))
      }
    }
  )
})
```

Here we are iterating over every Node of Map and finding out the grandchildren of each child in node. This, will help us in creating a Map with Grand-Child and Grand-Parents relationship.

Solution 2 (With Recursion):

Here we can use Immutable Scala Map as we are using Recursion.

```
val input = Map("A" -> Set("B","C"), "B" -> Set("D", "C"), "C" -> Set("E"))
val output = findGrandparents(input)

def findGrandparents(family: Map[String, Set[String]]): Map[String, Set[String]] = {
  family.foldLeft(Map[String, Set[String]]()){
    case (grandParents, oneFamily) => {
      val grandchildren: Set[String] = oneFamily._2.flatMap(member => family.get(member)).flatMap
      val res = grandchildren.map(child => {
        grandParents.get(child) match {
          case None => (child -> Set(oneFamily._1))
          case Some(x) => (child -> (x + oneFamily._1))
        }
      }).toMap
      grandParents ++ res
    }
  }
}
```

This article is contributed by **Himanshu 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/identify-all-grand-parent-nodes-of-each-node-in-a-map/>

Chapter 57

Java 8 | Arrays parallelSort() method with Examples

Java 8 | Arrays parallelSort() method with Examples - GeeksforGeeks

Java 8 introduced a new method called as **parallelSort()** in **java.util.Arrays** Class. It uses Parallel Sorting of array elements

Algorithm of parallelSort()

1. The array is divided into sub-arrays and that sub-arrays is again divided into their sub-arrays, until the minimum level of detail in a set of array.
2. Arrays are sorted individually by multiple thread.
3. The parallel sort uses Fork/Join Concept for sorting.
4. Sorted sub-arrays are then merged.

Syntax :

1. For sorting data in ascending order :

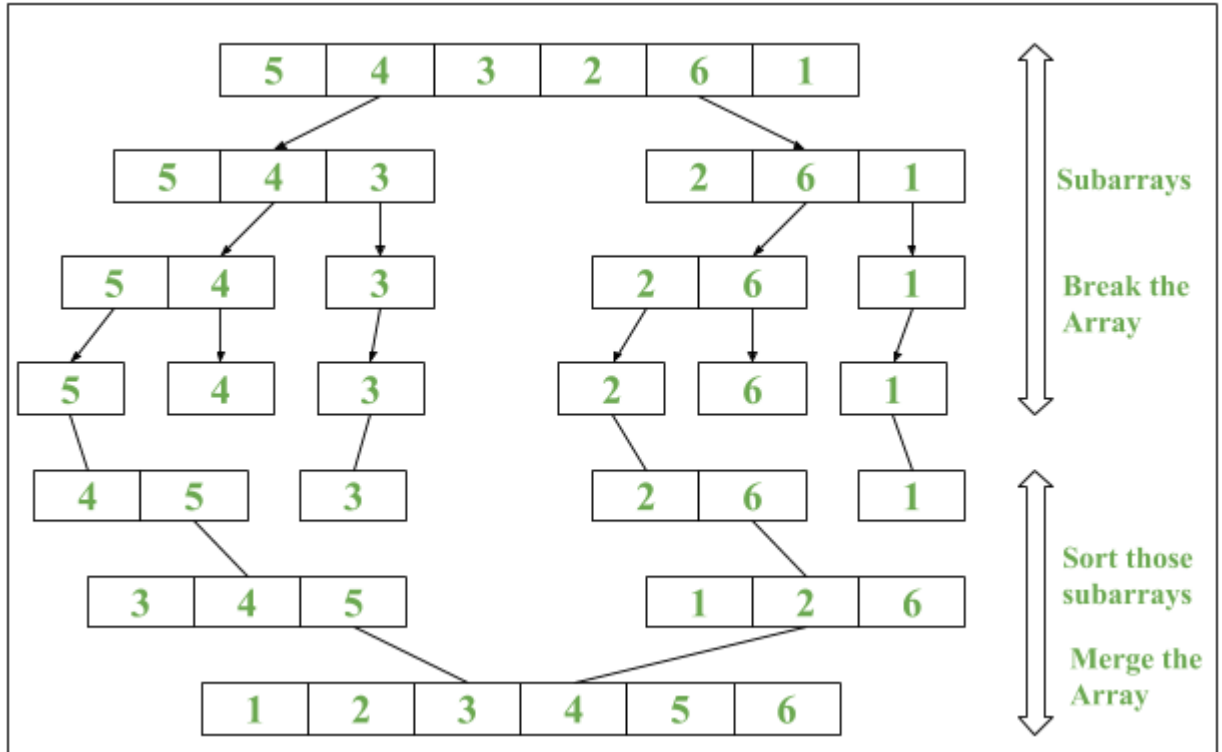
```
public static void parallelSort(Object obj[])
```

2. For sorting data in specified range in ascending order :

```
public static void parallelSort(Object obj[], int from, int to)
```

Advantage :

parallelSort() method uses concept of **MultiThreading** which makes the sorting **faster** as compared to normal sorting method.

Example

Below are the program that will illustrate the use of `Arrays.parallelSort()`:

Program 1: To demonstrate use of Parallel Sort

```
// Java program to demonstrate
// Arrays.parallelSort() method

import java.util.Arrays;

public class ParallelSort {
    public static void main(String[] args)
    {
        // Creating an array
        int numbers[] = { 9, 8, 7, 6, 3, 1 };

        // Printing unsorted Array
        System.out.print("Unsorted Array: ");
        // Iterating the Elements using stream
        Arrays.stream(numbers)
            .forEach(n -> System.out.print(n + " "));
        System.out.println();
    }
}
```

```
// Using Arrays.parallelSort()
Arrays.parallelSort(numbers);

// Printing sorted Array
System.out.print("Sorted Array: ");
// Iterating the Elements using stream
Arrays.stream(numbers)
    .forEach(n -> System.out.print(n + " "));
}
}
```

Output:

Unsorted Array: 9 8 7 6 3 1

Sorted Array: 1 3 6 7 8 9

Time Complexity is $O(n \log n)$

Program 2: To demonstrate use of Parallel Sort w.r.t. Series Sort (Normal Sort)

```
// Java program to demonstrate impact
// of Parallel Sort vs Serial Sort

import java.util.Arrays;
import java.util.Random;

public class ParallelSort {
    public static void main(String[] args)
    {
        // Creating an array
        int numbers[] = new int[100];

        // Iterating Loop till i = 1000
        // with interval of 10
        for (int i = 0; i < 1000; i += 10) {

            System.out.println("\nFor iteration number: "
                               + (i / 10 + 1));

            // Random Int Array Generation
            Random rand = new Random();

            for (int j = 0; j < 100; j++) {
                numbers[j] = rand.nextInt();
            }

            // Start and End Time of Arrays.sort()
        }
    }
}
```

```
        long startTime = System.nanoTime();

        // Performing Serial Sort
        Arrays.sort(numbers);

        long endTime = System.nanoTime();

        // Printing result of Serial Sort
        System.out.println("Start and End Time in Serial (in ns): "
            + startTime + ":" + endTime);
        System.out.println("Time taken by Serial Sort(in ns): "
            + (endTime - startTime));

        // Start and End Time of Arrays.parallelSort()
        startTime = System.nanoTime();

        // Performing Parallel Sort
        Arrays.parallelSort(numbers);

        endTime = System.nanoTime();

        // Printing result of Parallel Sort
        System.out.println("Start and End Time in parallel (in ns): "
            + startTime + ":" + endTime);
        System.out.println("Time taken by Parallel Sort(in ns): "
            + (endTime - startTime));
        System.out.println();
    }
}
}
```

Output:

```
For iteration number: 1
Start and End Time in Serial (in ns): 3951000637977:3951000870361
Time taken by Serial Sort(in ns): 232384
Start and End Time in parallel (in ns): 3951000960823:3951000971044
Time taken by Parallel Sort(in ns): 10221
```

```
For iteration number: 2
Start and End Time in Serial (in ns): 3951001142284:3951001201757
Time taken by Serial Sort(in ns): 59473
Start and End Time in parallel (in ns): 3951001256643:3951001264039
Time taken by Parallel Sort(in ns): 7396
.
.
```



```
.  
For iteration number: 99  
Start and End Time in Serial (in ns): 3951050723541:3951050731520  
Time taken by Serial Sort(in ns): 7979  
Start and End Time in parallel (in ns): 3951050754238:3951050756130  
Time taken by Parallel Sort(in ns): 1892
```

```
For iteration number: 100  
Start and End Time in Serial (in ns): 3951050798392:3951050804741  
Time taken by Serial Sort(in ns): 6349  
Start and End Time in parallel (in ns): 3951050828544:3951050830582  
Time taken by Parallel Sort(in ns): 2038
```

Note : Different time intervals will be printed But parallel sort will be done before normal sort.

Environment: 2.6 GHz Intel Core i7, java version 8

Source

<https://www.geeksforgeeks.org/java-8-arrays-parallel-sort-method-with-examples/>

Chapter 58

Java Applet | Implementing Flood Fill algorithm

Java Applet | Implementing Flood Fill algorithm - GeeksforGeeks

Flood Fill Algorithm is to replace a certain closed or a similarly coloured field with a specified color. The use of the FloodFill algorithm can be seen in paints and other games such as minesweeper.

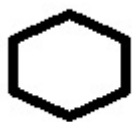
In this article, FloodFill is used for a connected area by a specified colour, in Java Applet by using the FloodFill algorithm.

There are two approaches that can be used:

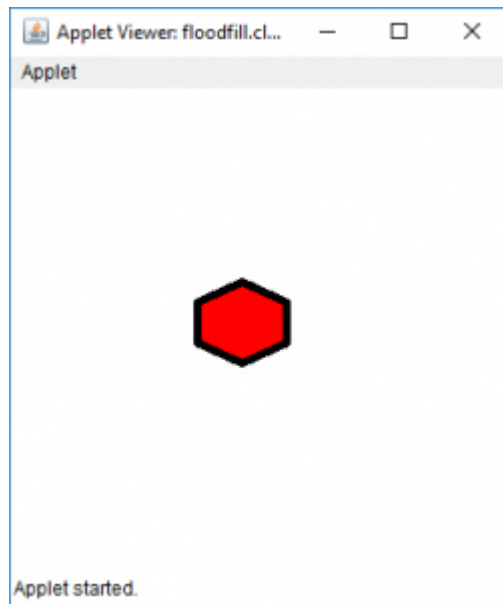
1. Recursive approach (limited usage as it crashes for a larger area)
2. Using queue (more reliable)

Examples:

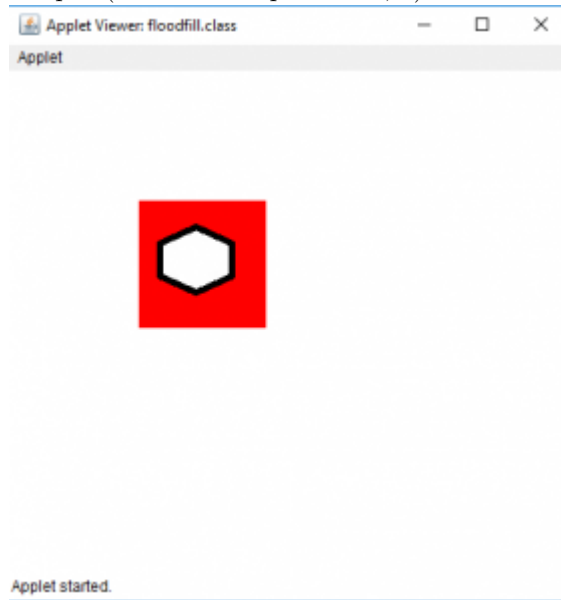
1. For image 1:



- Output (floodfilled at position 35, 35):



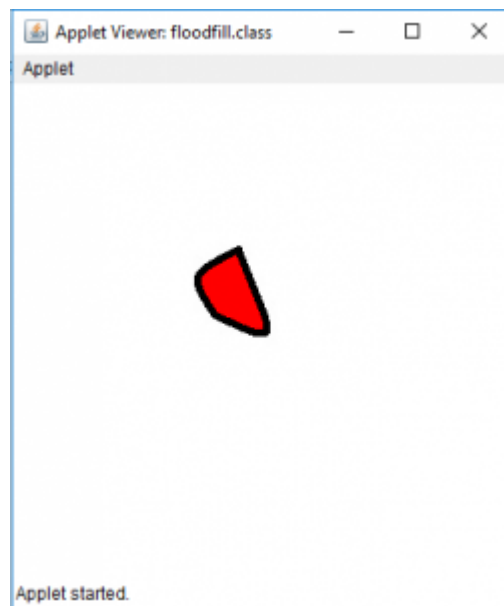
- Output (floodfilled at position 1, 1):



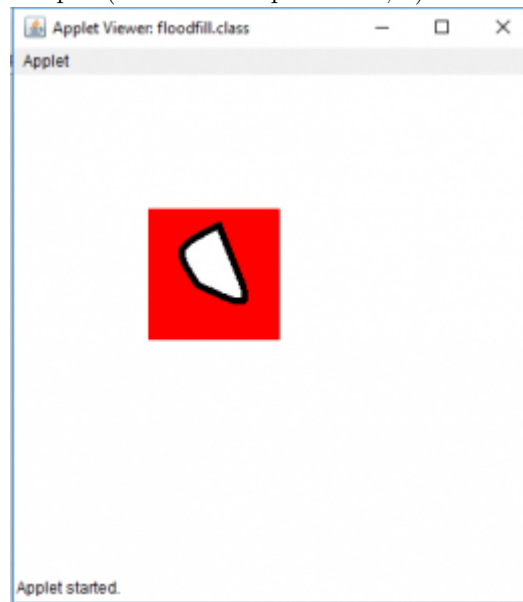
2. For image 2:



- Output(floodfilled at position 35, 35) :



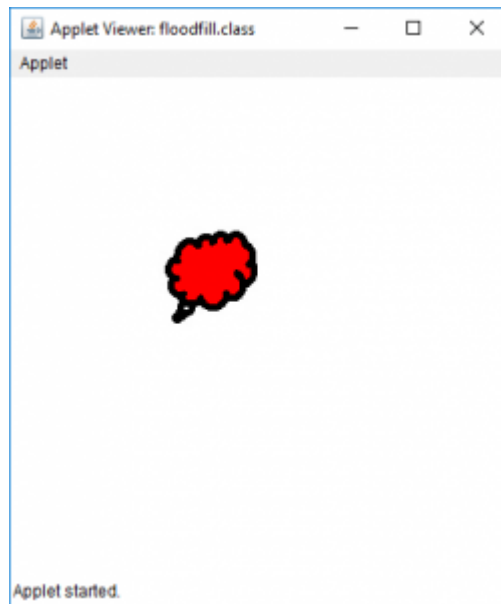
- Output (floodfilled at position 1, 1):



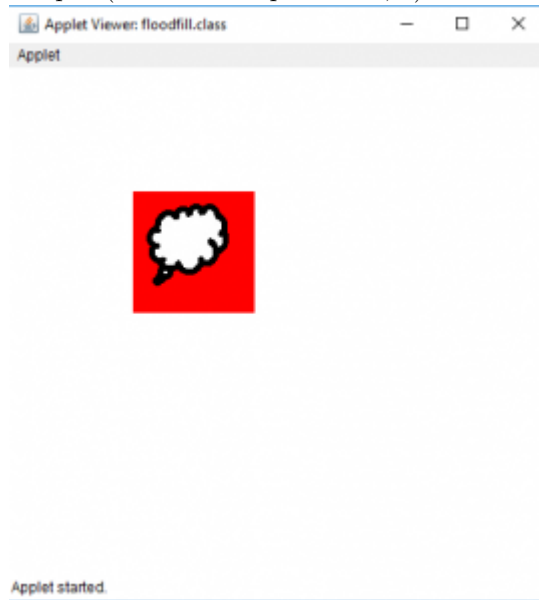
3. For image 3:



- Output(floodfilled at position 35, 35) :



- Output (floodfilled at position 1, 1):



Program 1: To implement floodfill algorithm in Java Applet using recursion:

Note: To run the program, use an offline IDE such as [Netbeans](#), Eclipse, etc. Please download the input images and put them along with the class file. Otherwise, the program might yield an “Can’t read the input file” error.

```
// Java Program to implement floodfill algorithm  
// in Java Applet(using recursion)
```

```
import java.awt.*;
import javax.swing.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.ImageIO;

public class floodfill extends JApplet {

    public void init()
    {
    }

    // paint function
    public void paint(Graphics g)
    {
        BufferedImage i = null;
        try {
            // Input the image to be used for FloodFill
            // The output is shown for 3 images
            // image1, image2 and image2
            i = ImageIO.read(new File("image1.jpg"));

            // floodfill with color red at point 35, 35
            // get color of image at 35, 35
            Color c = new Color(i.getRGB(35, 35));
            flood(i, g, 35, 35, c, Color.red);

            // draw the image after floodfill
            g.drawImage(i, 100, 100, this);
        }
        catch (Exception e) {
            JOptionPane.showMessageDialog(this, e.getMessage());
        }

        // draw the image after floodfill
        g.drawImage(i, 100, 100, this);
    }

    // function to floodfill the image
    public void flood(BufferedImage i,
                     Graphics g,
                     int x,
                     int y,
                     Color c,
                     Color c1)
    {
        if (x >= 1 && y >= 1
            && x < i.getWidth()
```

```
&& y < i.getHeight()) {  
    // find the color at point x, y  
    Color c2 = new Color(i.getRGB(x, y));  
  
    // if there is no boundary (the color is almost  
    // same as the color of the point where  
    // floodfill is to be applied  
    if (Math.abs(c2.getGreen() - c.getGreen()) < 30  
        && Math.abs(c2.getRed() - c.getRed()) < 30  
        && Math.abs(c2.getBlue() - c.getBlue()) < 30) {  
        // change the color of the pixel of image  
        i.setRGB(x, y, c1.getRGB());  
  
        g.drawImage(i, 100, 100, this);  
  
        // floodfill in all possible directions  
        flood(i, g, x, y + 1, c, c1);  
        flood(i, g, x + 1, y, c, c1);  
        flood(i, g, x - 1, y, c, c1);  
        flood(i, g, x, y - 1, c, c1);  
    }  
}  
}
```

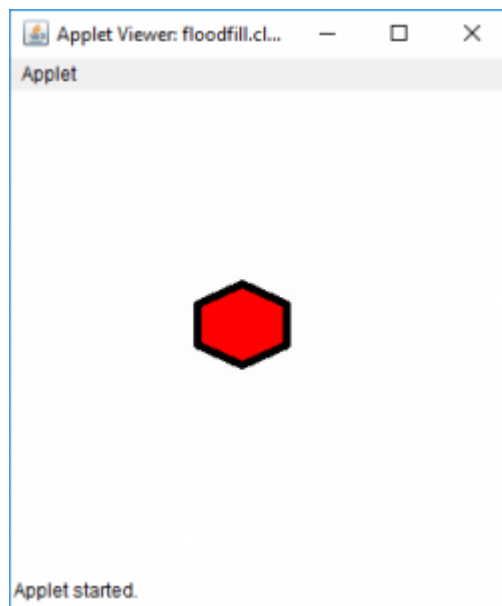
Output:

- for image 1:

Input:



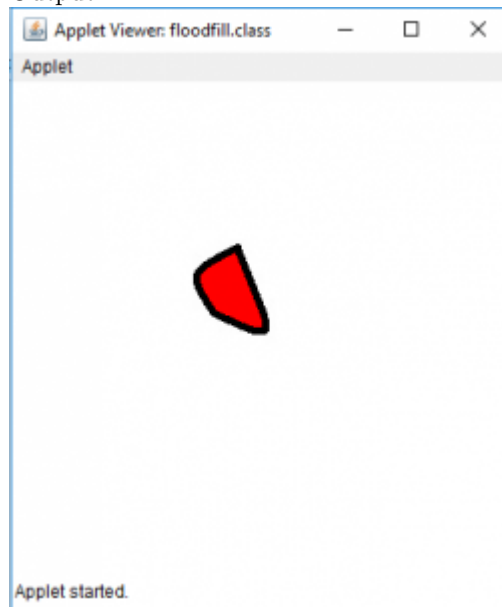
Output :



- for image 2:
Input:



Output :



- for image 3:

Input:



Output :



Note: If a larger area is flood filled (at coordinate 1, 1) using the recursive approach, then recursive algorithm might get **crashed**.

Example:

floodfill the larger side of image

Input:

Output:

Explanation:

Since the area to be covered is very large,
therefore only some part is covered by the algorithm,
and after that the program gets crashed.

Program 2: To implement floodfill algorithm in Java Applet using queue:

Note: To run the program, use an offline IDE such as Netbeans, Eclipse, etc. Please download the input images and put them along with the class file. Otherwise, the program might yield an “Can’t read the input file” error.

```
// Java Program to implement floodfill algorithm
// in Java Applet(using queue)
import java.awt.*;
import javax.swing.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.ImageIO;

public class floodfill extends JApplet {

    public void init()
    {
    }

    // paint function
    public void paint(Graphics g)
    {
        BufferedImage i = null;
        try {
            // Input the image to be used for FloodFill
            // The output is shown for 3 images
            // image1, image2 and image2
            i = ImageIO.read(new File("image1.jpg"));

            // floodfill with color red at point 1, 1
            // get color of image at 1, 1
            // if 35, 35 point is floodfilled it will floodfill
            // the smaller area
            Color c = new Color(i.getRGB(1, 1));
            flood(i, g, 1, 1, c, Color.red);

            // draw the image after floodfill
            g.drawImage(i, 100, 100, this);
        }
        catch (Exception e) {
            JOptionPane.showMessageDialog(this, e.getMessage());
        }

        // draw the image after floodfill
        g.drawImage(i, 100, 100, this);
    }

    // function to floodfill the image using queue
    public void flood(BufferedImage i,
                     Graphics g,
                     int x1,
                     int y1,
                     Color c,
```

```
        Color c1)
{
    // create a stack using array
    int stx[] = new int[100000];
    int sty[] = new int[100000], f, r, x, y;

    // create a front and rear
    f = r = 0;

    // initilize them
    stx[0] = x1;
    sty[0] = y1;

    // while front is greater than rear
    while (f >= r) {
        // pop element out
        x = stx[r];
        y = sty[r++];
        if (x >= 1 && y >= 1
            && x < i.getWidth()
            && y < i.getHeight()) {
            // find the color at point x, y
            Color c2 = new Color(i.getRGB(x, y));

            // if there is no boundary (the color is almost
            // same as the color of the point where
            // floodfill is to be applied

            if (Math.abs(c2.getGreen() - c.getGreen()) < 30
                && Math.abs(c2.getRed() - c.getRed()) < 30
                && Math.abs(c2.getBlue() - c.getBlue()) < 30) {

                // change the color of the pixel of image
                i.setRGB(x, y, c1.getRGB());

                g.drawImage(i, 100, 100, this);

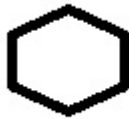
                // floodfill in all possible directions
                // store them in queue
                stx[f] = x;
                sty[f++] = y + 1;
                stx[f] = x;
                sty[f++] = y - 1;
                stx[f] = x + 1;
                sty[f++] = y;
                stx[f] = x - 1;
                sty[f++] = y;
            }
        }
    }
}
```

```
        }  
    }  
}
```

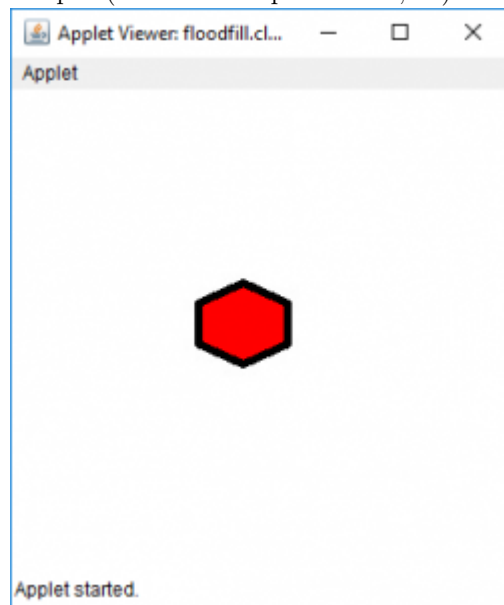
Output:

– **For image 1:**

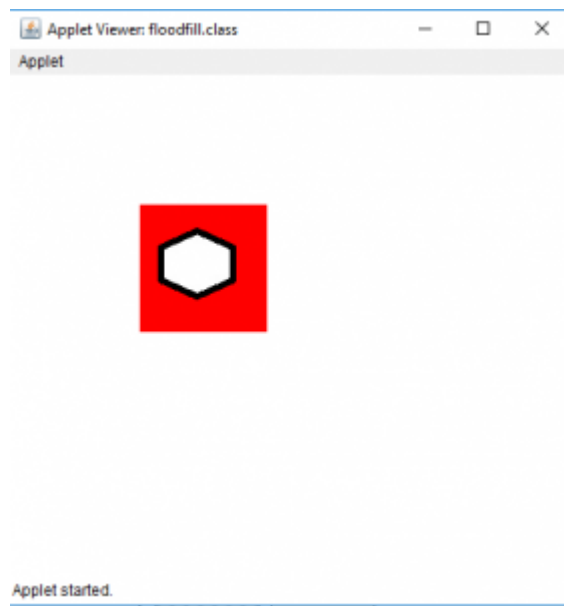
Input:



Output (floodfilled at position 35, 35):



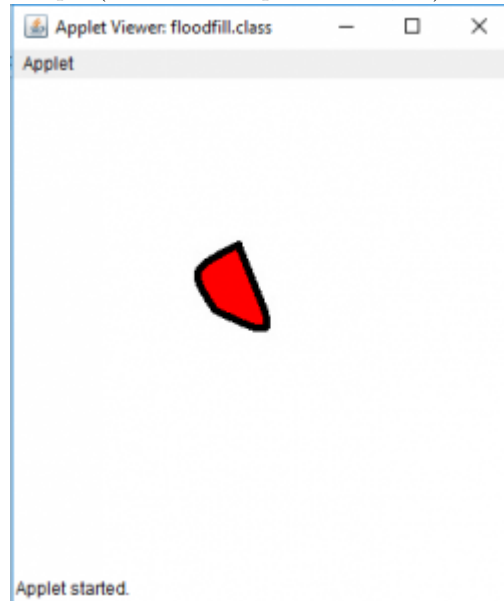
Output (floodfilled at position 1, 1):



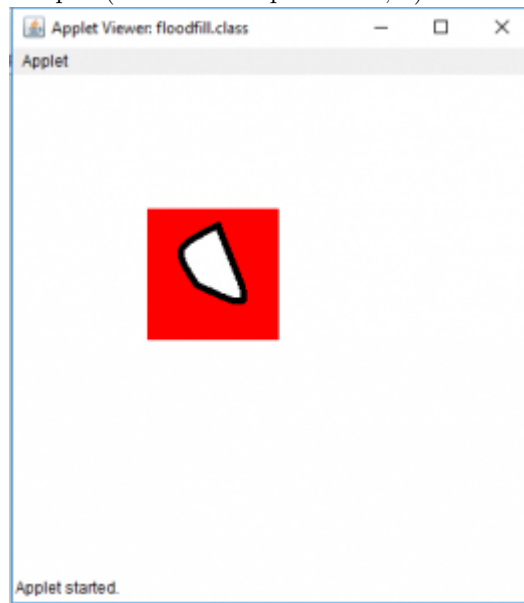
- For image 2:
Input:



Output(floodfilled at position 35, 35) :



Output (floodfilled at position 1, 1):

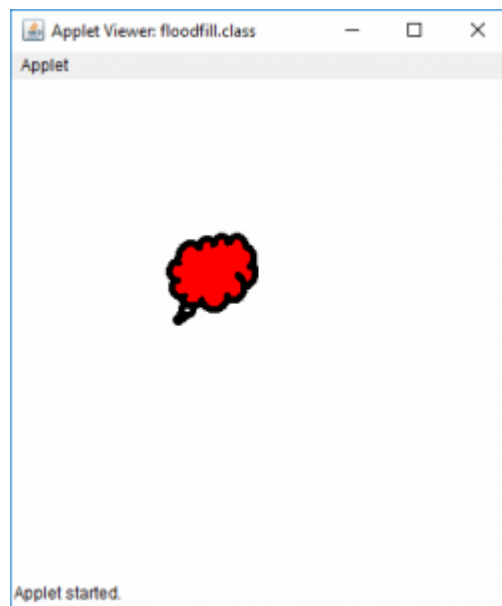


– For image 3:

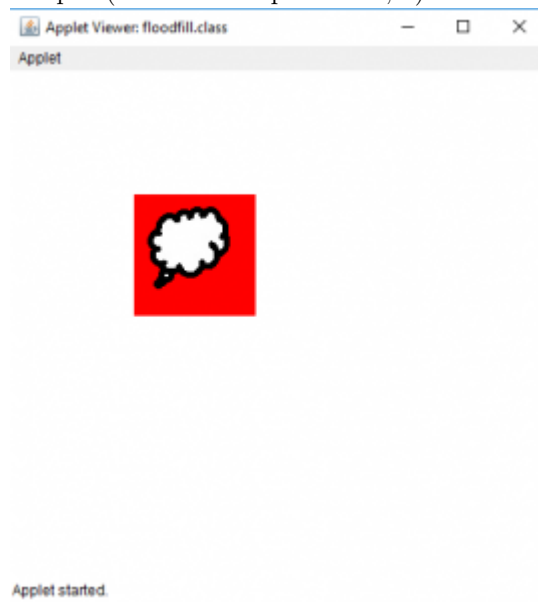
Input:



Output(floodfilled at position 35, 35) :



Output (floodfilled at position 1, 1):



Source

<https://www.geeksforgeeks.org/java-applet-implementing-flood-fill-algorithm/>

Chapter 59

Josephus problem | Set 1 (A $O(n)$ Solution)

Josephus problem | Set 1 (A $O(n)$ Solution) - GeeksforGeeks

In computer science and mathematics, the [Josephus Problem](#) (or [Josephus permutation](#)) is a theoretical problem. Following is the problem statement:

There are n people standing in a circle waiting to be executed. The counting out begins at some point in the circle and proceeds around the circle in a fixed direction. In each step, a certain number of people are skipped and the next person is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last person remains, who is given freedom. Given the total number of persons n and a number k which indicates that $k-1$ persons are skipped and k th person is killed in circle. The task is to choose the place in the initial circle so that you are the last one remaining and so survive.

For example, if $n = 5$ and $k = 2$, then the safe position is 3. Firstly, the person at position 2 is killed, then person at position 4 is killed, then person at position 1 is killed. Finally, the person at position 5 is killed. So the person at position 3 survives.

If $n = 7$ and $k = 3$, then the safe position is 4. The persons at positions 3, 6, 2, 7, 5, 1 are killed in order, and person at position 4 survives.

The problem has following recursive structure.

```
josephus(n, k) = (josephus(n - 1, k) + k-1) % n + 1
josephus(1, k) = 1
```

After the first person (k th from beginning) is killed, $n-1$ persons are left. So we call $josephus(n - 1, k)$ to get the position with $n-1$ persons. But the position returned by $josephus(n - 1, k)$ will consider the position starting from $k\%n + 1$. So, we must make adjustments to the position returned by $josephus(n - 1, k)$.

Following is simple recursive implementation of the Josephus problem. The implementation simply follows the recursive structure mentioned above.

C

```
#include <stdio.h>

int josephus(int n, int k)
{
    if (n == 1)
        return 1;
    else
        /* The position returned by josephus(n - 1, k) is adjusted because the
           recursive call josephus(n - 1, k) considers the original position
           k%n + 1 as position 1 */
        return (josephus(n - 1, k) + k - 1) % n + 1;
}

// Driver Program to test above function
int main()
{
    int n = 14;
    int k = 2;
    printf("The chosen place is %d", josephus(n, k));
    return 0;
}
```

Java

```
// Java code for Josephus Problem
import java.io.*;

class GFG {

    static int josephus(int n, int k)
    {
        if (n == 1)
            return 1;
        else
            /* The position returned by josephus(n - 1, k)
               is adjusted because the recursive call
               josephus(n - 1, k) considers the original
               position k%n + 1 as position 1 */
            return (josephus(n - 1, k) + k - 1) % n + 1;
    }

    // Driver Program to test above function
```

```
public static void main(String[] args)
{
    int n = 14;
    int k = 2;
    System.out.println("The chosen place is " + josephus(n, k));
}
}
```

// This code is contributed by Prerna Saini

Python3

```
# Python code for Josephus Problem

def josephus(n, k):

    if (n == 1):
        return 1
    else:

        # The position returned by
        # josephus(n - 1, k) is adjusted
        # because the recursive call
        # josephus(n - 1, k) considers
        # the original position
        # k%n + 1 as position 1
        return (josephus(n - 1, k) + k-1) % n + 1

# Driver Program to test above function

n = 14
k = 2

print("The chosen place is ", josephus(n, k))

# This code is contributed by
# Sumit Sadhakar
```

C#

```
// C# code for Josephus Problem
using System;

class GFG {

    static int josephus(int n, int k)
```

```
{
    if (n == 1)
        return 1;
    else
        /* The position returned
        by josephus(n - 1, k) is
        adjusted because the
        recursive call josephus(n
        - 1, k) considers the
        original position k%n + 1
        as position 1 */
        return (josephus(n - 1, k)
                + k-1) % n + 1;
}

// Driver Program to test above
// function
public static void Main()
{
    int n = 14;
    int k = 2;
    Console.WriteLine("The chosen "
        + "place is " + josephus(n, k));
}

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

PHP

```
<?php
// PHP code for
// Josephus Problem

function josephus($n, $k)
{
    if ($n == 1)
        return 1;
    else
        /* The position returned by
        josephus(n - 1, k) is
        adjusted because the
        recursive call josephus
        (n - 1, k) considers the
        original position k%n + 1
        as position 1 */
        return (josephus($n - 1, $k) +
                $k - 1) % $n + 1;
}
```

```
}  
  
    // Driver Code  
    $n = 14;  
    $k = 2;  
    echo "The chosen place is ", josephus($n, $k);  
  
// This code is contributed by ajit.  
?>
```

Output:

The chosen place is 13

Time Complexity: $O(n)$

[Josephus problem | Set 2 \(A Simple Solution when \$k = 2\$ \)](#)

Source:

http://en.wikipedia.org/wiki/Josephus_problem

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

Source

<https://www.geeksforgeeks.org/josephus-problem-set-1-a-on-solution/>

Chapter 60

Last non-zero digit of a factorial

Last non-zero digit of a factorial - GeeksforGeeks

Given a number n , find the last non-zero digit in $n!$.

Examples:

```
Input   : n = 5
Output  : 2
5! = 5 * 4 * 3 * 2 * 1 = 120
Last non-zero digit in 120 is 2.
```

```
Input   : n = 33
Output  : 4
```

A **Simple Solution** is to first find $n!$, then find last non-zero digit of n . This solution doesn't work for even slightly large numbers due to arithmetic overflow.

A **Better Solution** is based on below recursive formula

```
Let D(n) be the last non-zero digit in n!
If tens digit (or second last digit) of n is odd
    D(n) = 4 * D(floor(n/5)) * D(Unit digit of n)
If tens digit (or second last digit) of n is even
    D(n) = 6 * D(floor(n/5)) * D(Unit digit of n)
```

Illustration of formula:

For the numbers less than 10 we can easily find the last non-zero digit by above simple solution, i.e., first computing $n!$, then finding last digit.

$D(1) = 1$, $D(2) = 2$, $D(3) = 6$, $D(4) = 4$, $D(5) = 2$,
 $D(6) = 2$, $D(7) = 4$, $D(8) = 2$, $D(9) = 8$.

D(1) to D(9) are assumed to be precomputed.

Example 1: $n = 27$ [Second last digit is even]:

$$\begin{aligned} D(27) &= 6 * D(\text{floor}(27/5)) * D(7) \\ &= 6 * D(5) * D(7) \\ &= 6 * 2 * 4 \\ &= 48 \end{aligned}$$

Last non-zero digit is 8

Example 2: $n = 33$ [Second last digit is odd]:

$$\begin{aligned} D(33) &= 4 * D(\text{floor}(33/5)) * D(3) \\ &= 4 * D(6) * 6 \\ &= 4 * 2 * 6 \\ &= 48 \end{aligned}$$

Last non-zero digit is 8

How does above formula work?

The below explanation provides intuition behind the formula. Readers may refer <http://math.stackexchange.com/questions/130352/last-non-zero-digit-of-a-factorial> for complete proof.

$$14! = 14 * 13 * 12 * 11 * 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1$$

Since we are asked about last non-zero digit, we remove all 5's and equal number of 2's from factors of 14!. We get following:

$$14! = 14 * 13 * 12 * 11 * 2 * 9 * 8 * 7 * 6 * 3 * 2 * 1$$

Now we can get last non-zero digit by multiplying last digits of above factors!

In $n!$, number of 2's are always more than number of 5's. To remove trailing 0's, we remove 5's and equal number of 2's.

Let $a = \text{floor}(n/5)$, $b = n \% 5$. After removing equal number of 5's and 2's, we can reduce the problem from $n!$ to $2^a * a! * b!$

$$D(n) = 2^a * D(a) * D(b)$$

Implementation:

C++

```
// C++ program to find last non-zero digit in n!
#include<bits/stdc++.h>
using namespace std;

// Initialize values of last non-zero digit of
// numbers from 0 to 9
int dig[] = {1, 1, 2, 6, 4, 2, 2, 4, 2, 8};

int lastNonODigit(int n)
{
    if (n < 10)
        return dig[n];

    // Check whether tens (or second last) digit
    // is odd or even
    // If n = 375, So n/10 = 37 and (n/10)%10 = 7
    // Applying formula for even and odd cases.
    if (((n/10)%10)%2 == 0)
        return (6*lastNonODigit(n/5)*dig[n%10]) % 10;
    else
        return (4*lastNonODigit(n/5)*dig[n%10]) % 10;
}

// Driver code
int main()
{
    int n = 14;
    cout << lastNonODigit(n);
    return 0;
}
```

Java

```
// Java program to find last
// non-zero digit in n!

class GFG
{
    // Initialize values of last non-zero digit of
    // numbers from 0 to 9
    static int dig[] = {1, 1, 2, 6, 4, 2, 2, 4, 2, 8};

    static int lastNonODigit(int n)
    {
        if (n < 10)
            return dig[n];

        // Check whether tens (or second last)
```

```
// digit is odd or even
// If n = 375, So n/10 = 37 and
// (n/10)%10 = 7 Applying formula for
// even and odd cases.
if (((n / 10) % 10) % 2 == 0)
    return (6 * lastNonODigit(n / 5)
            * dig[n % 10]) % 10;
else
    return (4 * lastNonODigit(n / 5)
            * dig[n % 10]) % 10;
}

// Driver code
public static void main (String[] args)
{
    int n = 14;
    System.out.print(lastNonODigit(n));
}
// This code is contributed by Anant Agarwal.
```

Python3

```
# Python program to find
# last non-zero digit in n!

# Initialize values of
# last non-zero digit of
# numbers from 0 to 9
dig= [1, 1, 2, 6, 4, 2, 2, 4, 2, 8]

def lastNonODigit(n):
    if (n < 10):
        return dig[n]

    # Check whether tens (or second last) digit
    # is odd or even
    # If n = 375, So n/10 = 37 and (n/10)%10 = 7
    # Applying formula for even and odd cases.
    if (((n//10)%10)%2 == 0):
        return (6*lastNonODigit(n//5)*dig[n%10]) % 10
    else:
        return (4*lastNonODigit(n//5)*dig[n%10]) % 10
    return 0

# driver code
n = 14
```



```
print(lastNon0Digit(n))
```

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

C#

```
// C# program to find last  
// non-zero digit in n!  
using System;  
  
class GFG {  
  
    // Initialize values of last non-zero  
    // digit of numbers from 0 to 9  
    static int []dig = {1, 1, 2, 6, 4, 2, 2, 4, 2, 8};  
  
    static int lastNon0Digit(int n)  
    {  
        if (n < 10)  
            return dig[n];  
  
        // Check whether tens (or second  
        // last) digit is odd or even  
        // If n = 375, So n/10 = 37 and  
        // (n/10)%10 = 7 Applying formula  
        // for even and odd cases.  
        if (((n / 10) % 10) % 2 == 0)  
            return (6 * lastNon0Digit(n / 5) *  
                    dig[n % 10]) % 10;  
        else  
            return (4 * lastNon0Digit(n / 5) *  
                    dig[n % 10]) % 10;  
    }  
  
    // Driver code  
    public static void Main ()  
    {  
        int n = 14;  
        Console.WriteLine(lastNon0Digit(n));  
    }  
}  
  
// This code is contributed by Nitin Mittal.
```

PHP

```
<?php
```

```
// PHP program to find last
// non-zero digit in n!

// Initialize values of
// last non-zero digit of
// numbers from 0 to 9
$dig = array(1, 1, 2, 6, 4,
            2, 2, 4, 2, 8);

function lastNonODigit($n)
{
    global $dig;
    if ($n < 10)
        return $dig[$n];

    // Check whether tens(or second
    // last) digit is odd or even
    // If n = 375, So n/10 = 37 and
    // (n/10)%10 = 7
    // Applying formula for even
    // and odd cases.
    if (((($n / 10) % 10) % 2 == 0)
        return (6 * lastNonODigit($n / 5) *
                $dig[$n % 10]) % 10;
    else
        return (4 * lastNonODigit($n / 5) *
                $dig[$n % 10]) % 10;
}

// Driver code
$n = 14;
echo(lastNonODigit($n));

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

Output:

2

Improved By : [nitin mittal](#), [jit_t](#)

Source

<https://www.geeksforgeeks.org/last-non-zero-digit-factorial/>

Chapter 61

Leaf nodes from Preorder of a Binary Search Tree (Using Recursion)

Leaf nodes from Preorder of a Binary Search Tree (Using Recursion) - GeeksforGeeks

Given Preorder traversal of a Binary Search Tree. Then the task is print leaf nodes of the Binary Search Tree from the given preorder.

Examples :

Input : preorder[] = {890, 325, 290, 530, 965};
Output : 290 530 965

Tree represented is,

```
      890
     /  \
    325   965
   /  \
  290  530
```

Input : preorder[] = { 3, 2, 4 };
Output : 2 4

In this post, a simple recursive solution is discussed. The idea is to use two min and max variables and taking i (index in input array), the index for given preorder array, and recursively creating root node and correspondingly checking if left and right are existing or not. This method return boolean variable, and if both left and right are false it simply means that left and right are null hence it must be a leaf node so print it right there and return back true as root at that index existed.

C++

```
// Recursive C++ program to find leaf
// nodes from given preorder traversal
#include<bits/stdc++.h>
using namespace std;

// Print the leaf node from
// the given preorder of BST.
bool isLeaf(int pre[], int &i, int n,
            int min, int max)
{
    if (i >= n)
        return false;

    if (pre[i] > min && pre[i] < max) {
        i++;

        bool left = isLeaf(pre, i, n, min, pre[i-1]);
        bool right = isLeaf(pre, i, n, pre[i-1], max);

        if (!left && !right)
            cout << pre[i-1] << " ";

        return true;
    }
    return false;
}

void printLeaves(int preorder[], int n)
{
    int i = 0;
    isLeaf(preorder, i, n, INT_MIN, INT_MAX);
}

// Driver code
int main()
{
    int preorder[] = { 890, 325, 290, 530, 965 };
    int n = sizeof(preorder)/sizeof(preorder[0]);
    printLeaves(preorder, n);
    return 0;
}
```

PHP

```
<?php
```

```
// Recursive PHP program to
// find leaf nodes from given
// preorder traversal

// Print the leaf node from
// the given preorder of BST.

function isLeaf($pre, &$i, $n,
               $min, $max)
{
    if ($i >= $n)
        return false;

    if ($pre[$i] > $min &&
        $pre[$i] < $max)
    {
        $i++;

        $left = isLeaf($pre, $i, $n,
                       $min, $pre[$i - 1]);
        $right = isLeaf($pre, $i, $n,
                        $pre[$i - 1], $max);

        if (!$left && !$right)
            echo $pre[$i - 1] , " ";

        return true;
    }
    return false;
}

function printLeaves($preorder, $n)
{
    $i = 0;
    isLeaf($preorder, $i, $n,
           PHP_INT_MIN, PHP_INT_MAX);
}

// Driver code
$preorder = array (890, 325, 290,
                   530, 965 );
$n = sizeof($preorder);
printLeaves($preorder, $n);

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

Output :

290 530 965

Improved By : [jit_t](#)

Source

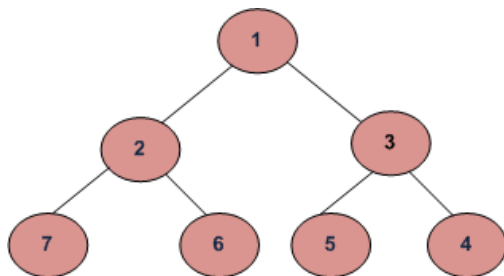
<https://www.geeksforgeeks.org/leaf-nodes-preorder-binary-search-treeusing-recursion/>

Chapter 62

Level order traversal in spiral form

Level order traversal in spiral form - GeeksforGeeks

Write a function to print spiral order traversal of a tree. For below tree, function should print 1, 2, 3, 4, 5, 6, 7.



Method 1 (Recursive)

This problem can be seen as an extension of the [level order traversal](#) post.

To print the nodes in spiral order, nodes at different levels should be printed in alternating order. An additional Boolean variable *ltr* is used to change printing order of levels. If *ltr* is 1 then `printGivenLevel()` prints nodes from left to right else from right to left. Value of *ltr* is flipped in each iteration to change the order.

Function to print level order traversal of tree

```
printSpiral(tree)
    bool ltr = 0;
    for d = 1 to height(tree)
        printGivenLevel(tree, d, ltr);
        ltr ~= ltr /*flip ltr*/
```

Function to print all nodes at a given level

```
printGivenLevel(tree, level, ltr)
if tree is NULL then return;
if level is 1, then
    print(tree->data);
else if level greater than 1, then
    if(ltr)
        printGivenLevel(tree->left, level-1, ltr);
        printGivenLevel(tree->right, level-1, ltr);
    else
        printGivenLevel(tree->right, level-1, ltr);
        printGivenLevel(tree->left, level-1, ltr);
```

Following is C implementation of above algorithm.

C

```
// C program for recursive level order traversal in spiral form
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

/* Function prototypes */
void printGivenLevel(struct node* root, int level, int ltr);
int height(struct node* node);
struct node* newNode(int data);

/* Function to print spiral traversal of a tree*/
void printSpiral(struct node* root)
{
    int h = height(root);
    int i;

    /*ltr -> Left to Right. If this variable is set,
       then the given level is traversed from left to right. */
    bool ltr = false;
    for(i=1; i<=h; i++)
    {
```



```
    printGivenLevel(root, i, ltr);

    /*Revert ltr to traverse next level in opposite order*/
    ltr = !ltr;
}
}

/* Print nodes at a given level */
void printGivenLevel(struct node* root, int level, int ltr)
{
    if(root == NULL)
        return;
    if(level == 1)
        printf("%d ", root->data);
    else if (level > 1)
    {
        if(ltr)
        {
            printGivenLevel(root->left, level-1, ltr);
            printGivenLevel(root->right, level-1, ltr);
        }
        else
        {
            printGivenLevel(root->right, level-1, ltr);
            printGivenLevel(root->left, level-1, ltr);
        }
    }
}

/* Compute the "height" of a tree -- the number of
   nodes along the longest path from the root node
   down to the farthest leaf node.*/
int height(struct node* node)
{
    if (node==NULL)
        return 0;
    else
    {
        /* compute the height of each subtree */
        int lheight = height(node->left);
        int rheight = height(node->right);

        /* use the larger one */
        if (lheight > rheight)
            return(lheight+1);
        else return(rheight+1);
    }
}
```

```
/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node = (struct node*)
        malloc(sizeof(struct node));

    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return(node);
}

/* Driver program to test above functions*/
int main()
{
    struct node *root = newNode(1);
    root->left      = newNode(2);
    root->right     = newNode(3);
    root->left->left = newNode(7);
    root->left->right = newNode(6);
    root->right->left = newNode(5);
    root->right->right = newNode(4);
    printf("Spiral Order traversal of binary tree is \n");
    printSpiral(root);

    return 0;
}
```

Java

```
// Java program for recursive level order traversal in spiral form

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
class Node
{
    int data;
    Node left, right;

    public Node(int d)
    {
        data = d;
        left = right = null;
    }
}
```

```
class BinaryTree
{
    Node root;

    // Function to print the spiral traversal of tree
    void printSpiral(Node node)
    {
        int h = height(node);
        int i;

        /* ltr -> left to right. If this variable is set then the
           given label is transversed from left to right */
        boolean ltr = false;
        for (i = 1; i <= h; i++)
        {
            printGivenLevel(node, i, ltr);

            /*Revert ltr to traverse next level in opposite order*/
            ltr = !ltr;
        }
    }

    /* Compute the "height" of a tree -- the number of
       nodes along the longest path from the root node
       down to the farthest leaf node.*/
    int height(Node node)
    {
        if (node == null)
            return 0;
        else
        {
            /* compute the height of each subtree */
            int lheight = height(node.left);
            int rheight = height(node.right);

            /* use the larger one */
            if (lheight > rheight)
                return (lheight + 1);
            else
                return (rheight + 1);
        }
    }

    /* Print nodes at a given level */
    void printGivenLevel(Node node, int level, boolean ltr)
    {
```

```
        if (node == null)
            return;
        if (level == 1)
            System.out.print(node.data + " ");
        else if (level > 1)
        {
            if (ltr != false)
            {
                printGivenLevel(node.left, level - 1, ltr);
                printGivenLevel(node.right, level - 1, ltr);
            }
            else
            {
                printGivenLevel(node.right, level - 1, ltr);
                printGivenLevel(node.left, level - 1, ltr);
            }
        }
    }
}
/* Driver program to test the above functions */
public static void main(String[] args)
{
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(7);
    tree.root.left.right = new Node(6);
    tree.root.right.left = new Node(5);
    tree.root.right.right = new Node(4);
    System.out.println("Spiral order traversal of Binary Tree is ");
    tree.printSpiral(tree.root);
}

// This code has been contributed by Mayank Jaiswal(mayank_24)
```

Output:

```
Spiral Order traversal of binary tree is
1 2 3 4 5 6 7
```

Time Complexity: Worst case time complexity of the above method is $O(n^2)$. Worst case occurs in case of skewed trees.

Method 2 (Iterative)

We can print spiral order traversal in $O(n)$ time and $O(n)$ extra space. The idea is to use two stacks. We can use one stack for printing from left to right and other stack for printing from right to left. In every iteration, we have nodes of one level in one of the stacks. We

print the nodes, and push nodes of next level in other stack.

C++

```
// C++ implementation of a O(n) time method for spiral order traversal
#include <iostream>
#include <stack>
using namespace std;

// Binary Tree node
struct node
{
    int data;
    struct node *left, *right;
};

void printSpiral(struct node *root)
{
    if (root == NULL) return;    // NULL check

    // Create two stacks to store alternate levels
    stack<struct node*> s1; // For levels to be printed from right to left
    stack<struct node*> s2; // For levels to be printed from left to right

    // Push first level to first stack 's1'
    s1.push(root);

    // Keep printing while any of the stacks has some nodes
    while (!s1.empty() || !s2.empty())
    {
        // Print nodes of current level from s1 and push nodes of
        // next level to s2
        while (!s1.empty())
        {
            struct node *temp = s1.top();
            s1.pop();
            cout << temp->data << " ";

            // Note that is right is pushed before left
            if (temp->right)
                s2.push(temp->right);
            if (temp->left)
                s2.push(temp->left);
        }

        // Print nodes of current level from s2 and push nodes of
        // next level to s1
        while (!s2.empty())
```

```
{
    struct node *temp = s2.top();
    s2.pop();
    cout << temp->data << " ";

    // Note that is left is pushed before right
    if (temp->left)
        s1.push(temp->left);
    if (temp->right)
        s1.push(temp->right);
}
}

// A utility function to create a new node
struct node* newNode(int data)
{
    struct node* node = new struct node;
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return(node);
}

int main()
{
    struct node *root = newNode(1);
    root->left      = newNode(2);
    root->right     = newNode(3);
    root->left->left = newNode(7);
    root->left->right = newNode(6);
    root->right->left = newNode(5);
    root->right->right = newNode(4);
    cout << "Spiral Order traversal of binary tree is \n";
    printSpiral(root);

    return 0;
}
```

Java

```
// Java implementation of an O(n) approach of level order
// traversal in spiral form

import java.util.*;

// A Binary Tree node
```

```
class Node
{
    int data;
    Node left, right;

    public Node(int item)
    {
        data = item;
        left = right = null;
    }
}

class BinaryTree
{
    static Node root;

    void printSpiral(Node node)
    {
        if (node == null)
            return;    // NULL check

        // Create two stacks to store alternate levels
        Stack<Node> s1 = new Stack<Node>(); // For levels to be printed from right to left
        Stack<Node> s2 = new Stack<Node>(); // For levels to be printed from left to right

        // Push first level to first stack 's1'
        s1.push(node);

        // Keep printing while any of the stacks has some nodes
        while (!s1.empty() || !s2.empty())
        {
            // Print nodes of current level from s1 and push nodes of
            // next level to s2
            while (!s1.empty())
            {
                Node temp = s1.peek();
                s1.pop();
                System.out.print(temp.data + " ");

                // Note that right is pushed before left
                if (temp.right != null)
                    s2.push(temp.right);

                if (temp.left != null)
                    s2.push(temp.left);
            }
        }
    }
}
```

```
// Print nodes of current level from s2 and push nodes of
// next level to s1
while (!s2.empty())
{
    Node temp = s2.peek();
    s2.pop();
    System.out.print(temp.data + " ");

    // Note that is left is pushed before right
    if (temp.left != null)
        s1.push(temp.left);
    if (temp.right != null)
        s1.push(temp.right);
}
}

public static void main(String[] args)
{
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(7);
    tree.root.left.right = new Node(6);
    tree.root.right.left = new Node(5);
    tree.root.right.right = new Node(4);
    System.out.println("Spiral Order traversal of Binary Tree is ");
    tree.printSpiral(root);
}

// This code has been contributed by Mayank Jaiswal(mayank_24)
```

Output:

```
Spiral Order traversal of binary tree is
1 2 3 4 5 6 7
```

Please write comments if you find any bug in the above program/algorithm; or if you want to share more information about spiral traversal.

Source

<https://www.geeksforgeeks.org/level-order-traversal-in-spiral-form/>

Chapter 63

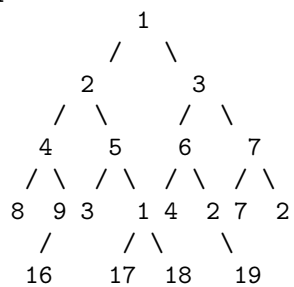
Level order traversal with direction change after every two levels | Recursive Approach

Level order traversal with direction change after every two levels | Recursive Approach - GeeksforGeeks

Given a binary tree, print the level order traversal in such a way that first two levels are printed from left to right, next two levels are printed from right to left, then next two from left to right and so on. So, the problem is to reverse the direction of level order traversal of the binary tree after every two levels.

Examples:

Input :



Output :

```
1
2 3
7 6 5 4
2 7 2 4 1 3 9 8
16 17 18 19
```

In the above example, the first two levels are printed from left to right, next two levels are printed from right to left, and then the last level is printed from left to right.

Approach: In the previous [post](#), level order traversal using queue and stack has been done to print the elements. A **recursive** method has been used over here to print the elements in every level. Traverse every level in the tree, for every level, check the direction. Use a flag to know the direction of traversal in the tree. If the **flag** is set to *true*, print the nodes from right to left in the particular level. If the **flag** is set to **false**, print the nodes in that level from left to right. Initially, the flag is set to False, after every 2 levels, flag changes its value to true and vice versa.

Below is the implementation of the above approach.

```
// C++ program level order traversal
// with direction change
// after every two levels
#include <bits/stdc++.h>
using namespace std;

struct node {
    int data;
    node *left, *right;
} * temp;

// inserts new node
node* newNode(int data)
{
    temp = new node;
    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// function to print current level
void printCurrLevel(node* root, int level, bool flag)
{
    if (!root)
        return;

    if (level == 1) {
        cout << root->data << " ";
        return;
    }
}
```

```
    else {
        // If the flag is true, we have to print
        // level from RIGHT to LEFT.
        if (flag) {
            printCurrLevel(root->right, level - 1, flag);
            printCurrLevel(root->left, level - 1, flag);
        }

        // If the flag is false, we have to print
        // level from LEFT to RIGHT.
        else {
            printCurrLevel(root->left, level - 1, flag);
            printCurrLevel(root->right, level - 1, flag);
        }
    }
}

// This function returns the height of tree.
int height(node* root)
{
    if (!root)
        return 0;

    // left subtree
    int lh = height(root->left);

    // right subtree
    int rh = height(root->right);

    return 1 + max(lh, rh);
}

// Function to traverse level-wise and
// print nodes
void modifiedLevelOrder(node* root)
{
    int h = height(root);

    // Variable to choose direction.
    bool flag = false;
    for (int i = 1; i <= h; i++) {
        printCurrLevel(root, i, flag);
        cout << endl;

        // change direction after every two levels.
        if (i % 2 == 0)
            flag = !flag;
    }
}
```

```
}

// Driver Code
int main()
{
    // create tree that is given
    // in the example
    node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->left->left->left = newNode(8);
    root->left->left->right = newNode(9);
    root->left->right->left = newNode(3);
    root->left->right->right = newNode(1);
    root->right->left->left = newNode(4);
    root->right->left->right = newNode(2);
    root->right->right->left = newNode(7);
    root->right->right->right = newNode(2);
    root->left->right->left->left = newNode(16);
    root->left->right->left->right = newNode(17);
    root->right->left->right->left = newNode(18);
    root->right->right->left->right = newNode(19);

    modifiedLevelOrder(root);
    return 0;
}
```

Output:

```
1
2 3
7 6 5 4
2 7 2 4 1 3 9 8
16 17 18 19
```

Source

<https://www.geeksforgeeks.org/level-order-traversal-with-direction-change-after-every-two-levels-recursive-approach/>

Chapter 64

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: HVVH

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", "VHVH", "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 a starting string 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  $x + y \leq k$ 
        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 C 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

HVVH

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/lexicographically-kth-smallest-way-reach-given-coordinate-origin/>

Chapter 65

Longest consecutive sequence in Binary tree

Longest consecutive sequence in Binary tree - GeeksforGeeks

Given a Binary Tree find the length of the longest path which comprises of nodes with consecutive values in increasing order. Every node is considered as a path of length 1.

Examples:

In below diagram binary tree with longest consecutive path(LCP) are shown :

We can solve above problem recursively. At each node we need information of its parent node, if current node has value one more than its parent node then it makes a consecutive path, at each node we will compare node's value with its parent value and update the longest consecutive path accordingly.

For getting the value of parent node, we will pass the (node_value + 1) as an argument to the recursive method and compare the node value with this argument value, if satisfies, update the current length of consecutive path otherwise reinitialize current path length by 1.

Please see below code for better understanding :

```
// C/C++ program to find longest consecutive
// sequence in binary tree
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to left
   child and a pointer to right child */
```

```
struct Node
{
    int data;
    Node *left, *right;
};

// A utility function to create a node
Node* newNode(int data)
{
    Node* temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// Utility method to return length of longest
// consecutive sequence of tree
void longestConsecutiveUtil(Node* root, int curLength,
                           int expected, int& res)
{
    if (root == NULL)
        return;

    // if root data has one more than its parent
    // then increase current length
    if (root->data == expected)
        curLength++;
    else
        curLength = 1;

    // update the maximum by current length
    res = max(res, curLength);

    // recursively call left and right subtree with
    // expected value 1 more than root data
    longestConsecutiveUtil(root->left, curLength,
                           root->data + 1, res);
    longestConsecutiveUtil(root->right, curLength,
                           root->data + 1, res);
}

// method returns length of longest consecutive
// sequence rooted at node root
int longestConsecutive(Node* root)
{
    if (root == NULL)
        return 0;
```

```
    int res = 0;

    // call utility method with current length 0
    longestConsecutiveUtil(root, 0, root->data, res);

    return res;
}

// Driver code to test above methods
int main()
{
    Node* root = newNode(6);
    root->right = newNode(9);
    root->right->left = newNode(7);
    root->right->right = newNode(10);
    root->right->right->right = newNode(11);

    printf("%d\n", longestConsecutive(root));
    return 0;
}
```

Output:

3

Also discussed on below link:

[Maximum Consecutive Increasing Path Length in Binary Tree](#)

Source

<https://www.geeksforgeeks.org/longest-consecutive-sequence-binary-tree/>

Chapter 66

Max sum of M non-overlapping subarrays of size K

Max sum of M non-overlapping subarrays of size K - GeeksforGeeks

Given an array and two numbers M and K. We need to find **sum of max M subarrays of size K (non-overlapping)** in the array. (Order of array remains unchanged). K is the size of subarrays and M is the count of subarray. It may be assumed that size of array is more than $m*k$. If total array size is not multiple of k, then we can take partial last array.

Examples :

Input: N = 7, M = 3, K = 1
arr[] = {2, 10, 7, 18, 5, 33, 0};
Output: 61
Explanation: subsets are: 33, 18, 10 (3 subsets of size 1)

Input: N = 4, M = 2, K = 2
arr[] = {3, 2, 100, 1};
Output: 106
Explanation: subsets are: (3, 2), (100, 1) 2 subsets of size 2

Here we can see that the we need to find M subarrays each of size K so,

1. We create a presum array, which contains in each index sum of all elements from 'index' to 'index + K' in the given array. And size of the sum array will be $n+1-k$.
2. Now if we include the subarray of size k, then we can not include any of the elements of that subarray again in any other subarray as it will create overlapping subarrays. So we make recursive call by excluding the k elements of included subarray.
3. if we exclude a subarray then we can use the next k-1 elements of that subarray in other subarrays so we will make recursive call by just excluding the first element of that subarray.
4. At last return the max(included sum, excluded sum).

C++

```
// C++ program to find Max sum of M non-overlapping
// subarray of size K in an Array
#include <bits/stdc++.h>
using namespace std;

// calculating presum of array. presum[i]
// is going to store prefix sum of subarray
// of size k beginning with arr[i]
void calculatePresumArray(int presum[],
                        int arr[], int n, int k)
{
    for (int i=0; i<k; i++)
        presum[i] += arr[i];

    // store sum of array index i to i+k
    // in presum array at index i of it.
    for (int i = 1; i <= n - k; i++)
        presum[i] += presum[i-1] + arr[i+k-1] -
                        arr[i-1];
}

// calculating maximum sum of m non overlapping array
int maxSumMnonOverlappingSubarray(int presum[],
                                int m, int size, int k, int start)
{
    // if m is zero then no need any array
    // of any size so return 0.
    if (m == 0)
        return 0;

    // if start is greater then the size
    // of presum array return 0.
    if (start > size - 1)
        return 0;

    int mx = 0;

    // if including subarray of size k
    int includeMax = presum[start] +
                    maxSumMnonOverlappingSubarray(presum,
                                                    m - 1, size, k, start + k);

    // if excluding element and searching
    // in all next possible subarrays
    int excludeMax =
        maxSumMnonOverlappingSubarray(presum,
```

```

        m, size, k, start + 1);

    // return max
    return max(includeMax, excludeMax);
}

// Driver code
int main()
{
    int arr[] = { 2, 10, 7, 18, 5, 33, 0 };
    int n = sizeof(arr)/sizeof(arr[0]);

    int m = 3, k = 1;

    int presum[n + 1 - k] = { 0 };
    calculatePresumArray(presum, arr, n, k);

    // resulting presum array will have a size = n+1-k
    cout << maxSumMnonOverlappingSubarray(presum,
                                           m, n + 1 - k, k, 0);

    return 0;
}

```

Java

```

// Java program to find Max sum
// of M non-overlapping subarray
// of size K in an Array

import java.io.*;

class GFG
{
    // calculating presum of array.
    // presum[i] is going to store
    // prefix sum of subarray of
    // size k beginning with arr[i]
    static void calculatePresumArray(int presum[],
                                     int arr[],
                                     int n, int k)
    {
        for (int i = 0; i < k; i++)
            presum[0] += arr[i];

        // store sum of array index i to i+k
        // in presum array at index i of it.
        for (int i = 1; i <= n - k; i++)

```



```
        presum[i] += presum[i - 1] + arr[i + k - 1] -
                        arr[i - 1];
    }

    // calculating maximum sum of
    // m non overlapping array
    static int maxSumMnonOverlappingSubarray(int presum[],
                                              int m, int size,
                                              int k, int start)
    {
        // if m is zero then no need
        // any array of any size so
        // return 0.
        if (m == 0)
            return 0;

        // if start is greater then the
        // size of presum array return 0.
        if (start > size - 1)
            return 0;

        int mx = 0;

        // if including subarray of size k
        int includeMax = presum[start] +
            maxSumMnonOverlappingSubarray(presum,
                                           m - 1, size, k, start + k);

        // if excluding element and searching
        // in all next possible subarrays
        int excludeMax =
            maxSumMnonOverlappingSubarray(presum,
                                           m, size, k, start + 1);

        // return max
        return Math.max(includeMax, excludeMax);
    }

    // Driver code
    public static void main (String[] args)
    {
        int arr[] = { 2, 10, 7, 18, 5, 33, 0 };
        int n = arr.length;
        int m = 3, k = 1;
        int presum[] = new int[n + 1 - k] ;
        calculatePresumArray(presum, arr, n, k);

        // resulting presum array
```

```
// will have a size = n+1-k
System.out.println(maxSumMnonOverlappingSubarray(presum,
                                                    m, n + 1 - k, k, 0));
}
}

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

C#

```
// C# program to find Max sum of M
// non-overlapping subarray of size
// K in an Array
using System;

class GFG {

    // calculating presum of array.
    // presum[i] is going to store
    // prefix sum of subarray of
    // size k beginning with arr[i]
    static void calculatePresumArray(int []presum,
                                     int []arr, int n, int k)
    {
        for (int i = 0; i < k; i++)
            presum[0] += arr[i];

        // store sum of array index i to i+k
        // in presum array at index i of it.
        for (int i = 1; i <= n - k; i++)
            presum[i] += presum[i - 1] + arr[i + k - 1]
                        - arr[i - 1];
    }

    // calculating maximum sum of
    // m non overlapping array
    static int maxSumMnonOverlappingSubarray(
        int []presum, int m, int size,
        int k, int start)
    {

        // if m is zero then no need
        // any array of any size so
        // return 0.
        if (m == 0)
            return 0;

        // if start is greater then the
```

```
// size of presum array return 0.
if (start > size - 1)
    return 0;

//int mx = 0;

// if including subarray of size k
int includeMax = presum[start] +
    maxSumMnonOverlappingSubarray(presum,
        m - 1, size, k, start + k);

// if excluding element and searching
// in all next possible subarrays
int excludeMax =
    maxSumMnonOverlappingSubarray(presum,
        m, size, k, start + 1);

// return max
return Math.Max(includeMax, excludeMax);
}

// Driver code
public static void Main ()
{
    int []arr = { 2, 10, 7, 18, 5, 33, 0 };
    int n = arr.Length;
    int m = 3, k = 1;
    int []presum = new int[n + 1 - k] ;
    calculatePresumArray(presum, arr, n, k);

    // resulting presum array
    // will have a size = n+1-k
    Console.WriteLine(
        maxSumMnonOverlappingSubarray(presum,
            m, n + 1 - k, k, 0));
}

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

Output :

61

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/max-sum-of-m-non-overlapping-subarrays-of-size-k/>

Chapter 67

Maximize array elements upto given number

Maximize array elements upto given number - GeeksforGeeks

Given an array of integers, a number and a maximum value, task is to compute the maximum value that can be obtained from the array elements. Every value on the array traversing from the beginning can be either added to or subtracted from the result obtained from previous index such that at any point the result is not less than 0 and not greater than the given maximum value. For index 0 take previous result equal to given number. In case of no possible answer print -1.

Examples :

```
Input : arr[] = {2, 1, 7}
        Number = 3
        Maximum value = 7
```

```
Output : 7
The order of addition and subtraction
is: 3(given number) - 2(arr[0]) -
1(arr[1]) + 7(arr[2]).
```

```
Input : arr[] = {3, 10, 6, 4, 5}
        Number = 1
        Maximum value = 15
```

```
Output : 9
The order of addition and subtraction
is: 1 + 3 + 10 - 6 - 4 + 5
```

Prerequisite : [Dynamic Programming](#) | [Recursion](#).

Naive Approach : Use recursion to find maximum value. At every index position there are two choices, either add current array element to value obtained so far from previous elements

or subtract current array element from value obtained so far from previous elements. Start from index 0, add or subtract arr[0] from given number and recursively call for next index along with updated number. When entire array is traversed, compare the updated number with overall maximum value of number obtained so far.

Below is the implementation of above approach :

C++

```
// CPP code to find maximum
// value of number obtained by
// using array elements recursively.
#include <bits/stdc++.h>
using namespace std;

// Utility function to find maximum possible value
void findMaxValUtil(int arr[], int n, int num,
                    int maxLimit, int ind, int& ans)
{
    // If entire array is traversed, then compare
    // current value in num to overall maximum
    // obtained so far.
    if (ind == n) {
        ans = max(ans, num);
        return;
    }

    // Case 1: Subtract current element from value so
    // far if result is greater than or equal to zero.
    if (num - arr[ind] >= 0)
    {
        findMaxValUtil(arr, n, num - arr[ind],
                        maxLimit, ind + 1, ans);
    }

    // Case 2 : Add current element to value so far
    // if result is less than or equal to maxLimit.
    if (num + arr[ind] <= maxLimit)
    {
        findMaxValUtil(arr, n, num + arr[ind],
                        maxLimit, ind + 1, ans);
    }
}

// Function to find maximum possible
// value that can be obtained using
// array elements and given number.
int findMaxVal(int arr[], int n,
```

```
        int num, int maxLimit)
{
    // variable to store maximum value
    // that can be obtained.
    int ans = 0;

    // variable to store current index position.
    int ind = 0;

    // call to utility function to find maximum
    // possible value that can be obtained.
    findMaxValUtil(arr, n, num, maxLimit, ind, ans);

    return ans;
}

// Driver code
int main()
{
    int num = 1;
    int arr[] = { 3, 10, 6, 4, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int maxLimit = 15;

    cout << findMaxVal(arr, n, num, maxLimit);
    return 0;
}
```

Java

```
// Java code to find maximum
// value of number obtained by
// using array elements recursively.
import java.io.*;
import java.lang.*;

public class GFG {

    // variable to store maximum value
    // that can be obtained.
    static int ans;

    // Utility function to find maximum
    // possible value
    static void findMaxValUtil(int []arr, int n, int num,
                                int maxLimit, int ind)
    {
```

```
// If entire array is traversed, then compare
// current value in num to overall maximum
// obtained so far.
if (ind == n) {
    ans = Math.max(ans, num);
    return;
}

// Case 1: Subtract current element from value so
// far if result is greater than or equal to zero.
if (num - arr[ind] >= 0)
{
    findMaxValUtil(arr, n, num - arr[ind],
                    maxLimit, ind + 1);
}

// Case 2 : Add current element to value so far
// if result is less than or equal to maxLimit.
if (num + arr[ind] <= maxLimit)
{
    findMaxValUtil(arr, n, num + arr[ind],
                    maxLimit, ind + 1);
}
}

// Function to find maximum possible
// value that can be obtained using
// array elements and given number.
static int findMaxVal(int []arr, int n,
                      int num, int maxLimit)
{

    // variable to store current index position.
    int ind = 0;

    // call to utility function to find maximum
    // possible value that can be obtained.
    findMaxValUtil(arr, n, num, maxLimit, ind);

    return ans;
}

// Driver code
public static void main(String args[])
{
    int num = 1;
```



```
int []arr = { 3, 10, 6, 4, 5 };
int n = arr.length;
int maxLimit = 15;

System.out.print(findMaxVal(arr, n, num,
                           maxLimit));
}
}

// This code is contributed by Manish Shaw
// (manishshaw1)
```

Python3

```
# Python3 code to find maximum
# value of number obtained by
# using array elements recursively.

# Utility def to find
# maximum possible value

# variable to store maximum value
# that can be obtained.
ans = 0;
def findMaxValUtil(arr, n, num, maxLimit, ind):
    global ans

    # If entire array is traversed,
    # then compare current value
    # in num to overall maximum
    # obtained so far.
    if (ind == n) :
        ans = max(ans, num)
        return

    # Case 1: Subtract current element
    # from value so far if result is
    # greater than or equal to zero.
    if (num - arr[ind] >= 0) :
        findMaxValUtil(arr, n, num - arr[ind],
                        maxLimit, ind + 1)

    # Case 2 : Add current element to
    # value so far if result is less
    # than or equal to maxLimit.
    if (num + arr[ind] <= maxLimit) :
        findMaxValUtil(arr, n, num + arr[ind],
                        maxLimit, ind + 1)
```

```
# def to find maximum possible
# value that can be obtained using
# array elements and given number.
def findMaxVal(arr, n, num, maxLimit) :
    global ans
    # variable to store
    # current index position.
    ind = 0

    # call to utility def to
    # find maximum possible value
    # that can be obtained.
    findMaxValUtil(arr, n, num, maxLimit, ind)
    return ans
```

```
# Driver code
num = 1
arr = [3, 10, 6, 4, 5]
n = len(arr)
maxLimit = 15

print (findMaxVal(arr, n, num, maxLimit))

# This code is contributed by Manish Shaw
# (manishshaw1)
```

C#

```
// C# code to find maximum
// value of number obtained by
// using array elements recursively.
using System;
using System.Collections.Generic;

class GFG {

    // Utility function to find maximum
    // possible value
    static void findMaxValUtil(int []arr, int n, int num,
                                int maxLimit, int ind, ref int ans)
    {

        // If entire array is traversed, then compare
        // current value in num to overall maximum
        // obtained so far.
        if (ind == n) {
```

```
        ans = Math.Max(ans, num);
        return;
    }

    // Case 1: Subtract current element from value so
    // far if result is greater than or equal to zero.
    if (num - arr[ind] >= 0)
    {
        findMaxValUtil(arr, n, num - arr[ind],
                        maxLimit, ind + 1, ref ans);
    }

    // Case 2 : Add current element to value so far
    // if result is less than or equal to maxLimit.
    if (num + arr[ind] <= maxLimit)
    {
        findMaxValUtil(arr, n, num + arr[ind],
                        maxLimit, ind + 1, ref ans);
    }
}

// Function to find maximum possible
// value that can be obtained using
// array elements and given number.
static int findMaxVal(int []arr, int n,
                      int num, int maxLimit)
{
    // variable to store maximum value
    // that can be obtained.
    int ans = 0;

    // variable to store current index position.
    int ind = 0;

    // call to utility function to find maximum
    // possible value that can be obtained.
    findMaxValUtil(arr, n, num, maxLimit, ind,
                    ref ans);

    return ans;
}

// Driver code
public static void Main()
{
    int num = 1;
    int []arr = { 3, 10, 6, 4, 5 };
```

```
        int n = arr.Length;
        int maxLimit = 15;

        Console.Write(findMaxVal(arr, n, num,
                                maxLimit));
    }
}
```

```
// This code is contributed by Manish Shaw
// (manishshaw1)
```

PHP

```
<?php
// PHP code to find maximum
// value of number obtained by
// using array elements recursively.

// Utility function to find
// maximum possible value
function findMaxValUtil($arr, $n,
                        $num, $maxLimit,
                        $ind, &$amp;ans)
{
    // If entire array is traversed,
    // then compare current value
    // in num to overall maximum
    // obtained so far.
    if ($ind == $n)
    {
        $ans = max($ans, $num);
        return;
    }

    // Case 1: Subtract current element
    // from value so far if result is
    // greater than or equal to zero.
    if ($num - $arr[$ind] >= 0)
    {
        findMaxValUtil($arr, $n,
                        $num - $arr[$ind],
                        $maxLimit, $ind + 1,
                        $ans);
    }

    // Case 2 : Add current element to
    // value so far if result is less
    // than or equal to maxLimit.
```

```
        if ($num + $arr[$ind] <= $maxLimit)
        {
            findMaxValUtil($arr, $n,
                           $num + $arr[$ind],
                           $maxLimit, $ind + 1,
                           $ans);
        }
    }

    // Function to find maximum possible
    // value that can be obtained using
    // array elements and given number.
    function findMaxVal($arr, $n,
                       $num, $maxLimit)
    {
        // variable to store maximum value
        // that can be obtained.
        $ans = 0;

        // variable to store
        // current index position.
        $ind = 0;

        // call to utility function to
        // find maximum possible value
        // that can be obtained.
        findMaxValUtil($arr, $n, $num,
                       $maxLimit, $ind, $ans);

        return $ans;
    }

    // Driver code
    $num = 1;
    $arr = array(3, 10, 6, 4, 5);
    $n = count($arr);
    $maxLimit = 15;

    echo (findMaxVal($arr, $n, $num, $maxLimit));

    //This code is contributed by Manish Shaw
    //(manishshaw1)
    ?>
```

Output:

Time Complexity : $O(2^n)$.

Note : For small values of $n \leq 20$, this solution will work. But as array size increases, this will not be an optimal solution.

An **efficient** solution is to use [Dynamic Programming](#). Observe that the value at every step is constrained between 0 and `maxLimit` and hence, the required maximum value will also lie in this range. At every index position, after `arr[i]` is added to or subtracted from result, the new value of result will also lie in this range. Lets try to build the solution backwards. Suppose the required maximum possible value is x , where $0 \leq x \leq \text{maxLimit}$. This value x is obtained by either adding or subtracting `arr[n-1]` to/from the value obtained until index position $n-2$. The same reason can be given for value obtained at index position $n-2$ that it depends on value at index position $n-3$ and so on. The resulting recurrence relation can be given as :

```
Check can x be obtained from arr[0..n-1]:
    Check can x - arr[n-1] be obtained from arr[0..n-2]
    || Check can x + arr[n-1] be obtained from arr[0..n-2]
```

A boolean DP table can be created in which `dp[i][j]` is 1 if value j can be obtained using `arr[0..i]` and 0 if not. For each index position, start from $j = 0$ and move to value `maxLimit`, and set `dp[i][j]` either 0 or 1 as described above. Find the maximum possible value that can be obtained at index position $n-1$ by finding maximum j when $i = n-1$ and `dp[n-1][j] = 1`.

C++

```
// C++ program to find maximum value of
// number obtained by using array
// elements by using dynamic programming.
#include <bits/stdc++.h>
using namespace std;

// Function to find maximum possible
// value of number that can be
// obtained using array elements.
int findMaxVal(int arr[], int n,
               int num, int maxLimit)
{
    // Variable to represent current index.
    int ind;

    // Variable to show value between
    // 0 and maxLimit.
    int val;

    // Table to store whether a value can
    // be obtained or not upto a certain index.
    // 1. dp[i][j] = 1 if value j can be
```

```
//    obtained upto index i.
// 2. dp[i][j] = 0 if value j cannot be
//    obtained upto index i.
int dp[n][maxLimit+1];

for(ind = 0; ind < n; ind++)
{
    for(val = 0; val <= maxLimit; val++)
    {
        // Check for index 0 if given value
        // val can be obtained by either adding
        // to or subtracting arr[0] from num.
        if(ind == 0)
        {
            if(num - arr[ind] == val ||
               num + arr[ind] == val)
            {
                dp[ind][val] = 1;
            }
            else
            {
                dp[ind][val] = 0;
            }
        }
        else
        {
            // 1. If arr[ind] is added to
            // obtain given val then val-
            // arr[ind] should be obtainable
            // from index ind-1.
            // 2. If arr[ind] is subtracted to
            // obtain given val then val+arr[ind]
            // should be obtainable from index ind-1.
            // Check for both the conditions.
            if(val - arr[ind] >= 0 &&
               val + arr[ind] <= maxLimit)
            {
                // If either of one condition is true,
                // then val is obtainable at index ind.
                dp[ind][val] = dp[ind-1][val-arr[ind]] ||
                               dp[ind-1][val+arr[ind]];
            }
            else if(val - arr[ind] >= 0)
            {
                dp[ind][val] = dp[ind-1][val-arr[ind]];
            }
            else if(val + arr[ind] <= maxLimit)
            {

```

```
        dp[ind][val] = dp[ind-1][val+arr[ind]];
    }
    else
    {
        dp[ind][val] = 0;
    }
}
}

// Find maximum value that is obtained
// at index n-1.
for(val = maxLimit; val >= 0; val--)
{
    if(dp[n-1][val])
    {
        return val;
    }
}

// If no solution exists return -1.
return -1;
}

// Driver Code
int main()
{
    int num = 1;
    int arr[] = {3, 10, 6, 4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    int maxLimit = 15;

    cout << findMaxVal(arr, n, num, maxLimit);
    return 0;
}
```

Java

```
// Java program to find maximum
// value of number obtained by
// using array elements by using
// dynamic programming.
import java.io.*;
```

```
class GFG
{
    // Function to find maximum
```



```
// possible value of number
// that can be obtained
// using array elements.
static int findMaxVal(int []arr, int n,
                     int num, int maxLimit)
{
    // Variable to represent
    // current index.
    int ind;

    // Variable to show value
    // between 0 and maxLimit.
    int val;

    // Table to store whether
    // a value can be obtained
    // or not upto a certain
    // index 1. dp[i,j] = 1 if
    // value j can be obtained
    // upto index i.
    // 2. dp[i,j] = 0 if value j
    // cannot be obtained upto index i.
    int [][]dp = new int[n][maxLimit + 1];

    for(ind = 0; ind < n; ind++)
    {
        for(val = 0; val <= maxLimit; val++)
        {
            // Check for index 0 if given
            // value val can be obtained
            // by either adding to or
            // subtracting arr[0] from num.
            if(ind == 0)
            {
                if(num - arr[ind] == val ||
                   num + arr[ind] == val)
                {
                    dp[ind][val] = 1;
                }
                else
                {
                    dp[ind][val] = 0;
                }
            }
            else
            {
                // 1. If arr[ind] is added
```

```
// to obtain given val then
// val- arr[ind] should be
// obtainable from index
// ind-1.
// 2. If arr[ind] is subtracted
// to obtain given val then
// val+arr[ind] should be
// obtainable from index ind-1.
// Check for both the conditions.
if(val - arr[ind] >= 0 &&
    val + arr[ind] <= maxLimit)
{

    // If either of one condition
    // is true, then val is
    // obtainable at index ind.
    if(dp[ind - 1][val - arr[ind]] == 1
        || dp[ind - 1][val + arr[ind]] == 1)
        dp[ind][val] = 1;

}
else if(val - arr[ind] >= 0)
{
    dp[ind][val] = dp[ind - 1][val -
                                arr[ind]];
}
else if(val + arr[ind] <= maxLimit)
{
    dp[ind][val] = dp[ind - 1][val +
                                arr[ind]];
}
else
{
    dp[ind][val] = 0;
}
}
}

// Find maximum value that
// is obtained at index n-1.
for(val = maxLimit; val >= 0; val--)
{
    if(dp[n - 1][val] == 1)
    {
        return val;
    }
}
```

```
        // If no solution
        // exists return -1.
        return -1;
    }

    // Driver Code
    public static void main(String args[])
    {
        int num = 1;
        int []arr = new int[]{3, 10, 6, 4, 5};
        int n = arr.length;
        int maxLimit = 15;

        System.out.print(findMaxVal(arr, n,
                                    num, maxLimit));
    }
}
```

// This code is contributed
// by Manish Shaw(manishshaw1)

C#

```
// C# program to find maximum value of
// number obtained by using array
// elements by using dynamic programming.
using System;

class GFG {

    // Function to find maximum possible
    // value of number that can be
    // obtained using array elements.
    static int findMaxVal(int []arr, int n,
                          int num, int maxLimit)
    {

        // Variable to represent current index.
        int ind;

        // Variable to show value between
        // 0 and maxLimit.
        int val;

        // Table to store whether a value can
        // be obtained or not upto a certain
        // index 1. dp[i,j] = 1 if value j
```

```
// can be obtained upto index i.
// 2. dp[i,j] = 0 if value j cannot be
// obtained upto index i.
int [,]dp = new int[n,maxLimit+1];

for(ind = 0; ind < n; ind++)
{
    for(val = 0; val <= maxLimit; val++)
    {
        // Check for index 0 if given
        // value val can be obtained
        // by either adding to or
        // subtracting arr[0] from num.
        if(ind == 0)
        {
            if(num - arr[ind] == val ||
               num + arr[ind] == val)
            {
                dp[ind,val] = 1;
            }
            else
            {
                dp[ind,val] = 0;
            }
        }
        else
        {
            // 1. If arr[ind] is added
            // to obtain given val then
            // val- arr[ind] should be
            // obtainable from index
            // ind-1.
            // 2. If arr[ind] is subtracted
            // to obtain given val then
            // val+arr[ind] should be
            // obtainable from index ind-1.
            // Check for both the conditions.
            if(val - arr[ind] >= 0 &&
               val + arr[ind] <= maxLimit)
            {

                // If either of one condition
                // is true, then val is
                // obtainable at index ind.
                if(dp[ind-1,val-arr[ind]] == 1
                   || dp[ind-1,val+arr[ind]] == 1)
                    dp[ind,val] = 1;
            }
        }
    }
}
```

```
        }
        else if(val - arr[ind] >= 0)
        {
            dp[ind,val] = dp[ind-1,val-arr[ind]];
        }
        else if(val + arr[ind] <= maxLimit)
        {
            dp[ind,val] = dp[ind-1,val+arr[ind]];
        }
        else
        {
            dp[ind,val] = 0;
        }
    }
}

// Find maximum value that is obtained
// at index n-1.
for(val = maxLimit; val >= 0; val--)
{
    if(dp[n-1,val] == 1)
    {
        return val;
    }
}

// If no solution exists return -1.
return -1;
}

// Driver Code
static void Main()
{
    int num = 1;
    int []arr = new int[]{3, 10, 6, 4, 5};
    int n = arr.Length;
    int maxLimit = 15;

    Console.Write(
        findMaxVal(arr, n, num, maxLimit));
}

// This code is contributed by Manish Shaw
// (manishshaw1)
```

Output:

9

Time Complexity : $O(n * \text{maxLimit})$, where n is the size of array and maxLimit is the given max value.

Auxiliary Space : $O(n * \text{maxLimit})$, n is the size of array and maxLimit is the given max value.

Optimization : The space required can be reduced to $O(2 * \text{maxLimit})$. Note that at every index position, we are only using values from previous row. So we can create a table with two rows, in which one of the rows store result for previous iteration and other for the current iteration.

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/maximize-array-elements-upto-given-number/>

Chapter 68

Maximum length of segments of 0's and 1's

Maximum length of segments of 0's and 1's - GeeksforGeeks

Given a string comprising of ones and zeros. The task is to find the maximum length of the segments of string such that a number of 1 in each segment is greater than 0.

Note: Each segment taken should be distinct. Index starts from 0.

Examples:

Input: str = "100110001010001"

Output: 9

First segment from index 0 to 4 (10011), total length = 5

Second segment from index 8 to 10 (101), total length = 3

Third segment from index 14 till 14 (1), total length = 1,

Hence answer is $5 + 3 + 1 = 9$

Input: str = "0010111101100000"

Output: 13

The maximum length can be formed by taking segment from index 0 till index 12 (0010111101100),
i.e. of total length = 13

Approach:

1. If start == n, limiting condition arises, return 0.
2. Run a loop from start till n, computing for each subarray till n.
3. If character is 1 then increment the count of 1 else increment the count of 0.
4. If count of 1 is greater than 0, recursively call the function for index (k+1) i.e. next index and add the remaining length i.e. k-start+1.

5. Else only recursively call the function for next index k+1.
6. Return dp[start].

Below is the implementation of above approach:

```
// C++ implementation of above approach
#include <bits/stdc++.h>
using namespace std;

// Recursive Function to find total length of the array
// Where 1 is greater than zero
int find(int start, string adj, int n, int dp[])
{
    // If reaches till end
    if (start == n)
        return 0;

    // If dp is saved
    if (dp[start] != -1)
        return dp[start];

    dp[start] = 0;
    int one = 0, zero = 0, k;

    // Finding for each length
    for (k = start; k < n; k++) {

        // If the character scanned is 1
        if (adj[k] == '1')
            one++;
        else
            zero++;

        // If one is greater than zero, add total
        // length scanned till now
        if (one > zero)
            dp[start] = max(dp[start], find(k + 1, adj, n, dp)
                            + k - start + 1);

        // Continue with next length
        else
            dp[start] = max(dp[start], find(k + 1, adj, n, dp));
    }

    // Return the value for start index
    return dp[start];
}
```



```
// Driver Code
int main()
{
    string adj = "100110001010001";

    // Size of string
    int n = adj.size();
    int dp[n + 1];
    memset(dp, -1, sizeof(dp));
    // Calling the function to find the value of function

    cout << find(0, adj, n, dp) << endl;

    return 0;
}
```

Output:

9

Source

<https://www.geeksforgeeks.org/maximum-length-of-segments-of-0s-and-1s/>

Chapter 69

Memoization (1D, 2D and 3D)

Memoization (1D, 2D and 3D) - GeeksforGeeks

Most of the Dynamic Programming problems are solved in two ways:

1. **Tabulation:** Bottom Up
2. **Memoization:** Top Down

One of the easier approaches to solve most of the problems in DP is to write the recursive code at first and then write the Bottom-up Tabulation Method or Top-down Memoization of the recursive function. The steps to write the DP solution of Top-down approach to any problem is to:

1. Write the recursive code
2. Memoize the return value and use it to reduce recursive calls.

1-D Memoization

The first step will be to write the recursive code. In the program below, a program related to recursion where only one parameter changes its value has been shown. Since only one parameter is non-constant, this method is known as 1-D memoization. E.g., the Fibonacci series problem to find the N-th term in the Fibonacci series. The recursive approach has been discussed over [here](#).

Given below is the recursive code to find the N-th term:

C++

```
// C++ program to find the Nth term
// of Fibonacci series
#include <bits/stdc++.h>
using namespace std;
```

```
// Fibonacci Series using Recursion
int fib(int n)
{
    // Base case
    if (n <= 1)
        return n;

    // recursive calls
    return fib(n - 1) + fib(n - 2);
}

// Driver Code
int main()
{
    int n = 6;
    printf("%d", fib(n));
    return 0;
}
```

Java

```
// Java program to find the
// Nth term of Fibonacci series
import java.io.*;

class GFG
{
    // Fibonacci Series
    // using Recursion
    static int fib(int n)
    {
        // Base case
        if (n <= 1)
            return n;

        // recursive calls
        return fib(n - 1) +
            fib(n - 2);
    }

    // Driver Code
    public static void main (String[] args)
    {
        int n = 6;
        System.out.println(fib(n));
    }
}
```

```
}  
}  
  
// This code is contributed  
// by ajit  
  
C#  
  
// C# program to find  
// the Nth term of  
// Fibonacci series  
using System;  
  
class GFG  
{  
  
// Fibonacci Series  
// using Recursion  
static int fib(int n)  
{  
    // Base case  
    if (n <= 1)  
        return n;  
  
    // recursive calls  
    return fib(n - 1) +  
           fib(n - 2);  
}  
// Driver Code  
static public void Main ()  
{  
    int n = 6;  
    Console.WriteLine(fib(n));  
}  
}  
  
// This code is contributed  
// by akt_mit
```

PHP

```
<?php  
// PHP program to find  
// the Nth term of  
// Fibonacci series  
// using Recursion
```

```

function fib($n)
{
    // Base case
    if ($n <= 1)
        return $n;

    // recursive calls
    return fib($n - 1) +
        fib($n - 2);
}

// Driver Code
$n = 6;
echo fib($n);

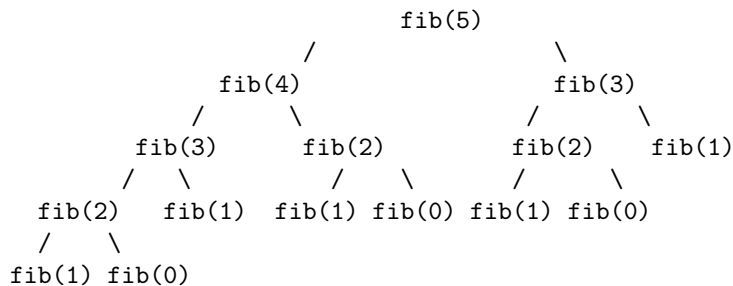
// This code is contributed
// by ajit
?>

```

Output:

8

A common observation is that this implementation does a lot of **repeated work** (see the following recursion tree). So this will consume a lot of time for finding the N-th Fibonacci number if done.



In the above tree fib(3), fib(2), fib(1), fib(0) all are called more than once.

The following problem has been solved using [Tabulation](#) method.

In the program below, the steps to write a **Top-Down approach program** has been explained. Some modifications in the recursive program will reduce the complexity of the program and give the desired result. If fib(x) has not occurred previously, then we store

the value of **fib(x)** in an array **term** at index **x** and return **term[x]**. By memoizing the return value of **fib(x)** at index **x** of an array, reduce the number of recursive calls at the next step when **fib(x)** has already been called. So without doing further recursive calls to compute the value of **fib(x)**, **return term[x]** when **fib(x)** has already been computed previously to avoid a lot of repeated work as shown in the tree.

Given below is the memoized recursive code to find the N-th term.

C++

```
// CPP program to find the Nth term
// of Fibonacci series
#include <bits/stdc++.h>
using namespace std;
int term[1000];
// Fibonacci Series using memoized Recursion
int fib(int n)
{
    // base case
    if (n <= 1)
        return n;

    // if fib(n) has already been computed
    // we do not do further recursive calls
    // and hence reduce the number of repeated
    // work
    if (term[n] != 0)
        return term[n];

    else {

        // store the computed value of fib(n)
        // in an array term at index n to
        // so that it does not needs to be
        // precomputed again
        term[n] = fib(n - 1) + fib(n - 2);

        return term[n];
    }
}

// Driver Code
int main()
{
    int n = 6;
    printf("%d", fib(n));
    return 0;
}
```

Java

```
// Java program to find
// the Nth term of
// Fibonacci series
import java.io.*;

class GFG
{
    // Fibonacci Series using
    // memoized Recursion
    static int fib(int n)
    {
        int []term = new int [1000];

        // base case
        if (n <= 1)
            return n;

        // if fib(n) has already
        // been computed we do not
        // do further recursive
        // calls and hence reduce
        // the number of repeated
        // work
        if (term[n] != 0)
            return term[n];

        else
        {
            // store the computed value
            // of fib(n) in an array
            // term at index n to so that
            // it does not needs to be
            // precomputed again
            term[n] = fib(n - 1) +
                    fib(n - 2);

            return term[n];
        }
    }
}

// Driver Code
public static void main (String[] args)
{
    int n = 6;
```

```
        System.out.println(fib(n));
    }
}

// This code is contributed by ajit
```

Output:

8

If the recursive code has been written once, then memoization is just modifying the recursive program and storing the return values to avoid repetitive calls of functions which have been computed previously.

2-D Memoization

In the above program, the recursive function had only **one argument whose value was not constant** after every function call. Below, an implementation where the recursive program has two non-constant arguments has been shown.

For e.g., Program to solve the standard Dynamic Problem [LCS](#) problem when two strings are given. The general recursive solution of the problem is to generate all subsequences of both given sequences and find the longest matching subsequence. Total possible combinations will be 2^n . Hence recursive solution will take $O(2^n)$. The approach to write the recursive solution has been discussed [here](#).

Given below is the recursive solution to the LCS problem:

C++

```
// A Naive recursive implementation of LCS problem
#include <bits/stdc++.h>

int max(int a, int b);

// Returns length of LCS for X[0..m-1], Y[0..n-1]
int lcs(char* X, char* Y, int m, int n)
{
    if (m == 0 || n == 0)
        return 0;
    if (X[m - 1] == Y[n - 1])
        return 1 + lcs(X, Y, m - 1, n - 1);
    else
        return max(lcs(X, Y, m, n - 1),
                    lcs(X, Y, m - 1, n));
}

// Utility function to get max of 2 integers
```



```

int max(int a, int b)
{
    return (a > b) ? a : b;
}

// Driver Code
int main()
{
    char X[] = "AGGTAB";
    char Y[] = "GXTXAYB";

    int m = strlen(X);
    int n = strlen(Y);

    printf("Length of LCS is %dn", lcs(X, Y, m, n));

    return 0;
}

```

Output:

Length of LCS is 4n

Output:

Length of LCS is 4

Considering the above implementation, the following is a partial recursion tree for input strings “AXYT” and “AYZX”

```

                    lcs("AXYT", "AYZX")
                      /      \
        lcs("AXY", "AYZX")    lcs("AXYT", "AYZ")
          /      \            /      \
lcs("AX", "AYZX") lcs("AXY", "AYZ") lcs("AXY", "AYZ") lcs("AXYT", "AY")

```

In the above partial recursion tree, **lcs(“AXY”, “AYZ”)** is being solved twice. On drawing the complete recursion tree, it has been observed that there are many subproblems which are solved again and again. So this problem has Overlapping Substructure property and recomputation of same subproblems can be avoided by either using Memoization or Tabulation. The tabulation method has been discussed [here](#).

A common point of observation to use memoization in the recursive code will be the **two non-constant arguments M and N** in every function call. The function has 4 arguments, but 2 arguments are constant which do not affect the Memoization. The repetitive calls occur

for N and M which have been called previously. So use a 2-D array to store the computed **lcs(m, n) value at arr[m-1][n-1]** as the string index starts from 0. Whenever the function with the same argument m and n are called again, we do not perform any further recursive call and return arr[m-1][n-1] as the previous computation of the lcs(m, n) has already been stored in arr[m-1][n-1], hence reducing the recursive calls that happen more than once.

Below is the implementation of the Memoization approach of the recursive code.

C++

```
// C++ program to memoize
// recursive implementation of LCS problem
#include <bits/stdc++.h>
int arr[1000][1000];
int max(int a, int b);

// Returns length of LCS for X[0..m-1], Y[0..n-1] */
// memoization applied in recursive solution
int lcs(char* X, char* Y, int m, int n)
{
    // base case
    if (m == 0 || n == 0)
        return 0;

    // if the same state has already been
    // computed
    if (arr[m - 1][n - 1] != -1)
        return arr[m - 1][n - 1];

    // if equal, then we store the value of the
    // function call
    if (X[m - 1] == Y[n - 1]) {

        // store it in arr to avoid further repetitive
        // work in future function calls
        arr[m - 1][n - 1] = 1 + lcs(X, Y, m - 1, n - 1);
        return arr[m - 1][n - 1];
    }
    else {
        // store it in arr to avoid further repetitive
        // work in future function calls
        arr[m - 1][n - 1] = max(lcs(X, Y, m, n - 1),
                                lcs(X, Y, m - 1, n));
        return arr[m - 1][n - 1];
    }
}

// Utility function to get max of 2 integers
```

```

int max(int a, int b)
{
    return (a > b) ? a : b;
}

// Driver Code
int main()
{
    memset(arr, -1, sizeof(arr));
    char X[] = "AGGTAB";
    char Y[] = "GXTXAYB";

    int m = strlen(X);
    int n = strlen(Y);

    printf("Length of LCS is %d", lcs(X, Y, m, n));

    return 0;
}

```

Output:

Length of LCS is 4

3-D Memoization

In the above program, the recursive function had only **two** arguments whose value were not constant after every function call. Below, an implementation where the recursive program has **three non-constant arguments** is done.

For e.g., Program to solve the standard Dynamic Problem LCS problem for three strings. The general recursive solution of the problem is to generate all subsequences of both given sequences and find the longest matching subsequence. Total possible combinations will be 3^n . Hence recursive solution will take $O(3^n)$.

Given below is the recursive solution to the LCS problem:

C++

```

// A recursive implementation of LCS problem
// of three strings
#include <bits/stdc++.h>
int max(int a, int b);

// Returns length of LCS for X[0..m-1], Y[0..n-1]
int lcs(char* X, char* Y, char* Z, int m, int n, int o)
{
    // base case

```

```

    if (m == 0 || n == 0 || o == 0)
        return 0;

    // if equal, then check for next combination
    if (X[m - 1] == Y[n - 1] and Y[n - 1] == Z[o - 1]) {

        // recursive call
        return 1 + lcs(X, Y, Z, m - 1, n - 1, o - 1);
    }
    else {

        // return the maximum of the three other
        // possible states in recursion
        return max(lcs(X, Y, Z, m, n - 1, o),
                   max(lcs(X, Y, Z, m - 1, n, o),
                       lcs(X, Y, Z, m, n, o - 1)));
    }
}

// Utility function to get max of 2 integers
int max(int a, int b)
{
    return (a > b) ? a : b;
}

// Driver Code
int main()
{
    char X[] = "geeks";
    char Y[] = "geeksfor";
    char Z[] = "geeksforge";
    int m = strlen(X);
    int n = strlen(Y);
    int o = strlen(Z);
    printf("Length of LCS is %d", lcs(X, Y, Z, m, n, o));

    return 0;
}

```

Output:

Length of LCS is 5

The tabulation method has been shown [here](#). On drawing the recursion tree completely, it has been noticed that there are many overlapping sub-problems which are been calculated

multiple times. Since the function parameter has three non-constant parameters, hence a 3-D array will be used to memoize the value that was returned when `lcs(x, y, z, m, n, o)` for any value of `m`, `n` and `o` was called so that if `lcs(x, y, z, m, n, o)` is again called for the same value of `m`, `n` and `o` then the function will return the already stored value as it has been computed previously in the recursive call. `arr[m][n][o]` stores the value returned by the `lcs(x, y, z, m, n, o)` function call. The only modification that needs to be done in the recursive program is to store the return value of `(m, n, o)` state of the recursive function. The rest remains the same in the above recursive program.

Below is the implementation of the Memoization approach of the recursive code:

C++

```
// A memoize recursive implementation of LCS problem
#include <bits/stdc++.h>
int arr[100][100][100];
int max(int a, int b);

// Returns length of LCS for X[0..m-1], Y[0..n-1] */
// memoization applied in recursive solution
int lcs(char* X, char* Y, char* Z, int m, int n, int o)
{
    // base case
    if (m == 0 || n == 0 || o == 0)
        return 0;

    // if the same state has already been
    // computed
    if (arr[m - 1][n - 1][o - 1] != -1)
        return arr[m - 1][n - 1][o - 1];

    // if equal, then we store the value of the
    // function call
    if (X[m - 1] == Y[n - 1] and Y[n - 1] == Z[o - 1]) {

        // store it in arr to avoid further repetitive work
        // in future function calls
        arr[m - 1][n - 1][o - 1] = 1 + lcs(X, Y, Z, m - 1,
                                           n - 1, o - 1);

        return arr[m - 1][n - 1][o - 1];
    }
    else {

        // store it in arr to avoid further repetitive work
        // in future function calls
        arr[m - 1][n - 1][o - 1] =
            max(lcs(X, Y, Z, m, n - 1, o),
               max(lcs(X, Y, Z, m - 1, n, o),
```

```
        lcs(X, Y, Z, m, n, o - 1));
    return arr[m - 1][n - 1][o - 1];
}

// Utility function to get max of 2 integers
int max(int a, int b)
{
    return (a > b) ? a : b;
}

// Driver Code
int main()
{
    memset(arr, -1, sizeof(arr));
    char X[] = "geeks";
    char Y[] = "geeksfor";
    char Z[] = "geeksforgeeks";
    int m = strlen(X);
    int n = strlen(Y);
    int o = strlen(Z);
    printf("Length of LCS is %d", lcs(X, Y, Z, m, n, o));

    return 0;
}
```

Output:

Length of LCS is 5

Note: The array used to Memoize is initialized to some value (say -1) before the function call to mark if the function with the same parameters has been previously called or not.

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/memoization-1d-2d-and-3d/>

Chapter 70

Merge Sort for Linked Lists in JavaScript

Merge Sort for Linked Lists in JavaScript - GeeksforGeeks

Prerequisite: [Merge Sort for Linked Lists](#)

Merge sort is often preferred for sorting a linked list. The slow random-access performance of a linked list makes some other algorithms (such as quicksort) perform poorly, and others (such as heapsort) completely impossible.

In this post, Merge sort for linked list is implemented using JavaScript.

Examples:

Input : 5 -> 4 -> 3 -> 2 -> 1

Output : 1 -> 2 -> 3 -> 4 -> 5

Input : 10 -> 20 -> 3 -> 2 -> 1

Output : 1 -> 2 -> 3 -> 10 -> 20

```
<script>

// Create Node of LinkedList
function Node(data) {
    this.node = data;
    this.next = null;
}

// To initialize a linkedlist
function LinkedList(list) {
    this.head = list || null
}

// Function to insert The new Node into the linkedList
```

```
LinkedList.prototype.insert = function(data) {

    // Check if the linked list is empty
    // so insert first node and lead head
    // points to generic node
    if (this.head === null)
        this.head = new Node(data);

    else {

        // If linked list is not empty, insert the node
        // at the end of the linked list
        let list = this.head;
        while (list.next) {
            list = list.next;
        }

        // Now here list pointer points to last
        // node let's insert out new node in it
        list.next = new Node(data)
    }
}

// Function to print linkedList
LinkedList.prototype.iterate = function() {

    // First we will check whether out
    // linked list is empty or node
    if (this.head === null)
        return null;

    // If linked list is not empty we will
    // iterate from each Node and prints
    // it's value store in "data" property

    let list = this.head;

    // we will iterate until our list variable
    // contains the "Next" value of the last Node
    // i.e-> null
    while (list) {
        document.write(list.node)
        if (list.next)
            document.write(' -> ')
        list = list.next
    }
}
```



```
// Function to mergesort a linked list
LinkedList.prototype.mergeSort = function(list) {

    if (list.next === null)
        return list;

    let count = 0;
    let countList = list
    let leftPart = list;
    let leftPointer = list;
    let rightPart = null;
    let rightPointer = null;

    // Counting the nodes in the received linkedlist
    while (countList.next !== null) {
        count++;
        countList = countList.next;
    }

    // counting the mid of the linked list
    let mid = Math.floor(count / 2)
    let count2 = 0;

    // separating the left and right part with
    // respect to mid node in the linked list
    while (count2 < mid) {
        count2++;
        leftPointer = leftPointer.next;
    }

    rightPart = new LinkedList(leftPointer.next);
    leftPointer.next = null;

    // Here are two linked list which
    // contains the left most nodes and right
    // most nodes of the mid node
    return this._mergeSort(this.mergeSort(leftPart),
                           this.mergeSort(rightPart.head))
}

// Merging both lists in sorted manner
LinkedList.prototype._mergeSort = function(left, right) {

    // Create a new empty linked list
    let result = new LinkedList()

    let resultPointer = result.head;
    let pointerLeft = left;
```

```
    let pointerRight = right;

    // If true then add left most node value in result,
    // increment left pointer else do the same in
    // right linked list.
    // This loop will be executed until pointer's of
    // a left node or right node reached null
    while (pointerLeft && pointerRight) {
        let tempNode = null;

        // Check if the right node's value is greater than
        // left node's value
        if (pointerLeft.node > pointerRight.node) {
            tempNode = pointerRight.node
            pointerRight = pointerRight.next;
        }
        else {
            tempNode = pointerLeft.node
            pointerLeft = pointerLeft.next;
        }

        if (result.head == null) {
            result.head = new Node(tempNode)
            resultPointer = result.head
        }
        else {
            resultPointer.next = new Node(tempNode)
            resultPointer = resultPointer.next
        }
    }

    // Add the remaining elements in the last of resultant
    // linked list
    resultPointer.next = pointerLeft;
    while (resultPointer.next)
        resultPointer = resultPointer.next

    resultPointer.next = pointerRight

    // Result is the new sorted linked list
    return result.head;
}

// Initialize the object
let l = new LinkedList();
l.insert(10)
l.insert(20)
```

```
l.insert(3)
l.insert(2)
l.insert(1)
// Print the linked list
l.iterate()

// Sort the linked list
l.head = LinkedList.prototype.mergeSort(l.head)

document.write('<br> After sorting : ');

// Print the sorted linked list
l.iterate()
</script>
```

Output

```
10 -> 20 -> 3 -> 2 -> 1
After sorting : 1 -> 2 -> 3 -> 10 -> 20
```

Source

<https://www.geeksforgeeks.org/merge-sort-linked-lists-javascript/>

Chapter 71

Minimum steps to reach a destination

Minimum steps to reach a destination - GeeksforGeeks

Given a number line from $-\infty$ to $+\infty$. You start at 0 and can go either to the left or to the right. The condition is that in i 'th move, you take i steps.

- a) Find if you can reach a given number x
- b) Find the most optimal way to reach a given number x , if we can indeed reach it. For example, 3 can be reached in 2 steps, (0, 1) (1, 3) and 4 can be reached in 3 steps (0, -1), (-1, 1) (1, 4).

Source: [Flipkart Interview Question](#)

The important thing to note is we can reach any destination as it is always possible to make a move of length 1. At any step i , we can move forward i , then backward $i + 1$.

Below is a recursive solution suggested by Arpit Thapar [here](#).

- 1) Since distance of $+5$ and -5 from 0 is same, hence we find answer for absolute value of destination.
- 2) We use a recursive function which takes as arguments:
 - i) Source Vertex
 - ii) Value of last step taken
 - iii) Destination
- 3) If at any point source vertex = destination; return number of steps.
- 4) Otherwise we can go in both of the possible directions. Take the minimum of steps in both cases.

From any vertex we can go to :
(current source + last step +1) and
(current source - last step -1)

- 5) If at any point, absolute value of our position exceeds the absolute value of our destination then it is intuitive that the shortest path is not possible from here. Hence we make the value

of steps INT_MAX, so that when i take the minimum of both possibilities, this one gets eliminated.

If we don't use this last step, the program enters into an INFINITE recursion and gives RUN TIME ERROR.

Below is the implementation of above idea. Note that the solution only counts steps.

C++

```
// C++ program to count number of
// steps to reach a point
#include<bits/stdc++.h>
using namespace std;

// Function to count number of steps
// required to reach a destination

// source -> source vertex
// step -> value of last step taken
// dest -> destination vertex
int steps(int source, int step, int dest)
{
    // base cases
    if (abs(source) > (dest))
        return INT_MAX;
    if (source == dest) return step;

    // at each point we can go either way

    // if we go on positive side
    int pos = steps(source + step + 1,
                    step + 1, dest);

    // if we go on negative side
    int neg = steps(source - step - 1,
                    step + 1, dest);

    // minimum of both cases
    return min(pos, neg);
}

// Driver code
int main()
{
    int dest = 11;
    cout << "No. of steps required to reach "
          << dest << " is "
          << steps(0, 0, dest);
}
```

```
    return 0;
}
```

Java

```
// Java program to count number of
// steps to reach a point
import java.io.*;

class GFG
{
    // Function to count number of steps
    // required to reach a destination

    // source -> source vertex
    // step -> value of last step taken
    // dest -> destination vertex
    static int steps(int source, int step,
                     int dest)
    {
        // base cases
        if (Math.abs(source) > (dest))
            return Integer.MAX_VALUE;

        if (source == dest)
            return step;

        // at each point we can go either way

        // if we go on positive side
        int pos = steps(source + step + 1,
                        step + 1, dest);

        // if we go on negative side
        int neg = steps(source - step - 1,
                        step + 1, dest);

        // minimum of both cases
        return Math.min(pos, neg);
    }

    // Driver Code
    public static void main(String[] args)
    {
        int dest = 11;
        System.out.println("No. of steps required"+
                           " to reach " + dest +
```

```
        " is " + steps(0, 0, dest));  
    }  
}
```

// This code is contributed by Prerna Saini

Python3

```
# python program to count number of  
# steps to reach a point  
import sys
```

```
# Function to count number of steps  
# required to reach a destination
```

```
# source -> source vertex  
# step -> value of last step taken  
# dest -> destination vertex  
def steps(source, step, dest):
```

```
    #base cases  
    if (abs(source) > (dest)) :  
        return sys.maxsize
```

```
    if (source == dest):  
        return step
```

```
    # at each point we can go  
    # either way
```

```
    # if we go on positive side  
    pos = steps(source + step + 1,  
                step + 1, dest)
```

```
    # if we go on negative side  
    neg = steps(source - step - 1,  
                step + 1, dest)
```

```
    # minimum of both cases  
    return min(pos, neg)
```

```
# Driver Code  
dest = 11;  
print("No. of steps required",  
      " to reach " ,dest ,  
      " is " , steps(0, 0, dest));
```

This code is contributed by Sam007.

C#

```
// C# program to count number of
// steps to reach a point
using System;

class GFG
{
    // Function to count number of steps
    // required to reach a destination

    // source -> source vertex
    // step -> value of last step taken
    // dest -> destination vertex
    static int steps(int source, int step,
                     int dest)
    {
        // base cases
        if (Math.Abs(source) > (dest))
            return int.MaxValue;

        if (source == dest)
            return step;

        // at each point we can go either way

        // if we go on positive side
        int pos = steps(source + step + 1,
                        step + 1, dest);

        // if we go on negative side
        int neg = steps(source - step - 1,
                        step + 1, dest);

        // minimum of both cases
        return Math.Min(pos, neg);
    }

    // Driver Code
    public static void Main()
    {
        int dest = 11;
        Console.WriteLine("No. of steps required"+
                           " to reach " + dest +
                           " is " + steps(0, 0, dest));
    }
}
```



```
    }  
}  
  
// This code is contributed by Sam007
```

PHP

```
<?php  
// PHP program to count number  
// of steps to reach a point  
  
// Function to count number  
// of steps required to reach  
// a destination  
  
// source -> source vertex  
// step -> value of last step taken  
// dest -> destination vertex  
function steps($source, $step, $dest)  
{  
    // base cases  
    if (abs($source) > ($dest))  
        return PHP_INT_MAX;  
    if ($source == $dest)  
        return $step;  
  
    // at each point we  
    // can go either way  
  
    // if we go on positive side  
    $pos = steps($source + $step + 1,  
                $step + 1, $dest);  
  
    // if we go on negative side  
    $neg = steps($source - $step - 1,  
                $step + 1, $dest);  
  
    // minimum of both cases  
    return min($pos, $neg);  
}  
  
// Driver code  
$dest = 11;  
echo "No. of steps required to reach ",  
     $dest, " is ", steps(0, 0, $dest);  
  
// This code is contributed by aj_36  
?>
```

Output :

No. of steps required to reach 11 is 5

Thanks to Arpit Thapar for providing above algorithm and implementation.

Optimized Solution : [Find minimum moves to reach target on an infinite line](#)

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

Improved By : [Sam007](#), [jit_t](#)

Source

<https://www.geeksforgeeks.org/minimum-steps-to-reach-a-destination/>

Chapter 72

Minimum tiles of sizes in powers of two to cover whole area

Minimum tiles of sizes in powers of two to cover whole area - GeeksforGeeks

Given an area of $N \times M$. You have infinite number of tiles of size $2^i \times 2^i$, where $i = 0, 1, 2, \dots$ so on. The task is to find minimum number of tiles required to fill the given area with tiles.

Examples:

Input : $N = 5, M = 6$.

Output : 9

Area of 5×6 can be covered with minimum 9 tiles.

6 tiles of 1×1 , 2 tiles of 2×2 , 1 tile of 4×4 .

Input : $N = 10, M = 5$.

Output : 14

The idea is to divide the given area into nearest $2^i \times 2^i$.

Lets divide the problem into cases:

Case 1: if N is odd and M is even, fill the a row or column with M number of 1×1 tiles. Then count the minimum number of tiles for $N/2 \times M/2$ size of area. Similarly, if M is odd and N is even, add N to our answer and find minimum number of tiles for $N/2 \times M/2$ area.

Case 2: If N and M both are odd, fill one row and one column, so add $N + M - 1$ to the answer and find minimum number of tiles required to fill $N/2 \times M/2$ area.

Case 3: If N and M both are even, calculate the minimum number of tiles required to fill area of $N/2 \times M/2$ area. Because halving both the dimensions doesn't change the number of tiles required.

Below is the implementation of this approach:

C++

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

int minTiles(int n, int m)
{
    // base case, when area is 0.
    if (n == 0 || m == 0)
        return 0;

    // If n and m both are even, calculate tiles for n/2 x m/2
    // Halving both dimensions doesn't change the number of tiles
    else if (n%2 == 0 && m%2 == 0)
        return minTiles(n/2, m/2);

    // If n is even and m is odd
    // Use a row of 1x1 tiles
    else if (n%2 == 0 && m%2 == 1)
        return (n + minTiles(n/2, m/2));

    // If n is odd and m is even
    // Use a column of 1x1 tiles
    else if (n%2 == 1 && m%2 == 0)
        return (m + minTiles(n/2, m/2));

    // If n and m are odd
    // add row + column number of tiles
    else
        return (n + m - 1 + minTiles(n/2, m/2));
}

// Driven Program
int main()
{
    int n = 5, m = 6;

    cout << minTiles(n, m) << endl;
    return 0;
}
```

Java

```
// Java code for Minimum tiles of
// sizes in powers of two to cover
```

```
// whole area

class GFG {

    static int minTiles(int n, int m)
    {
        // base case, when area is 0.
        if (n == 0 || m == 0)
            return 0;

        // If n and m both are even,
        // calculate tiles for n/2 x m/2
        // Halving both dimensions doesn't
        // change the number of tiles
        else if (n % 2 == 0 && m % 2 == 0)
            return minTiles(n / 2, m / 2);

        // If n is even and m is odd
        // Use a row of 1x1 tiles
        else if (n % 2 == 0 && m % 2 == 1)
            return (n + minTiles(n / 2, m / 2));

        // If n is odd and m is even
        // Use a column of 1x1 tiles
        else if (n % 2 == 1 && m % 2 == 0)
            return (m + minTiles(n / 2, m / 2));

        // If n and m are odd
        // add row + column number of tiles
        else
            return (n + m - 1 + minTiles(n / 2, m / 2));
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 5, m = 6;
        System.out.println(minTiles(n, m));
    }
}

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

Python3

```
def minTiles(n, m):

    # base case, when area is 0.
```

```
if n == 0 or m == 0:
    return 0

# If n and m both are even, calculate
# tiles for n/2 x m/2
# Halving both dimensions doesn't
# change the number of tiles
elif n%2 == 0 and m%2 == 0:
    return minTiles(int(n/2), int(m/2))

# If n is even and m is odd
# Use a row of 1x1 tiles
elif n % 2 == 0 and m % 2 == 1:
    return (n + minTiles(int(n/2), int(m/2)))

# If n is odd and m is even
# Use a column of 1x1 tiles
elif n % 2 == 1 and m % 2 == 0:
    return (m + minTiles(int(n/2), int(m/2)))

# If n and m are odd add
# row + column number of tiles
else:
    return (n + m - 1 + minTiles(int(n/2), int(m/2)))

# Driven Program
n = 5
m = 6
print (minTiles(n, m))

# This code is contributed
# by Shreyanshi Arun.
```

C#

```
// C# code for Minimum tiles of
// sizes in powers of two to cover
// whole area
using System;

class GFG {

    static int minTiles(int n, int m)
    {

        // base case, when area is 0.
        if (n == 0 || m == 0)
            return 0;
```

```
// If n and m both are even,
// calculate tiles for n/2 x m/2
// Halving both dimensions doesn't
// change the number of tiles
else if (n % 2 == 0 && m % 2 == 0)
    return minTiles(n / 2, m / 2);

// If n is even and m is odd
// Use a row of 1x1 tiles
else if (n % 2 == 0 && m % 2 == 1)
    return (n + minTiles(n / 2, m / 2));

// If n is odd and m is even
// Use a column of 1x1 tiles
else if (n % 2 == 1 && m % 2 == 0)
    return (m + minTiles(n / 2, m / 2));

// If n and m are odd
// add row + column number of tiles
else
    return (n + m - 1 + minTiles(n / 2, m / 2));
}

// Driver code
public static void Main()
{
    int n = 5, m = 6;

    Console.WriteLine(minTiles(n, m));
}

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

PHP

```
<?php
// PHP program for Minimum tiles of
// sizes in powers of two to cover
// whole area

function minTiles($n, $m)
{
    // base case, when area is 0.
    if ($n == 0 or $m == 0)
        return 0;
```

```
// If n and m both are even,
// calculate tiles for n/2 x m/2
// Halving both dimensions doesn't
// change the number of tiles
else if ($n % 2 == 0 and
        $m % 2 == 0)
    return minTiles($n / 2, $m / 2);

// If n is even and m is odd
// Use a row of 1x1 tiles
else if ($n % 2 == 0 and $m % 2 == 1)
    return floor($n + minTiles($n / 2,
                               $m / 2));

// If n is odd and m is even
// Use a column of 1x1 tiles
else if ($n % 2 == 1 and
        $m % 2 == 0)
    return ($m + minTiles($n / 2,
                          $m / 2));

// If n and m are odd
// add row + column number of tiles
else
    return floor($n + $m - 1 +
                minTiles($n / 2, $m / 2));
}

// Driver Code
$n = 5; $m = 6;
echo minTiles($n, $m);

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

Output:

9

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/minimum-tiles-of-sizes-in-powers-of-two-to-cover-whole-area/>

Chapter 73

Modular exponentiation (Recursive)

Modular exponentiation (Recursive) - GeeksforGeeks

Given three numbers a, b and c, we need to find $(a^b) \% c$

Now why do “% c” after exponentiation, because a^b will be really large even for relatively small values of a, b and that is a problem because the data type of the language that we try to code the problem, will most probably not let us store such a large number.

Examples:

Input : a = 2312 b = 3434 c = 6789
Output : 6343

Input : a = -3 b = 5 c = 89
Output : 24

The idea is based on below properties.

Property 1:

$(m * n) \% p$ has a very interesting property:
 $(m * n) \% p = ((m \% p) * (n \% p)) \% p$

Property 2:

if b is even:

$(a^b) \% c = ((a^{b/2}) * (a^{b/2})) \% c$? this suggests divide and conquer

if b is odd:

$(a^b) \% c = (a * (a^{b-1})) \% c$

Property 3:

If we have to return the mod of a negative number x whose absolute value is less than y:
then $(x + y) \% y$ will do the trick

Note:

Also as the product of $(a^{b/2}) * (a^{b/2})$ and $a * (a^{b-1})$ may cause overflow, hence we must be careful about those scenarios

C

```
// Recursive C program to compute modular power
#include <stdio.h>

int exponentMod(int A, int B, int C)
{
    // Base cases
    if (A == 0)
        return 0;
    if (B == 0)
        return 1;

    // If B is even
    long y;
    if (B % 2 == 0) {
        y = exponentMod(A, B / 2, C);
        y = (y * y) % C;
    }

    // If B is odd
    else {
        y = A % C;
        y = (y * exponentMod(A, B - 1, C) % C) % C;
    }

    return (int)((y + C) % C);
}

// Driver program to test above functions
int main()
{
    int A = 2, B = 5, C = 13;
    printf("Power is %d", exponentMod(A, B, C));
    return 0;
}
```

Java

```
// Recursive Java program
// to compute modular power
import java.io.*;
```

```

class GFG
{
static int exponentMod(int A,
                      int B, int C)
{

    // Base cases
    if (A == 0)
        return 0;
    if (B == 0)
        return 1;

    // If B is even
    long y;
    if (B % 2 == 0)
    {
        y = exponentMod(A, B / 2, C);
        y = (y * y) % C;
    }

    // If B is odd
    else
    {
        y = A % C;
        y = (y * exponentMod(A, B - 1,
                           C) % C) % C;
    }

    return (int)((y + C) % C);
}

// Driver Code
public static void main(String args[])
{
    int A = 2, B = 5, C = 13;
    System.out.println("Power is " +
                      exponentMod(A, B, C));
}
}

// This code is contributed
// by Swetank Modi.

```

Python3

```

# Recursive Python program
# to compute modular power
def exponentMod(A, B, C):

```

```
# Base Cases
if (A == 0):
    return 0
if (B == 0):
    return 1

# If B is Even
y = 0
if (B % 2 == 0):
    y = exponentMod(A, B / 2, C)
    y = (y * y) % C

# If B is Odd
else:
    y = A % C
    y = (y * exponentMod(A, B - 1,
                        C) % C) % C

return ((y + C) % C)

# Driver Code
A = 2
B = 5
C = 13
print("Power is", exponentMod(A, B, C))

# This code is contributed
# by Swetank Modi.
```

C#

```
// Recursive C# program
// to compute modular power
class GFG
{
static int exponentMod(int A, int B, int C)
{

    // Base cases
    if (A == 0)
        return 0;
    if (B == 0)
        return 1;

    // If B is even
    long y;
    if (B % 2 == 0)
    {
```

```
        y = exponentMod(A, B / 2, C);
        y = (y * y) % C;
    }

    // If B is odd
    else
    {
        y = A % C;
        y = (y * exponentMod(A, B - 1,
                               C) % C) % C;
    }

    return (int)((y + C) % C);
}

// Driver Code
public static void Main()
{
    int A = 2, B = 5, C = 13;
    System.Console.WriteLine("Power is " +
                             exponentMod(A, B, C));
}
}
```

// This code is contributed
// by mits

PHP

```
<?php
// Recursive PHP program to
// compute modular power
function exponentMod($A, $B, $C)
{
    // Base cases
    if ($A == 0)
        return 0;
    if ($B == 0)
        return 1;

    // If B is even
    if ($B % 2 == 0)
    {
        $y = exponentMod($A, $B / 2, $C);
        $y = ($y * $y) % $C;
    }

    // If B is odd
```

```
    else
    {
        $y = $A % $C;
        $y = ($y * exponentMod($A, $B - 1,
                               $C) % $C) % $C;
    }

    return (($y + $C) % $C);
}
```

```
// Driver Code
$A = 2;
$B = 5;
$C = 13;
echo "Power is " . exponentMod($A, $B, $C);

// This code is contributed
// by Swetank Modi.
?>
```

Output:

Power is 6

Iterative modular exponentiation.

Improved By : [swetankmodi](#), [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/modular-exponentiation-recursive/>

Chapter 74

Moser-de Bruijn Sequence

Moser-de Bruijn Sequence - GeeksforGeeks

Given an integer 'n', print the first 'n' terms of the Moser-de Bruijn Sequence.

The **Moser-de Bruijn sequence** is the sequence obtained by adding up the distinct powers of the number 4 (For example 1, 4, 16, 64, etc).

Examples :

Input : 5

Output : 0 1 4 5 16

Input : 10

Output : 0 1 4 5 16 17 20 21 64 65

It is observed that the terms of the sequence follow the recurrence relation :

$$1) S(2 * n) = 4 * S(n)$$

$$2) S(2 * n + 1) = 4 * S(n) + 1$$

$$\text{with } S(0) = 0 \text{ and } S(1) = 1$$

It may be noted here that any number which is the sum of non-distinct powers of 4 are not a part of the sequence. For example, 8 is not a part of the sequence because it is formed as the sum of non-distinct powers of 4, that are 4 and 4.

Thus, any number which is not a power of 4 and is present in the sequence must be the sum of the distinct powers of 4.

For example, 21 is a part of the sequence, even though it is not a power of 4 because it is the sum of the distinct powers of 4, that are 1, 4 and 16.

Employ the recurrence relation discussed above to generate the sequence, efficiently.

C++

```
// CPP code to generate first 'n' terms
// of the Moser-de Bruijn Sequence
#include <bits/stdc++.h>
using namespace std;

// Function to generate nth term
// of Moser-de Bruijn Sequence
int gen(int n)
{
    // S(0) = 0
    if (n == 0)
        return 0;

    // S(1) = 1
    else if (n == 1)
        return 1;

    // S(2 * n) = 4 * S(n)
    else if (n % 2 == 0)
        return 4 * gen(n / 2);

    // S(2 * n + 1) = 4 * S(n) + 1
    else if (n % 2 == 1)
        return 4 * gen(n / 2) + 1;
}

// Generating the first 'n' terms
// of Moser-de Bruijn Sequence
void moserDeBruijn(int n)
{
    for (int i = 0; i < n; i++)
        cout << gen(i) << " ";
    cout << "\n";
}

// Driver Code
int main()
{
    int n = 15;
    cout << "First " << n << " terms of "
         << "Moser-de Bruijn Sequence : \n";
    moserDeBruijn(n);
    return 0;
}
```

Java


```
// Java code to generate first 'n' terms
// of the Moser-de Bruijn Sequence

class GFG
{
    // Function to generate nth term
    // of Moser-de Bruijn Sequence
    public static int gen(int n)
    {
        // S(0) = 0
        if (n == 0)
            return 0;

        // S(1) = 1
        else if (n == 1)
            return 1;

        // S(2 * n) = 4 * S(n)
        else if (n % 2 == 0)
            return 4 * gen(n / 2);

        // S(2 * n + 1) = 4 * S(n) + 1
        else if (n % 2 == 1)
            return 4 * gen(n / 2) + 1;
        return 0;
    }

    // Generating the first 'n' terms
    // of Moser-de Bruijn Sequence
    public static void moserDeBruijn(int n)
    {
        for (int i = 0; i < n; i++)
            System.out.print(gen(i) + " ");
        System.out.println();
    }

    // Driver Code
    public static void main(String args[])
    {
        int n = 15;
        System.out.println("First " + n +
                           " terms of " +
                           "Moser-de Bruijn Sequence : ");
        moserDeBruijn(n);
    }
}
```

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

Python3

```
# Python code to generate first
# 'n' terms of the Moser-de
# Bruijn Sequence

# Function to generate nth term
# of Moser-de Bruijn Sequence
def gen(n):

    # S(0) = 0
    if n == 0:
        return 0

    # S(1) = 1
    elif n == 1:
        return 1

    # S(2 * n) = 4 * S(n)
    elif n % 2 == 0:
        return 4 * gen(n // 2)

    # S(2 * n + 1) = 4 * S(n) + 1
    elif n % 2 == 1:
        return 4 * gen(n // 2) + 1

# Generating the first 'n' terms
# of Moser-de Bruijn Sequence
def moserDeBruijn(n):
    for i in range(n):
        print(gen(i), end = " ")

# Driver Program
n = 15
print("First", n, "terms of ",
      "Moser-de Bruijn Sequence:")
moserDeBruijn(n)

# This code is contributed by Shrikant13
```

C#

```
// C# code to generate first 'n' terms
// of the Moser-de Bruijn Sequence
```

```
using System;

class GFG {

    // Function to generate nth term
    // of Moser-de Bruijn Sequence
    public static int gen(int n)
    {

        // S(0) = 0
        if (n == 0)
            return 0;

        // S(1) = 1
        else if (n == 1)
            return 1;

        // S(2 * n) = 4 * S(n)
        else if (n % 2 == 0)
            return 4 * gen(n / 2);

        // S(2 * n + 1) = 4 * S(n) + 1
        else if (n % 2 == 1)
            return 4 * gen(n / 2) + 1;
        return 0;
    }

    // Generating the first 'n' terms
    // of Moser-de Bruijn Sequence
    public static void moserDeBruijn(int n)
    {
        for (int i = 0; i < n; i++)
            Console.WriteLine(gen(i) + " ");
    }

    // Driver Code
    public static void Main()
    {
        int n = 15;
        Console.WriteLine("First " + n +
                           " terms of " +
                           "Moser-de Bruijn Sequence : ");
        moserDeBruijn(n);
    }
}

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

PHP

```
<?php
// PHP code to generate first 'n' terms
// of the Moser-de Bruijn Sequence

// Function to generate nth term
// of Moser-de Bruijn Sequence
function gen($n)
{
    // S(0) = 0
    if ($n == 0)
        return 0;

    // S(1) = 1
    else if ($n == 1)
        return 1;

    // S(2 * n) = 4 * S(n)
    else if ($n % 2 == 0)
        return 4 * gen($n / 2);

    // S(2 * n + 1) = 4 * S(n) + 1
    else if ($n % 2 == 1)
        return 4 * gen($n / 2) + 1;
}

// Generating the first 'n' terms
// of Moser-de Bruijn Sequence
function moserDeBruijn($n)
{
    for ($i = 0; $i < $n; $i++)
        echo(gen($i) . " ");
    echo("\n");
}

// Driver Code
$n = 15;
echo("First " . $n . " terms of " .
    "Moser-de Bruijn Sequence : \n");
echo(moserDeBruijn($n));

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

Output :

First 15 terms of Moser-de Bruijn Sequence :
0 1 4 5 16 17 20 21 64 65 68 69 80 81 84

Dynamic Programming Implementation:

C++

```
// CPP code to generate first 'n' terms
// of the Moser-de Bruijn Sequence
#include <bits/stdc++.h>
using namespace std;

// Function to generate nth term
// of Moser-de Bruijn Sequence
int gen(int n)
{
    int S[n+1];

    S[0] = 0;
    S[1] = 1;

    for (int i = 2; i <= n; i++)
    {
        //  $S(2 * n) = 4 * S(n)$ 
        if (i % 2 == 0)
            S[i] = 4 * S[i / 2];

        //  $S(2 * n + 1) = 4 * S(n) + 1$ 
        else
            S[i] = 4 * S[i / 2] + 1;
    }

    return S[n];
}

// Generating the first 'n' terms
// of Moser-de Bruijn Sequence
void moserDeBruijn(int n)
{
    for (int i = 0; i < n; i++)
        cout << gen(i) << " ";
    cout << "\n";
}

// Driver Code
int main()
```

```
{
    int n = 15;
    cout << "First " << n << " terms of "
         << "Moser-de Bruijn Sequence : \n";
    moserDeBruijn(n);
    return 0;
}
```

Java

```
// Java code to generate first 'n' terms
// of the Moser-de Bruijn Sequence
```

```
class GFG
{
    // Function to generate nth term
    // of Moser-de Bruijn Sequence
    static int gen(int n)
    {
        int []S = new int [n + 1];

        S[0] = 0;
        if(n != 0)
            S[1] = 1;

        for (int i = 2; i <= n; i++)
        {
            //  $S(2 * n) = 4 * S(n)$ 
            if (i % 2 == 0)
                S[i] = 4 * S[i / 2];

            //  $S(2 * n + 1) = 4 * S(n) + 1$ 
            else
                S[i] = 4 * S[i/2] + 1;
        }

        return S[n];
    }

    // Generating the first 'n' terms
    // of Moser-de Bruijn Sequence
    static void moserDeBruijn(int n)
    {
        for (int i = 0; i < n; i++)
            System.out.print(gen(i)+" ");
    }
}
```

```
// Driver Code
public static void main(String[] args)
{
    int n = 15;
    System.out.println("First " + n +
                       " terms of " +
                       "Moser-de Bruijn Sequence : ");
    moserDeBruijn(n);
}
}
```

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

python3

```
# python3 code to generate first 'n' terms
# of the Moser-de Bruijn Sequence

# Function to generate nth term
# of Moser-de Bruijn Sequence
def gen( n ):
    S = [0, 1]
    for i in range(2, n+1):

        #  $S(2 * n) = 4 * S(n)$ 
        if i % 2 == 0:
            S.append(4 * S[int(i / 2)]);

        #  $S(2 * n + 1) = 4 * S(n) + 1$ 
        else:
            S.append(4 * S[int(i / 2)] + 1);
    z = S[n];
    return z;

# Generating the first 'n' terms
# of Moser-de Bruijn Sequence
def moserDeBruijn(n):
    for i in range(n):
        print(gen(i), end = " ")

# Driver Code
n = 15
print("First", n, "terms of ",
      "Moser-de Bruijn Sequence:")
moserDeBruijn(n)
```

This code is contributed by mits.

C#

```
// C# code to generate first 'n' terms
// of the Moser-de Bruijn Sequence
using System;
```

```
class GFG
```

```
{
```

```
// Function to generate nth term
```

```
// of Moser-de Bruijn Sequence
```

```
static int gen(int n)
```

```
{
```

```
    int []S = new int [n + 1];
```

```
    S[0] = 0;
```

```
    if(n != 0)
```

```
        S[1] = 1;
```

```
    for (int i = 2; i <= n; i++)
```

```
    {
```

```
        //  $S(2 * n) = 4 * S(n)$ 
```

```
        if (i % 2 == 0)
```

```
            S[i] = 4 * S[i / 2];
```

```
        //  $S(2 * n + 1) = 4 * S(n) + 1$ 
```

```
        else
```

```
            S[i] = 4 * S[i/2] + 1;
```

```
    }
```

```
    return S[n];
```

```
}
```

```
// Generating the first 'n' terms
```

```
// of Moser-de Bruijn Sequence
```

```
static void moserDeBruijn(int n)
```

```
{
```

```
    for (int i = 0; i < n; i++)
```

```
        Console.Write(gen(i)+" ");
```

```
}
```

```
// Driver Code
```

```
public static void Main()
```

```
{
```

```
    int n = 15;
```



```
        Console.WriteLine("First " + n +  
                           " terms of " +  
                           "Moser-de Bruijn Sequence : ");  
        moserDeBruijn(n);  
    }  
}
```

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

PHP

```
<?php  
// PHP code to generate first 'n' terms  
// of the Moser-de Bruijn Sequence  
  
// Function to generate nth term  
// of Moser-de Bruijn Sequence  
function gen( $n)  
{  
    $S = array();  
  
    $S[0] = 0;  
    $S[1] = 1;  
  
    for ( $i = 2; $i <= $n; $i++)  
    {  
        //  $S(2 * n) = 4 * S(n)$   
        if ($i % 2 == 0)  
            $S[$i] = 4 * $S[$i / 2];  
  
        //  $S(2 * n + 1) = 4 * S(n) + 1$   
        else  
            $S[$i] = 4 * $S[$i/2] + 1;  
    }  
  
    return $S[$n];  
}  
  
// Generating the first 'n' terms  
// of Moser-de Bruijn Sequence  
function moserDeBruijn( $n)  
{  
    for ( $i = 0; $i < $n; $i++)  
        echo gen($i) , " ";  
    echo "\n";  
}
```

```
// Driver Code
$n = 15;
echo "First " , $n , " terms of "
    , "Moser-de Bruijn Sequence : \n";
moserDeBruijn($n);

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

Output :

```
First 15 terms of Moser-de Bruijn Sequence :
0 1 4 5 16 17 20 21 64 65 68 69 80 81 84
```

Improved By : [shrikanth13](#), [Smitha Dinesh Semwal](#), [jit_t](#), [jaideeppyne1997](#), [vt_m](#), more
[Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/moser-de-bruijn-sequence/>

Chapter 75

Mutual Recursion with example of Hofstadter Female and Male sequences

Mutual Recursion with example of Hofstadter Female and Male sequences - GeeksforGeeks

Mutual recursion is a variation [recursion](#). Two functions are called mutually recursive if the first function makes a recursive call to the second function and the second function, in turn, calls the first one.

In software development this concept is used in circular dependency which is a relation between two or more modules which either directly or indirectly depend on each other to function properly. Such modules are also known as mutually recursive.

A great example of mutual recursion would be implementing the Hofstadter Sequence.

Hofstadter Sequence

In mathematics, a Hofstadter sequence is a member of a family of related integer sequences defined by **non-linear** recurrence relations. In this example we are going to focus on **Hofstadter Female and Male sequences**:

C

```
// C program to implement Hofstadter Sequence
// using mutual recursion
#include <stdio.h>
```

```
int hofstaderFemale(int);
int hofstaderMale(int);

// Female function
int hofstaderFemale(int n)
{
    if (n < 0)
        return;
    else
        return (n == 0) ? 1 : n - hofstaderFemale(n - 1);
}

// Male function
int hofstaderMale(int n)
{
    if (n < 0)
        return;
    else
        return (n == 0) ? 0 : n - hofstaderMale(n - 1);
}

// hard coded driver function to run the program
int main()
{
    int i;
    printf("F: ");
    for (i = 0; i < 20; i++)
        printf("%d ", hofstaderFemale(i));

    printf("\n");

    printf("M: ");
    for (i = 0; i < 20; i++)
        printf("%d ", hofstaderMale(i));

    return 0;
}
```

Java

```
// Java program to implement Hofstadter
// Sequence using mutual recursion
import java .io.*;

class GFG {

    // Female function
```

```
static int hofstaderFemale(int n)
{
    if (n < 0)
        return 0;
    else
        return (n == 0) ? 1 : n -
            hofstaderFemale(n - 1);
}

// Male function
static int hofstaderMale(int n)
{
    if (n < 0)
        return 0;
    else
        return (n == 0) ? 0 : n -
            hofstaderMale(n - 1);
}

// Driver Code
static public void main (String[] args)
{
    int i;
    System.out.print("F: ");
    for (i = 0; i < 20; i++)
        System.out.print(hofstaderFemale(i)
                           + " ");

    System.out.println();

    System.out.print("M: ");
    for (i = 0; i < 20; i++)
        System.out.print(hofstaderMale(i)
                           + " ");
}

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

Python3

```
# Python program to implement
# Hofstadter Sequence using
# mutual recursion

# Female function
def hofstaderFemale(n):
    if n < 0:
```

```
        return;
    else:
        val = 1 if n == 0 else (
            n - hofstaderFemale(n - 1))
        return val

# Male function
def hofstaderMale(n):
    if n < 0:
        return;
    else:
        val = 0 if n == 0 else (
            n - hofstaderMale(n - 1))
        return val

# Driver code
print("F:", end = " ")
for i in range(0, 20):
    print(hofstaderFemale(i), end = " ")

print("\n")
print("M:", end = " ")
for i in range(0, 20):
    print(hofstaderMale(i), end = " ")

# This code is contributed
# by Shantanu Sharma
```

C#

```
// C# program to implement Hofstadter
// Sequence using mutual recursion
using System;

class GFG {

    // Female function
    static int hofstaderFemale(int n)
    {
        if (n < 0)
            return 0;
        else
            return (n == 0) ? 1 : n -
                hofstaderFemale(n - 1);
    }

    // Male function
    static int hofstaderMale(int n)
```

```
{
    if (n < 0)
        return 0;
    else
        return (n == 0) ? 0 : n -
            hofstaderMale(n - 1);
}

// Driver Code
static public void Main ()
{
    int i;
    Console.WriteLine("F: ");
    for (i = 0; i < 20; i++)
        Console.Write(hofstaderFemale(i) + " ");

    Console.WriteLine();

    Console.WriteLine("M: ");
    for (i = 0; i < 20; i++)
        Console.Write(hofstaderMale(i) + " ");
}
}

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

PHP

```
<?php
// PHP program to implement
// Hofstadter Sequence
// using mutual recursion

//function hofstaderFemale(int);
//int hofstaderMale(int);

// Female function
function hofstaderFemale($n)
{
    if ($n < 0)
        return;
    else
        return ($n == 0) ? 1 : $n - hofstaderFemale($n - 1);
}

// Male function
function hofstaderMale($n)
{

```

```
if ($n < 0)
    return;
else
    return ($n == 0) ? 0 : $n - hofstaderMale($n - 1);
}

// Driver Code
$i;
echo "F: ";
for ($i = 0; $i < 20; $i++)
    echo hofstaderFemale($i), " ";

echo "\n";

echo "M: ";
for ($i = 0; $i < 20; $i++)
    echo hofstaderMale($i), " ";

// This code contributed by Ajit
?>
```

Output:

```
F: 1 0 2 1 3 2 4 3 5 4 6 5 7 6 8 7 9 8 10 9
M: 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10
```

Disadvantages of Circular Dependency/Mutual Recursion:

1. Circular dependencies can cause a domino effect when a small local change in one module spreads into other modules and has unwanted global effects
2. Circular dependencies can also result in infinite recursions or other unexpected failures.
3. Circular dependencies may also cause memory leaks by preventing certain very primitive automatic garbage collectors (those that use reference counting) from deallocating unused objects.

References: [Wikipedia](#)

Improved By : [jit_t](#), [vt_m](#), [Shantanu Sharma](#).

Source

<https://www.geeksforgeeks.org/mutual-recursion-example-hofstadter-female-male-sequences/>

Chapter 76

N Queen in $O(n)$ space

N Queen in $O(n)$ space - GeeksforGeeks

Given n , of a $n \times n$ chessboard, find the proper placement of queens on chessboard.

Previous Approach : [N Queen](#)

Algorithm :

```
Place(k, i)
// Returns true if a queen can be placed
// in kth row and ith column. Otherwise it
// returns false. X[] is a global array
// whose first (k-1) values have been set.
// Abs( ) returns absolute value of r
{
    for j := 1 to k-1 do

        // Two in the same column
        // or in the same diagonal
        if ((x[j] == i) or
            (abs(x[j] - i) = Abs(j - k)))
            then return false;

    return true;
}
```

Algorithm nQueens(k, n) :

```
// Using backtracking, this procedure prints all
// possible placements of n queens on an  $n \times n$ 
```

```
// chessboard so that they are nonattacking.
{
    for i:= 1 to n do
    {
        if Place(k, i) then
        {
            x[k] = i;
            if (k == n)
                write (x[1:n]);
            else
                NQueens(k+1, n);
        }
    }
}
```

C++

```
// CPP code to for n Queen placement
#include <bits/stdc++.h>

#define breakLine cout << "\n-----\n";
#define MAX 10

using namespace std;

int arr[MAX], no;

void nQueens(int k, int n);
bool canPlace(int k, int i);
void display(int n);

// Function to check queens placement
void nQueens(int k, int n){
    for (int i = 1; i <= n; i++){
        if (canPlace(k, i)){
            arr[k] = i;
            if (k == n)
                display(n);
            else
                nQueens(k + 1, n);
        }
    }
}

// Helper Function to check if queen can be placed
bool canPlace(int k, int i){
    for (int j = 1; j <= k - 1; j++){
```

```
        if (arr[j] == i ||
            (abs(arr[j] - i) == abs(j - k)))
            return false;
    }
    return true;
}

// Function to display placed queen
void display(int n){
    breakLine
    cout << "Arrangement No. " << ++no;
    breakLine

    for (int i = 1; i <= n; i++){
        for (int j = 1; j <= n; j++){
            if (arr[i] != j)
                cout << "\t_";
            else
                cout << "\tQ";
        }
        cout << endl;
    }

    breakLine
}

// Driver Code
int main(){
    int n = 4;
    nQueens(1, n);
    return 0;
}
```

Output:

Arrangement No. 1

-	Q	-	-
-	-	-	Q
Q	-	-	-
-	-	Q	-

Arrangement No. 2

-	-	Q	-
Q	-	-	-
-	-	-	Q
-	Q	-	-

Source

<https://www.geeksforgeeks.org/n-queen-in-on-space/>

Chapter 77

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 to 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 <= 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 :

5

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 78

Number of handshakes such that a person shakes hands only once

Number of handshakes such that a person shakes hands only once - GeeksforGeeks

There are N number of persons in a party, find the total number of handshake such that a person can handshake only once.

Examples:

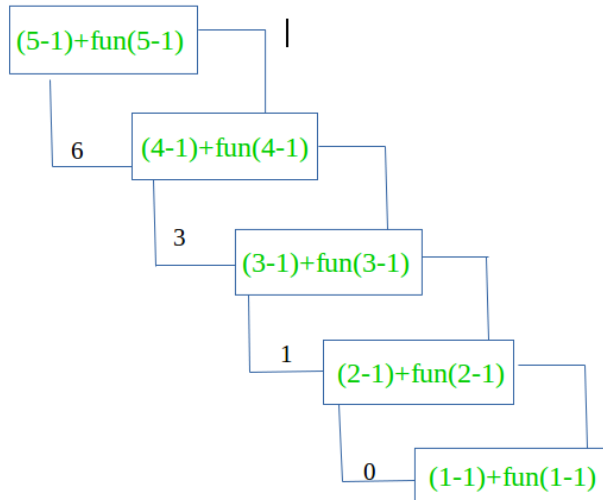
Input : 5
Output : 10

Input : 9
Output : 36

We can see a recursive nature in the problem.

```
// n-th person has (n-1) choices and after  
// n-th person chooses a person, problem  
// recurs for n-1.  
handshake(n) = (n-1) + handshake(n-1)  
  
// Base case  
handshake(0) = 0
```

Let assume that number persons are 5.



So here total number of handshake is 10.

Below is implementation of above recursive formula.

C++

```
// Recursive CPP program to count total
// number of handshakes when a person
// can shake hand with only one.
#include <stdio.h>

// function to find all possible handshakes
int handshake(int n)
{
    // when n becomes 0 that means all the
    // persons have done handshake with other
    if (n == 0)
        return 0;
    else
        return (n - 1) + handshake(n - 1);
}

int main()
{
    int n = 9;
    printf("%d", handshake(n));
    return 0;
}
```

Java

```
// Recursive Java program to
// count total number of
// handshakes when a person
// can shake hand with only one.
import java.io.*;

class GFG
{
    // function to find all
    // possible handshakes
    static int handshake(int n)
    {
        // when n becomes 0 that
        // means all the persons
        // have done handshake
        // with other
        if (n == 0)
            return 0;
        else
            return (n - 1) + handshake(n - 1);
    }

    // Driver Code
    public static void main (String[] args)
    {
        int n = 9;
        System.out.print(handshake(n));
    }
}

// This code is contributed
// by chandan_jnu
```

C#

```
// Recursive C# program to
// count total number of
// handshakes when a person
// can shake hand with only one.
using System;

class GFG
{
```

```
// function to find all
// possible handshakes
static int handshake(int n)
{
    // when n becomes 0 that
    // means all the persons
    // have done handshake
    // with other
    if (n == 0)
        return 0;
    else
        return (n - 1) + handshake(n - 1);
}

// Driver Code
public static void Main (String []args)
{
    int n = 9;
    Console.WriteLine(handshake(n));
}
}

// This code is contributed
// by Arnab Kundu
```

Python3

```
# Recursive Python program
# to count total number of
# handshakes when a person
# can shake hand with only one.

# function to find all
# possible handshakes
def handshake(n):

    # when n becomes 0 that means
    # all the persons have done
    # handshake with other
    if (n == 0):
        return 0
    else:
        return (n - 1) + handshake(n - 1)

# Driver Code
n = 9
print(handshake(n))
```



```
# This code is contributed
# by Shivi_Aggarwal
```

PHP

```
<?php
// Recursive PHP program to
// count total number of
// handshakes when a person
// can shake hand with only one.

// function to find all
// possible handshakes
function handshake($n)
{
    // when n becomes 0 that means
    // all the persons have done
    // handshake with other
    if ($n == 0)
        return 0;
    else
        return ($n - 1) + handshake($n - 1);
}

// Driver Code
$n = 9;
echo(handshake($n));

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

Output:

36

We can come up with a **direct formula** by expanding the recursion.

$$\begin{aligned}\text{handshake}(n) &= (n-1) + \text{handshake}(n-1) \\ &= (n-1) + (n-2) + \text{handshake}(n-2) \\ &= (n-1) + (n-2) + \dots + 1 + 0 \\ &= n * (n - 1)/2\end{aligned}$$

C++

```
// Recursive CPP program to count total
// number of handshakes when a person
// can shake hand with only one.
#include <stdio.h>

// function to find all possible handshakes
int handshake(int n)
{
    return n * (n - 1)/2;
}

int main()
{
    int n = 9;
    printf("%d", handshake(n));
    return 0;
}
```

Java

```
// Recursive Java program to
// count total number of
// handshakes when a person
// can shake hand with only one.
class GFG
{
    // function to find all
    // possible handshakes
    static int handshake(int n)
    {
        return n * (n - 1) / 2;
    }

    // Driver code
    public static void main(String args[])
    {
        int n = 9;
        System.out.println(handshake(n));
    }
}

// This code is contributed
// by Arnab Kundu
```

Python3

```
# Recursive Python program
```

```
# to count total number of
# handshakes when a person
# can shake hand with only one.

# function to find all
# possible handshakes
def handshake(n):

    return int(n * (n - 1) / 2)

# Driver Code
n = 9
print(handshake(n))

# This code is contributed
# by Shivi_Aggarwal
```

PHP

```
<?php
// Recursive PHP program to
// count total number of
// handshakes when a person
// can shake hand with only one.

// function to find all
// possible handshakes
function handshake($n)
{
    return $n * ($n - 1) / 2;
}

// Driver Code
$n = 9;
echo(handshake($n));

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

Output:

36

Improved By : [Chandan_Kumar](#), [andrew1234](#), [Shivi_Aggarwal](#)

Source

<https://www.geeksforgeeks.org/number-of-handshakes-such-that-a-person-shakes-hands-only-once/>

Chapter 79

Number of non-negative integral solutions of sum equation

Number of non-negative integral solutions of sum equation - GeeksforGeeks

Given a number n (number of variables) and val (sum of the variables), find out how many such non negative integral solutions are possible.

Examples :

Input : $n = 5, val = 1$
Output : 5
Explanation:
 $x_1 + x_2 + x_3 + x_4 + x_5 = 1$
Number of possible solution are :
(0 0 0 0 1), (0 0 0 1 0), (0 0 1 0 0),
(0 1 0 0 0), (1 0 0 0 0)
Total number of possible solutions are 5

Input : $n = 5, val = 4$
Output : 70
Explanation:
 $x_1 + x_2 + x_3 + x_4 + x_5 = 4$
Number of possible solution are:
(1 1 1 1 0), (1 0 1 1 1), (0 1 1 1 1),
(2 1 0 0 1), (2 2 0 0 0)..... so on.....
Total numbers of possible solutions are 70

Asked in: Microsoft Interview

1. Make a recursive function call to `countSolutions(int n, int val)`
2. Call this Solution function `countSolutions(n-1, val-i)` until $n = 1$ and $val \geq 0$ and then return 1.

Below is the implementation of above approach:

C++

```
// CPP program to find the numbers
// of non negative integral solutions
#include<iostream>
using namespace std;

// return number of non negative
// integral solutions
int countSolutions(int n, int val)
{
    // initialize total = 0
    int total = 0;

    // Base Case if n = 1 and val >= 0
    // then it should return 1
    if (n == 1 && val >= 0)
        return 1;

    // iterate the loop till equal the val
    for (int i = 0; i <= val; i++){

        // total solution of of equations
        // and again call the recursive
        // function Solutions(variable,value)
        total += countSolutions(n-1, val-i);
    }

    // return the total no possible solution
    return total;
}

// driver code
int main(){

    int n = 5;
    int val = 20;

    cout<<countSolutions(n, val);
}

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

Java

```
// Java program to find the numbers
```

```
// of non negative integral solutions
class GFG {

    // return number of non negative
    // integral solutions
    static int countSolutions(int n, int val) {

        // initialize total = 0
        int total = 0;

        // Base Case if n = 1 and val >= 0
        // then it should return 1
        if (n == 1 && val >= 0)
            return 1;

        // iterate the loop till equal the val
        for (int i = 0; i <= val; i++) {

            // total solution of of equations
            // and again call the recursive
            // function Solutions(variable, value)
            total += countSolutions(n - 1, val - i);
        }

        // return the total no possible solution
        return total;
    }

    // Driver code
    public static void main(String[] args) {
        int n = 5;
        int val = 20;

        System.out.print(countSolutions(n, val));
    }
}

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

Python3

```
# Python3 program to find the numbers
# of non negative integral solutions

# return number of non negative
# integral solutions
def countSolutions(n, val):
```

```
# initialize total = 0
total = 0

# Base Case if n = 1 and val >= 0
# then it should return 1
if n == 1 and val >= 0:
    return 1

# iterate the loop till equal the val
for i in range(val+1):

    # total solution of of equations
    # and again call the recursive
    # function Solutions(variable,value)
    total += countSolutions(n-1, val-i)

# return the total no possible solution
return total

# driver code
n = 5
val = 20
print(countSolutions(n, val))
```

C#

```
// C# program to find the numbers
// of non negative integral solutions
using System;

class GFG {

    // return number of non negative
    // integral solutions
    static int countSolutions(int n, int val) {

        // initialize total = 0
        int total = 0;

        // Base Case if n = 1 and val >= 0
        // then it should return 1
        if (n == 1 && val >= 0)
            return 1;

        // iterate the loop till equal the val
        for (int i = 0; i <= val; i++) {

            // total solution of of equations
```



```
        // and again call the recursive
        // function Solutions(variable, value)
        total += countSolutions(n - 1, val - i);
    }

    // return the total no possible solution
    return total;
}

// Driver code
public static void Main()
{
    int n = 5;
    int val = 20;

    Console.WriteLine(countSolutions(n, val));
}

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

PHP

```
<?php
// PHP program to find the numbers
// of non negative integral solutions

// return number of non negative
// integral solutions
function countSolutions($n, $val)
{
    // initialize total = 0
    $total = 0;

    // Base Case if n = 1 and
    // val >= 0 then it should
    // return 1
    if ($n == 1 && $val >= 0)
        return 1;

    // iterate the loop
    // till equal the val
    for ($i = 0; $i <= $val; $i++)
    {

        // total solution of equations
        // and again call the recursive
        // function Solutions(variable,value)
```

```
        $total += countSolutions($n - 1,
                                $val - $i);
    }

    // return the total
    // no possible solution
    return $total;
}

// Driver Code
$n = 5;
$val = 20;

echo countSolutions($n, $val);

// This code is contributed by nitin mittal.
?>
```

Output :

10626

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/number-of-non-negative-integral-solutions-of-sum-equation/>

Chapter 80

Number of ways to divide a given number as a set of integers in decreasing order

Number of ways to divide a given number as a set of integers in decreasing order - Geeks-forGeeks

Given two numbers **a** and **m**. The task is to find the number of ways in which **a** can be represented by a set

such that

and the summation of these numbers is equal to **a**. Also $1 \leq c \leq m$ (maximum size of the set cannot exceed **m**).

Examples:

Input : a = 4, m = 4

Output : 2 -> ({4}, {3, 1})

Note: {2, 2} is not a valid set as values are not in decreasing order

Input : a = 7, m = 5

Output : 5 -> ({7}, {6, 1}, {5, 2}, {4, 3}, {4, 2, 1})

Approach: This problem can be solved by Divide and Conquer using a recursive approach which follows the following conditions:

- If **a** is equal to zero, one solution has been found.
- If **a** > 0 and **m** == 0, this set violates the condition as no further values can be added in the set.
- If calculation has already been done for given values of **a**, **m** and **prev** (last value included in the current set), return that value.
- Start a loop from **i** = **a** till 0 and if **i** < **prev**, count the number of solutions if we include **i** in the current set and return it.

Below is the implementation of the above approach:

```
# Python3 code to calculate the number of ways
# in which a given number can be represented
# as set of finite numbers

# Import function to initialize the dictionary
from collections import defaultdict

# Initialize dictionary which is used
# to check if given solution is already
# visited or not to avoid
# calculating it again
visited = defaultdict(lambda : False)

# Initialize dictionary which is used to
# store the number of ways in which solution
# can be obtained for given values
numWays = defaultdict(lambda : 0)

# This function returns the total number
# of sets which satisfy given criteria
# a --> number to be divided into sets
# m --> maximum possible size of the set
# x --> previously selected value
def countNumOfWays(a, m, prev):

    # number is divided properly and
    # hence solution is obtained
    if a == 0:
        return 1

    # Solution can't be obtained
    elif a > 0 and m == 0:
        return 0

    # Return the solution if it has
    # already been calculated
    elif visited[(a, m, prev)] == True:
        return numWays[(a, m, prev)]

    else:
        visited[(a, m, prev)] = True

        for i in range(a, -1, -1):
            # Continue only if current value is
            # smaller compared to previous value
            if i < prev:
```

```
        numWays[(a,m,prev)] += countNumOfWays(a-i,m-1,i)

    return numWays[(a, m, prev)]

# Values of 'a' and 'm' for which
# solution is to be found
# MAX_CONST is extremely large value
# used for first comparison in the function
a, m, MAX_CONST = 7, 5, 10**5
print(countNumOfWays(a, m, MAX_CONST))
```

Output:

5

Time Complexity: $O(a \cdot \log(a))$

Source

<https://www.geeksforgeeks.org/number-of-ways-to-divide-a-given-number-as-a-set-of-integers-in-decreasing-order/>

Chapter 81

Partition given string in such manner that i'th substring is sum of (i-1)'th and (i-2)'th substring

Partition given string in such manner that i'th substring is sum of (i-1)'th and (i-2)'th substring - GeeksforGeeks

Partition given string in such manner that i'th substring is sum of (i-1)'th and (i-2)'nd substring.

Examples:

Input : "11235813"
Output : ["1", "1", "2", "3", "5", "8", "13"]

Input : "1111223"
Output : ["1", "11", "12", "23"]

Input : "1111213"
Output : ["11", "1", "12", "13"]

Input : "11121114"
Output : []

1. Iterate through the given string by picking 3 numbers (first, seconds and third) at a time starting from one digit each.
2. If first + second = third, recursively call check() with second as first and third as second. The third is picked based on next possible number of digits. (The result of addition of two

- numbers can have a max. of second's & third's digits + 1)
3. Else, first increment the third (by adding more digits) till the limit (Here limit is sum of first and second).
 4. After incrementing third, following cases arise.
 - a) When doesn't match, increment the second offset.
 - b) When doesn't match, increment the first offset.
 - c) Note: Once a call to check() is made after incrementing the third offset, do not alter the second or first, as those are already finalized.
 5. When the end of the string is reached and the condition is satisfied, add "second" and "third" to the empty list. While rolling back the recursive stack, prepend the "first" to the list so the order is preserved.

```
// Java program to check if we can partition a
// string in a way that value of i-th string is
// sum of (i-1)-th and (i-2)-th substrings.
import java.util.LinkedList;

public class SumArray {

    private static LinkedList<Integer> resultList =
        new LinkedList<>();

    private static boolean check(char[] chars, int offset1,
        int offset2, int offset3, boolean freezeFirstAndSecond) {

        // Find subarrays according to given offsets
        int first = intOf(subArr(chars, 0, offset1));
        int second = intOf(subArr(chars, offset1, offset2));
        int third = intOf(subArr(chars, offset1 + offset2, offset3));

        // If condition is satisfied for current subarrays
        if (first + second == third) {

            // If whole array is covered.
            if (offset1 + offset2 + offset3 >= chars.length) {
                addIntermediateResults(first, second, third);
                return true;
            }

            // Check if remaining array also satisfies the condition
            boolean result = check(subArr(chars, offset1,
                chars.length - offset1), offset2, offset3,
                Math.max(offset2, offset3), true);
            if (result)
                addIntermediateResults(first, second, third);
            return result;
        }
    }
}
```

```
// If not satisfied, try incrementing third
if (isValidOffset(offset1, offset2, 1 + offset3, chars.length)) {
    if (check(chars, offset1, offset2, 1 + offset3, false))
        return true;
}

// If first and second have been finalized, do not
// alter already computed results
if (freezeFirstAndSecond)
    return false;

// If first and second are not finalized
if (isValidOffset(offset1, 1 + offset2, Math.max(offset1,
    1 + offset2), chars.length)) {

    // Try incrementing second
    if (check(chars, offset1, 1 + offset2,
        Math.max(offset1, 1 + offset2), false))
        return true;
}

// Try incrementing first
if (isValidOffset(1 + offset1, offset2, Math.max(1 + offset1,
    offset2), chars.length)) {
    if (check(chars, 1 + offset1, offset2, Math.max(1 + offset1,
        offset2), false))
        return true;
}
return false;
}

private static void addIntermediateResults(int first,
    int second, int third) {
    if (resultList.isEmpty()) {
        resultList.add(second);
        resultList.add(third);
    }
    resultList.addFirst(first);
}

// Check if given three offsets are valid (Within array length
// and third offset can represent sum of first two)
private static boolean isValidOffset(int offset1, int offset2,
    int offset3, int length) {
    return (offset1 + offset2 + offset3 <= length &&
        (offset3 == Math.max(offset1, offset2) ||
            offset3 == 1 + Math.max(offset1, offset2)));
}
```



```
// To get a subarray with starting from given
// index and offset
private static char[] subArr(char[] chars, int index, int offset) {
    int trueOffset = Math.min(chars.length - index, offset);
    char[] destArr = new char[trueOffset];
    System.arraycopy(chars, index, destArr, 0, trueOffset);
    return destArr;
}

private static int intOf(char... chars) {
    return Integer.valueOf(new String(chars));
}

public static void main(String[] args) {
    String numStr = "11235813";
    char[] chars = numStr.toCharArray();
    System.out.println(check(chars, 1, 1, 1, false));
    System.out.println(resultList);
}
}
```

Output:

```
true
[1, 1, 2, 3, 5, 8, 13]
```

Source

<https://www.geeksforgeeks.org/partition-given-string-manner-ith-substring-sum-1th-2th-substring/>

Chapter 82

Perform n steps to convert every digit of a number in the format [count][digit]

Perform n steps to convert every digit of a number in the format [count][digit] - Geeks-forGeeks

Given a number *num* as a string and a number N. The task is to write a program which converts the given number *num* to another number after performing N steps. At each step, every digit of num will be written in the format [count][digit] in the new number, where *count* is the number of times a digit occurs consecutively in num.

Examples:

Input: num = "123"; n = 3

Output: 1321123113

For, n = 1: 123 becomes 1 time 1, 1 time 2, 1 time 3, hence number 111213

For, n = 2: 3 times 1, 1 time 2, 1 time 1, 1 time 3, hence number 31121113

For, n = 3: 1 time 3, 2 times 1, 1 time 2, 3 times 1, 1 time 3, hence number 1321123113

Input: num = "1213"; n = 1

Output: 11121113

Approach: Parse the string's characters as a single digit and maintain a count for that digit till a different digit is found. Once a different digit is found, add the count of the digit to the new string and number to it. Once the string is parsed completely, recur for the function again with this new string till n steps are done.

Below is the implementation of the above approach:

```
// C++ program to convert number
```

```
// to the format [count][digit] at every step
#include <bits/stdc++.h>
using namespace std;

// Function to perform every step
void countDigits(string st, int n)
{
    // perform N steps
    if (n > 0) {
        int cnt = 1, i;
        string st2 = "";

        // Traverse in the string
        for (i = 1; i < st.length(); i++) {
            if (st[i] == st[i - 1])
                cnt++;
            else {
                st2 += ('0' + cnt);
                st2 += st[i - 1];
                cnt = 1;
            }
        }

        // for last digit
        st2 += ('0' + cnt);
        st2 += st[i - 1];

        // recur for current string
        countDigits(st2, --n);
    }

    else
        cout << st;
}

// Driver Code
int main()
{
    string num = "123";
    int n = 3;

    countDigits(num, n);

    return 0;
}
```

Chapter 82. Perform n steps to convert every digit of a number in the format [count][digit]

Output:

1321123113

Source

<https://www.geeksforgeeks.org/perform-n-steps-to-convert-every-digit-of-a-number-in-the-format-countdigit/>

Chapter 83

Power Set in Lexicographic order

Power Set in Lexicographic order - GeeksforGeeks

This article is about generating [Power set](#) in lexicographical order.

Examples :

Input : abc
Output : a ab abc ac b bc c

The idea is to sort array first. After sorting, one by one fix characters and recursively generates all subsets starting from them. After every recursive call, we remove last character so that next permutation can be generated.

C++

```
// CPP program to generate power set in
// lexicographic order.
#include <bits/stdc++.h>
using namespace std;

// str : Stores input string
// n : Length of str.
// curr : Stores current permutation
// index : Index in current permutation, curr
void permuteRec(string str, int n,
               int index = -1, string curr = "")
{
    // base case
```

```
    if (index == n)
        return;

    cout << curr << "\n";
    for (int i = index + 1; i < n; i++) {

        curr += str[i];
        permuteRec(str, n, i, curr);

        // backtracking
        curr = curr.erase(curr.size() - 1);
    }
    return;
}

// Generates power set in lexicographic
// order.
void powerSet(string str)
{
    sort(str.begin(), str.end());
    permuteRec(str, str.size());
}

// Driver code
int main()
{
    string str = "cab";
    powerSet(str);
    return 0;
}
```

PHP

```
<?php
// PHP program to generate power
// set in lexicographic order.

// str : Stores input string
// n : Length of str.
// curr : Stores current permutation
// index : Index in current permutation, curr
function permuteRec($str, $n, $index = -1,
                    $curr = "")
{
    // base case
    if ($index == $n)
        return;
```

```
    echo $curr."\n";
    for ($i = $index + 1; $i < $n; $i++)
    {

        $curr=$curr.$str[$i];
        permuteRec($str, $n, $i, $curr);

        // backtracking
        $curr = "";
    }
    return;
}

// Generates power set in lexicographic
// order.
function powerSet($str)
{

    $str = str_split($str);
    sort($str);
    permuteRec($str, sizeof($str));
}

// Driver code
$str = "cab";
powerSet($str);

// This code is contributed by Mithun Kumar
?>
```

Output :

```
a
ab
abc
ac
b
bc
c
```

Improved By : [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/powet-set-lexicographic-order/>

Chapter 84

Practice Questions for Recursion | Set 1

Practice Questions for Recursion | Set 1 - GeeksforGeeks

Explain the functionality of following functions.

Question 1

```
int fun1(int x, int y)
{
    if(x == 0)
        return y;
    else
        return fun1(x - 1, x + y);
}
```

Answer: The function fun() calculates and returns $((1 + 2 \dots + x-1 + x) + y)$ which is $x(x+1)/2 + y$. For example if x is 5 and y is 2, then fun should return $15 + 2 = 17$.

Question 2

```
void fun2(int arr[], int start_index, int end_index)
{
    if(start_index >= end_index)
        return;
    int min_index;
    int temp;

    /* Assume that minIndex() returns index of minimum value in
       array arr[start_index...end_index] */
    min_index = minIndex(arr, start_index, end_index);
```



```
temp = arr[start_index];
arr[start_index] = arr[min_index];
arr[min_index] = temp;

fun2(arr, start_index + 1, end_index);
}
```

Answer: The function fun2() is a recursive implementation of [Selection Sort](#).

Source

<https://www.geeksforgeeks.org/practice-questions-for-recursion/>

Chapter 85

Practice Questions for Recursion | Set 2

Practice Questions for Recursion | Set 2 - GeeksforGeeks

Explain the functionality of following functions.

Question 1

```
/* Assume that n is greater than or equal to 1 */
int fun1(int n)
{
    if(n == 1)
        return 0;
    else
        return 1 + fun1(n/2);
}
```

Answer: The function calculates and returns $\lfloor \log_2(n) \rfloor$. For example, if n is between 8 and 15 then fun1() returns 3. If n is between 16 to 31 then fun1() returns 4.

Question 2

```
/* Assume that n is greater than or equal to 0 */
void fun2(int n)
{
    if(n == 0)
        return;

    fun2(n/2);
    printf("%d", n%2);
}
```

Answer: The function `fun2()` prints binary equivalent of `n`. For example, if `n` is 21 then `fun2()` prints 10101.

Note that above functions are just for practicing recursion, they are not the ideal implementation of the functionality they provide.

Source

<https://www.geeksforgeeks.org/practice-questions-for-recursion-set-2/>

Chapter 86

Practice Questions for Recursion | Set 3

Practice Questions for Recursion | Set 3 - GeeksforGeeks

Explain the functionality of below recursive functions.

Question 1

```
void fun1(int n)
{
    int i = 0;
    if (n > 1)
        fun1(n-1);
    for (i = 0; i < n; i++)
        printf(" * ");
}
```

Answer: Total numbers of stars printed is equal to $1 + 2 + \dots + (n-2) + (n-1) + n$, which is $n(n+1)/2$.

Question 2

```
#define LIMIT 1000
void fun2(int n)
{
    if (n <= 0)
        return;
    if (n > LIMIT)
        return;
    printf("%d ", n);
    fun2(2*n);
    printf("%d ", n);
}
```

Answer: For a positive n , $\text{fun2}(n)$ prints the values of n , $2n$, $4n$, $8n$... while the value is smaller than LIMIT . After printing values in increasing order, it prints same numbers again in reverse order. For example $\text{fun2}(100)$ prints 100, 200, 400, 800, 800, 400, 200, 100. If n is negative, the function is returned immediately.

Improved By : [hoangdang](#)

Source

<https://www.geeksforgeeks.org/practice-questions-for-recursion-set-3/>

Chapter 87

Practice Questions for Recursion | Set 4

Practice Questions for Recursion | Set 4 - GeeksforGeeks

Question 1

Predict the output of following program.

```
#include<stdio.h>
void fun(int x)
{
    if(x > 0)
    {
        fun(--x);
        printf("%d\t", x);
        fun(--x);
    }
}

int main()
{
    int a = 4;
    fun(a);
    getchar();
    return 0;
}
```

Output: 0 1 2 0 3 0 1

```
fun(4);
/
```

```
        fun(3), print(3), fun(2)(prints 0 1)
      /
    fun(2), print(2), fun(1)(prints 0)
  /
fun(1), print(1), fun(0)(does nothing)
/
fun(0), print(0), fun(-1) (does nothing)
```

Question 2

Predict the output of following program. What does the following fun() do in general?

```
int fun(int a[],int n)
{
    int x;
    if(n == 1)
        return a[0];
    else
        x = fun(a, n-1);
    if(x > a[n-1])
        return x;
    else
        return a[n-1];
}

int main()
{
    int arr[] = {12, 10, 30, 50, 100};
    printf(" %d ", fun(arr, 5));
    getchar();
    return 0;
}
```

Output: 100

fun() returns the maximum value in the input array a[] of size n.

Question 3

Predict the output of following program. What does the following fun() do in general?

```
int fun(int i)
{
    if ( i%2 ) return (i++);
    else return fun(fun( i - 1 ));
}

int main()
{
    printf(" %d ", fun(200));
    getchar();
}
```

```
    return 0;  
}
```

Output: 199

If n is odd then returns n, else returns (n-1). Eg., for n = 12, you get 11 and for n = 11 you get 11. The statement *“return i++;”* returns value of i only as it is a post increment.

Source

<https://www.geeksforgeeks.org/practice-questions-for-recursion-set-4/>

Chapter 88

Practice Questions for Recursion | Set 5

Practice Questions for Recursion | Set 5 - GeeksforGeeks

Question 1

Predict the output of following program. What does the following fun() do in general?

```
#include<stdio.h>

int fun(int a, int b)
{
    if (b == 0)
        return 0;
    if (b % 2 == 0)
        return fun(a+a, b/2);

    return fun(a+a, b/2) + a;
}

int main()
{
    printf("%d", fun(4, 3));
    getchar();
    return 0;
}
```

Output: 12

It calculates $a*b$ (a multiplied b).

Question 2

In question 1, if we replace $+$ with $*$ and replace return 0 with return 1, then what does the changed function do? Following is the changed function.

```
#include<stdio.h>

int fun(int a, int b)
{
    if (b == 0)
        return 1;
    if (b % 2 == 0)
        return fun(a*a, b/2);

    return fun(a*a, b/2)*a;
}

int main()
{
    printf("%d", fun(4, 3));
    getchar();
    return 0;
}
```

Output: 64

It calculates a^b (a raised to power b).

Question 3

Predict the output of following program. What does the following fun() do in general?

```
#include<stdio.h>

int fun(int n)
{
    if (n > 100)
        return n - 10;
    return fun(fun(n+11));
}

int main()
{
    printf(" %d ", fun(99));
    getchar();
    return 0;
}
```

Output: 91

```
fun(99) = fun(fun(110)) since 99 < 100
        = fun(100)      since 110 > 100
        = fun(fun(111)) since 100 < 100
        = fun(101)      since 111 > 100
        = 91             since 101 > 100
```

Returned value of fun() is 91 for all integer arguments $n \leq 101$, and $n - 10$ for $n > 101$. This function is known as [McCarthy 91 function](#).

Source

<https://www.geeksforgeeks.org/practice-questions-for-recursion-set-5/>

Chapter 89

Practice Questions for Recursion | Set 6

Practice Questions for Recursion | Set 6 - GeeksforGeeks

Question 1

Consider the following recursive C function. Let *len* be the length of the string *s* and *num* be the number of characters printed on the screen, give the relation between *num* and *len* where *len* is always greater than 0.

```
void abc(char *s)
{
    if(s[0] == '\0')
        return;

    abc(s + 1);
    abc(s + 1);
    printf("%c", s[0]);
}
```

Following is the relation between *num* and *len*.

$$\text{num} = 2^{\text{len}} - 1$$

```
s[0] is 1 time printed
s[1] is 2 times printed
s[2] is 4 times printed
s[i] is printed 2i times
s[strlen(s)-1] is printed 2(strlen(s)-1) times
total = 1+2+...+2(strlen(s)-1)
       = (2strlen(s)) - 1
```

For example, the following program prints 7 characters.

```
#include<stdio.h>

void abc(char *s)
{
    if(s[0] == '\\0')
        return;

    abc(s + 1);
    abc(s + 1);
    printf("%c", s[0]);
}

int main()
{
    abc("xyz");
    return 0;
}
```

Thanks to bharat nag for suggesting this solution.

Question 2

```
#include<stdio.h>
int fun(int count)
{
    printf("%d\\n", count);
    if(count < 3)
    {
        fun(fun(fun(++count)));
    }
    return count;
}

int main()
{
    fun(1);
    return 0;
}
```

Output:

```
1
2
3
```

3
3
3
3

The `main()` function calls `fun(1)`. `fun(1)` prints “1” and calls `fun(fun(fun(2)))`. `fun(2)` prints “2” and calls `fun(fun(fun(3)))`. So the function call sequence becomes `fun(fun(fun(fun(fun(3)))))`. `fun(3)` prints “3” and returns 3 (note that count is not incremented and no more functions are called as the if condition is not true for count 3). So the function call sequence reduces to `fun(fun(fun(fun(3))))`. `fun(3)` again prints “3” and returns 3. So the function call again reduces to `fun(fun(fun(3)))` which again prints “3” and reduces to `fun(fun(3))`. This continues and we get “3” printed 5 times on the screen.

Source

<https://www.geeksforgeeks.org/practice-questions-for-recursion-set-6/>

Chapter 90

Practice Questions for Recursion | Set 7

Practice Questions for Recursion | Set 7 - GeeksforGeeks

Question 1 Predict the output of the following program. What does the following fun() do in general?

```
#include <stdio.h>

int fun ( int n, int *fp )
{
    int t, f;

    if ( n <= 1 )
    {
        *fp = 1;
        return 1;
    }
    t = fun ( n-1, fp );
    f = t + *fp;
    *fp = t;
    return f;
}

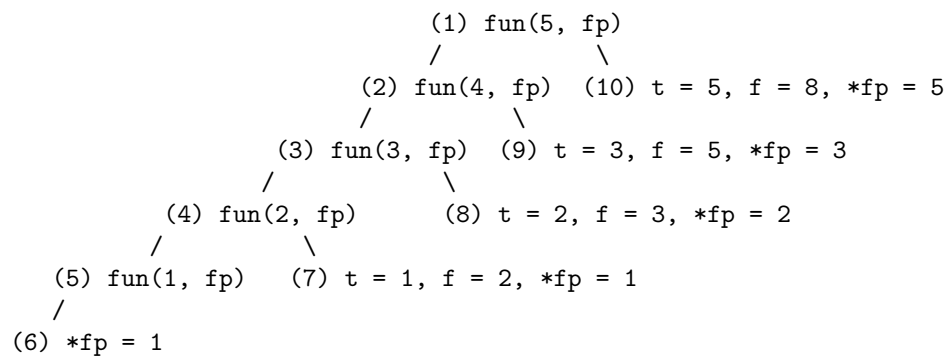
int main()
{
    int x = 15;
    printf("%d\n",fun(5, &x));

    return 0;
}
```

Output:

8

The program calculates nth Fibonacci Number. The statement `t = fun (n-1, fp)` gives the (n-1)th Fibonacci number and `*fp` is used to store the (n-2)th Fibonacci Number. Initial value of `*fp` (which is 15 in the above program) doesn't matter. Following recursion tree shows all steps from 1 to 10, for execution of `fun(5, &x)`.



Question 2: Predict the output of the following program.

```
#include <stdio.h>

void fun(int n)
{
    if(n > 0)
    {
        fun(n-1);
        printf("%d ", n);
        fun(n-1);
    }
}

int main()
{
    fun(4);
    return 0;
}
```

Output

1 2 1 3 1 2 1 4 1 2 1 3 1 2 1


```
        fun(4)
      /
    fun(3), print(4), fun(3) [fun(3) prints 1 2 1 3 1 2 1]
  /
fun(2), print(3), fun(2) [fun(2) prints 1 2 1]
/
fun(1), print(2), fun(1) [fun(1) prints 1]
/
fun(0), print(1), fun(0) [fun(0) does nothing]
```

Source

<https://www.geeksforgeeks.org/practice-questions-for-recursion-set-7/>

Chapter 91

Practice questions for Linked List and Recursion

Practice questions for Linked List and Recursion - GeeksforGeeks

Assume the structure of a Linked List node is as follows.

```
struct Node
{
    int data;
    struct Node *next;
};
```

Explain the functionality of following C functions.

1. What does the following function do for a given Linked List?

```
void fun1(struct Node* head)
{
    if(head == NULL)
        return;

    fun1(head->next);
    printf("%d ", head->data);
}
```

fun1() prints the given Linked List in reverse manner. For Linked List 1->2->3->4->5, fun1() prints 5->4->3->2->1.

2. What does the following function do for a given Linked List ?

```
void fun2(struct Node* head)
```

```
{
    if(head== NULL)
        return;
    printf("%d ", head->data);

    if(head->next != NULL )
        fun2(head->next->next);
    printf("%d ", head->data);
}
```

fun2() prints alternate nodes of the given Linked List, first from head to end, and then from end to head. If Linked List has even number of nodes, then fun2() skips the last node. For Linked List 1->2->3->4->5, fun2() prints 1 3 5 5 3 1. For Linked List 1->2->3->4->5->6, fun2() prints 1 3 5 5 3 1.

Below is a complete running program to test above functions.

```
#include<stdio.h>
#include<stdlib.h>

/* A linked list node */
struct Node
{
    int data;
    struct Node *next;
};

/* Prints a linked list in reverse manner */
void fun1(struct Node* head)
{
    if(head == NULL)
        return;

    fun1(head->next);
    printf("%d ", head->data);
}

/* prints alternate nodes of a Linked List, first
   from head to end, and then from end to head. */
void fun2(struct Node* start)
{
    if(start == NULL)
        return;
    printf("%d ", start->data);

    if(start->next != NULL )
        fun2(start->next->next);
}
```

```
    printf("%d ", start->data);
}

/* UTILITY FUNCTIONS TO TEST fun1() and fun2() */
/* Given a reference (pointer to pointer) to the head
   of a list and an int, push a new node on the front
   of the list. */
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Drier program to test above functions */
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;

    /* Using push() to construct below list
       1->2->3->4->5 */
    push(&head, 5);
    push(&head, 4);
    push(&head, 3);
    push(&head, 2);
    push(&head, 1);

    printf("\n Output of fun1() for list 1->2->3->4->5 \n");
    fun1(head);

    printf("\n Output of fun2() for list 1->2->3->4->5 \n");
    fun2(head);

    getchar();
    return 0;
}
```

Source

<https://www.geeksforgeeks.org/practice-questions-for-linked-list-and-recursion/>

Chapter 92

Print 1 to 100 in C++, without loop and recursion

Print 1 to 100 in C++, without loop and recursion - GeeksforGeeks

Following is a C++ program that prints 1 to 100 without loop and without recursion.

```
#include <iostream>
using namespace std;

template<int N>
class PrintOneToN
{
public:
    static void print()
    {
        PrintOneToN<N-1>::print(); // Note that this is not recursion
        cout << N << endl;
    }
};

template<>
class PrintOneToN<1>
{
public:
    static void print()
    {
        cout << 1 << endl;
    }
};

int main()
{
    const int N = 100;
```

```
    PrintOneToN<N>::print();  
    return 0;  
}
```

Output:

```
1  
2  
3  
..  
..  
98  
99  
100
```

The program prints all numbers from 1 to n without using a loop and recursion. The concept used in this program is [Template Metaprogramming](#).

Let us see how this works. [Templates in C++](#) allow non-datatypes also as parameter. Non-datatype means a value, not a datatype. For example, in the above program, N is passed as a value which is not a datatype. A new instance of a generic class is created for every parameter and these classes are created at compile time. In the above program, when compiler sees the statement “PrintOneToN<>::print()” with N = 100, it creates an instance PrintOneToN<100>. In function PrintOneToN<100>::print(), another function PrintOneToN<99>::print() is called, therefore an instance PrintOneToN<99> is created. Similarly, all instances from PrintOneToN<100> to PrintOneToN<2> are created. PrintOneToN<1>::print() is already there and prints 1. The function PrintOneToN<2> prints 2 and so on. Therefore we get all numbers from 1 to N printed on the screen.

Following is **another approach** to print 1 to 100 without loop and recursion.

```
#include<iostream>  
using namespace std;  
  
class A  
{  
public:  
    static int a;  
    A()  
    { cout<<a++<<endl; }  
};  
  
int A::a = 1;  
  
int main()  
{  
    int N = 100;  
    A obj[N];  
    return 0;  
}
```

The output of this program is same as above program. In the above program, class A has a static variable 'a', which is incremented with every instance of A. The default constructor of class A prints the value of 'a'. When we create an array of objects of type A, the default constructor is called for all objects one by one. Value of 'a' is printed and incremented with every call. Therefore, we get all values from 1 to 100 printed on the screen.

Thanks to *Lakshmanan* for suggesting this approach.

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/output-of-c-program-set-18-3/>

Chapter 93

Print N-bit binary numbers having more 1's than 0's in all prefixes

Print N-bit binary numbers having more 1's than 0's in all prefixes - GeeksforGeeks

Given a positive integer n, print all n-bit binary numbers having more 1's than 0's for any prefix of the number.

Examples:

Input : n = 2
Output : 11 10

Input : n = 4
Output : 1111 1110 1101 1100 1011 1010

A simple but not efficient solution will be to generate all N-bit binary numbers and print those numbers that satisfy the conditions. The time complexity of this solution is exponential.

An efficient solution is to generate only those N-bit numbers that satisfy the given conditions. We use recursion. At each point in the recursion, we append 0 and 1 to the partially formed number and recur with one less digit.

C++

```
// CPP program to print all N-bit binary
#include <bits/stdc++.h>
using namespace std;
```

```
/* function to generate n digit numbers*/
void printRec(string number, int extraOnes,
              int remainingPlaces)
{
    /* if number generated */
    if (0 == remainingPlaces) {
        cout << number << " ";
        return;
    }

    /* Append 1 at the current number and reduce
       the remaining places by one */
    printRec(number + "1", extraOnes + 1,
              remainingPlaces - 1);

    /* If more ones than zeros, append 0 to the
       current number and reduce the remaining
       places by one*/
    if (0 < extraOnes)
        printRec(number + "0", extraOnes - 1,
                  remainingPlaces - 1);
}

void printNums(int n)
{
    string str = "";
    printRec(str, 0, n);
}

/*driver function*/
int main()
{
    int n = 4;
    printNums(n);
    return 0;
}
```

Java

```
// java program to print all N-bit binary
import java.io.*;

class GFG
{
    // function to generate n digit numbers
    static void printRec(String number, int extraOnes,
                          int remainingPlaces)
    {
```

```
// if number generated
if (0 == remainingPlaces)
{
    System.out.print( number + " ");
    return;
}

// Append 1 at the current number and
// reduce the remaining places by one
printRec(number + "1", extraOnes + 1,
          remainingPlaces - 1);

// If more ones than zeros, append 0 to the
// current number and reduce the remaining
// places by one
if (0 < extraOnes)
    printRec(number + "0", extraOnes - 1,
              remainingPlaces - 1);
}

static void printNums(int n)
{
    String str = "";
    printRec(str, 0, n);
}

// Driver function
public static void main (String[] args)
{
    int n = 4;
    printNums(n);
}
}
```

// This code is contributed by vt_m

C#

```
// C# program to print all N-bit binary
using System;

class GFG {

    // function to generate n digit numbers
    static void printRec(String number,
                          int extraOnes,
                          int remainingPlaces)
```

```
{

    // if number generated
    if (0 == remainingPlaces)
    {
        Console.Write( number + " ");
        return;
    }

    // Append 1 at the current number and
    // reduce the remaining places by one
    printRec(number + "1", extraOnes + 1,
            remainingPlaces - 1);

    // If more ones than zeros, append
    // 0 to the current number and
    // reduce the remaining places
    // by one
    if (0 < extraOnes)
        printRec(number + "0", extraOnes - 1,
            remainingPlaces - 1);
}

static void printNums(int n)
{
    String str = "";
    printRec(str, 0, n);
}

// Driver code
public static void Main ()
{
    int n = 4;
    printNums(n);
}
}
```

// This code is contributed by Nitin Mittal.

Output:

1111 1110 1101 1100 1011 1010

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/print-n-bit-binary-numbers-1s-0s-prefixes/>

Chapter 94

Print a pattern without using any loop

Print a pattern without using any loop - GeeksforGeeks

Given a number n, print following pattern without using any loop.

Examples :

Input: n = 16
Output: 16, 11, 6, 1, -4, 1, 6, 11, 16

Input: n = 10
Output: 10, 5, 0, 5, 10

We basically first reduce 5 one by one until we reach a negative or 0. After we reach 0 or negative, we one add 5 until we reach n.

Source: [Microsoft Interview Question](#).

The idea is to use recursion. It is an interesting question to try on your own.

Below is the Code. The code uses a flag variable to indicate whether we are moving toward 0 or we are moving toward the back to n.

C++

```
// C++ program to print pattern that first reduces 5 one
// by one, then adds 5. Without any loop
#include <iostream>
using namespace std;

// Recursive function to print the pattern.
// n indicates input value
```

```
// m indicates current value to be printed
// flag indicates whether we need to add 5 or
// subtract 5. Initially flag is true.
void printPattern(int n, int m, bool flag)
{
    // Print m.
    cout << m << " ";

    // If we are moving back toward the n and
    // we have reached there, then we are done
    if (flag == false && n ==m)
        return;

    // If we are moving toward 0 or negative.
    if (flag)
    {
        // If m is greater, then 5, recur with true flag
        if (m-5 > 0)
            printPattern(n, m-5, true);
        else // recur with false flag
            printPattern(n, m-5, false);
    }
    else // If flag is false.
        printPattern(n, m+5, false);
}

// Driver Program
int main()
{
    int n = 16;
    printPattern(n, n, true);
    return 0;
}
```

Java

```
// Java program to print pattern that first reduces 5 one
// by one, then adds 5. Without any loop
import java.io.*;

class GFG {

    // Recursive function to print the pattern.
    // n indicates input value
    // m indicates current value to be printed
    // flag indicates whether we need to add 5 or
    // subtract 5. Initially flag is true.
    static void printPattern(int n, int m, boolean flag)
```

```
{

    // Print m.
    System.out.print(m + " ");

    // If we are moving back toward the n and
    // we have reached there, then we are done
    if (flag == false && n == m)
        return;

    // If we are moving toward 0 or negative.
    if (flag) {

        // If m is greater, then 5, recur with
        // true flag
        if (m - 5 > 0)
            printPattern(n, m - 5, true);

        else // recur with false flag
            printPattern(n, m - 5, false);
    }

    else // If flag is false.
        printPattern(n, m + 5, false);
}

// Driver Program
public static void main(String[] args)
{
    int n = 16;
    printPattern(n, n, true);
}
}
// This code is contributed by vt_m
```

C#

```
// C# program to print pattern that first reduces 5 one
// by one, then adds 5. Without any loop
using System;

class GFG {

    // Recursive function to print the pattern.
    // n indicates input value
    // m indicates current value to be printed
    // flag indicates whether we need to add 5 or
    // subtract 5. Initially flag is true.
```



```
static void printPattern(int n, int m, bool flag)
{
    // Print m.
    Console.Write(m + " ");

    // If we are moving back toward the n and
    // we have reached there, then we are done
    if (flag == false && n == m)
        return;

    // If we are moving toward 0 or negative.
    if (flag) {

        // If m is greater, then 5, recur with
        // true flag
        if (m - 5 > 0)
            printPattern(n, m - 5, true);

        else // recur with false flag
            printPattern(n, m - 5, false);
    }

    else // If flag is false.
        printPattern(n, m + 5, false);
}

// Driver Program
public static void Main()
{
    int n = 16;
    printPattern(n, n, true);
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP program to print pattern
// that first reduces 5 one by one,
// then adds 5. Without any loop

// Recursive function to print
// the pattern. n indicates input
// value m indicates current value
// to be printed flag indicates whether
// we need to add 5 or subtract 5.
```

```
// Initially flag is true.
function printPattern($n, $m, $flag)
{
    // Print m.
    echo $m , " ";

    // If we are moving back
    // toward the n and we
    // have reached there,
    // then we are done
    if ($flag == false && $n == $m)
        return;

    // If we are moving
    // toward 0 or negative.
    if ($flag)
    {
        // If m is greater, then 5,
        // recur with true flag
        if ($m - 5 > 0)
            printPattern($n, $m - 5, true);

        // recur with false flag
        else
            printPattern($n, $m - 5, false);
    }

    // If flag is false.
    else
        printPattern($n, $m + 5, false);
}

// Driver Code
$n = 16;
printPattern($n, $n, true);

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

Python3

```
# Python program to print pattern
# that first reduces 5 one by one,
# then adds 5. Without any loop.

# Recursive function to print
# the pattern.n indicates
# input value m indicates
```

```
# current value to be printed
# flag indicates whether we
# need to add 5 or subtract 5.
# Initially flag is True.
def printPattern(n, m, flag):

    # Print m.
    print(m)

    # If we are moving back
    # toward the n and we
    # have reached there,
    # then we are done
    if flag == False and n == m:
        return
    # If we are moving
    # toward 0 or negative.
    if flag:
        # If m is greater, then 5,
        # recur with true flag
        if m - 5 > 0:
            printPattern(n, m - 5, True)
        else: # recur with false flag
            printPattern(n, m - 5, False)
    else: # If flag is false.
        printPattern(n, m + 5, False)

# Driver Code
n = 16
printPattern(n, n, True)

# This code is contributed
# by HrushikeshChoudhary
```

Output :

16, 11, 6, 1, -4, 1, 6, 11, 16

How to print above pattern without any extra variable and loop?

The above program works fine and prints the desired out but uses extra variables. We can use two print statements. First one before the recursive call that prints all decreasing sequence. Second one after the recursive call to print increasing sequence. Below is the implementation of the idea.

C++

```
// C++ program to print pattern that first reduces 5 one
// by one, then adds 5. Without any loop an extra variable.
```

```
#include <iostream>
using namespace std;

// Recursive function to print the pattern without any extra
// variable
void printPattern(int n)
{
    // Base case (When n becomes 0 or negative)
    if (n == 0 || n < 0)
    {
        cout << n << " ";
        return;
    }

    // First print decreasing order
    cout << n << " ";
    printPattern(n-5);

    // Then print increasing order
    cout << n << " ";
}

// Driver Program
int main()
{
    int n = 16;
    printPattern(n);
    return 0;
}
```

Java

```
// Java program to print pattern that first
// reduces 5 one by one, then adds 5.
// Without any loop an extra variable.

import java.io.*;

class GFG {

    // Recursive function to print the
    // pattern without any extra variable
    static void printPattern(int n)
    {

        // Base case (When n becomes 0 or
        // negative)
        if (n == 0 || n < 0) {
```

```
        System.out.print(n + " ");

        return;
    }

    // First print decreasing order
    System.out.print(n + " ");

    printPattern(n - 5);

    // Then print increasing order
    System.out.print(n + " ");
}

// Driver Program
public static void main(String[] args)
{
    int n = 16;

    printPattern(n);
}

// This code is contributed by vt_m
```

C#

```
// C# program to print pattern that first
// reduces 5 one by one, then adds 5.
// Without any loop an extra variable.

using System;

class GFG {

    // Recursive function to print the
    // pattern without any extra variable
    static void printPattern(int n)
    {

        // Base case (When n becomes 0 or
        // negative)
        if (n == 0 || n < 0) {

            Console.Write(n + " ");
```

```
        return;
    }

    // First print decreasing order
    Console.Write(n + " ");

    printPattern(n - 5);

    // Then print increasing order
    Console.Write(n + " ");
}

// Driver Program
public static void Main()
{
    int n = 16;

    printPattern(n);
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP program to print pattern
// that first reduces 5 one
// by one, then adds 5. Without
// any loop an extra variable.

// Recursive function to print the
// pattern without any extra variable
function printPattern( $n)
{
    // Base case (When n becomes
    // 0 or negative)
    if ($n == 0 or $n < 0)
    {
        echo $n , " ";
        return;
    }

    // First print decreasing order
    echo $n , " ";
    printPattern($n-5);
}
```

```
// Then print increasing order
echo $n , " ";
}

// Driver Code
$n = 16;
printPattern($n);

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

Output:

16, 11, 6, 1, -4, 1, 6, 11, 16

Thanks to AKSHAY RATHORE for suggesting above solution.

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

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

Source

<https://www.geeksforgeeks.org/print-a-pattern-without-using-any-loop/>

Chapter 95

Print all combinations of factors (Ways to factorize)

Print all combinations of factors (Ways to factorize) - GeeksforGeeks

Write a program to print all the combinations of factors of given number n.

Examples:

```
Input : 16
Output : 2 2 2 2
        2 2 4
        2 8
        4 4
```

```
Input : 12
Output : 2 2 3
        2 6
        3 4
```

To solve this problem we take one array of array of integers or list of list of integers to store all the factors combination possible for the given n. So, to achieve this we can have one recursive function which can store the factors combination in each of its iteration. And each of those list should be stored in the final result list.

Below is the implementation of the above approach .

```
// Java program to print all factors combination
import java.io.*;
import java.util.*;
class FactorsCombination {
```



```
// Returns a list containing all ways to factorize
// a number n.
public static List<List<Integer> > factComb(int n)
{
    // making list of lists to store all
    // possible combinations of factors
    List<List<Integer> > result_list =
        new ArrayList<List<Integer> >();
    List<Integer> list = new ArrayList<Integer>();

    // function to calculate all the combinations
    // of factors
    factorsListFunc(2, 1, n, result_list, list);
    return result_list;
}

// First is current factor to be considered.
// each_product is current product of factors.
public static void factorsListFunc(int first,
                                   int each_prod, int n,
                                   List<List<Integer> > result_list, List<Integer>
                                   single_result_list)
{
    // Terminating condition of this recursive
    // function
    if (first > n || each_prod > n)
        return;

    // When each_prod is equal to n, we get
    // the list of factors in 'single_result_
    // _list' so it is added to the result_
    // _list list .
    if (each_prod == n) {

        ArrayList<Integer> t =
            new ArrayList<Integer>(single_result_list);

        result_list.add(t);

        return;
    }

    // In this loop we first calculate factors
    // of n and then it's combination so that
    // we get the value n in a recursive way .
    for (int i = first; i < n; i++) {
        if (i * each_prod > n)
            break;
    }
}
```

```

        // if i divides n
        // properly then it
        // is factor of n
        if (n % i == 0) {

            // it is added to 'single_result_list' list
            single_result_list.add(i);

            // Here function is called recursively
            // and when (i*each_prod) is equal to n
            // we will store the 'single_result_list' (it is
            // basically the list containing all
            // combination of factors) into result_list.
            factorsListFunc(i, i * each_prod, n,
                result_list, single_result_list);

            // here we will empty the 'single_result_list'
            // List so that new combination of factors
            // get stored in it.
            single_result_list.remove(single_result_list.size() - 1);
        }
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 16;
        List<List<Integer>> resultant = factComb(n);

        // printing all possible combination
        // of factors stored in resultant list
        for (List<Integer> i : resultant) {
            for (int j : i)
                System.out.print(j + " ");
            System.out.println();
        }
    }
}

```

Output:

```

2 2 2 2
2 2 4
2 8
4 4

```

Source

<https://www.geeksforgeeks.org/print-combinations-factors-ways-factorize/>

Chapter 96

Print all distinct integers that can be formed by K numbers from a given array of N numbers

Print all distinct integers that can be formed by K numbers from a given array of N numbers
- GeeksforGeeks

Given an array of N elements and an integer K, print all the distinct integers which can be formed by choosing K numbers from the given N numbers. A number from an array can be chosen any number of times.

Examples:

Input: k = 2, a[] = {3, 8, 17, 5}

Output: The 10 distinct integers are:

6 8 10 11 13 16 20 22 25 34

The 2 elements chosen are:

3+3 = 6, 5+3 = 8, 5+5 = 10, 8+3 = 11, 8+5 = 13

8+8 = 16, 17+3 = 20, 17+5 = 22, 17+8 = 25, 17+17 = 34.

Input: k = 3, a[] = {3, 8, 17}

Output: The 10 distinct integers are:

9 14 19 23 24 28 33 37 42 51

Approach: The problem will be solved using recursion. All combinations are to be tried, when the count of elements selected is equal to k, then we keep the *num* formed in the [set](#) so that the repetitive elements are not counted twice. The function ***generateNumber(int count, int a[], int n, int num, int k)*** is a recursive function, in which the base case is when the count becomes K which signifies that K elements from the array have been chosen. *num* in the parameter signifies the number formed by count number of numbers. In the function, iterate over the array and for every element, call the recursive function with count as count+1 and num as num+a[i].

Below is the implementation of the above approach:

C++

```
// C++ program to print all distinct
// integers that can be formed by K numbers
// from a given array of N numbers.
#include <bits/stdc++.h>
using namespace std;

// stores all the distinct integers formed
set<int> s;

// Function to generate all possible numbers
void generateNumber(int count, int a[], int n,
                   int num, int k)
{
    // Base case when K elements
    // are chosen
    if (k == count) {
        // insert it in set
        s.insert(num);
        return;
    }

    // Choose every element and call the function
    for (int i = 0; i < n; i++) {
        generateNumber(count + 1, a, n, num + a[i], k);
    }
}

// Function to print the distinct integers
void printDistinctIntegers(int k, int a[], int n)
{
    generateNumber(0, a, n, 0, k);
    cout << "The " << s.size()
         << " distinct integers are:\n";

    // prints all the elements in the set
    while (!s.empty()) {
        cout << *s.begin() << " ";

        // erase the number after printing it
        s.erase(*s.begin());
    }
}

// Driver Code
int main()
```

```
{
    int a[] = { 3, 8, 17, 5 };
    int n = sizeof(a) / sizeof(a[0]);
    int k = 2;

    // Calling Function
    printDistinctIntegers(k, a, n);
    return 0;
}
```

Java

```
// Java program to print all
// distinct integers that can
// be formed by K numbers from
// a given array of N numbers.
import java.util.*;
import java.lang.*;

class GFG
{
    // stores all the distinct
    // integers formed
    static TreeSet<Integer> set =
        new TreeSet<Integer>();

    // Function to generate
    // all possible numbers
    public static void generateNumber(int count, int a[],
                                     int n, int num, int k)
    {
        // Base case when K
        // elements are chosen
        if(count == k)
        {
            set.add(num);
            return;
        }

        // Choose every element
        // and call the function
        for(int i = 0; i < n; i++)
            generateNumber(count + 1, a, n,
                          num + a[i], k);
    }

    // Function to print
    // the distinct integers
}
```

```
public static void printDistinctIntegers(int k,
                                         int a[], int n)
{
    generateNumber(0, a, n, 0, k);
    System.out.print("The" + " " + set.size() +
                    " " + "distinct integers are: ");
    System.out.println();
    Iterator<Integer> i = set.iterator();

    // prints all the
    // elements in the set
    while(set.isEmpty() == false)
    {
        while(i.hasNext())
        {
            System.out.print(i.next() + " ");
            //set.remove(i.next());
        }
    }
}

// Driver Code
public static void main (String[] args)
{
    int arr[] = {3, 8, 17, 5};
    int n = arr.length;
    int k = 2;

    // Calling Function
    printDistinctIntegers(k, arr, n);
}
}
```

Output:

```
The 10 distinct integers are:
6 8 10 11 13 16 20 22 25 34
```

Source

<https://www.geeksforgeeks.org/print-all-distinct-integers-that-can-be-formed-by-k-numbers-from-a-given-array-of->

Chapter 97

Print all increasing sequences of length k from first n natural numbers

Print all increasing sequences of length k from first n natural numbers - GeeksforGeeks

Given two positive integers n and k, print all increasing sequences of length k such that the elements in every sequence are from first n natural numbers.

Examples:

Input: k = 2, n = 3

Output: 1 2
1 3
2 3

Input: k = 5, n = 5

Output: 1 2 3 4 5

Input: k = 3, n = 5

Output: 1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5

We strongly recommend to minimize the browser and try this yourself first.

It's a good recursion question. The idea is to create an array of length k. The array stores current sequence. For every position in array, we check the previous element and one by one put all elements greater than the previous element. If there is no previous element (first position), we put all numbers from 1 to n.

Below is the implementation of above idea :

C++

```
// C++ program to print all increasing sequences of
// length 'k' such that the elements in every sequence
// are from first 'n' natural numbers.
#include<iostream>
using namespace std;

// A utility function to print contents of arr[0..k-1]
void printArr(int arr[], int k)
{
    for (int i=0; i<k; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// A recursive function to print all increasing sequences
// of first n natural numbers. Every sequence should be
// length k. The array arr[] is used to store current
// sequence.
void printSeqUtil(int n, int k, int &len, int arr[])
{
    // If length of current increasing sequence becomes k,
    // print it
    if (len == k)
    {
        printArr(arr, k);
        return;
    }

    // Decide the starting number to put at current position:
    // If length is 0, then there are no previous elements
    // in arr[]. So start putting new numbers with 1.
    // If length is not 0, then start from value of
    // previous element plus 1.
    int i = (len == 0)? 1 : arr[len-1] + 1;

    // Increase length
    len++;
```

```
// Put all numbers (which are greater than the previous
// element) at new position.
while (i<=n)
{
    arr[len-1] = i;
    printSeqUtil(n, k, len, arr);
    i++;
}

// This is important. The variable 'len' is shared among
// all function calls in recursion tree. Its value must be
// brought back before next iteration of while loop
len--;
}

// This function prints all increasing sequences of
// first n natural numbers. The length of every sequence
// must be k. This function mainly uses printSeqUtil()
void printSeq(int n, int k)
{
    int arr[k]; // An array to store individual sequences
    int len = 0; // Initial length of current sequence
    printSeqUtil(n, k, len, arr);
}

// Driver program to test above functions
int main()
{
    int k = 3, n = 7;
    printSeq(n, k);
    return 0;
}
```

Java

```
// Java program to print all
// increasing sequences of
// length 'k' such that the
// elements in every sequence
// are from first 'n'
// natural numbers.

class GFG {

    // A utility function to print
    // contents of arr[0..k-1]
    static void printArr(int[] arr, int k)
    {
```

```
        for (int i = 0; i < k; i++)
            System.out.print(arr[i] + " ");
        System.out.print("\n");
    }

    // A recursive function to print
    // all increasing sequences
    // of first n natural numbers.
    // Every sequence should be
    // length k. The array arr[] is
    // used to store current sequence
    static void printSeqUtil(int n, int k,
                            int len, int[] arr)
    {

        // If length of current increasing
        // sequence becomes k, print it
        if (len == k)
        {
            printArr(arr, k);
            return;
        }

        // Decide the starting number
        // to put at current position:
        // If length is 0, then there
        // are no previous elements
        // in arr[]. So start putting
        // new numbers with 1.
        // If length is not 0,
        // then start from value of
        // previous element plus 1.
        int i = (len == 0) ? 1 : arr[len - 1] + 1;

        // Increase length
        len++;

        // Put all numbers (which are
        // greater than the previous
        // element) at new position.
        while (i <= n)
        {
            arr[len - 1] = i;
            printSeqUtil(n, k, len, arr);
            i++;
        }

        // This is important. The
```

```
        // variable 'len' is shared among
        // all function calls in recursion
        // tree. Its value must be
        // brought back before next
        // iteration of while loop
        len--;
    }

    // This function prints all
    // increasing sequences of
    // first n natural numbers.
    // The length of every sequence
    // must be k. This function
    // mainly uses printSeqUtil()
    static void printSeq(int n, int k)
    {

        // An array to store
        // individual sequences
        int[] arr = new int[k];

        // Initial length of
        // current sequence
        int len = 0;
        printSeqUtil(n, k, len, arr);
    }

    // Driver Code
    static public void main (String[] args)
    {
        int k = 3, n = 7;
        printSeq(n, k);
    }
}

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

C#

```
// C# program to print all
// increasing sequences of
// length 'k' such that the
// elements in every sequence
// are from first 'n'
// natural numbers.
using System;

class GFG {
```

```
// A utility function to print
// contents of arr[0..k-1]
static void printArr(int[] arr, int k)
{
    for (int i = 0; i < k; i++)
        Console.Write(arr[i] + " ");
    Console.WriteLine();
}

// A recursive function to print
// all increasing sequences
// of first n natural numbers.
// Every sequence should be
// length k. The array arr[] is
// used to store current sequence
static void printSeqUtil(int n, int k,
                        int len, int[] arr)
{
    // If length of current increasing
    // sequence becomes k, print it
    if (len == k)
    {
        printArr(arr, k);
        return;
    }

    // Decide the starting number
    // to put at current position:
    // If length is 0, then there
    // are no previous elements
    // in arr[]. So start putting
    // new numbers with 1.
    // If length is not 0,
    // then start from value of
    // previous element plus 1.
    int i = (len == 0) ? 1 : arr[len - 1] + 1;

    // Increase length
    len++;

    // Put all numbers (which are
    // greater than the previous
    // element) at new position.
    while (i <= n)
    {
        arr[len - 1] = i;
```

```
        printSeqUtil(n, k, len, arr);
        i++;
    }

    // This is important. The
    // variable 'len' is shared among
    // all function calls in recursion
    // tree. Its value must be
    // brought back before next
    // iteration of while loop
    len--;
}

// This function prints all
// increasing sequences of
// first n natural numbers.
// The length of every sequence
// must be k. This function
// mainly uses printSeqUtil()
static void printSeq(int n, int k)
{

    // An array to store
    // individual sequences
    int[] arr = new int[k];

    // Initial length of
    // current sequence
    int len = 0;
    printSeqUtil(n, k, len, arr);
}

// Driver Code
static public void Main ()
{
    int k = 3, n = 7;
    printSeq(n, k);
}

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

PHP

```
<?php
// PHP program to print all
// increasing sequences of
// length 'k' such that the
```

```
// elements in every sequence
// are from first 'n' natural
// numbers.

// A utility function to
// print contents of arr[0..k-1]
function printArr($arr, $k)
{
    for ($i = 0; $i < $k; $i++)
        echo $arr[$i], " ";
    echo "\n";
}

// A recursive function to
// print all increasing
// sequences of first n
// natural numbers. Every
// sequence should be length
// k. The array arr[] is used
// to store current sequence.
function printSeqUtil($n, $k,
                    $len, $arr)
{
    // If length of current
    // increasing sequence
    // becomes k, print it
    if ($len == $k)
    {
        printArr($arr, $k);
        return;
    }

    // Decide the starting number
    // to put at current position:
    // If length is 0, then there
    // are no previous elements
    // in arr[]. So start putting
    // new numbers with 1. If length
    // is not 0, then start from value
    // of previous element plus 1.
    $i = ($len == 0)? 1 :
        $arr[$len - 1] + 1;

    // Increase length
    $len++;

    // Put all numbers (which are
    // greater than the previous
```

```
// element) at new position.
while ($i <= $n)
{
    $arr[$len-1] = $i;
    printSeqUtil($n, $k,
                 $len, $arr);
    $i++;
}

// This is important. The
// variable 'len' is shared
// among all function calls
// in recursion tree. Its
// value must be brought back
// before next iteration of
// while loop

$len--;
}

// This function prints all
// increasing sequences of
// first n natural numbers.
// The length of every sequence
// must be k. This function
// mainly uses printSeqUtil()
function printSeq($n, $k)
{
    $arr = array(); // An array to store
                   // individual sequences
    $len = 0; // Initial length of
              // current sequence
    printSeqUtil($n, $k,
                 $len, $arr);
}

// Driver Code
$k = 3;
$n = 7;
printSeq($n, $k);

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

Output:

```
1 2 3
1 2 4
```



```
1 2 5
1 2 6
1 2 7
1 3 4
1 3 5
1 3 6
1 3 7
1 4 5
1 4 6
1 4 7
1 5 6
1 5 7
1 6 7
2 3 4
2 3 5
2 3 6
2 3 7
2 4 5
2 4 6
2 4 7
2 5 6
2 5 7
2 6 7
3 4 5
3 4 6
3 4 7
3 5 6
3 5 7
3 6 7
4 5 6
4 5 7
4 6 7
5 6 7
```

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

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

Source

<https://www.geeksforgeeks.org/print-increasing-sequences-length-k-first-n-natural-numbers/>

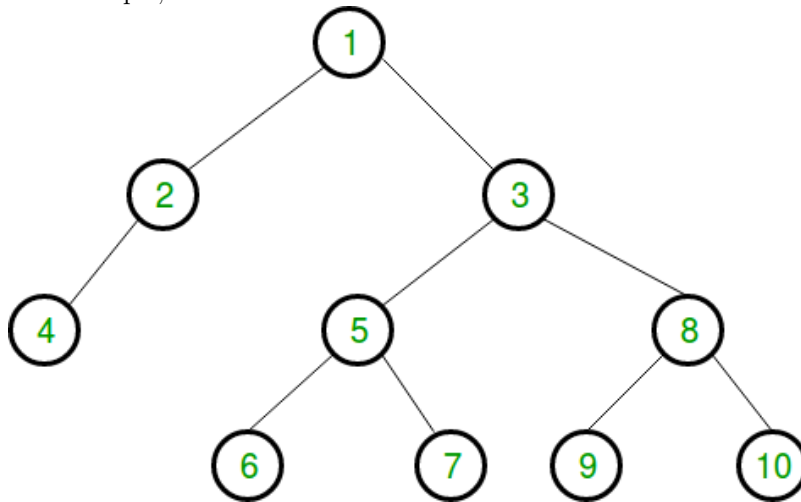
Chapter 98

Print all leaf nodes of a Binary Tree from left to right

Print all leaf nodes of a Binary Tree from left to right - GeeksforGeeks

Given a binary tree, we need to write a program to print all leaf nodes of the given binary tree from left to right. That is, the nodes should be printed in the order they appear from left to right in the given tree.

For Example,



For the above binary tree, output will be as shown below:

4 6 7 9 10

The idea to do this is similar to [DFS algorithm](#). Below is step by step algorithm to do this:

1. Check if given node is null. If null, then return from the function.

2. Check if it is a leaf node. If the node is a leaf node, then print its data.
3. If in above step, node is not a leaf node then check if left and right childs of node exists. If yes then call function for left and right childs of the node recursively.

Below is C++ implementation of above approach.

```
/* C++ program to print leaf nodes from left
   to right */
#include <iostream>
using namespace std;

// A Binary Tree Node
struct Node
{
    int data;
    struct Node *left, *right;
};

// function to print leaf
// nodes from left to right
void printLeafNodes(Node *root)
{
    // if node is null, return
    if (!root)
        return;

    // if node is leaf node, print its data
    if (!root->left && !root->right)
    {
        cout << root->data << " ";
        return;
    }

    // if left child exists, check for leaf
    // recursively
    if (root->left)
        printLeafNodes(root->left);

    // if right child exists, check for leaf
    // recursively
    if (root->right)
        printLeafNodes(root->right);
}

// Utility function to create a new tree node
Node* newNode(int data)
{
    Node *temp = new Node;
```

```
temp->data = data;
temp->left = temp->right = NULL;
return temp;
}

// Driver program to test above functions
int main()
{
    // Let us create binary tree shown in
    // above diagram
    Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->right->left = newNode(5);
    root->right->right = newNode(8);
    root->right->left->left = newNode(6);
    root->right->left->right = newNode(7);
    root->right->right->left = newNode(9);
    root->right->right->right = newNode(10);

    // print leaf nodes of the given tree
    printLeafNodes(root);

    return 0;
}
```

Output:

4 6 7 9 10

Time Complexity: $O(n)$, where n is number of nodes in the binary tree.

Source

<https://www.geeksforgeeks.org/print-leaf-nodes-left-right-binary-tree/>

Chapter 99

Print all longest common sub-sequences in lexicographical order

Print all longest common sub-sequences in lexicographical order - GeeksforGeeks

You are given two strings. Now you have to print all longest common sub-sequences in lexicographical order?

Examples:

Input : str1 = "abcabcaa", str2 = "acbacba"

Output: ababa
abaca
abcba
acaba
acaca
acbaa
acbca

This problem is an extension of [longest common subsequence](#). We first find length of LCS and store all LCS in 2D table using Memoization (or Dynamic Programming). Then we search all characters from 'a' to 'z' (to output sorted order) in both strings. If a character is found in both strings and current positions of character lead to LCS, we recursively search all occurrences with current LCS length plus 1.

Below is the implementation of algorithm.

```
// C++ program to find all LCS of two strings in
// sorted order.
#include<bits/stdc++.h>
```

```
#define MAX 100
using namespace std;

// length of lcs
int lcslen = 0;

// dp matrix to store result of sub calls for lcs
int dp[MAX][MAX];

// A memoization based function that returns LCS of
// str1[i..len1-1] and str2[j..len2-1]
int lcs(string str1, string str2, int len1, int len2,
        int i, int j)
{
    int &ret = dp[i][j];

    // base condition
    if (i==len1 || j==len2)
        return ret = 0;

    // if lcs has been computed
    if (ret != -1)
        return ret;

    ret = 0;

    // if characters are same return previous + 1 else
    // max of two sequences after removing i'th and j'th
    // char one by one
    if (str1[i] == str2[j])
        ret = 1 + lcs(str1, str2, len1, len2, i+1, j+1);
    else
        ret = max(lcs(str1, str2, len1, len2, i+1, j),
                  lcs(str1, str2, len1, len2, i, j+1));
    return ret;
}

// Function to print all routes common sub-sequences of
// length lcslen
void printAll(string str1, string str2, int len1, int len2,
             char data[], int indx1, int indx2, int currlcs)
{
    // if currlcs is equal to lcslen then print it
    if (currlcs == lcslen)
    {
        data[currlcs] = '\0';
        puts(data);
        return;
    }
}
```

```
}

// if we are done with all the characters of both string
if (indx1==len1 || indx2==len2)
    return;

// here we have to print all sub-sequences lexicographically,
// that's why we start from 'a' to 'z' if this character is
// present in both of them then append it in data[] and same
// remaining part
for (char ch='a'; ch<='z'; ch++)
{
    // done is a flag to tell that we have printed all the
    // subsequences corresponding to current character
    bool done = false;

    for (int i=indx1; i<len1; i++)
    {
        // if character ch is present in str1 then check if
        // it is present in str2
        if (ch==str1[i])
        {
            for (int j=indx2; j<len2; j++)
            {
                // if ch is present in both of them and
                // remaining length is equal to remaining
                // lcs length then add ch in sub-sequence
                if (ch==str2[j] &&
                    lcs(str1, str2, len1, len2, i, j) == lcslen-currlcs)
                {
                    data[currlcs] = ch;
                    printAll(str1, str2, len1, len2, data, i+1, j+1, currlcs+1);
                    done = true;
                    break;
                }
            }
        }
    }

    // If we found LCS beginning with current character.
    if (done)
        break;
}

}

// This function prints all LCS of str1 and str2
// in lexicographic order.
void printAllLCSSorted(string str1, string str2)
```

```
{
    // Find lengths of both strings
    int len1 = str1.length(), len2 = str2.length();

    // Find length of LCS
    memset(dp, -1, sizeof(dp));
    lcslen = lcs(str1, str2, len1, len2, 0, 0);

    // Print all LCS using recursive backtracking
    // data[] is used to store individual LCS.
    char data[MAX];
    printAll(str1, str2, len1, len2, data, 0, 0, 0);
}

// Driver program to run the case
int main()
{
    string str1 = "abcabcaa", str2 = "acbacba";
    printAllLCSSorted(str1, str2);
    return 0;
}
```

Output:

```
ababa
abaca
abcba
acaba
acaca
acbaa
acbca
```

Source

<https://www.geeksforgeeks.org/print-longest-common-sub-sequences-lexicographical-order/>

Chapter 100

Print all n-digit strictly increasing numbers

Print all n-digit strictly increasing numbers - GeeksforGeeks

Given number of digits n in a number, print all n-digit numbers whose digits are strictly increasing from left to right.

Examples:

Input: n = 2

Output:

01 02 03 04 05 06 07 08 09 12 13 14 15 16 17 18 19 23 24 25 26 27 28
29 34 35 36 37 38 39 45 46 47 48 49 56 57 58 59 67 68 69 78 79 89

Input: n = 3

Output:

012 013 014 015 016 017 018 019 023 024 025 026 027 028 029 034
035 036 037 038 039 045 046 047 048 049 056 057 058 059 067 068
069 078 079 089 123 124 125 126 127 128 129 134 135 136 137 138
139 145 146 147 148 149 156 157 158 159 167 168 169 178 179 189
234 235 236 237 238 239 245 246 247 248 249 256 257 258 259 267
268 269 278 279 289 345 346 347 348 349 356 357 358 359 367 368
369 378 379 389 456 457 458 459 467 468 469 478 479 489 567 568
569 578 579 589 678 679 689 789

Input: n = 1

Output: 0 1 2 3 4 5 6 7 8 9

The idea is to use recursion. We start from the leftmost position of a possible N-digit number and fill it from set of all digits greater than its previous digit. i.e. fill current position with

digits (i to 9] where i is its previous digit. After filling current position, we recurse for next position with strictly increasing numbers.

Below is implementation of above idea –

C++

```
// C++ program to print all n-digit numbers whose digits
// are strictly increasing from left to right
#include <bits/stdc++.h>
using namespace std;

// Function to print all n-digit numbers whose digits
// are strictly increasing from left to right.
// out --> Stores current output number as string
// start --> Current starting digit to be considered
void findStrictlyIncreasingNum(int start, string out, int n)
{
    // If number becomes N-digit, print it
    if (n == 0)
    {
        cout << out << " ";
        return;
    }

    // start from (prev digit + 1) till 9
    for (int i = start; i <= 9; i++)
    {
        // append current digit to number
        string str = out + to_string(i);

        // recurse for next digit
        findStrictlyIncreasingNum(i + 1, str, n - 1);
    }
}

// Driver code
int main()
{
    int n = 3;
    findStrictlyIncreasingNum(0, "", n);
    return 0;
}
```

Java

```
// Java program to print all n-digit numbers whose digits
```

```
// are strictly increasing from left to right
import java.io.*;

class Increasing
{
    // Function to print all n-digit numbers whose digits
    // are strictly increasing from left to right.
    // out --> Stores current output number as string
    // start --> Current starting digit to be considered
    void findStrictlyIncreasingNum(int start, String out, int n)
    {
        // If number becomes N-digit, print it
        if (n == 0)
        {
            System.out.print(out + " ");
            return;
        }

        // start from (prev digit + 1) till 9
        for (int i = start; i <= 9; i++)
        {
            // append current digit to number
            String str = out + Integer.toString(i);

            // recurse for next digit
            findStrictlyIncreasingNum(i + 1, str, n - 1);
        }
    }

    // Driver code for above function
    public static void main(String args[])throws IOException
    {
        Increasing obj = new Increasing();
        int n = 3;
        obj.findStrictlyIncreasingNum(0, " ", n);
    }
}
```

C#

```
// C# program to print all n-digit numbers
// whose digits are strictly increasing
// from left to right
using System;

class GFG {

    // Function to print all n-digit numbers
```

```
// whose digits are strictly increasing
// from left to right. out --> Stores
// current output number as string
// start --> Current starting digit to
// be considered
static void findStrictlyIncreasingNum(int start,
                                     string Out, int n)
{
    // If number becomes N-digit, print it
    if (n == 0)
    {
        Console.Write(Out + " ");
        return;
    }

    // start from (prev digit + 1) till 9
    for (int i = start; i <= 9; i++)
    {
        // append current digit to number
        string str = Out + Convert.ToInt32(i);

        // recurse for next digit
        findStrictlyIncreasingNum(i + 1, str, n - 1);
    }
}

// Driver code for above function
public static void Main()
{
    int n = 3;
    findStrictlyIncreasingNum(0, " ", n);
}

// This code is contributed by Sam007.
```

Output:

```
012 013 014 015 016 017 018 019 023 024 025 026 027 028 029 034
035 036 037 038 039 045 046 047 048 049 056 057 058 059 067 068
069 078 079 089 123 124 125 126 127 128 129 134 135 136 137 138
139 145 146 147 148 149 156 157 158 159 167 168 169 178 179 189
234 235 236 237 238 239 245 246 247 248 249 256 257 258 259 267
268 269 278 279 289 345 346 347 348 349 356 357 358 359 367 368
369 378 379 389 456 457 458 459 467 468 469 478 479 489 567 568
```

569 578 579 589 678 679 689 789

Exercise: Print all n-digit numbers whose digits are strictly decreasing from left to right.

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/print-all-n-digit-strictly-increasing-numbers/>

Chapter 101

Print all non-increasing sequences of sum equal to a given number x

Print all non-increasing sequences of sum equal to a given number x - GeeksforGeeks

Given a number x, print all possible non-increasing sequences with sum equals to x.

Examples:

Input: x = 3

Output: 1 1 1
2 1
3

Input: x = 4

Output: 1 1 1 1
2 1 1
2 2
3 1
4

We strongly recommend you to minimize your browser and try this yourself first.

The idea is to use a recursive function, an array `arr[]` to store all sequences one by one and an index variable `curr_idx` to store current next index in `arr[]`. Below is algorithm.

- 1) If current sum is equal to x, then print current sequence.
- 2) Place all possible numbers from 1 to `x-curr_sum` numbers at `curr_idx` in array. Here `curr_sum` is sum of current elements in `arr[]`. After placing a number, recur for `curr_sum + number` and `curr_idx+1`,

Below is the implementation of above steps.

C++

```
// C++ program to generate all non-increasing sequences
// of sum equals to x
#include<bits/stdc++.h>
using namespace std;

// Utility function to print array arr[0..n-1]
void printArr(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Recursive Function to generate all non-increasing sequences
// with sum x
// arr[]    --> Elements of current sequence
// curr_sum --> Current Sum
// curr_idx --> Current index in arr[]
void generateUtil(int x, int arr[], int curr_sum, int curr_idx)
{
    // If current sum is equal to x, then we found a sequence
    if (curr_sum == x)
    {
        printArr(arr, curr_idx);
        return;
    }

    // Try placing all numbers from 1 to x-curr_sum at current index
    int num = 1;

    // The placed number must also be smaller than previously placed
    // numbers, i.e., arr[curr_idx-1] if there exists a previous number
    while (num<=x-curr_sum && (curr_idx==0 || num<=arr[curr_idx-1]))
    {
        // Place number at curr_idx
        arr[curr_idx] = num;

        // Recur
        generateUtil(x, arr, curr_sum+num, curr_idx+1);

        // Try next number
        num++;
    }
}
```

```
// A wrapper over generateUtil()
void generate(int x)
{
    int arr[x]; // Array to store sequences on by one
    generateUtil(x, arr, 0, 0);
}

// Driver program
int main()
{
    int x = 5;
    generate(x);
    return 0;
}
```

Java

```
// Java program to generate all non-increasing
// sequences of sum equals to x
class GFG {

    // Utility function to print array
    // arr[0..n-1]
    static void printArr(int arr[], int n)
    {
        for (int i = 0; i < n; i++)
            System.out.printf("%d ", arr[i]);

        System.out.println("");
    }

    // Recursive Function to generate all
    // non-increasing sequences with sum x
    // arr[] --> Elements of current sequence
    // curr_sum --> Current Sum
    // curr_idx --> Current index in arr[]
    static void generateUtil(int x, int arr[],
                             int curr_sum, int curr_idx)
    {
        // If current sum is equal to x, then
        // we found a sequence
        if (curr_sum == x)
        {
            printArr(arr, curr_idx);
            return;
        }
    }
}
```



```
// Try placing all numbers from 1 to
// x-curr_sum at current index
int num = 1;

// The placed number must also be smaller
// than previously placed numbers, i.e.,
// arr[curr_idx-1] if there exists a
// previous number
while (num <= x - curr_sum &&
      (curr_idx == 0 ||
       num <= arr[curr_idx - 1]))
{

    // Place number at curr_idx
    arr[curr_idx] = num;

    // Recur
    generateUtil(x, arr, curr_sum+num,
                curr_idx + 1);

    // Try next number
    num++;
}

// A wrapper over generateUtil()
static void generate(int x)
{

    // Array to store sequences one by one
    int arr[] = new int [x];
    generateUtil(x, arr, 0, 0);
}

// Driver program
public static void main(String[] args)
{
    int x = 5;
    generate(x);
}

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

Python 3

```
# Python 3 program to generate all
```

```
# non-increasing sequences of sum
# equals to x

# Utility function to print array
# arr[0..n-1]
def printArr(arr, n):

    for i in range(0, n):
        print(arr[i], end = " ")

    print("")

# Recursive Function to generate
# all non-increasing sequences
# with sum x arr[] --> Elements
# of current sequence
# curr_sum --> Current Sum
# curr_idx --> Current index in
# arr[]
def generateUtil(x, arr, curr_sum,
                curr_idx):

    # If current sum is equal to x,
    # then we found a sequence
    if (curr_sum == x):

        printArr(arr, curr_idx)
        return

    # Try placing all numbers from
    # 1 to x-curr_sum at current
    # index
    num = 1

    # The placed number must also
    # be smaller than previously
    # placed numbers, i.e.,
    # arr[curr_idx-1] if there
    # exists a previous number
    while (num <= x - curr_sum and
           (curr_idx == 0 or
            num <= arr[curr_idx - 1])):

        # Place number at curr_idx
        arr[curr_idx] = num
```

```
# Recur
generateUtil(x, arr,
            curr_sum + num, curr_idx + 1)

# Try next number
num += 1

# A wrapper over generateUtil()
def generate(x):

    # Array to store sequences
    # on by one
    arr = [0] * x
    generateUtil(x, arr, 0, 0)

# Driver program
x = 5
generate(x)

# This code is contributed
# by Smitha.

C#

// C# program to generate all non-increasing
// sequences of sum equals to x
using System;

class GFG {

    // Utility function to print array
    // arr[0..n-1]
    static void printArr(int []arr, int n)
    {
        for (int i = 0; i < n; i++)
            Console.Write( arr[i]);

        Console.WriteLine();
    }

    // Recursive Function to generate all
    // non-increasing sequences with sum x
    // arr[] --> Elements of current sequence
    // curr_sum --> Current Sum
    // curr_idx --> Current index in arr[]
```

```
static void generateUtil(int x, int []arr,
                        int curr_sum, int curr_idx)
{
    // If current sum is equal to x, then
    // we found a sequence
    if (curr_sum == x)
    {
        printArr(arr, curr_idx);
        return;
    }

    // Try placing all numbers from 1 to
    // x-curr_sum at current index
    int num = 1;

    // The placed number must also be smaller
    // than previously placed numbers, i.e.,
    // arr[curr_idx-1] if there exists a
    // previous number
    while (num <= x - curr_sum &&
           (curr_idx == 0 ||
            num <= arr[curr_idx - 1]))
    {
        // Place number at curr_idx
        arr[curr_idx] = num;

        // Recur
        generateUtil(x, arr, curr_sum+num,
                    curr_idx + 1);

        // Try next number
        num++;
    }
}

// A wrapper over generateUtil()
static void generate(int x)
{
    // Array to store sequences one by one
    int []arr = new int [x];
    generateUtil(x, arr, 0, 0);
}

// Driver program
public static void Main()
```

```
{
    int x = 5;
    generate(x);
}

// This code is contributed by nitin mittal.
```

PHP

```
<?php
// PHP program to generate all
// non-increasing sequences
// of sum equals to x

// function to print array
// arr[0..n-1]
function printArr($arr, $n)
{
    for ($i = 0; $i < $n; $i++)
        echo $arr[$i] , " ";
    echo " \n";
}

// Recursive Function to generate
// all non-increasing sequences
// with sum x
// arr[] --> Elements of current sequence
// curr_sum --> Current Sum
// curr_idx --> Current index in arr[]
function generateUtil($x, $arr, $curr_sum,
                    $curr_idx)
{
    // If current sum is equal to x,
    // then we found a sequence
    if ($curr_sum == $x)
    {
        printArr($arr, $curr_idx);
        return;
    }

    // Try placing all numbers from
    // 1 to x-curr_sum at current index
    $num = 1;

    // The placed number must also
    // be smaller than previously
```

```
// placed numbers, i.e., arr[curr_idx-1]
// if there exists a previous number
while ($num <= $x - $curr_sum and
      ($curr_idx == 0 or $num <=
        $arr[$curr_idx - 1]))
{
    // Place number at curr_idx
    $arr[$curr_idx] = $num;

    // Recur
    generateUtil($x, $arr, $curr_sum +
                $num, $curr_idx + 1);

    // Try next number
    $num++;
}

// A wrapper over generateUtil()
function generate($x)
{
    // Array to store
    // sequences on by one
    $arr = array();
    generateUtil($x, $arr, 0, 0);
}

// Driver Code
$x = 5;
generate($x);

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

Output:

```
1 1 1 1 1
2 1 1 1
2 2 1
3 1 1
3 2
4 1
5
```

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

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

Source

<https://www.geeksforgeeks.org/print-all-non-increasing-sequences-of-sum-equal-to-a-given-number/>

Chapter 102

Print all possible combinations of r elements in a given array of size n

Print all possible combinations of r elements in a given array of size n - GeeksforGeeks

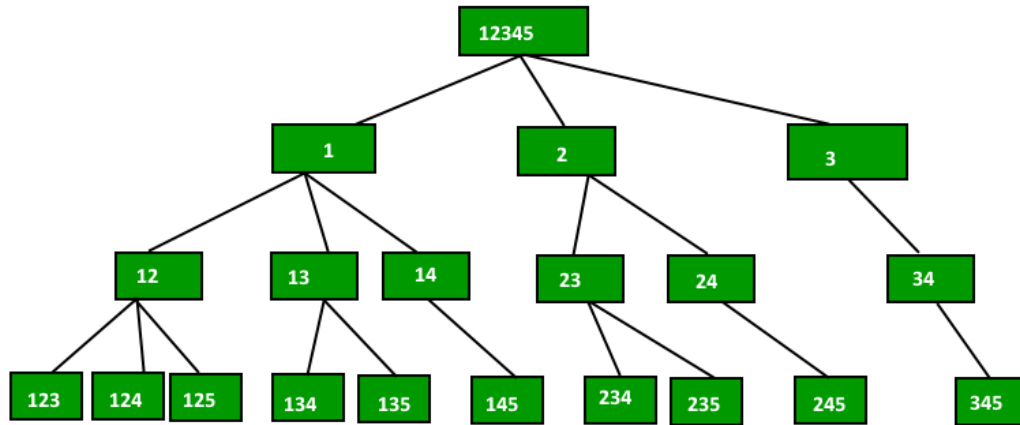
Given an array of size n, generate and print all possible combinations of r elements in array. For example, if input array is {1, 2, 3, 4} and r is 2, then output should be {1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4} and {3, 4}.

Following are two methods to do this.

Method 1 (Fix Elements and Recur)

We create a temporary array 'data[]' which stores all outputs one by one. The idea is to start from first index (index = 0) in data[], one by one fix elements at this index and recur for remaining indexes. Let the input array be {1, 2, 3, 4, 5} and r be 3. We first fix 1 at index 0 in data[], then recur for remaining indexes, then we fix 2 at index 0 and recur. Finally, we fix 3 and recur for remaining indexes. When number of elements in data[] becomes equal to r (size of a combination), we print data[].

Following diagram shows recursion tree for same input.



Following is implementation of above approach.

C

```
// Program to print all combination of size r in an array of size n
#include <stdio.h>
void combinationUtil(int arr[], int data[], int start, int end,
                    int index, int r);

// The main function that prints all combinations of size r
// in arr[] of size n. This function mainly uses combinationUtil()
void printCombination(int arr[], int n, int r)
{
    // A temporary array to store all combination one by one
    int data[r];

    // Print all combination using temprary array 'data[]'
    combinationUtil(arr, data, 0, n-1, 0, r);
}

/* arr[] ---> Input Array
   data[] ---> Temporary array to store current combination
   start & end ---> Staring and Ending indexes in arr[]
   index ---> Current index in data[]
   r ---> Size of a combination to be printed */
void combinationUtil(int arr[], int data[], int start, int end,
                    int index, int r)
{
    // Current combination is ready to be printed, print it
    if (index == r)
    {
```

```
        for (int j=0; j<r; j++)
            printf("%d ", data[j]);
        printf("\n");
        return;
    }

    // replace index with all possible elements. The condition
    // "end-i+1 >= r-index" makes sure that including one element
    // at index will make a combination with remaining elements
    // at remaining positions
    for (int i=start; i<=end && end-i+1 >= r-index; i++)
    {
        data[index] = arr[i];
        combinationUtil(arr, data, i+1, end, index+1, r);
    }
}

// Driver program to test above functions
int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int r = 3;
    int n = sizeof(arr)/sizeof(arr[0]);
    printCombination(arr, n, r);
}
```

Java

```
// Java program to print all combination of size r in an array of size n
import java.io.*;

class Permutation {

    /* arr[] ---> Input Array
    data[] ---> Temporary array to store current combination
    start & end ---> Starting and Ending indexes in arr[]
    index ---> Current index in data[]
    r ---> Size of a combination to be printed */
    static void combinationUtil(int arr[], int data[], int start,
                                int end, int index, int r)
    {
        // Current combination is ready to be printed, print it
        if (index == r)
        {
            for (int j=0; j<r; j++)
                System.out.print(data[j]+" ");
            System.out.println("");
            return;
        }
    }
}
```

```
    }

    // replace index with all possible elements. The condition
    // "end-i+1 >= r-index" makes sure that including one element
    // at index will make a combination with remaining elements
    // at remaining positions
    for (int i=start; i<=end && end-i+1 >= r-index; i++)
    {
        data[index] = arr[i];
        combinationUtil(arr, data, i+1, end, index+1, r);
    }
}

// The main function that prints all combinations of size r
// in arr[] of size n. This function mainly uses combinationUtil()
static void printCombination(int arr[], int n, int r)
{
    // A temporary array to store all combination one by one
    int data[]=new int[r];

    // Print all combination using temprary array 'data[]'
    combinationUtil(arr, data, 0, n-1, 0, r);
}

/*Driver function to check for above function*/
public static void main (String[] args) {
    int arr[] = {1, 2, 3, 4, 5};
    int r = 3;
    int n = arr.length;
    printCombination(arr, n, r);
}
}
```

/* This code is contributed by Devesh Agrawal */

C#

```
// C# program to print all
// combination of size r
// in an array of size n
using System;

class GFG
{
    /* arr[] ---> Input Array
    data[] ---> Temporary array to
                store current combination
    start & end ---> Staring and Ending
```

```

        indexes in arr[]
index ---> Current index in data[]
r ---> Size of a combination
        to be printed */
static void combinationUtil(int []arr, int []data,
                           int start, int end,
                           int index, int r)
{
    // Current combination is
    // ready to be printed,
    // print it
    if (index == r)
    {
        for (int j = 0; j < r; j++)
            Console.Write(data[j] + " ");
        Console.WriteLine("");
        return;
    }

    // replace index with all
    // possible elements. The
    // condition "end-i+1 >=
    // r-index" makes sure that
    // including one element
    // at index will make a
    // combination with remaining
    // elements at remaining positions
    for (int i = start; i <= end &&
         end - i + 1 >= r - index; i++)
    {
        data[index] = arr[i];
        combinationUtil(arr, data, i + 1,
                        end, index + 1, r);
    }
}

// The main function that prints
// all combinations of size r
// in arr[] of size n. This
// function mainly uses combinationUtil()
static void printCombination(int []arr,
                            int n, int r)
{
    // A temporary array to store
    // all combination one by one
    int []data = new int[r];

    // Print all combination
}
```

```
        // using temprary array 'data[]'
        combinationUtil(arr, data, 0,
                        n - 1, 0, r);
    }

    // Driver Code
    static public void Main ()
    {
        int []arr = {1, 2, 3, 4, 5};
        int r = 3;
        int n = arr.Length;
        printCombination(arr, n, r);
    }
}

// This code is contributed by m_kit
```

PHP

```
<?php
// Program to print all
// combination of size r
// in an array of size n

// The main function that
// prints all combinations
// of size r in arr[] of
// size n. This function
// mainly uses combinationUtil()
function printCombination($arr,
                        $n, $r)
{
    // A temporary array to
    // store all combination
    // one by one
    $data = array();

    // Print all combination
    // using temprary array 'data[]'
    combinationUtil($arr, $data, 0,
                    $n - 1, 0, $r);
}

/* arr[] ---> Input Array
data[] ---> Temporary array to
            store current combination
start & end ---> Staring and Ending
            indexes in arr[]
```

```
index ---> Current index in data[]
r ---> Size of a combination
        to be printed */
function combinationUtil($arr, $data, $start,
                        $end, $index, $r)

{
    // Current combination is ready
    // to be printed, print it
    if ($index == $r)
    {
        for ($j = 0; $j < $r; $j++)
            echo $data[$j];
        echo "\n";
        return;
    }

    // replace index with all
    // possible elements. The
    // condition "end-i+1 >=
    // r-index" makes sure that
    // including one element at
    // index will make a combination
    // with remaining elements at
    // remaining positions
    for ($i = $start;
        $i <= $end &&
        $end - $i + 1 >= $r - $index; $i++)
    {
        $data[$index] = $arr[$i];
        combinationUtil($arr, $data, $i + 1,
                        $end, $index + 1, $r);
    }
}

// Driver Code
$arr = array(1, 2, 3, 4, 5);
$r = 3;
$n = sizeof($arr);
printCombination($arr, $n, $r);

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

Output:

```
1 2 3
1 2 4
```

```
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5
```

How to handle duplicates?

Note that the above method doesn't handle duplicates. For example, if input array is {1, 2, 1} and r is 2, then the program prints {1, 2} and {2, 1} as two different combinations. We can avoid duplicates by adding following two additional things to above code.

- 1) Add code to sort the array before calling combinationUtil() in printCombination()
- 2) Add following lines at the end of for loop in combinationUtil()

```
    // Since the elements are sorted, all occurrences of an element
    // must be together
    while (arr[i] == arr[i+1])
        i++;
```

See [this](#) for an implementation that handles duplicates.

Method 2 (Include and Exclude every element)

Like the above method, We create a temporary array data[]. The idea here is similar to [Subset Sum Problem](#). We one by one consider every element of input array, and recur for two cases:

- 1) The element is included in current combination (We put the element in data[] and increment next available index in data[])
- 2) The element is excluded in current combination (We do not put the element and do not change index)

When number of elements in data[] become equal to r (size of a combination), we print it.

This method is mainly based on [Pascal's Identity](#), i.e. $n_{c_r} = n-1_{c_r} + n-1_{c_{r-1}}$

Following is implementation of method 2.

C

```
// Program to print all combination of size r in an array of size n
#include<stdio.h>
void combinationUtil(int arr[],int n,int r,int index,int data[],int i);

// The main function that prints all combinations of size r
// in arr[] of size n. This function mainly uses combinationUtil()
void printCombination(int arr[], int n, int r)
{
```

```
// A temporary array to store all combination one by one
int data[r];

// Print all combination using temprary array 'data[]'
combinationUtil(arr, n, r, 0, data, 0);
}

/* arr[] ---> Input Array
   n      ---> Size of input array
   r      ---> Size of a combination to be printed
   index  ---> Current index in data[]
   data[] ---> Temporary array to store current combination
   i      ---> index of current element in arr[] */
void combinationUtil(int arr[], int n, int r, int index, int data[], int i)
{
    // Current combination is ready, print it
    if (index == r)
    {
        for (int j=0; j<r; j++)
            printf("%d ",data[j]);
        printf("\n");
        return;
    }

    // When no more elements are there to put in data[]
    if (i >= n)
        return;

    // current is included, put next at next location
    data[index] = arr[i];
    combinationUtil(arr, n, r, index+1, data, i+1);

    // current is excluded, replace it with next (Note that
    // i+1 is passed, but index is not changed)
    combinationUtil(arr, n, r, index, data, i+1);
}

// Driver program to test above functions
int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int r = 3;
    int n = sizeof(arr)/sizeof(arr[0]);
    printCombination(arr, n, r);
    return 0;
}
```

Java


```
// Java program to print all combination of size r in an array of size n
import java.io.*;

class Permutation {

    /* arr[] ---> Input Array
    data[] ---> Temporary array to store current combination
    start & end ---> Starting and Ending indexes in arr[]
    index ---> Current index in data[]
    r ---> Size of a combination to be printed */
    static void combinationUtil(int arr[], int n, int r, int index,
                                int data[], int i)
    {
        // Current combination is ready to be printed, print it
        if (index == r)
        {
            for (int j=0; j<r; j++)
                System.out.print(data[j]+" ");
            System.out.println("");
            return;
        }

        // When no more elements are there to put in data[]
        if (i >= n)
            return;

        // current is included, put next at next location
        data[index] = arr[i];
        combinationUtil(arr, n, r, index+1, data, i+1);

        // current is excluded, replace it with next (Note that
        // i+1 is passed, but index is not changed)
        combinationUtil(arr, n, r, index, data, i+1);
    }

    // The main function that prints all combinations of size r
    // in arr[] of size n. This function mainly uses combinationUtil()
    static void printCombination(int arr[], int n, int r)
    {
        // A temporary array to store all combination one by one
        int data[]=new int[r];

        // Print all combination using temprary array 'data[]'
        combinationUtil(arr, n, r, 0, data, 0);
    }

    /*Driver function to check for above function*/
    public static void main (String[] args) {
```

```
        int arr[] = {1, 2, 3, 4, 5};
        int r = 3;
        int n = arr.length;
        printCombination(arr, n, r);
    }
}
/* This code is contributed by Devesh Agrawal */
```

C#

```
// C# program to print all
// combination of size r
// in an array of size n
using System;

class GFG
{
    /* arr[] ---> Input Array
    data[] ---> Temporary array to
                    store current combination
    start & end ---> Starting and Ending
                    indexes in arr[]
    index ---> Current index in data[]
    r ---> Size of a combination
            to be printed */
    static void combinationUtil(int []arr, int n,
                                int r, int index,
                                int []data, int i)
    {
        // Current combination is ready
        // to be printed, print it
        if (index == r)
        {
            for (int j = 0; j < r; j++)
                Console.Write(data[j] + " ");
            Console.WriteLine("");
            return;
        }

        // When no more elements are
        // there to put in data[]
        if (i >= n)
            return;

        // current is included, put
        // next at next location
        data[index] = arr[i];
```

```
        combinationUtil(arr, n, r,
                        index + 1, data, i + 1);

        // current is excluded, replace
        // it with next (Note that
        // i+1 is passed, but index
        // is not changed)
        combinationUtil(arr, n, r, index,
                        data, i + 1);
    }

    // The main function that prints
    // all combinations of size r
    // in arr[] of size n. This
    // function mainly uses combinationUtil()
    static void printCombination(int []arr,
                                int n, int r)
    {
        // A temporary array to store
        // all combination one by one
        int []data = new int[r];

        // Print all combination
        // using temporary array 'data[]'
        combinationUtil(arr, n, r, 0, data, 0);
    }

    // Driver Code
    static public void Main ()
    {
        int []arr = {1, 2, 3, 4, 5};
        int r = 3;
        int n = arr.Length;
        printCombination(arr, n, r);
    }
}
```

// This code is contributed by ajit

PHP

```
<?php
// Program to print all
// combination of size r
// in an array of size n

// The main function that prints
// all combinations of size r in
```

```
// arr[] of size n. This function
// mainly uses combinationUtil()
function printCombination($arr, $n, $r)
{
    // A temporary array to store
    // all combination one by one
    $data = Array();

    // Print all combination using
    // temprary array 'data[]'
    combinationUtil($arr, $n, $r,
                    0, $data, 0);
}

/* arr[] ---> Input Array
n ---> Size of input array
r ---> Size of a combination
    to be printed
index ---> Current index in data[]
data[] ---> Temporary array to store
    current combination
i ---> index of current element in arr[] */
function combinationUtil($arr, $n, $r,
                        $index, $data, $i)
{
    // Current combination
    // is ready, print it
    if ($index == $r)
    {
        for ($j = 0; $j < $r; $j++)
            echo $data[$j], " ";
        echo "\n";
        return;
    }

    // When no more elements are
    // there to put in data[]
    if ($i >= $n)
        return;

    // current is included, put
    // next at next location
    $data[$index] = $arr[$i];
    combinationUtil($arr, $n, $r,
                    $index + 1,
                    $data, $i + 1);

    // current is excluded, replace
```

```
// it with next (Note that i+1
// is passed, but index is not changed)
combinationUtil($arr, $n, $r,
                $index, $data, $i + 1);
}

// Driver Code
$arr = array(1, 2, 3, 4, 5);
$r = 3;
$n = sizeof($arr);
printCombination($arr, $n, $r);

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

Output :

```
1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5
```

How to handle duplicates in method 2?

Like method 1, we can follow two things to handle duplicates.

- 1) Add code to sort the array before calling combinationUtil() in printCombination()
- 2) Add following lines between two recursive calls of combinationUtil() in combinationUtil()

```
// Since the elements are sorted, all occurrences of an element
// must be together
while (arr[i] == arr[i+1])
    i++;
```

See [this](#) for an implementation that handles duplicates.

Below is another DFS based approach to solve this problem.

[Make all combinations of size k](#)

This article is contributed by **Bateesh**. 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/print-all-possible-combinations-of-r-elements-in-a-given-array-of-size-n/>

Chapter 103

Print all possible expressions that evaluate to a target

Print all possible expressions that evaluate to a target - GeeksforGeeks

Given a string that contains only digits from 0 to 9, and an integer value, **target**. Find out how many expressions are possible which evaluate to **target** using binary operator +, − and * in given string of digits.

Input : "123", Target : 6
Output : {"1+2+3", "1*2*3"}

Input : "125", Target : 7
Output : {"1*2+5", "12-5"}

This problem can be solved by putting all possible binary operator in mid between to digits and evaluating them and then check they evaluate to target or not.

- While writing the recursive code, we need to keep these variable as argument of recursive method – result vector, input string, current expression string, target value, position till which input is processed, current evaluated value and last value in evaluation.
- Last value is kept in recursion because of multiplication operation, while doing multiplication we need last value for correct evaluation.

See below example for better understanding –

Input is 125, suppose we have reached till 1+2 now,
Input = "125", current expression = "1+2",

position = 2, current val = 3, last = 2

Now when we go for multiplication, we need last value for evaluation as follows:

current val = current val - last + last * current val

First we subtract last and then add last * current val for evaluation, new last is last * current val.
current val = $3 - 2 + 2 * 5 = 11$
last = $2 * 5 = 10$

Another thing to note in below code is, we have ignored all numbers which start from 0 by imposing a condition as first condition inside the loop so that we will not process number like 03, 05 etc.

See the use of `c_str()` function, this function converts the C++ string into C char array, this function is used in below code because `atoi()` function expects a character array as an argument not the string. It converts character array to number.

```
// C++ program to find all possible expression which
// evaluate to target
#include <bits/stdc++.h>
using namespace std;

// Utility recursive method to generate all possible
// expressions
void getExprUtil(vector<string>& res, string curExp,
                string input, int target, int pos,
                int curVal, int last)
{
    // true if whole input is processed with some
    // operators
    if (pos == input.length())
    {
        // if current value is equal to target
        // then only add to final solution
        // if question is : all possible o/p then just
        // push_back without condition
        if (curVal == target)
            res.push_back(curExp);
        return;
    }

    // loop to put operator at all positions
    for (int i = pos; i < input.length(); i++)
    {
        // ignoring case which start with 0 as they
```



```
// are useless for evaluation
if (i != pos && input[pos] == '0')
    break;

// take part of input from pos to i
string part = input.substr(pos, i + 1 - pos);

// take numeric value of part
int cur = atoi(part.c_str());

// if pos is 0 then just send numeric value
// for next recursion
if (pos == 0)
    getExprUtil(res, curExp + part, input,
                target, i + 1, cur, cur);

// try all given binary operator for evaluation
else
{
    getExprUtil(res, curExp + "+" + part, input,
                target, i + 1, curVal + cur, cur);
    getExprUtil(res, curExp + "-" + part, input,
                target, i + 1, curVal - cur, -cur);
    getExprUtil(res, curExp + "*" + part, input,
                target, i + 1, curVal - last + last * cur,
                last * cur);
}
}

// Below method returns all possible expression
// evaluating to target
vector<string> getExprs(string input, int target)
{
    vector<string> res;
    getExprUtil(res, "", input, target, 0, 0, 0);
    return res;
}

// method to print result
void printResult(vector<string> res)
{
    for (int i = 0; i < res.size(); i++)
        cout << res[i] << " ";
    cout << endl;
}
```

```
// Driver code to test above methods
int main()
{
    string input = "123";
    int target = 6;
    vector<string> res = getExprs(input, target);
    printResult(res);

    input = "125";
    target = 7;
    res = getExprs(input, target);
    printResult(res);
    return 0;
}
```

Output:

```
1+2+3 1*2*3
1*2+5 12-5
```

Source

<https://www.geeksforgeeks.org/print-all-possible-expressions-that-evaluate-to-a-target/>

Chapter 104

Print all possible strings of length k that can be formed from a set of n characters

Print all possible strings of length k that can be formed from a set of n characters - Geeks-forGeeks

Given a set of characters and a positive integer k, print all possible strings of length k that can be formed from the given set.

Examples:

Input:

```
set[] = {'a', 'b'}, k = 3
```

Output:

```
aaa
aab
aba
abb
baa
bab
bba
bbb
```

Input:

```
set[] = {'a', 'b', 'c', 'd'}, k = 1
```

Output:

```
a
b
```

c
d

For a given set of size n , there will be n^k possible strings of length k . The idea is to start from an empty output string (we call it *prefix* in following code). One by one add all characters to *prefix*. For every character added, print all possible strings with current prefix by recursively calling for k equals to $k-1$.

Below is the implementation of above idea :

Java

```
// Java program to print all
// possible strings of length k

class GFG {

    // The method that prints all
    // possible strings of length k.
    // It is mainly a wrapper over
    // recursive function printAllKLengthRec()
    static void printAllKLength(char[] set, int k)
    {
        int n = set.length;
        printAllKLengthRec(set, "", n, k);
    }

    // The main recursive method
    // to print all possible
    // strings of length k
    static void printAllKLengthRec(char[] set,
                                    String prefix,
                                    int n, int k)
    {

        // Base case: k is 0,
        // print prefix
        if (k == 0)
        {
            System.out.println(prefix);
            return;
        }

        // One by one add all characters
        // from set and recursively
        // call for k equals to k-1
        for (int i = 0; i < n; ++i)
        {
```

```
        // Next character of input added
        String newPrefix = prefix + set[i];

        // k is decreased, because
        // we have added a new character
        printAllKLengthRec(set, newPrefix,
                           n, k - 1);
    }
}
```

// Driver Code

```
public static void main(String[] args)
{
    System.out.println("First Test");
    char[] set1 = {'a', 'b'};
    int k = 3;
    printAllKLength(set1, k);

    System.out.println("\nSecond Test");
    char[] set2 = {'a', 'b', 'c', 'd'};
    k = 1;
    printAllKLength(set2, k);
}
}
```

C#

```
// C# program to print all
// possible strings of length k
using System;

class GFG {

    // The method that prints all
    // possible strings of length k.
    // It is mainly a wrapper over
    // recursive function printAllKLengthRec()
    static void printAllKLength(char[] set, int k)
    {
        int n = set.Length;
        printAllKLengthRec(set, "", n, k);
    }

    // The main recursive method
    // to print all possible
    // strings of length k
    static void printAllKLengthRec(char[] set,
                                   String prefix,
```

```
int n, int k)
{
    // Base case: k is 0,
    // print prefix
    if (k == 0)
    {
        Console.WriteLine(prefix);
        return;
    }

    // One by one add all characters
    // from set and recursively
    // call for k equals to k-1
    for (int i = 0; i < n; ++i)
    {
        // Next character of input added
        String newPrefix = prefix + set[i];

        // k is decreased, because
        // we have added a new character
        printAllKLengthRec(set, newPrefix,
                           n, k - 1);
    }
}

// Driver Code
static public void Main ()
{
    Console.WriteLine("First Test");
    char[] set1 = {'a', 'b'};
    int k = 3;
    printAllKLength(set1, k);

    Console.WriteLine("\nSecond Test");
    char[] set2 = {'a', 'b', 'c', 'd'};
    k = 1;
    printAllKLength(set2, k);
}
}
```

// This code is contributed by Ajit.

Output:

First Test
aaa

aab
aba
abb
baa
bab
bba
bbb

Second Test

a
b
c
d

The above solution is mainly generalization of [this post](#).

This article is contributed by [Abhinav Ramana](#). 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/print-all-combinations-of-given-length/>

Chapter 105

Print all possible strings that can be made by placing spaces

Print all possible strings that can be made by placing spaces - GeeksforGeeks

Given a string you need to print all possible strings that can be made by placing spaces (zero or one) in between them.

Input: `str[] = "ABC"`

Output: ABC

AB C

A BC

A B C

Source: [Amazon Interview Experience | Set 158, Round 1 ,Q 1.](#)

The idea is to use recursion and create a buffer that one by one contains all output strings having spaces. We keep updating buffer in every recursive call. If the length of given string is 'n' our updated string can have maximum length of $n + (n-1)$ i.e. $2n-1$. So we create buffer size of $2n$ (one extra character for string termination).

We leave 1st character as it is, starting from the 2nd character, we can either fill a space or a character. Thus one can write a recursive function like below.

C/C++

```
// C++ program to print permutations of a given string with spaces.
#include <iostream>
#include <cstring>
using namespace std;

/* Function recursively prints the strings having space pattern.
   i and j are indices in 'str[]' and 'buff[]' respectively */
```



```
void printPatternUtil(char str[], char buff[], int i, int j, int n)
{
    if (i==n)
    {
        buff[j] = '\0';
        cout << buff << endl;
        return;
    }

    // Either put the character
    buff[j] = str[i];
    printPatternUtil(str, buff, i+1, j+1, n);

    // Or put a space followed by next character
    buff[j] = ' ';
    buff[j+1] = str[i];

    printPatternUtil(str, buff, i+1, j+2, n);
}

// This function creates buf[] to store individual output string and uses
// printPatternUtil() to print all permutations.
void printPattern(char *str)
{
    int n = strlen(str);

    // Buffer to hold the string containing spaces
    char buf[2*n]; // 2n-1 characters and 1 string terminator

    // Copy the first character as it is, since it will be always
    // at first position
    buf[0] = str[0];

    printPatternUtil(str, buf, 1, 1, n);
}

// Driver program to test above functions
int main()
{
    char *str = "ABCD";
    printPattern(str);
    return 0;
}
```

Java

```
// Java program to print permutations of a given string with spaces
import java.io.*;
```

```
class Permutation
{
    // Function recursively prints the strings having space pattern
    // i and j are indices in 'String str' and 'buf[]' respectively
    static void printPatternUtil(String str, char buf[], int i, int j, int n)
    {
        if(i == n)
        {
            buf[j] = '\0';
            System.out.println(buf);
            return;
        }

        // Either put the character
        buf[j] = str.charAt(i);
        printPatternUtil(str, buf, i+1, j+1, n);

        // Or put a space followed by next character
        buf[j] = ' ';
        buf[j+1] = str.charAt(i);

        printPatternUtil(str, buf, i+1, j+2, n);
    }

    // Function creates buf[] to store individual output string and uses
    // printPatternUtil() to print all permutations
    static void printPattern(String str)
    {
        int len = str.length();

        // Buffer to hold the string containing spaces
        // 2n-1 characters and 1 string terminator
        char[] buf = new char[2*len];

        // Copy the first character as it is, since it will be always
        // at first position
        buf[0] = str.charAt(0);
        printPatternUtil(str, buf, 1, 1, len);
    }

    // Driver program
    public static void main (String[] args)
    {
        String str = "ABCD";
        printPattern(str);
    }
}
```

Python

```
# Python program to print permutations of a given string with
# spaces.

# Utility function
def toString(List):
    s = ""
    for x in List:
        if x == '\0':
            break
        s += x
    return s

# Function recursively prints the strings having space pattern.
# i and j are indices in 'str[]' and 'buff[]' respectively
def printPatternUtil(string, buff, i, j, n):
    if i == n:
        buff[j] = '\0'
        print toString(buff)
        return

    # Either put the character
    buff[j] = string[i]
    printPatternUtil(string, buff, i+1, j+1, n)

    # Or put a space followed by next character
    buff[j] = ' '
    buff[j+1] = string[i]

    printPatternUtil(string, buff, i+1, j+2, n)

# This function creates buf[] to store individual output string
# and uses printPatternUtil() to print all permutations.
def printPattern(string):
    n = len(string)

    # Buffer to hold the string containing spaces
    buff = [0] * (2*n) # 2n-1 characters and 1 string terminator

    # Copy the first character as it is, since it will be always
    # at first position
    buff[0] = string[0]

    printPatternUtil(string, buff, 1, 1, n)

# Driver program
string = "ABCD"
```

```
printPattern(string)
```

```
# This code is contributed by BHAVYA JAIN
```

C#

```
// C# program to print permutations of a
// given string with spaces
using System;

class GFG {

    // Function recursively prints the
    // strings having space pattern
    // i and j are indices in 'String
    // str' and 'buf[]' respectively
    static void printPatternUtil(string str,
                                char []buf, int i, int j, int n)
    {
        if(i == n)
        {
            buf[j] = '\0';
            Console.WriteLine(buf);
            return;
        }

        // Either put the character
        buf[j] = str[i];
        printPatternUtil(str, buf, i+1, j+1, n);

        // Or put a space followed by next
        // character
        buf[j] = ' ';
        buf[j+1] = str[i];

        printPatternUtil(str, buf, i+1, j+2, n);
    }

    // Function creates buf[] to store
    // individual output string and uses
    // printPatternUtil() to print all
    // permutations
    static void printPattern(string str)
    {
        int len = str.Length;

        // Buffer to hold the string containing
        // spaces 2n-1 characters and 1 string
    }
}
```

```
// terminator
char []buf = new char[2*len];

// Copy the first character as it is,
// since it will be always at first
// position
buf[0] = str[0];
printPatternUtil(str, buf, 1, 1, len);
}

// Driver program
public static void Main ()
{
    string str = "ABCD";
    printPattern(str);
}

// This code is contributed by nitin mittal.
```

Output:

```
ABCD
ABC D
AB CD
AB C D
A BCD
A BC D
A B CD
A B C D
```

Time Complexity: Since number of Gaps are $n-1$, there are total $2^{(n-1)}$ patterns each having length ranging from n to $2n-1$. Thus overall complexity would be $O(n \cdot 2^n)$.

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

Improved By : [nitin mittal](#)

Source

<https://www.geeksforgeeks.org/print-possible-strings-can-made-placing-spaces/>

Chapter 106

Print all possible words from phone digits

Print all possible words from phone digits - GeeksforGeeks

Before advent of QWERTY keyboards, texts and numbers were placed on the same key. For example 2 has “ABC” if we wanted to write anything starting with ‘A’ we need to type key 2 once. If we wanted to type ‘B’, press key 2 twice and thrice for typing ‘C’. below is picture of such keypad.



Given a keypad as shown in diagram, and a n digit number, list all words which are possible by pressing these numbers.

For example if input number is 234, possible words which can be formed are (Alphabetical order):

*adg adh adi aeg aeh aei afg afh afi bdg bdh bdi beg beh bei bfg bfh bfi cdg cdh cdi ceg ceh cei
cfg cfh cfi*

Let's do some calculations first. How many words are possible with seven digits with each digit representing n letters? For first digit we have at most four choices, and for each choice for first letter, we have at most four choices for second digit and so on. So it's simple maths it will be $O(4^n)$. Since keys 0 and 1 don't have any corresponding alphabet and many characters have 3 characters, 4^n would be the upper bound of number of words and not the minimum words.

Now let's do some examples.

For number above 234. Do you see any pattern? Yes, we notice that the last character always either G,H or I and whenever it resets its value from I to G, the digit at the left of it gets changed.

Similarly whenever the second last alphabet resets its value, the third last alphabet gets changes and so on. First character resets only once when we have generated all words. This can be looked from other end also. That is to say whenever character at position i changes, character at position i+1 goes through all possible characters and it creates ripple effect till we reach at end.

Since 0 and 1 don't have any characters associated with them. we should break as there will no iteration for these digits.

Let's take the second approach as it will be easy to implement it using recursion. We go till the end and come back one by one. Perfect condition for recursion. let's search for base case.

When we reach at the last character, we print the word with all possible characters for last digit and return. Simple base case.

When we reach at the last character, we print the word with all possible characters for last digit and return. Simple base case.

Following is C implementation of recursive approach to print all possible word corresponding to a n digit input number. Note that input number is represented as an array to simplify the code.

```
#include <stdio.h>
#include <string.h>

// hashTable[i] stores all characters that correspond to digit i in phone
const char hashTable[10][5] = {"", "", "abc", "def", "ghi", "jkl",
                               "mno", "pqrs", "tuv", "wxyz"};

// A recursive function to print all possible words that can be obtained
// by input number[] of size n. The output words are one by one stored
// in output[]
void printWordsUtil(int number[], int curr_digit, char output[], int n)
{
    // Base case, if current output word is prepared
    int i;
    if (curr_digit == n)
    {
```

```
        printf("%s ", output);
        return ;
    }

    // Try all 3 possible characters for current digir in number[]
    // and recur for remaining digits
    for (i=0; i<strlen(hashTable[number[curr_digit]]); i++)
    {
        output[curr_digit] = hashTable[number[curr_digit]][i];
        printWordsUtil(number, curr_digit+1, output, n);
        if (number[curr_digit] == 0 || number[curr_digit] == 1)
            return;
    }
}

// A wrapper over printWordsUtil(). It creates an output array and
// calls printWordsUtil()
void printWords(int number[], int n)
{
    char result[n+1];
    result[n] = '\0';
    printWordsUtil(number, 0, result, n);
}

//Driver program
int main(void)
{
    int number[] = {2, 3, 4};
    int n = sizeof(number)/sizeof(number[0]);
    printWords(number, n);
    return 0;
}
```

Output:

```
adg adh adi aeg aeh aei afg afh afi bdg
bdh bdi beg beh bei bfg bfh bfi cdg cdh
cdi ceg ceh cei cfg cfh cfi
Process returned 0 (0x0)   execution time : 0.025 s
Press any key to continue.
```

Time Complexity: Time complexity of above code is $O(4^n)$ where n is number of digits in input number.

Reference:

[Buy Programming Interviews Exposed: Secrets to Landing Your Next Job 3rd Edition from Flipkart.com](#)

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

Improved By : [ABHISHEKSINGH15BCE1009](#)

Source

<https://www.geeksforgeeks.org/find-possible-words-phone-digits/>

Chapter 107

Print all sequences starting with n and consecutive difference limited to k

Print all sequences starting with n and consecutive difference limited to k - GeeksforGeeks

Given three positive integer **n**, **s** and **k**. The task is to print all possible sequence of length s, starting with n and the absolute difference between consecutive element is less than k.

Examples :

Input : n = 5, s = 3, k = 2

Output :

5 5 5

5 5 6

5 5 4

5 6 6

5 6 7

5 6 5

5 4 4

5 4 5

5 4 3

Input : n = 3, s = 2, k = 1

Output :

3 3

Observe, to get the absolute difference between consecutive element less than k, we can increase from 0 to $k - 1$. Similarly, we can decrease the next element from 1 to $k - 1$.

Now, to form the required sequence, we will first push 'n' to the vector. And then try to fill

the other element of the sequence by making recursive call for each element in the sequence. At each recursive call we run a loop from 0 to $k - 1$ and add $(n + i)$ to the sequence. Once we make the sequence of size 's', we will print the whole sequence and return back to the recursively calling function and remove $(n + i)$.

Similarly, we can run loop from 1 to $k - 1$ and insert $(n - i)$ to next element position.

To check the number of remaining element required we will pass $\text{size} - 1$ to recursive call and when size become 0, we will print the whole sequence.

Below is the implementation of this approach:

C++

```
// CPP Program all sequence of length s
// starting with n such that difference
// between consecutive element is less than k.
#include <bits/stdc++.h>
using namespace std;

// Recursive function to print all sequence
// of length s starting with n such that
// difference between consecutive element
// is less than k.
void printSequence(vector<int>& v, int n,
                  int s, int k)
{
    // If size become 0, print the sequence.
    if (s == 0) {
        for (int i = 0; i < v.size(); i++)
            cout << v[i] << " ";
        cout << endl;
        return;
    }

    // Increment the next element and make
    // recursive call after inserting the
    // (n + i) to the sequence.
    for (int i = 0; i < k; i++) {
        v.push_back(n + i);
        printSequence(v, n + i, s - 1, k);
        v.pop_back();
    }

    // Decrementing the next element and
    // make recursive call after inserting
    // the (n - i) to the sequence.
    for (int i = 1; i < k; i++) {
        v.push_back(n - i);
        printSequence(v, n - i, s - 1, k);
    }
}
```

```
        v.pop_back();
    }
}

// Wrapper Function
void wrapper(int n, int s, int k)
{
    vector<int> v;
    v.push_back(n);
    printSequence(v, n, s - 1, k);
}

// Driven Program
int main()
{
    int n = 5, s = 3, k = 2;
    wrapper(n, s, k);
    return 0;
}
```

Java

```
// Java Program all sequence of length s
// starting with n such that difference
// between consecutive element is less than k.
import java.io.*;
import java.util.*;

public class GFG {

    static List<Integer> v = new ArrayList<Integer>();
    // Recursive function to print all sequence
    // of length s starting with n such that
    // difference between consecutive element
    // is less than k.
    static void printSequence(int n,
                               int s, int k)
    {
        // If size become 0, print the sequence.
        if (s == 0) {
            for (int i = 0; i < v.size(); i++)
                System.out.print(v.get(i) + " ");
            System.out.println();
            return;
        }

        // Increment the next element and make
        // recursive call after inserting the
```

```
// (n + i) to the sequence.
for (int i = 0; i < k; i++) {
    v.add(n + i);
    printSequence(n + i, s - 1, k);
    v.remove(v.size() - 1);
}

// Decrementing the next element and'
// make recursive call after inserting
// the (n - i) to the sequence.
for (int i = 1; i < k; i++) {
    v.add(n - i);
    printSequence(n - i, s - 1, k);
    v.remove(v.size() - 1);
}
}

// Wrapper Function
static void wrapper(int n, int s, int k)
{
    v.add(n);
    printSequence(n, s - 1, k);
}

// Driven Program
public static void main(String args[])
{
    int n = 5, s = 3, k = 2;
    wrapper(n, s, k);
}

// This code is contributed by Manish Shaw
// (manishshaw1)
```

Python3

```
# Python3 Program all sequence of length s
# starting with n such that difference
# between consecutive element is less than k.

# Recursive function to prall sequence
# of length s starting with n such that
# difference between consecutive element
# is less than k.
def printSequence(v, n, s, k):

    # If size become 0, prthe sequence.
```

```
if (s == 0) :
    for i in range(0, len(v)):
        print ("{} ".format(v[i]), end="")
    print ("")
    return;

# Increment the next element and make
# recursive call after inserting the
# (n + i) to the sequence.
for i in range(0,k):
    v.append(n + i)
    printSequence(v, n + i, s - 1, k)
    v.pop()

# Decrementing the next element and
# make recursive call after inserting
# the (n - i) to the sequence.
for i in range(1,k):
    v.append(n - i)
    printSequence(v, n - i, s - 1, k)
    v.pop()

# Wrapper Function
def wrapper(n, s, k):
    v = []
    v.append(n)
    printSequence(v, n, s - 1, k)

# Driven Program
n = 5; s = 3; k = 2;
wrapper(n, s, k);

# This code is contributed by
# Manish Shaw(manishshaw1)
```

C#

```
// C# Program all sequence of length s
// starting with n such that difference
// between consecutive element is less than k.
using System;
using System.Collections.Generic;
using System.Linq;
using System.Collections;
```

```
class GFG {

    // Recursive function to print all sequence
    // of length s starting with n such that
    // difference between consecutive element
    // is less than k.
    static void printSequence(ref List<int> v, int n,
                               int s, int k)
    {
        // If size become 0, print the sequence.
        if (s == 0) {
            for (int i = 0; i < v.Count; i++)
                Console.Write(v[i] + " ");
            Console.WriteLine();
            return;
        }

        // Increment the next element and make
        // recursive call after inserting the
        // (n + i) to the sequence.
        for (int i = 0; i < k; i++) {
            v.Add(n + i);
            printSequence(ref v, n + i, s - 1, k);
            v.RemoveAt(v.Count - 1);
        }

        // Decrementing the next element and
        // make recursive call after inserting
        // the (n - i) to the sequence.
        for (int i = 1; i < k; i++) {
            v.Add(n - i);
            printSequence(ref v, n - i, s - 1, k);
            v.RemoveAt(v.Count - 1);
        }
    }

    // Wrapper Function
    static void wrapper(int n, int s, int k)
    {
        List<int> v = new List<int>();
        v.Add(n);
        printSequence(ref v, n, s - 1, k);
    }

    // Driven Program
    public static void Main()
    {
```

```
        int n = 5, s = 3, k = 2;
        wrapper(n, s, k);
    }
}

// This code is contributed by Manish Shaw
// (manishshaw1)
```

PHP

```
<?php
// PHP Program all sequence of
// length s starting with n
// such that difference between
// consecutive element is less than k.

// Recursive function to print
// all sequence of length s
// starting with n such that
// difference between consecutive
// element is less than k.
function printSequence($v, $n,
                      $s, $k)
{
    // If size become 0,
    // print the sequence.
    if ($s == 0)
    {
        for ($i = 0; $i < count($v); $i++)
            echo ($v[$i]. " ");
        echo ("\n");
        return;
    }

    // Increment the next element
    // and make recursive call
    // after inserting the (n + i)
    // to the sequence.
    for ($i = 0; $i < $k; $i++)
    {
        array_push($v, $n + $i);
        printSequence($v, $n + $i,
                      $s - 1, $k);
        array_pop($v);
    }

    // Decrementing the next element
    // and make recursive call after
```



```
// inserting the (n - i) to the
// sequence.
for ($i = 1; $i < $k; $i++)
{
    array_push($v, $n - $i);
    printSequence($v, $n - $i,
                  $s - 1, $k);
    array_pop($v);
}

// Wrapper Function
function wrapper($n, $s, $k)
{
    $v = array();
    array_push($v, $n);
    printSequence($v, $n, $s - 1, $k);
}

// Driver Code
$n = 5;
$s = 3;
$k = 2;
wrapper($n, $s, $k);

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output :

```
5 5 5
5 5 6
5 5 4
5 6 6
5 6 7
5 6 5
5 4 4
5 4 5
5 4 3
```

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/print-sequences-starting-n-consecutive-difference-limited-k/>

Chapter 108

Print all subsequences of a string

Print all subsequences of a string - GeeksforGeeks

Given a string, we have to find out all subsequences of it. A String is a subsequence of a given String, that is generated by deleting some character of a given string without changing its order.

Examples:

Input : abc

Output : a, b, c, ab, bc, ac, abc

Input : aaa

Output : a, aa, aaa

Explanation :

Step 1: Iterate over the entire String

Step 2: Iterate from the end of string
in order to generate different substring
add the substring to the list

Step 3: Drop kth character from the substring obtained
from above to generate different subsequence.

Step 4: if the subsequence is not in the list then recur.

Below is the implementation of the approach.

```
// Java Program to print all subsequence of a
// given string.
import java.util.HashSet;

public class Subsequence {

    // set to store all the subsequences
    static HashSet<String> st = new HashSet<>();

    // It computes all the subsequence of an string
    static void subsequence(String str)
    {
        // iterate over the entire string
        for (int i = 0; i < str.length(); i++) {

            // iterate from the end of the string
            // to generate substrings
            for (int j = str.length(); j > i; j--) {
                String sub_str = str.substring(i, j);

                if (!st.contains(sub_str))
                    st.add(sub_str);

                // drop kth character in the substring
                // and if its not in the set then recur
                for (int k = 1; k < sub_str.length() - 1; k++) {
                    StringBuffer sb = new StringBuffer(sub_str);

                    // drop character from the string
                    sb.deleteCharAt(k);
                    if (!st.contains(sb))
                        ;
                    subsequence(sb.toString());
                }
            }
        }
    }

    // Driver code
    public static void main(String[] args)
    {
        String s = "aabc";
        subsequence(s);
        System.out.println(st);
    }
}
```

Output:

[aa, a, ab, bc, ac, b, aac, abc, c, aab, aabc]

Alternate Solution :

One by one fix characters and recursively generates all subsets starting from them. After every recursive call, we remove last character so that next permutation can be generated.

```
// CPP program to generate power set in
// lexicographic order.
#include <bits/stdc++.h>
using namespace std;

// str : Stores input string
// n : Length of str.
// curr : Stores current permutation
// index : Index in current permutation, curr
void printSubSeqRec(string str, int n,
                   int index = -1, string curr = "")
{
    // base case
    if (index == n)
        return;

    cout << curr << "\n";
    for (int i = index + 1; i < n; i++) {

        curr += str[i];
        printSubSeqRec(str, n, i, curr);

        // backtracking
        curr = curr.erase(curr.size() - 1);
    }
    return;
}

// Generates power set in lexicographic
// order.
void printSubSeq(string str)
{
    printSubSeqRec(str, str.size());
}

// Driver code
int main()
{
    string str = "cab";
    printSubSeq(str);
    return 0;
}
```

```
}
```

Source

<https://www.geeksforgeeks.org/print-subsequences-string/>

Chapter 109

Print all the combinations of a string in lexicographical order

Print all the combinations of a string in lexicographical order - GeeksforGeeks

Given a string str, print of all the combinations of a string in lexicographical order.

Examples:

Input: str = "ABC"

Output:

A
AB
ABC
AC
ACB
B
BA
BAC
BC
BCA
C
CA
CAB
CB
CBA

Input: ED

Output:

D
DE
E

ED

Approach: Count the occurrences of all the characters in the string using a map, then using recursion all the possible combinations can be printed. Store the elements and their counts in two different arrays. Three arrays are used, input[] array which has the characters, count[] array has the count of characters and result[] is a temporary array which is used in [recursion](#) to generate all the combinations. Using [recursion](#) and [backtracking](#) all the combinations can be printed.

Below is the implementation of the above approach.

```
// C++ program to find all combinations
// of a string in lexicographical order
#include <bits/stdc++.h>
using namespace std;

// function to print string
void printResult(char* result, int len)
{
    for (int i = 0; i <= len; i++)
        cout << result[i];
    cout << endl;
}

// Method to found all combination
// of string it is based in tree
void stringCombination(char result[], char str[], int count[],
                       int level, int size, int length)
{
    // return if level is equal size of string
    if (level == size)
        return;

    for (int i = 0; i < length; i++) {

        // if occurrence of char is 0 then
        // skip the iteration of loop
        if (count[i] == 0)
            continue;

        // decrease the char occurrence by 1
        count[i]--;

        // store the char in result
        result[level] = str[i];

        // print the string till level
        printResult(result, level);
    }
}
```

```
        // call the function from level +1
        stringCombination(result, str, count,
                           level + 1, size, length);

        // backtracking
        count[i]++;
    }
}

void combination(string str)
{
    // declare the map for store
    // each char with occurrence
    map<char, int> mp;

    for (int i = 0; i < str.size(); i++) {
        if (mp.find(str[i]) != mp.end())
            mp[str[i]] = mp[str[i]] + 1;
        else
            mp[str[i]] = 1;
    }

    // initialize the input array
    // with all unique char
    char* input = new char[mp.size()];

    // initialize the count array with
    // occurrence the unique char
    int* count = new int[mp.size()];

    // temporary char array for store the result
    char* result = new char[str.size()];

    map<char, int>::iterator it = mp.begin();
    int i = 0;

    for (it; it != mp.end(); it++) {
        // store the element of input array
        input[i] = it->first;

        // store the element of count array
        count[i] = it->second;
        i++;
    }
}
```



```
// size of map(no of unique char)
int length = mp.size();

// size of original string
int size = str.size();

// call function for print string combination
stringCombination(result, input, count,
                  0, size, length);
}

// Driver code
int main()
{
    string str = "ABC";
    cin >> str;

    combination(str);

    return 0;
}
```

Output:

```
A
AB
ABC
AC
ACB
B
BA
BAC
BC
BCA
C
CA
CAB
CB
CBA
```

Source

<https://www.geeksforgeeks.org/print-all-the-combinations-of-a-string-in-lexicographical-order/>

Chapter 110

Print alternate nodes of a linked list using recursion

Print alternate nodes of a linked list using recursion - GeeksforGeeks

Given a linked list, print alternate nodes of this linked list.

Examples :

Input : 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10
Output : 1 -> 3 -> 5 -> 7 -> 9

Input : 10 -> 9
Output : 10

Recursive Approach :

1. Initialize a static variable(say flag)
2. If flag is odd print the node
3. increase head and flag by 1, and recurse for next nodes.

C++

```
// CPP code to print alternate nodes
// of a linked list using recursion
#include <bits/stdc++.h>
using namespace std;

// A linked list node
struct Node {
    int data;
    struct Node* next;
};
```

```
// Inserting node at the beginning
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node =
        (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

// Function to print alternate nodes of linked list.
// The boolean flag isOdd is used to find if the current
// node is even or odd.
void printAlternate(struct Node* node, bool isOdd=true)
{
    if (node == NULL)
        return;
    if (isOdd == true)
        cout << node->data << " ";
    printAlternate(node->next, !isOdd);
}

// Driver code
int main()
{
    // Start with the empty list
    struct Node* head = NULL;

    // construct below list
    // 1->2->3->4->5->6->7->8->9->10

    push(&head, 10);
    push(&head, 9);
    push(&head, 8);
    push(&head, 7);
    push(&head, 6);
    push(&head, 5);
    push(&head, 4);
    push(&head, 3);
    push(&head, 2);
    push(&head, 1);

    printAlternate(head);

    return 0;
}
```

Output:

1 3 5 7 9

Source

<https://www.geeksforgeeks.org/print-alternate-nodes-linked-list-using-recursion/>

Chapter 111

Print sums of all subsets of a given set

Print sums of all subsets of a given set - GeeksforGeeks

Given an array of integers, print sums of all subsets in it. Output sums can be printed in any order.

Examples :

Input : arr[] = {2, 3}
Output: 0 2 3 5

Input : arr[] = {2, 4, 5}
Output : 0 2 4 5 6 7 9 11

Method 1 (Recursive)

We can recursively solve this problem. There are total 2^n subsets. For every element, we consider two choices, we include it in a subset and we don't include it in a subset. Below is recursive solution based on this idea.

C++

```
// C++ program to print sums of all possible
// subsets.
#include<bits/stdc++.h>
using namespace std;

// Prints sums of all subsets of arr[l..r]
void subsetSums(int arr[], int l, int r,
                int sum=0)
```

```
{
    // Print current subset
    if (l > r)
    {
        cout << sum << " ";
        return;
    }

    // Subset including arr[l]
    subsetSums(arr, l+1, r, sum+arr[l]);

    // Subset excluding arr[l]
    subsetSums(arr, l+1, r, sum);
}

// Driver code
int main()
{
    int arr[] = {5, 4, 3};
    int n = sizeof(arr)/sizeof(arr[0]);

    subsetSums(arr, 0, n-1);
    return 0;
}
```

Java

```
// Java program to print sums
// of all possible subsets.
import java .io.*;

class GFG
{
    // Prints sums of all
    // subsets of arr[l..r]
    static void subsetSums(int []arr, int l,
                           int r, int sum )
    {
        // Print current subset
        if (l > r)
        {
            System.out.print(sum + " ");
            return;
        }

        // Subset including arr[l]
```

```
        subsetSums(arr, l + 1, r,
                    sum + arr[l]);

        // Subset excluding arr[l]
        subsetSums(arr, l + 1, r, sum);
    }

    // Driver code
    public static void main (String[] args)
    {
        int []arr = {5, 4, 3};
        int n = arr.length;

        subsetSums(arr, 0, n - 1, 0);
    }
}

// This code is contributed by anuj_67
```

Python3

```
# Python3 program to print sums of
# all possible subsets.

# Prints sums of all subsets of arr[l..r]
def subsetSums(arr, l, r, sum = 0):

    # Print current subset
    if l > r:
        print (sum, end = " ")
        return

    # Subset including arr[l]
    subsetSums(arr, l + 1, r, sum + arr[l])

    # Subset excluding arr[l]
    subsetSums(arr, l + 1, r, sum)

# Driver code
arr = [5, 4, 3]
n = len(arr)
subsetSums(arr, 0, n - 1)

# This code is contributed by Shreyanshi Arun.
```

C#

```
// C# program to print sums of all possible
```

```
// subsets.
using System;

class GFG {

    // Prints sums of all subsets of
    // arr[l..r]
    static void subsetSums(int []arr, int l,
                           int r, int sum )
    {

        // Print current subset
        if (l > r)
        {
            Console.Write(sum + " ");
            return;
        }

        // Subset including arr[l]
        subsetSums(arr, l+1, r, sum + arr[l]);

        // Subset excluding arr[l]
        subsetSums(arr, l+1, r, sum);
    }

    // Driver code
    public static void Main ()
    {
        int []arr = {5, 4, 3};
        int n = arr.Length;

        subsetSums(arr, 0, n-1,0);
    }
}

// This code is contributed by anuj_67
```

PHP

```
<?php
// PHP program to print sums
// of all possible subsets.

// Prints sums of all
// subsets of arr[l..r]
function subsetSums($arr, $l,
                    $r, $sum = 0)
{
```



```
// Print current subset
if ($l > $r)
{
    echo $sum , " ";
    return;
}

// Subset including arr[l]
subsetSums($arr, $l + 1, $r,
           $sum + $arr[$l]);

// Subset excluding arr[l]
subsetSums($arr, $l + 1, $r, $sum);
}

// Driver code
$arr = array(5, 4, 3);
$n = count($arr);

subsetSums($arr, 0, $n - 1);

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

Output :

12 9 8 5 7 4 3 0

Method 2 (Iterative)

As discussed above, there are total 2^n subsets. The idea is generate loop from 0 to $2^n - 1$. For every number, pick all array elements which correspond to 1s in binary representation of current number.

C++

```
// Iterative C++ program to print sums of all
// possible subsets.
#include<bits/stdc++.h>
using namespace std;

// Prints sums of all subsets of array
void subsetSums(int arr[], int n)
{
    // There are total  $2^n$  subsets
    long long total = 1<<n;

    // Consider all numbers from 0 to  $2^n - 1$ 
```

```
for (long long i=0; i<total; i++)
{
    long long sum = 0;

    // Consider binary representation of
    // current i to decide which elements
    // to pick.
    for (int j=0; j<n; j++)
        if (i & (1<<j))
            sum += arr[j];

    // Print sum of picked elements.
    cout << sum << " ";
}

// Driver code
int main()
{
    int arr[] = {5, 4, 3};
    int n = sizeof(arr)/sizeof(arr[0]);

    subsetSums(arr, n);
    return 0;
}
```

Output :

0 5 4 9 3 8 7 12

Thanks to cfh for suggesting above iterative solution in a comment.

Note: We haven't actually created sub-sets to find their sums rather we have just used recursion to find sum of non-contiguous sub-sets of the given set.

The above mentioned techniques can be used to perform various operations on sub-sets like multiplication, division, XOR, OR, etc, without actually creating and storing the sub-sets and thus making the program memory efficient.

Improved By : [vt_m](#)

Source

<https://www.geeksforgeeks.org/print-sums-subsets-given-set/>

Chapter 112

Product of 2 Numbers using Recursion

Product of 2 Numbers using Recursion - GeeksforGeeks

Given two number x and y find product using recursion.

Examples :

Input : x = 5, y = 2
Output : 10

Input : x = 100, y = 5
Output : 500

Method

- 1) If x is less than y, swap the two variables value
- 2) Recursively find y times the sum of x
- 3) If any of them become zero, return 0

C++

```
// C++ Program to find Product
// of 2 Numbers using Recursion
#include <bits/stdc++.h>
using namespace std;

// recursive function to calculate
// multiplication of two numbers
int product(int x, int y)
{
```

```
// if x is less than
// y swap the numbers
if (x < y)
    return product(y, x);

// iteratively calculate
// y times sum of x
else if (y != 0)
    return (x + product(x, y - 1));

// if any of the two numbers is
// zero return zero
else
    return 0;
}

// Driver Code
int main()
{
    int x = 5, y = 2;
    cout << product(x, y);
    return 0;
}
```

Java

```
// Java Program to find Product
// of 2 Numbers using Recursion
import java.io.*;
import java.util.*;

class GFG
{
    // recursive function to calculate
    // multiplication of two numbers
    static int product(int x, int y)
    {
        // if x is less than
        // y swap the numbers
        if (x < y)
            return product(y, x);

        // iteratively calculate
        // y times sum of x
        else if (y != 0)
            return (x + product(x, y - 1));
    }
}
```

```
        // if any of the two numbers is
        // zero return zero
        else
            return 0;
    }

    // Driver Code
    public static void main (String[] args)
    {
        int x = 5, y = 2;
        System.out.println(product(x, y));
    }
}

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

Python3

```
# Python3 to find Product of
# 2 Numbers using Recursion

# recursive function to calculate
# multiplication of two numbers
def product( x , y ):
    # if x is less than y swap
    # the numbers
    if x < y:
        return product(y, x)

    # iteratively calculate y
    # times sum of x
    elif y != 0:
        return (x + product(x, y - 1))

    # if any of the two numbers is
    # zero return zero
    else:
        return 0

# Driver code
x = 5
y = 2
print( product(x, y))

# This code is contributed
# by Abhishek Sharma44.
```

C#

```
// C# Program to find Product
// of 2 Numbers using Recursion
using System;

class GFG
{
    // recursive function to calculate
    // multiplication of two numbers
    static int product(int x, int y)
    {
        // if x is less than
        // y swap the numbers
        if (x < y)
            return product(y, x);

        // iteratively calculate
        // y times sum of x
        else if (y != 0)
            return (x + product(x, y - 1));

        // if any of the two numbers is
        // zero return zero
        else
            return 0;
    }

    // Driver code
    public static void Main ()
    {
        int x = 5, y = 2;
        Console.WriteLine(product(x, y));
    }
}

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

PHP

```
<?php
// PHP Program to find Product
// of 2 Numbers using Recursion

// recursive function to calculate
// multiplication of two numbers
function product($x, $y)
{
    // if x is less than
```

```
// y swap thenumbers
if ($x < $y)
    return product($y, $x);

// iteratively calculate
// y times sum of x
else if ($y != 0)
    return ($x + product($x, $y - 1));

// if any of the two numbers is
// zero return zero
else
    return 0;
}

// Driver Code
$x = 5; $y = 2;
echo(product($x, $y));

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

Output :

10

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/product-2-numbers-using-recursion/>

Chapter 113

Program for Chocolate and Wrapper Puzzle

Program for Chocolate and Wrapper Puzzle - GeeksforGeeks

Given following three values, the task is to find the total number of maximum chocolates you can eat.

1. money : Money you have to buy chocolates
2. price : Price of a chocolate
3. wrap : Number of wrappers to be returned for getting one extra chocolate.

It may be assumed that all given values are positive integers and greater than 1.

Examples:

Input : money = 16, price = 2, wrap = 2

Output : 15

Price of a chocolate is 2. You can buy 8 chocolates from amount 16. You can return 8 wrappers back and get 4 more chocolates. Then you can return 4 wrappers and get 2 more chocolates. Finally you can return 2 wrappers to get 1 more chocolate.

Input : money = 15, price = 1, wrap = 3

Output : 22

We buy and eat 15 chocolates

We return 15 wrappers and get 5 more chocolates.

We return 3 wrappers, get 1 chocolate and eat it (keep 2 wrappers). Now we have 3 wrappers. Return 3 and get 1 more chocolate.

So total chocolates = 15 + 5 + 1 + 1

Input : money = 20, price = 3, wrap = 5
Output : 7

Source: [Puzzle 22 | \(Maximum Chocolates\)](#)

A **naive method** is to continuously count number of chocolates by returning wrappers until wrappers left didn't become less than required to get a chocolate. Below is the implementation of this method.

C++

```
// Recursive C++ program to find maximum
// number of chocolates
#include <iostream>
using namespace std;

// Returns number of chocolates we can
// have from given number of chocolates
// and number of wrappers required to
// get a chocolate.
int countRec(int choc, int wrap)
{
    // If number of chocolates is less than
    // number of wrappers required.
    if (choc < wrap)
        return 0;

    // We can immediately get newChoc using
    // wrappers of choc.
    int newChoc = choc/wrap;

    // Now we have "newChoc + choc%wrap" wrappers.
    return newChoc + countRec(newChoc + choc%wrap,
                              wrap);
}

// Returns maximum number of chocolates we can eat
// with given money, price of chocolate and number
// of wrappers required to get a chocolate.
int countMaxChoco(int money, int price, int wrap)
{
    // We can directly buy below number of chocolates
    int choc = money/price;

    // countRec returns number of chocolates we can
    // have from given number of chocolates
    return choc + countRec(choc, wrap);
}
```

```
}

// Driver code
int main()
{
    int money = 15 ; // total money
    int price = 1; // cost of each candy
    int wrap = 3 ; // no of wrappers needs to be
    // exchanged for one chocolate.

    cout << countMaxChoco(money, price, wrap);
    return 0;
}
```

Java

```
// Recursive java program to find maximum
// number of chocolates

import java.io.*;

class GFG {

    // Returns number of chocolates we can
    // have from given number of chocolates
    // and number of wrappers required to
    // get a chocolate.
    static int countRec(int choc, int wrap)
    {

        // If number of chocolates is less than
        // number of wrappers required.
        if (choc < wrap)
            return 0;

        // We can immediatly get newChoc using
        // wrappers of choc.
        int newChoc = choc / wrap;

        // Now we have "newChoc + choc%wrap"
        // wrappers.
        return newChoc + countRec(newChoc +
                                   choc % wrap, wrap);
    }

    // Returns maximum number of chocolates
    // we can eat with given money, price of
    // chocolate and number of wrappers
}
```

```
// required to get a chocolate.
static int countMaxChoco(int money,
                        int price, int wrap)
{
    // We can directly buy below number of
    // chocolates
    int choc = money/price;

    // countRec returns number of chocolates
    // we can have from given number of
    // chocolates
    return choc + countRec(choc, wrap);
}

// Driver code
public static void main (String[] args)
{
    int money = 15 ; // total money
    int price = 1; // cost of each candy

    // no of wrappers needs to be
    // exchanged for one chocolate.
    int wrap = 3 ;
    System.out.println(
        countMaxChoco(money, price, wrap));
}

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

C#

```
// Recursive C# program to find maximum
// number of chocolates

using System;

class GFG {
    // Returns number of chocolates we can
    // have from given number of chocolates
    // and number of wrappers required to
    // get a chocolate.
    static int countRec(int choc, int wrap)
    {
        // If number of chocolates is less than
```

```
// number of wrappers required.
if (choc < wrap)
    return 0;

// We can immediatly get newChoc using
// wrappers of choc.
int newChoc = choc / wrap;

// Now we have "newChoc + choc%wrap"
// wrappers.
return newChoc + countRec(newChoc +
                          choc % wrap, wrap);
}

// Returns maximum number of chocolates
// we can eat with given money, price of
// chocolate and number of wrappers
// required to get a chocolate.
static int countMaxChoco(int money,
                        int price, int wrap)
{
    // We can directly buy below number of
    // chocolates
    int choc = money/price;

    // countRec returns number of chocolates
    // we can have from given number of
    // chocolates
    return choc + countRec(choc, wrap);
}

// Driver code
public static void Main ()
{
    int money = 15 ; // total money
    int price = 1; // cost of each candy

    // no of wrappers needs to be
    // exchanged for one chocolate.
    int wrap = 3 ;
    Console.WriteLine(
        countMaxChoco(money, price, wrap));
}

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

PHP

```
<?php
// Recursive PHP program to find maximum
// number of chocolates

// Returns number of chocolates we can
// have from given number of chocolates
// and number of wrappers required to
// get a chocolate.
function countRec($choc, $wrap)
{
    // If number of chocolates is less than
    // number of wrappers required.
    if ($choc < $wrap)
        return 0;

    // We can immediatly get newChoc using
    // wrappers of choc.
    $newChoc = $choc/$wrap;

    // Now we have "newChoc + choc%wrap" wrappers.
    return $newChoc + countRec($newChoc + $choc % $wrap,
                               $wrap);
}

// Returns maximum number of chocolates we can eat
// with given money, price of chocolate and number
// of wrappers required to get a chocolate.
function countMaxChoco($money, $price, $wrap)
{
    // We can directly buy below
    // number of chocolates
    $choc = $money/$price;

    // countRec returns number
    // of chocolates we can
    // have from given number
    // of chocolates
    return floor($choc + countRec($choc, $wrap));
}

// Driver code
// total money
$money = 15;
```

```
// cost of each candy
$price = 1;

// no of wrappers needs to be
$wrap = 3 ;

// exchanged for one chocolate.
echo countMaxChoco($money, $price, $wrap);

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

Output :

22

An **efficient solution** is to use a direct formula to find the number of chocolates.

Find initial number of chocolates by
dividing the amount with per piece cost.
i.e. $\text{choc} = \text{money} / \text{wrap}$

then apply below formula
 $\text{choc} += (\text{choc} - 1) / (\text{wrap} - 1)$

In above naive implementation, we noticed that after finding initial number of chocolates, we recursively divide the number of chocolates with the number of wrappers required. until we left with 1 chocolate or wrapper.

We are recomputing the values i.e. $((\text{choc} / \text{wrap} + \text{choc} \% \text{wrap}) / \text{wrap})$ until we get 1.

It is observed that, we can get the result by just reducing the values of chocolates and wrappers by 1 and then divide them to get the result **$(\text{choc}-1)/(\text{wrap}-1)$**

C++

```
// Efficient C++ program to find maximum
// number of chocolates
#include <iostream>
using namespace std;

// Returns maximum number of chocolates we can eat
// with given money, price of chocolate and number
// of wrapprices required to get a chocolate.
int countMaxChoco(int money, int price, int wrap)
{
    // Corner case
    if (money < price)
```

```
        return 0;

        // First find number of chocolates that
        // can be purchased with the given amount
        int choc = money / price;

        // Now just add number of chocolates with the
        // chocolates gained by wrapprices
        choc = choc + (choc - 1) / (wrap - 1);
        return choc;
    }

    // Driver code
    int main()
    {
        int money = 15 ; // total money
        int price = 1; // cost of each candy
        int wrap = 3 ; // no of wrappers needs to be
        // exchanged for one chocolate.

        cout << countMaxChoco(money, price, wrap);
        return 0;
    }
```

Java

```
// Efficient Java program to find maximum
// number of chocolates
import java.io.*;

class GFG {

    // Returns maximum number of chocolates
    // we can eat with given money, price
    // of chocolate and number of wrapprices
    // required to get a chocolate.
    static int countMaxChoco(int money,
                             int price, int wrap)
    {

        // Corner case
        if (money < price)
            return 0;

        // First find number of chocolates
        // that can be purchased with the
        // given amount
        int choc = money / price;
```

```
        // Now just add number of chocolates
        // with the chocolates gained by
        // wrapprices
        choc = choc + (choc - 1) / (wrap - 1);
        return choc;
    }

    // Driver code
    public static void main (String[] args)
    {

        // total money
        int money = 15;

        // cost of each candy
        int price = 1;

        // no of wrappers needs to be
        int wrap = 3 ;

        // exchanged for one chocolate.
        System.out.println(
            countMaxChoco(money, price, wrap));
    }
}

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

C#

```
// Efficient C# program to find maximum
// number of chocolates
using System;

class GFG {

    // Returns maximum number of chocolates
    // we can eat with given money, price
    // of chocolate and number of wrapprices
    // required to get a chocolate.
    static int countMaxChoco(int money,
                             int price, int wrap)
    {

        // Corner case
        if (money < price)
            return 0;

    }
}
```



```
// First find number of chocolates
// that can be purchased with the
// given amount
int choc = money / price;

// Now just add number of chocolates
// with the chocolates gained by
// wrapprices
choc = choc + (choc - 1) / (wrap - 1);
return choc;
}

// Driver code
public static void Main ()
{

    // total money
    int money = 15;

    // cost of each candy
    int price = 1;

    // no of wrappers needs to be
    int wrap = 3 ;

    // exchanged for one chocolate.
    Console.WriteLine(
        countMaxChoco(money, price, wrap));
}

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

PHP

```
<?php
// Efficient PHP program to find maximum
// number of chocolates

// Returns maximum number
// of chocolates we can eat
// with given money, price
// of chocolate and number
// of wrapprices required
// to get a chocolate.
function countMaxChoco($money, $price, $wrap)
{
```

```
// Corner case
if ($money < $price)
return 0;

// First find number
// of chocolates that
// can be purchased
// with the given amount
$choc = $money / $price;

// Now just add number
// of chocolates with the
// chocolates gained by wrapprices
$choc = $choc + ($choc - 1) /
           ($wrap - 1);
return $choc;
}

// Driver code

// total money
$money = 15 ;

// cost of each candy
$price = 1;

// no of wrappers needs to be
// exchanged for one chocolate.
$wrap = 3 ;
echo countMaxChoco($money, $price, $wrap);

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

Output :

22

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

Source

<https://www.geeksforgeeks.org/program-chocolate-wrapper-puzzle/>

Chapter 114

Program for Sum the digits of a given number

Program for Sum the digits of a given number - GeeksforGeeks

Given a number, find sum of its digits.

Examples :

Input : n = 687

Output : 21

Input : n = 12

Output : 3

Below are the solutions to get sum of the digits.

1. Iterative:

C

```
// C program to compute sum of digits in
// number.
# include<stdio.h>

/* Function to get sum of digits */
int getSum(int n)
{
    int sum = 0;
    while (n != 0)
    {
        sum = sum + n % 10;
```

```
        n = n/10;
    }
    return sum;
}

int main()
{
    int n = 687;
    printf(" %d ", getSum(n));
    return 0;
}
```

Java

```
// Java program to compute
// sum of digits in number.
import java.io.*;

class GFG {

    /* Function to get sum of digits */
    static int getSum(int n)
    {
        int sum = 0;

        while (n != 0)
        {
            sum = sum + n % 10;
            n = n/10;
        }

        return sum;
    }

    // Driver program
    public static void main(String[] args)
    {
        int n = 687;

        System.out.println(getSum(n));
    }
}

// This code is contributed by Gitanjali
```

Python3

```
# Python 3 program to
```

```
# compute sum of digits in
# number.

# Function to get sum of digits
def getSum(n):

    sum = 0
    while (n != 0):

        sum = sum + int(n % 10)
        n = int(n/10)

    return sum

n = 687
print(getSum(n))
```

C#

```
// C# program to compute
// sum of digits in number.
using System;

class GFG
{
    /* Function to get sum of digits */
    static int getSum(int n)
    {
        int sum = 0;

        while (n != 0)
        {
            sum = sum + n % 10;
            n = n/10;
        }

        return sum;
    }

    // Driver program
    public static void Main()
    {
        int n = 687;
        Console.Write(getSum(n));
    }
}

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP Code to compute sum
// of digits in number.

// Function to get
// $sum of digits
function getsum($n)
{
    $sum = 0;
    while ($n != 0)
    {
        $sum = $sum + $n % 10;
        $n = $n/10;
    }
    return $sum;
}

// Driver Code
$n = 687;
$res = getsum($n);
echo("$res");

// This code is contributed by
// Smitha Dinesh Semwal.
?>
```

Output :

21

How to compute in single line?

Below function has three lines instead of one line but it calculates sum in line. It can be made one line function if we pass pointer to sum.

C

```
# include<stdio.h>
/* Function to get sum of digits */
int getSum(int n)
{
    int sum;

    /* Single line that calculates sum */
    for (sum = 0; n > 0; sum += n % 10, n /= 10);
```

```
    return sum;
}

int main()
{
    int n = 687;
    printf(" %d ", getSum(n));
    return 0;
}
```

Java

```
// Java program to compute
// sum of digits in number.
import java.io.*;

class GFG {

    /* Function to get sum of digits */
    static int getSum(int n)
    {
        int sum;

        /* Single line that calculates sum */
        for (sum = 0; n > 0; sum += n % 10,
            n /= 10);

        return sum;
    }

    // Driver code
    public static void main(String[] args)
    {
        int n = 687;

        System.out.println(getSum(n));
    }
}

// This code is contributed by Gitanjali
```

Python3

```
# Function to get sum of digits

def getSum(n):
```

```
sum = 0

# Single line that calculates sum
while(n > 0):
    sum += int(n%10)
    n = int(n/10)

return sum

# Driver code

n = 687
print(getSum(n))

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

C#

```
// C# program to compute
// sum of digits in number.
using System;

class GFG
{
    static int getSum(int n)
    {
        int sum;

        /* Single line that calculates sum */
        for (sum = 0; n > 0; sum += n % 10,
            n /= 10);

        return sum;
    }

    // Driver code
    public static void Main()
    {
        int n = 687;
        Console.Write(getSum(n));
    }
}

// This code is contributed by Sam007
```

PHP


```
<?php
// PHP Code for Sum the
// digits of a given number

// Function to get sum of digits
function getsum($n)
{

    // Single line that calculates $sum
    for ($sum = 0; $n > 0; $sum += $n % 10,
        $n /= 10);

    return $sum;
}

// Driver Code
$n = 687;
echo(getsum($n));

// This code is contributed by
// Smitha Dinesh Semwal.
?>
```

Output :

21

2. Recursive

Thanks to ayesha for providing the below recursive solution.

C

```
int sumDigits(int no)
{
    return no == 0 ? 0 : no%10 + sumDigits(no/10) ;
}

int main(void)
{
    printf("%d", sumDigits(687));
    return 0;
}
```

Java

```
// Java program to compute
// sum of digits in number.
```

```
import java.io.*;

class GFG {

    /* Function to get sum of digits */
    static int sumDigits(int no)
    {
        return no == 0 ? 0 : no%10 +
            sumDigits(no/10) ;
    }

    // Driver code
    public static void main(String[] args)
    {
        System.out.println(sumDigits(687));
    }
}

// This code is contributed by Gitanjali
```

Python3

```
# Python program to compute
# sum of digits in number.

def sumDigits(no):
    return 0 if no == 0 else int(no%10) + sumDigits(int(no/10))

# Driver code
print(sumDigits(687))

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

C#

```
// C# program to compute
// sum of digits in number.
using System;

class GFG
{
    /* Function to get sum of digits */
    static int sumDigits(int no)
    {
        return no == 0 ? 0 : no % 10 +
            sumDigits(no / 10);
    }
}
```

```
    }

    // Driver code
    public static void Main()
    {
        Console.Write(sumDigits(687));
    }
}

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP program to compute
// sum of digits in number.
function sumDigits($no)
{
    return $no == 0 ? 0 : $no % 10 +
        sumDigits($no / 10) ;
}

// Driver Code
echo sumDigits(687);

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

Output :

21

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/program-for-sum-the-digits-of-a-given-number/>

Chapter 115

Program for length of a string using recursion

Program for length of a string using recursion - GeeksforGeeks

Given a string calculate length of the string using recursion.

Examples:

Input : str = "abcd"
Output :4

Input : str = "GEEKSFORGEEKS"
Output :13

We have discussed [5 Different methods to find length of a string in C++](#)
[How to find length of a string without string.h and loop in C?](#)

Algorithm

```
recLen(str)
{
    If str is NULL
        return 0
    Else
        return 1 + recLen(str + 1)
}
```

C++

```
// CPP program to calculate length of
```

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

/* Function to calculate length */
int recLen(char* str)
{
    // if we reach at the end of the string
    if (*str == '\0')
        return 0;
    else
        return 1 + recLen(str + 1);
}

/* Driver program to test above function */
int main()
{
    char str[] = "GeeksforGeeks";
    cout << recLen(str);
    return 0;
}
```

Java

```
// java program to calculate length of
// a string using recursion
import java.util.*;

public class GFG{

    /* Function to calculate length */
    private static int recLen(String str)
    {
        // if we reach at the end of the string
        if (str.equals(""))
            return 0;
        else
            return recLen(str.substring(1)) + 1;
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {

        String str ="GeeksforGeeks";
        System.out.println(recLen(str));
    }
}
```

```
    }  
}  
  
// This code is contributed by Sam007.
```

Python3

```
# Python program to calculate  
# length of a string using  
# recursion  
str = "GeeksforGeeks"  
  
# Function to  
# calculate length  
def recLen(i) :  
  
    global str  
  
    # if we reach at the  
    # end of the string  
    if (i == len(str)) :  
        return 0  
    else :  
        return 1 + recLen(i + 1)  
  
# Driver Code  
print (recLen(0))  
  
# This code is contributed by  
# Manish Shaw(manishshaw1)
```

C#

```
// C# program to calculate length of  
// a string using recursion  
using System;  
  
public class GFG{  
  
    /* Function to calculate length */  
    private static int recLen(string str)  
    {  
  
        // if we reach at the end of the string  
        if (str.Equals(""))  
            return 0;  
        else
```

```
        return recLen(str.Substring(1)) + 1;
    }

    /* Driver program to test above function */
    public static void Main()
    {

        string str = "GeeksforGeeks";
        Console.WriteLine(recLen(str));
    }
}

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

PHP

```
<?php
// PHP program to calculate
// length of a string using
// recursion

// Function to
// calculate length
function recLen(&$str, $i)
{
    // if we reach at the
    // end of the string
    if ($i == strlen($str))
        return 0;
    else
        return 1 +
            recLen($str,
                $i + 1);
}

// Driver Code
$str = "GeeksforGeeks";
echo (recLen($str, 0));

// This code is contributed by
// Manish Shaw(manishshaw1)
?>
```

Output:

Time Complexity : $O(n)$

Auxiliary Space : $O(n)$ for recursion call stack.

Since this solution requires extra space and function call overhead, it is not recommended to use it in practice.

Improved By : [manishshaw1](#)

Source

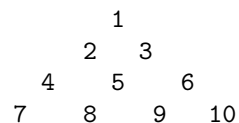
<https://www.geeksforgeeks.org/program-for-length-of-a-string-using-recursion/>

Chapter 116

Program to find amount of water in a given glass

Program to find amount of water in a given glass - GeeksforGeeks

There are some glasses with equal capacity as 1 litre. The glasses are kept as follows:



You can put water to only top glass. If you put more than 1 litre water to 1st glass, water overflows and fills equally in both 2nd and 3rd glasses. Glass 5 will get water from both 2nd glass and 3rd glass and so on.

If you have X litre of water and you put that water in top glass, how much water will be contained by jth glass in ith row?

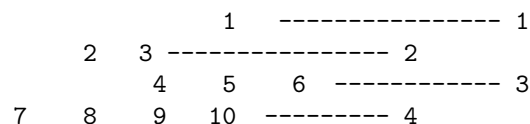
Example. If you will put 2 litre on top.

1st – 1 litre

2nd – 1/2 litre

3rd – 1/2 litre

The approach is similar to Method 2 of the [Pascal's Triangle](#). If we take a closer look at the problem, the problem boils down to [Pascal's Triangle](#).



Each glass contributes to the two glasses down the glass. Initially, we put all water in first glass. Then we keep 1 litre (or less than 1 litre) in it, and move rest of the water to two glasses down to it. We follow the same process for the two glasses and all other glasses till i th row. There will be $i*(i+1)/2$ glasses till i th row.

C++

```
// Program to find the amount of water in j-th glass
// of i-th row
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Returns the amount of water in jth glass of ith row
float findWater(int i, int j, float X)
{
    // A row number i has maximum i columns. So input
    // column number must be less than i
    if (j > i)
    {
        printf("Incorrect Inputn");
        exit(0);
    }

    // There will be i*(i+1)/2 glasses till ith row
    // (including ith row)
    float glass[i * (i + 1) / 2];

    // Initialize all glasses as empty
    memset(glass, 0, sizeof(glass));

    // Put all water in first glass
    int index = 0;
    glass[index] = X;

    // Now let the water flow to the downward glasses
    // till the row number is less than or/ equal to i (given row)
    // correction : X can be zero for side glasses as they have lower rate to fill
    for (int row = 1; row <= i ; ++row)
    {
        // Fill glasses in a given row. Number of
        // columns in a row is equal to row number
        for (int col = 1; col <= row; ++col, ++index)
        {
            // Get the water from current glass
            X = glass[index];

            // Keep the amount less than or equal to
            // capacity in current glass
```

```
        glass[index] = (X >= 1.0f) ? 1.0f : X;

        // Get the remaining amount
        X = (X >= 1.0f) ? (X - 1) : 0.0f;

        // Distribute the remaining amount to
        // the down two glasses
        glass[index + row] += X / 2;
        glass[index + row + 1] += X / 2;
    }
}

// The index of jth glass in ith row will
// be i*(i-1)/2 + j - 1
return glass[i*(i-1)/2 + j - 1];
}

// Driver program to test above function
int main()
{
    int i = 2, j = 2;
    float X = 2.0; // Total amount of water

    printf("Amount of water in jth glass of ith row is: %f",
           findWater(i, j, X));

    return 0;
}
```

Java

```
// Program to find the amount
/// of water in j-th glass
// of i-th row
import java.lang.*;

class GFG
{
    // Returns the amount of water
    // in jth glass of ith row
    static float findWater(int i, int j,
                           float X)
    {
        // A row number i has maximum i
        // columns. So input column
        // number must be less than i
        if (j > i)
        {
```

```
        System.out.println("Incorrect Input");
        System.exit(0);
    }

    // There will be i*(i+1)/2 glasses
    // till ith row (including ith row)
    int ll = Math.round((i * (i + 1) ));
    float[] glass = new float[ll + 2];

    // Put all water in first glass
    int index = 0;
    glass[index] = X;

    // Now let the water flow to the
    // downward glasses till the row
    // number is less than or/ equal
    // to i (given row)
    // correction : X can be zero for side
    // glasses as they have lower rate to fill
    for (int row = 1; row <= i ; ++row)
    {
        // Fill glasses in a given row. Number of
        // columns in a row is equal to row number
        for (int col = 1;
             col <= row; ++col, ++index)
        {
            // Get the water from current glass
            X = glass[index];

            // Keep the amount less than or
            // equal to capacity in current glass
            glass[index] = (X >= 1.0f) ? 1.0f : X;

            // Get the remaining amount
            X = (X >= 1.0f) ? (X - 1) : 0.0f;

            // Distribute the remaining amount
            // to the down two glasses
            glass[index + row] += X / 2;
            glass[index + row + 1] += X / 2;
        }
    }

    // The index of jth glass in ith
    // row will be i*(i-1)/2 + j - 1
    return glass[(int)(i * (i - 1) /
                        2 + j - 1)];
}
```

```
// Driver Code
public static void main(String[] args)
{
    int i = 2, j = 2;
    float X = 2.0f; // Total amount of water
    System.out.println("Amount of water in jth " +
                       "glass of ith row is: " +
                       findWater(i, j, X));
}
}

// This code is contributed by mits
```

Python3

```
# Program to find the amount
# of water in j-th glass of
# i-th row

# Returns the amount of water
# in jth glass of ith row
def findWater(i, j, X):
    # A row number i has maximum
    # i columns. So input column
    # number must be less than i
    if (j > i):
        print("Incorrect Input");
        return;

    # There will be i*(i+1)/2
    # glasses till ith row
    # (including ith row)
    # and Initialize all glasses
    # as empty
    glass = [0]*int(i *(i + 1) / 2);

    # Put all water
    # in first glass
    index = 0;
    glass[index] = X;

    # Now let the water flow to
    # the downward glasses till
    # the row number is less
    # than or/ equal to i (given
    # row) correction : X can be
    # zero for side glasses as
```

```
# they have lower rate to fill
for row in range(1,i):
    # Fill glasses in a given
    # row. Number of columns
    # in a row is equal to row number
    for col in range(1,row+1):
        # Get the water
        # from current glass
        X = glass[index];

        # Keep the amount less
        # than or equal to
        # capacity in current glass
        glass[index] = 1.0 if (X >= 1.0) else X;

        # Get the remaining amount
        X = (X - 1) if (X >= 1.0) else 0.0;

        # Distribute the remaining
        # amount to the down two glasses
        glass[index + row] += (X / 2);
        glass[index + row + 1] += (X / 2);
        index+=1;

# The index of jth glass
# in ith row will
# be i*(i-1)/2 + j - 1
return glass[int(i * (i - 1) / 2 + j - 1)];

# Driver Code
if __name__ == "__main__":

    i = 2;
    j = 2;
    X = 2.0;
    # Total amount of water

    res=repr(findWater(i, j, X));
    print("Amount of water in jth glass of ith row is:",res.ljust(8,'0'));
    # This Code is contributed by mits
```

C#

```
// Program to find the amount
// of water in j-th glass
// of i-th row
using System;

class GFG
```

```
{
// Returns the amount of water
// in jth glass of ith row
static float findWater(int i, int j,
float X)
{
// A row number i has maximum i
// columns. So input column
// number must be less than i
if (j > i)
{
Console.WriteLine("Incorrect Input");
Environment.Exit(0);
}

// There will be i*(i+1)/2 glasses
// till ith row (including ith row)
int ll = (int)Math.Round((double)(i * (i + 1)));
float[] glass = new float[ll + 2];

// Put all water in first glass
int index = 0;
glass[index] = X;

// Now let the water flow to the
// downward glasses till the row
// number is less than or/ equal
// to i (given row)
// correction : X can be zero
// for side glasses as they have
// lower rate to fill
for (int row = 1; row <= i; ++row) { // Fill glasses in a given row. // Number of columns
in a row // is equal to row number for (int col = 1; col <= row; ++col, ++index) { // Get
the water from current glass X = glass[index]; // Keep the amount less than // or equal to
capacity in // current glass glass[index] = (X >= 1.0f) ?
1.0f : X;

// Get the remaining amount
X = (X >= 1.0f) ? (X - 1) : 0.0f;

// Distribute the remaining amount
// to the down two glasses
glass[index + row] += X / 2;
glass[index + row + 1] += X / 2;
}
}

// The index of jth glass in ith
// row will be i*(i-1)/2 + j - 1
return glass[(int)(i * (i - 1) /
2 + j - 1)];
}
```

```
}
// Driver Code
static void Main()
{
    int i = 2, j = 2;
    float X = 2.0f; // Total amount of water
    Console.WriteLine("Amount of water in jth " +
        "glass of ith row is: " +
        findWater(i, j, X));
}
}
```

// This code is contributed by mits

PHP

```
<?php
// Program to find the amount
// of water in j-th glass of
// i-th row

// Returns the amount of water
// in jth glass of ith row
function findWater($i, $j, $X)
{
    // A row number i has maximum
    // i columns. So input column
    // number must be less than i
    if ($j > $i)
    {
        echo "Incorrect Input\n";
        return;
    }

    // There will be i*(i+1)/2
    // glasses till ith row
    // (including ith row)
    // and Initialize all glasses
    // as empty
    $glass = array_fill(0, (int)($i *
        ($i + 1) / 2), 0);

    // Put all water
    // in first glass
    $index = 0;
    $glass[$index] = $X;

    // Now let the water flow to
```



```

// the downward glasses till
// the row number is less
// than or/ equal to i (given
// row) correction : X can be
// zero for side glasses as
// they have lower rate to fill
for ($row = 1; $row < $i ; ++$row)
{
    // Fill glasses in a given
    // row. Number of columns
    // in a row is equal to row number
    for ($col = 1;
        $col <= $row; ++$col, ++$index)
    {
        // Get the water
        // from current glass
        $X = $glass[$index];

        // Keep the amount less
        // than or equal to
        // capacity in current glass
        $glass[$index] = ($X >= 1.0) ?
            1.0 : $X;

        // Get the remaining amount
        $X = ($X >= 1.0) ?
            ($X - 1) : 0.0;

        // Distribute the remaining
        // amount to the down two glasses
        $glass[$index + $row] += (double)($X / 2);
        $glass[$index + $row + 1] += (double)($X / 2);
    }
}

// The index of jth glass
// in ith row will
// be  $i*(i-1)/2 + j - 1$ 
return $glass[(int)($i * ($i - 1) /
    2 + $j - 1)];
}

// Driver Code
$i = 2;
$j = 2;
$X = 2.0; // Total amount of water
echo "Amount of water in jth " ,
    "glass of ith row is: ".

```

```
        str_pad(findWater($i, $j,
                        $X), 8, '0');

// This Code is contributed by mits
?>
```

Output:

Amount of water in jth glass of ith row is: 0.500000

Time Complexity: $O(i*(i+1)/2)$ or $O(i^2)$

Auxiliary Space: $O(i*(i+1)/2)$ or $O(i^2)$

This article is compiled by **Rahul** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Improved By : [Pradeep Joshi](#), [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/find-water-in-a-glass/>

Chapter 117

Program to find the minimum (or maximum) element of an array

Program to find the minimum (or maximum) element of an array - GeeksforGeeks

Given an array, write functions to find minimum and maximum elements in it.

C++

```
// CPP program to find minimum (or maximum) element
// in an array.
#include <iostream>
using namespace std;

int getMin(int arr[], int n)
{
    int res = arr[0];
    for (int i = 1; i < n; i++)
        res = min(res, arr[i]);
    return res;
}

int getMax(int arr[], int n)
{
    int res = arr[0];
    for (int i = 1; i < n; i++)
        res = max(res, arr[i]);
    return res;
}
```

```
int main()
{
    int arr[] = { 12, 1234, 45, 67, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Minimum element of array: " << getMin(arr, n) << "\n";
    cout << "Maximum element of array: " << getMax(arr, n);
    return 0;
}
```

Java

```
// Java program to find minimum (or maximum)
// element in an array.
import java.io.*;

class GFG {

    static int getMin(int arr[], int n)
    {
        int res = arr[0];

        for (int i = 1; i < n; i++)
            res = Math.min(res, arr[i]);
        return res;
    }

    static int getMax(int arr[], int n)
    {
        int res = arr[0];

        for (int i = 1; i < n; i++)
            res = Math.max(res, arr[i]);
        return res;
    }

    // Driver code
    public static void main (String[] args)
    {
        int arr[] = { 12, 1234, 45, 67, 1 };
        int n = arr.length;
        System.out.println( "Minimum element"
            + " of array: " + getMin(arr, n));
        System.out.println( "Maximum element"
            + " of array: " + getMax(arr, n));
    }
}

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

Python3

```
# Python3 program to find minimum
# (or maximum) element in an array

# Minimum Function
def getMin(arr, n):
    res = arr[0]
    for i in range(1,n):
        res = min(res, arr[i])
    return res

# Maximum Function
def getMax(arr, n):
    res = arr[0]
    for i in range(1,n):
        res = max(res, arr[i])
    return res

# Driver Program
arr = [12, 1234, 45, 67, 1]
n = len(arr)
print ("Minimum element of array:", getMin(arr, n))
print ("Maximum element of array:", getMax(arr, n))

# This code is contributed
# by Shreyanshi Arun.
```

C#

```
// C# program to find
// minimum (or maximum)
// element in an array.
using System;

class GFG
{
    static int getMin(int []arr,
                      int n)
    {
        int res = arr[0];

        for (int i = 1; i < n; i++)
            res = Math.Min(res, arr[i]);
        return res;
    }
}
```

```
static int getMax(int []arr,
                  int n)
{
    int res = arr[0];

    for (int i = 1; i < n; i++)
        res = Math.Max(res, arr[i]);
    return res;
}

// Driver code
public static void Main ()
{
    int []arr = {12, 1234, 45, 67, 1};
    int n = arr.Length;
    Console.WriteLine("Minimum element" +
                      " of array: " +
                      getMin(arr, n) + "\n" );
    Console.WriteLine("Maximum element" +
                      " of array: " +
                      getMax(arr, n));
}

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

PHP

```
<?php
// PHP program to find minimum
// (or maximum) element in an
// array.

function getMin($arr, $n)
{
    $res = $arr[0];
    for ($i = 1; $i < $n; $i++)
        $res = min($res, $arr[$i]);
    return $res;
}

function getMax($arr, $n)
{
    $res = $arr[0];
    for ($i = 1; $i < $n; $i++)
        $res = max($res, $arr[$i]);
    return $res;
}
```

```
// Driver Code
$arr = array(12, 1234, 45, 67, 1);
$n = sizeof($arr);
echo "Minimum element of array: "
    , getMin($arr, $n), "\n";
echo "Maximum element of array: "
    , getMax($arr, $n);

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

Output:

```
Minimum element of array: 1
Maximum element of array: 1234
```

Time Complexity: $O(n)$

Recursive Solution

C++

```
// CPP program to find minimum (or maximum) element
// in an array.
#include <iostream>
using namespace std;

int getMin(int arr[], int n)
{
    // If there is single element, return it.
    // Else return minimum of first element and
    // minimum of remaining array.
    return (n == 1) ? arr[0] : min(arr[0], getMin(arr + 1, n - 1));
}

int getMax(int arr[], int n)
{
    // If there is single element, return it.
    // Else return maximum of first element and
    // maximum of remaining array.
    return (n == 1) ? arr[0] : max(arr[0], getMax(arr + 1, n - 1));
}

int main()
{
    int arr[] = { 12, 1234, 45, 67, 1 };
}
```

```
int n = sizeof(arr) / sizeof(arr[0]);
cout << "Minimum element of array: " << getMin(arr, n) << "\n";
cout << "Maximum element of array: " << getMax(arr, n);
return 0;
}
```

Output:

```
Min of array: 1
Max of array: 1234
```

Using Library functions:

We can use [min_element\(\)](#) and [max_element\(\)](#) to find minimum and maximum of array.

C++

```
// CPP program to find minimum (or maximum) element
// in an array.
#include <iostream>
using namespace std;

int getMin(int arr[], int n)
{
    return *min_element(arr, arr + n);
}

int getMax(int arr[], int n)
{
    return *max_element(arr, arr + n);
}

int main()
{
    int arr[] = { 12, 1234, 45, 67, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Minimum element of array: " << getMin(arr, n) << "\n";
    cout << "Maximum element of array: " << getMax(arr, n);
    return 0;
}
```

Output:

```
Min of array: 1
Max of array: 1234
```

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

Source

<https://www.geeksforgeeks.org/program-find-minimum-maximum-element-array/>

Chapter 118

Program to implement Collatz Conjecture

Program to implement Collatz Conjecture - GeeksforGeeks

Given a positive integer n , the task is to find whether this number reaches to 1 after performing following two operations:-

1. If n is even, then $n = n/2$.
2. If n is odd, then $n = 3*n + 1$.
3. Repeat above steps, until it becomes 1.

For example, for $n = 12$, we get the sequence 12, 6, 3, 10, 5, 16, 8, 4, 2, 1.

Examples:

Input : $n = 4$
Output : Yes

Input : $n = 5$
Output : Yes

The idea is to simply follow given rules and recursively call function with reduced values until it reaches 1. If a value is seen again during recursion, then there is a cycle and we can't reach 1. In this case, we return false.

C++

```
// C++ program to implement Collatz Conjecture
#include<bits/stdc++.h>
```

```
using namespace std;

// Function to find if n reaches to 1 or not.
bool isToOneRec(int n, unordered_set<int> &s)
{
    if (n == 1)
        return true;

    // If there is a cycle formed, we can't r
    // reach 1.
    if (s.find(n) != s.end())
        return false;

    // If n is odd then pass n = 3n+1 else n = n/2
    return (n % 2)? isToOneRec(3*n + 1, s) :
                isToOneRec(n/2, s);
}

// Wrapper over isToOneRec()
bool isToOne(int n)
{
    // To store numbers visited using recursive calls.
    unordered_set<int> s;

    return isToOneRec(n, s);
}

// Drivers code
int main()
{
    int n = 5;
    isToOne(n) ? cout << "Yes" : cout << "No";
    return 0;
}
```

Output:

Yes

The above program is inefficient. The idea is to use [Collatz Conjecture](#). It states that if n is a positive then somehow it will reach to 1 after a certain amount of time. So, by using this fact it can be done in $O(1)$ i.e. just check if n is a positive integer or not. Note that the answer would be false for negative numbers. For negative numbers, the above operations would keep number negative and it would never reach 1.

C++

```
// C++ program to implement Collatz Conjecture
#include<bits/stdc++.h>
using namespace std;

// Function to find if n reaches to 1 or not.
bool isToOne(int n)
{
    // Return true if n is positive
    return (n > 0);
}

// Drivers code
int main()
{
    int n = 5;
    isToOne(n) ? cout << "Yes" : cout <<"No";
    return 0;
}
```

Java

```
// Java program to implement Collatz
// Conjecture
class GFG {

    // Function to find if n reaches
    // to 1 or not.
    static boolean isToOne(int n)
    {

        // Return true if n is positive
        return (n > 0);
    }

    // Drivers code
    public static void main(String[] args)
    {
        int n = 5;

        if(isToOne(n) == true)
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}

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

Python 3

```
# Python 3 program to implement
# Collatz Conjecture

# Function to find if n
# reaches to 1 or not.
def isToOne(n):

    # Return true if n
    # is positive
    return (n > 0)

# Drivers code
n = 5

if isToOne(n) == True:
    print("Yes")
else:
    print("No")

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

C#

```
// C# program to implement
// Collatz Conjecture
using System;

class GFG {

    // Function to find if n
    // reaches to 1 or not.
    static bool isToOne(int n)
    {

        // Return true if n
        // is positive
        return (n > 0);
    }

    // Drivers code
    public static void Main()
    {
        int n = 5;
```

```
        if(isToOne(n) == true)
            Console.Write("Yes") ;
        else
            Console.Write("No");
    }
}
```

// This code is contributed
// by Smitha.

Output:

Yes

We strongly recommend to refer below problem as an exercise:

Maximum Collatz sequence length

Improved By : [Smitha Dinesh Semwal](#)

Source

<https://www.geeksforgeeks.org/c-program-implement-collatz-conjecture/>

Chapter 119

Program to reverse a string (Iterative and Recursive)

Program to reverse a string (Iterative and Recursive) - GeeksforGeeks

Given a string, write a recursive program to reverse it.

Input : s = "abc"
Output : s = "cba"

Input : s = "geeksforgeeks"
Output : s = "skeegrofeskeeg"

Method 1 (Using Stack)

C++

```
// C++ program to reverse a string using stack
#include <bits/stdc++.h>
using namespace std;

void recursiveReverse(string &str)
{
    stack<char> st;
    for (int i=0; i<str.length(); i++)
        st.push(str[i]);
```

```
        for (int i=0; i<str.length(); i++) {
            str[i] = st.top();
            st.pop();
        }
    }

// Driver program
int main()
{
    string str = "geeksforgeeks";
    recursiveReverse(str);
    cout << str;
    return 0;
}
```

Output:

skeegrofkskeeg

Time complexity : $O(n)$

Auxiliary Space : $O(n)$

Method 2 (Iterative)

C++

```
// A Simple Iterative C++ program to reverse
// a string
#include <bits/stdc++.h>
using namespace std;

// Function to reverse a string
void reverseStr(string& str)
{
    int n = str.length();

    // Swap character starting from two
    // corners
    for (int i = 0; i < n / 2; i++)
        swap(str[i], str[n - i - 1]);
}

// Driver program
int main()
{
    string str = "geeksforgeeks";
    reverseStr(str);
}
```



```
    cout << str;
    return 0;
}
```

Java

```
// A Simple Java program
// to reverse a string
import java.util.Scanner;

public class reverseStr
{
    // Function to reverse
    // a string
    void stringReverse()
    {
        String str = "geeksforgeeks";
        int length = str.length();
        StringBuffer revString = new StringBuffer();
        for (int i = length - 1;
             i >= 0; i--)
        {
            revString.append(str.charAt(i));
        }
        System.out.println(revString);
    }

    // Driver Code
    public static void main(String []args)
    {
        reverseStr s= new reverseStr();
        s.stringReverse();
    }
}

// This code is contributed
// by prabhat kumar singh
```

Python

```
# A Simple python program
# to reverse a string

# Function to
# reverse a string
def reverseStr(str):
    n = len(str)
```

```
# initialising a empty
# string 'str1'
str1 = ''
i = n - 1
while i >= 0:

    # copy str
    # to str1
    str1 += str[i]
    i -= 1
print(str1)

# Driver Code
def main():
    str = "geeksforgeeks";
    reverseStr(str);

if __name__=="__main__":
    main()

# This code is contributed
# by prabhat kumar singh
```

PHP

```
<?php
// A Simple Iterative PHP
// program to reverse
// a string

// Function to reverse a string
function reverseStr(&$str)
{
    $n = strlen($str);

    // Swap character starting
    // from two corners
    for ($i = 0; $i < $n / 2; $i++)
        //swap the string
        list($str[$i],
            $str[$n - $i - 1]) = array($str[$n - $i - 1],
            $str[$i]);
}

// Driver Code
$str = "geeksforgeeks";
```

```
reverseStr($str);
echo $str;

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

Output:

skeegrofskeeg

Time complexity : $O(n)$

Auxiliary Space : $O(1)$

Method 3 (Iterative using two pointers)

C++

```
// A Simple Iterative C++ program to reverse
// a string
#include <bits/stdc++.h>
using namespace std;

// Function to reverse a string
void reverseStr(string& str)
{
    int n = str.length();

    // Swap character starting from two
    // corners
    for (int i=0, j=n-1; i<j; i++,j--)
        swap(str[i], str[j]);
}

// Driver program
int main()
{
    string str = "geeksforgeeks";
    reverseStr(str);
    cout << str;
    return 0;
}
```

PHP

```
<?php
```

```
// A Simple Iterative PHP
// program to reverse a string

// Function to reverse a string
function reverseStr (&$str)
{
    $n = strlen($str);

    // Swap character starting
    // from two corners
    for ($i = 0, $j = $n - 1;
        $i < $j; $i++, $j--)
        //swap function
        list($str[$i],
            $str[$j]) = array($str[$j],
                              $str[$i]);
}

// Driver Code
$str = "geeksforgeeks";
reverseStr($str);
echo $str;

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

Output:

skeegrofkskeeg

Time complexity : $O(n)$

Auxiliary Space : $O(1)$

Method 4 (Recursive)

C++

```
// Recursive C++ program to reverse a string
#include <bits/stdc++.h>
using namespace std;

void recursiveReverse(string &str, int i = 0)
{
    int n = str.length();
    if (i == n / 2)
        return;
}
```

```
        swap(str[i], str[n - i - 1]);
        recursiveReverse(str, i + 1);
    }

    // Driver program
    int main()
    {
        string str = "geeksforgeeks";
        recursiveReverse(str);
        cout << str;
        return 0;
    }
```

Output:

skeegrofskeeg

Time complexity : $O(n)$

Auxiliary Space : $O(n)$

Method 5 (Library function)

C++

```
// A quickly written program for reversing a string
// using reverse()
#include<bits/stdc++.h>
using namespace std;
int main()
{
    string str = "geeksforgeeks";

    // Reverse str[beign..end]
    reverse(str.begin(),str.end());

    cout << str;
    return 0;
}
```

Python

```
# A Simple python program
# to reverse a string

# Function to
```

```
# reverse a string
def reverseStr(str):

    # print the string
    # from last
    print(str[::-1])

# Driver Code
def main():
    str = "geeksforgeeks";
    reverseStr(str);

if __name__=="__main__":
    main()

# This code is contributed
# by prabhat kumar singh
```

Output:

skeegrofkskeeg

Improved By : [jit_t](#), prabhat kumar singh

Source

<https://www.geeksforgeeks.org/program-reverse-string-iterative-recursive/>

Chapter 120

Python | Convert a nested list into a flat list

Python | Convert a nested list into a flat list - GeeksforGeeks

The task is to convert a nested list into a single list in python i.e no matter how many levels of nesting is there in python list, all the nested has to be removed in order to convert it to a single containing all the values of all the lists inside the outermost brackets but without any brackets inside.

Examples:

```
Input : l = [1, 2, [3, 4, [5, 6] ], 7, 8, [9, [10] ] ]
Output : l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Input : l = [[[ 'item1', 'item2']], [['item3', 'item4']]]
Output : l = ['item1', 'item2', 'itm3, 'item4"]
```

We use recursion because the levels of nesting cannot be predetermined.

```
# Python code to flat a nested list with
# multiple levels of nesting allowed.

# input list
l = [1, 2, [3, 4, [5, 6]], 7, 8, [9, [10]]]

# output list
output = []

# function used for removing nested
# lists in python.
def reemovNestings(l):
    for i in l:
```

```
        if type(i) == list:
            reemovNestings(i)
        else:
            output.append(i)

# Driver code
print ('The original list: ', l)
reemovNestings(l)
print ('The list after removing nesting: ', output)
```

Output:

```
The original list:  [1, 2, [3, 4, [5, 6]], 7, 8, [9, [10]]]
The list after removing nesting:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Source

<https://www.geeksforgeeks.org/python-convert-a-nested-list-into-a-flat-list/>

Chapter 121

Recaman's sequence

Recaman's sequence - GeeksforGeeks

Given an integer n. Print first n elements of [Recaman's sequence](#).

Examples:

Input : n = 6
Output : 0, 1, 3, 6, 2, 7

Input : n = 17
Output : 0, 1, 3, 6, 2, 7, 13, 20, 12, 21,
11, 22, 10, 23, 9, 24, 8

It is basically a function with domain and co-domain as natural numbers and 0. It is recursively defined as below:

Specifically, let $a(n)$ denote the $(n+1)$ -th term. (0 being already there).

The rule says:

```
a(0) = 0,  
if n > 0 and the number is not  
    already included in the sequence,  
    a(n) = a(n - 1) - n  
else  
    a(n) = a(n-1) + n.
```

Below is simple implementation where we store all n Recaman Sequence numbers in an array. We compute next number using recursive formula mentioned above.

C++

```
// C++ program to print n-th number in Recaman's
// sequence
#include <bits/stdc++.h>
using namespace std;

// Prints first n terms of Recaman sequence
int recaman(int n)
{
    // Create an array to store terms
    int arr[n];

    // First term of the sequence is always 0
    arr[0] = 0;
    printf("%d, ", arr[0]);

    // Fill remaining terms using recursive
    // formula.
    for (int i=1; i< n; i++)
    {
        int curr = arr[i-1] - i;
        int j;
        for (j = 0; j < i; j++)
        {
            // If arr[i-1] - i is negative or
            // already exists.
            if ((arr[j] == curr) || curr < 0)
            {
                curr = arr[i-1] + i;
                break;
            }
        }

        arr[i] = curr;
        printf("%d, ", arr[i]);
    }
}

// Driver code
int main()
{
    int n = 17;
    recaman(n);
    return 0;
}
```

Java

```
// Java program to print n-th number in Recaman's
```

```
// sequence
import java.io.*;

class GFG {

    // Prints first n terms of Recaman sequence
    static void recaman(int n)
    {
        // Create an array to store terms
        int arr[] = new int[n];

        // First term of the sequence is always 0
        arr[0] = 0;
        System.out.print(arr[0]+" ");

        // Fill remaining terms using recursive
        // formula.
        for (int i = 1; i < n; i++)
        {
            int curr = arr[i - 1] - i;
            int j;
            for (j = 0; j < i; j++)
            {
                // If arr[i-1] - i is negative or
                // already exists.
                if ((arr[j] == curr) || curr < 0)
                {
                    curr = arr[i - 1] + i;
                    break;
                }
            }

            arr[i] = curr;
            System.out.print(arr[i]+" ");
        }
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 17;
        recaman(n);
    }
}

// This code is contributed by vt_m
```

Python 3

```
# Python 3 program to print n-th
# number in Recaman's sequence

# Prints first n terms of Recaman
# sequence
def recaman(n):

    # Create an array to store terms
    arr = [0] * n

    # First term of the sequence
    # is always 0
    arr[0] = 0
    print(arr[0], end=" ")

    # Fill remaining terms using
    # recursive formula.
    for i in range(1, n):

        curr = arr[i-1] - i
        for j in range(0, i):

            # If arr[i-1] - i is
            # negative or already
            # exists.
            if ((arr[j] == curr) or curr < 0):
                curr = arr[i-1] + i
                break

        arr[i] = curr
        print(arr[i], end=" ")

# Driver code
n = 17

recaman(n)

# This code is contributed by Smitha.
```

C#

```
// C# program to print n-th number in Recaman's
// sequence
using System;
```

```
class GFG {

    // Prints first n terms of Recaman sequence
    static void recaman(int n)
    {
        // Create an array to store terms
        int []arr = new int[n];

        // First term of the sequence is always 0
        arr[0] = 0;
        Console.Write(arr[0]+" ");

        // Fill remaining terms using recursive
        // formula.
        for (int i = 1; i < n; i++)
        {
            int curr = arr[i - 1] - i;
            int j;
            for (j = 0; j < i; j++)
            {
                // If arr[i-1] - i is negative or
                // already exists.
                if ((arr[j] == curr) || curr < 0)
                {
                    curr = arr[i - 1] + i;
                    break;
                }
            }

            arr[i] = curr;
            Console.Write(arr[i]+" ");
        }
    }

    // Driver code
    public static void Main ()
    {
        int n = 17;
        recaman(n);
    }
}

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

PHP

```
<?php
// PHP program to print n-th
// number in Recaman's sequence

// Prints first n terms
// of Recaman sequence
function recaman($n)
{

    // First term of the
    // sequence is always 0
    $arr[0] = 0;
    echo $arr[0], ", ";

    // Fill remaining terms
    // using recursive formula.
    for ($i = 1; $i < $n; $i++)
    {
        $curr = $arr[$i - 1] - $i;
        $j;
        for ($j = 0; $j < $i; $j++)
        {

            // If arr[i-1] - i
            // is negative or
            // already exists.
            if (($arr[$j] == $curr) || $curr < 0)
            {
                $curr = $arr[$i-1] + $i;
                break;
            }
        }

        $arr[$i] = $curr;
        echo $arr[$i], ", ";
    }
}

// Driver Code
$n = 17;
recaman($n);

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

Output:

0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9, 24, 8,

Time Complexity : $O(n^2)$

Auxiliary Space : $O(n)$

Optimizations :

We can use hashing to store previously computed values and can make this program work in $O(n)$ time.

```
// C++ program to print n-th number in Recaman's
// sequence
#include <bits/stdc++.h>
using namespace std;

// Prints first n terms of Recaman sequence
void recaman(int n)
{
    if (n <= 0)
        return;

    // Print first term and store it in a hash
    printf("%d, ", 0);
    unordered_set<int> s;
    s.insert(0);

    // Print remaining terms using recursive
    // formula.
    int prev = 0;
    for (int i=1; i< n; i++)
    {
        int curr = prev - i;

        // If arr[i-1] - i is negative or
        // already exists.
        if (curr < 0 || s.find(curr) != s.end())
            curr = prev + i;

        s.insert(curr);

        printf("%d, ", curr);
        prev = curr;
    }
}

// Driver code
int main()
{
    int n = 17;
    recaman(n);
}
```

```
    return 0;  
}
```

Output:

0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9, 24, 8,

Time Complexity : $O(n)$

Auxiliary Space : $O(n)$

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

Source

<https://www.geeksforgeeks.org/recamans-sequence/>

Chapter 122

Recursion

Recursion - GeeksforGeeks

What is Recursion?

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are [Towers of Hanoi \(TOH\)](#), [Inorder/Preorder/Postorder Tree Traversals](#), [DFS of Graph](#), etc.

What is base condition in recursion?

In recursive program, the solution to base case is provided and solution of bigger problem is expressed in terms of smaller problems.

```
int fact(int n)
{
    if (n <= 1) // base case
        return 1;
    else
        return n*fact(n-1);
}
```

In the above example, base case for $n \leq 1$ is defined and larger value of number can be solved by converting to smaller one till base case is reached.

How a particular problem is solved using recursion?

The idea is represent a problem in terms of one or more smaller problems, and add one or more base conditions that stop recursion. For example, we compute factorial n if we know factorial of $(n-1)$. Base case for factorial would be $n = 0$. We return 1 when $n = 0$.

Why Stack Overflow error occurs in recursion?

If base case is not reached or not defined, then stack overflow problem may arise. Let us take an example to understand this.

```
int fact(int n)
{
    // wrong base case (it may cause
    // stack overflow).
    if (n == 100)
        return 1;

    else
        return n*fact(n-1);
}
```

If fact(10) is called, it will call fact(9), fact(8), fact(7) and so on but number will never reach 100. So, the base case is not reached. If the memory is exhausted by these functions on stack, it will cause stack overflow error.

What is the difference between direct and indirect recursion?

A function fun is called direct recursive if it calls the same function fun. A function fun is called indirect recursive if it calls another function say fun_new and fun_new calls fun directly or indirectly. Difference between direct and indirect recursion has been illustrated in Table 1.

```
// An example of direct recursion
void directRecFun()
{
    // Some code....

    directRecFun();

    // Some code...
}

// An example of indirect recursion
void indirectRecFun1()
{
    // Some code...

    indirectRecFun2();

    // Some code...
}
void indirectRecFun2()
{
    // Some code...

    indirectRecFun1();
}
```

```
    // Some code...
}
```

What is difference between tailed and non-tailed recursion?

A recursive function is tail recursive when recursive call is the last thing executed by the function. Please refer [tail recursion article](#) for details.

How memory is allocated to different function calls in recursion?

When any function is called from `main()`, the memory is allocated to it on stack. A recursive function calls itself, the memory for called function is allocated on top of memory allocated to calling function and different copy of local variables is created for each function call. When the base case is reached, the function returns its value to the function by whom it is called and memory is de-allocated and the process continues.

Let us take the example how recursion works by taking a simple function.

C++

```
// A C++ program to demonstrate working of
// recursion
#include<bits/stdc++.h>
using namespace std;

void printFun(int test)
{
    if (test < 1)
        return;
    else
    {
        cout << test << " ";
        printFun(test-1);    // statement 2
        cout << test << " ";
        return;
    }
}

int main()
{
    int test = 3;
    printFun(test);
}
```

Java

```
// A Java program to demonstrate working of
// recursion
class GFG{
static void printFun(int test)
```

```
{
    if (test < 1)
        return;
    else
    {
        System.out.printf("%d ",test);
        printFun(test-1); // statement 2
        System.out.printf("%d ",test);
        return;
    }
}

public static void main(String[] args)
{
    int test = 3;
    printFun(test);
}
}

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

Python3

```
# A Python 3 program to
# demonstrate working of
# recursion

def printFun(test):

    if (test < 1):
        return
    else:

        print( test,end = " ")
        printFun(test-1) # statement 2
        print( test,end = " ")
        return

test = 3
printFun(test)

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

C#

```
// A C# program to demonstrate
// working of recursion
using System;

class GFG {

    // function to demonstrate
    // working of recursion
    static void printFun(int test)
    {
        if (test < 1)
            return;
        else
        {
            Console.Write(test + " ");

            // statement 2
            printFun(test - 1);

            Console.Write(test + " ");
            return;
        }
    }

    // Driver Code
    public static void Main(String[] args)
    {
        int test = 3;
        printFun(test);
    }
}

// This code is contributed by Anshul Aggarwal.
```

PHP

```
<?php
// PHP program to demonstrate
// working of recursion

// function to demonstrate
// working of recursion
function printFun($test)
{
    if ($test < 1)
        return;
    else
    {
```

```
        echo("$test ");

        // statement 2
        printFun($test-1);

        echo("$test ");
        return;
    }
}

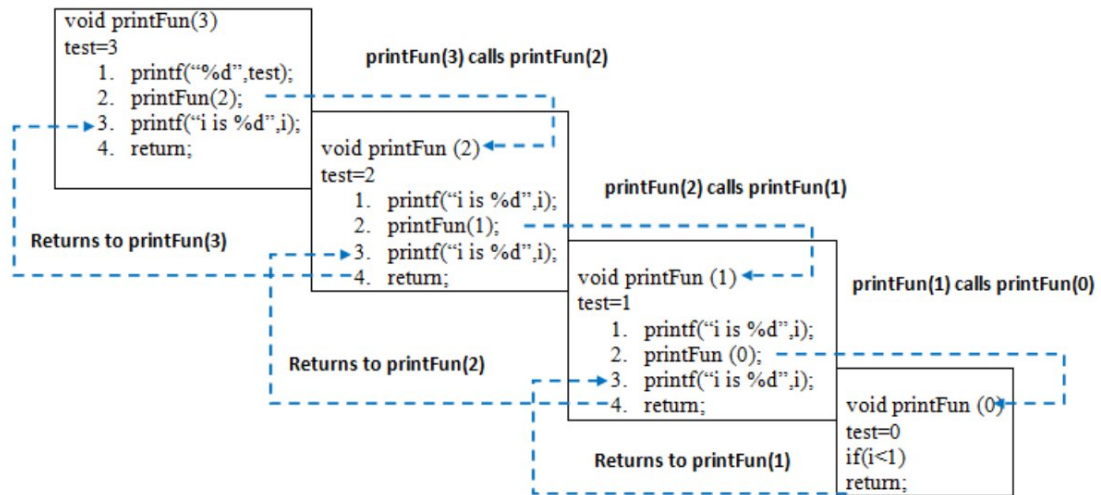
// Driver Code
$test = 3;
printFun($test);

// This code is contributed by
// Smitha Dinesh Semwal.
?>
```

Output :

3 2 1 1 2 3

When **printFun(3)** is called from `main()`, memory is allocated to **printFun(3)** and a local variable `test` is initialized to 3 and statement 1 to 4 are pushed on the stack as shown in below diagram. It first prints '3'. In statement 2, **printFun(2)** is called and memory is allocated to **printFun(2)** and a local variable `test` is initialized to 2 and statement 1 to 4 are pushed in the stack. Similarly, **printFun(2)** calls **printFun(1)** and **printFun(1)** calls **printFun(0)**. **printFun(0)** goes to if statement and it return to **printFun(1)**. Remaining statements of **printFun(1)** are executed and it returns to **printFun(2)** and so on. In the output, value from 3 to 1 are printed and then 1 to 3 are printed. The memory stack has been shown in below diagram.



What are the disadvantages of recursive programming over iterative programming?

Note that both recursive and iterative programs have same problem solving powers, i.e., every recursive program can be written iteratively and vice versa is also true. Recursive program has greater space requirements than iterative program as all functions will remain in stack until base case is reached. It also has greater time requirements because of function calls and return overhead.

What are the advantages of recursive programming over iterative programming?

Recursion provides a clean and simple way to write code. Some problems are inherently recursive like tree traversals, Tower of Hanoi, etc. For such problems it is preferred to write recursive code. We can write such codes also iteratively with the help of stack data structure. For example refer [Inorder Tree Traversal without Recursion](#), [Iterative Tower of Hanoi](#).

Output based practice problems for beginners:

[Practice Questions for Recursion | Set 1](#)
[Practice Questions for Recursion | Set 2](#)
[Practice Questions for Recursion | Set 3](#)
[Practice Questions for Recursion | Set 4](#)
[Practice Questions for Recursion | Set 5](#)
[Practice Questions for Recursion | Set 6](#)
[Practice Questions for Recursion | Set 7](#)

Quiz on Recursion

Coding Practice on Recursion:

All Articles on Recursion

[Recursive Practice Problems with Solutions](#)

Improved By : [Anshul_Aggarwal](#)

Source

<https://www.geeksforgeeks.org/recursion/>

Chapter 123

Recursive Approach to find nth node from the end in the linked list

Recursive Approach to find nth node from the end in the linked list - GeeksforGeeks

Find the nth node from the end in the given linked list using a recursive approach.

Examples:

Input : list: 4->2->1->5->3

n = 2

Output : 5

Algorithm:

```
findNthFromLast(head, n, count, nth_last)
    if head == NULL then
        return

    findNthFromLast(head->next, n, count, nth_last)
    count = count + 1
    if count == n then
        nth_last = head

findNthFromLastUtil(head, n)
    Initialize nth_last = NULL
    Initialize count = 0
```

```
findNthFromLast(head, n, &count, &nth_last)
```

```
if nth_last != NULL then
    print nth_last->data
else
    print "Node does not exists"
```

Note: Parameters **count** and **nth_last** will be pointer variables in **findNthFromLast()**.

C++

```
// C++ implementation to recursively find the nth node from
// the last of the linked list
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
// structure of a node of a linked list
struct Node {
    int data;
    Node* next;
};
```

```
// function to get a new node
Node* getNode(int data)
{
    // allocate space
    Node* newNode = new Node;

    // put in data
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

```
// function to recursively find the nth node from
// the last of the linked list
void findNthFromLast(Node* head, int n, int* count,
                    Node** nth_last)
{
    // if list is empty
    if (!head)
        return;

    // recursive call
    findNthFromLast(head->next, n, count, nth_last);

    // increment count
```

```
*count = *count + 1;

// if true, then head is the nth node from the last
if (*count == n)
    *nth_last = head;
}

// utility function to find the nth node from
// the last of the linked list
void findNthFromLastUtil(Node* head, int n)
{
    // Initialize
    Node* nth_last = NULL;
    int count = 0;

    // find nth node from the last
    findNthFromLast(head, n, &count, &nth_last);

    // if node exists, then print it
    if (nth_last != NULL)
        cout << "Nth node from last is: "
              << nth_last->data;
    else
        cout << "Node does not exists";
}

// Driver program to test above
int main()
{
    // linked list: 4->2->1->5->3
    Node* head = getNode(4);
    head->next = getNode(2);
    head->next->next = getNode(1);
    head->next->next->next = getNode(5);
    head->next->next->next->next = getNode(3);

    int n = 2;

    findNthFromLastUtil(head, n);

    return 0;
}
```

Java

```
// Java implementation to recursively
// find the nth node from the last
// of the linked list
import java.util.*;
```

```
class GFG
{
    static int count = 0, data = 0;

    // a node of a linked list
    static class Node
    {
        int data;
        Node next;
    }

    // function to get a new node
    static Node getNode(int data)
    {
        // allocate space
        Node newNode = new Node();

        // put in data
        newNode.data = data;
        newNode.next = null;
        return newNode;
    }

    // function to recursively
    // find the nth node from
    // the last of the linked list
    static void findNthFromLast(Node head, int n,
        Node nth_last)
    {
        // if list is empty
        if (head == null)
            return;

        // recursive call
        findNthFromLast(head.next, n, nth_last);

        // increment count
        count = count + 1;

        // if true, then head is the
        // nth node from the last
        if (count == n)
        {
            data = head.data;
        }
    }

    // utility function to find
    // the nth node from the last
    // of the linked list
    static void findNthFromLastUtil(Node head, int n)
    {

```

```
// Initialize
Node nth_last = new Node();
count = 0;

// find nth node from the last
findNthFromLast(head, n, nth_last);

// if node exists, then print it
if (nth_last != null)
    System.out.println("Nth node from last is: " +
        data);
else
    System.out.println("Node does not exists");
}

// Driver Code
public static void main(String args[])
{
    // linked list: 4.2.1.5.3
    Node head = getNode(4);
    head.next = getNode(2);
    head.next.next = getNode(1);
    head.next.next.next = getNode(5);
    head.next.next.next.next = getNode(3);

    int n = 2;

    findNthFromLastUtil(head, n);
}
}
```

// This code is contributed
// by Arnab Kundu

Output:

Nth node from last is: 5

Time Complexity: $O(n)$, where 'n' is the number of nodes in the linked list.

Improved By : [andrew1234](#)

Source

<https://www.geeksforgeeks.org/recursive-approach-to-find-nth-node-from-the-end-in-the-linked-list/>

Chapter 124

Recursive Bubble Sort

Recursive Bubble Sort - GeeksforGeeks

Background :

[Bubble Sort](#) is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Example:

First Pass:

(5 1 4 2 8) -> (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps since $5 > 1$.

(1 5 4 2 8) -> (1 4 5 2 8), Swap since $5 > 4$

(1 4 5 2 8) -> (1 4 2 5 8), Swap since $5 > 2$

(1 4 2 5 8) -> (1 4 2 5 8), Now, since these elements are already in order ($8 > 5$), algorithm does not swap them.

Second Pass:

(1 4 2 5 8) -> (1 4 2 5 8)

(1 4 2 5 8) -> (1 2 4 5 8), Swap since $4 > 2$

(1 2 4 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) -> (1 2 4 5 8)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:

(1 2 4 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) -> (1 2 4 5 8)

(1 2 4 5 8) -> (1 2 4 5 8)

Following is iterative Bubble sort algorithm :

```
// Iterative Bubble Sort
bubbleSort(arr[], n)
```

```
{
    for (i = 0; i < n-1; i++)

        // Last i elements are already in place
        for (j = 0; j < n-i-1; j++)
            swap(arr[j], arr[j+1]);
}
```

See [Bubble Sort](#) for more details.

How to implement it recursively?

Recursive Bubble Sort has no performance/implementation advantages, but can be a good question to check one's understanding of Bubble Sort and recursion.

If we take a closer look at Bubble Sort algorithm, we can notice that in first pass, we move largest element to end (Assuming sorting in increasing order). In second pass, we move second largest element to second last position and so on.

Recursion Idea.

1. Base Case: If array size is 1, return.
2. Do One Pass of normal Bubble Sort. This pass fixes last element of current subarray.
3. Recur for all elements except last of current subarray.

Below is implementation of above idea.

C/C++

```
// C/C++ program for recursive implementation
// of Bubble sort
#include <bits/stdc++.h>
using namespace std;

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    // Base case
    if (n == 1)
        return;

    // One pass of bubble sort. After
    // this pass, the largest element
    // is moved (or bubbled) to end.
    for (int i=0; i<n-1; i++)
        if (arr[i] > arr[i+1])
            swap(arr[i], arr[i+1]);

    // Largest element is fixed,
```

```
        // recur for remaining array
        bubbleSort(arr, n-1);
    }

    /* Function to print an array */
    void printArray(int arr[], int n)
    {
        for (int i=0; i < n; i++)
            printf("%d ", arr[i]);
        printf("\n");
    }

    // Driver program to test above functions
    int main()
    {
        int arr[] = {64, 34, 25, 12, 22, 11, 90};
        int n = sizeof(arr)/sizeof(arr[0]);
        bubbleSort(arr, n);
        printf("Sorted array : \n");
        printArray(arr, n);
        return 0;
    }
```

Java

```
// Java program for recursive implementation
// of Bubble sort

import java.util.Arrays;

public class GFG
{
    // A function to implement bubble sort
    static void bubbleSort(int arr[], int n)
    {
        // Base case
        if (n == 1)
            return;

        // One pass of bubble sort. After
        // this pass, the largest element
        // is moved (or bubbled) to end.
        for (int i=0; i<n-1; i++)
            if (arr[i] > arr[i+1])
            {
                // swap arr[i], arr[i+1]
                int temp = arr[i];
                arr[i] = arr[i+1];
            }
    }
}
```



```
        arr[i+1] = temp;
    }

    // Largest element is fixed,
    // recur for remaining array
    bubbleSort(arr, n-1);
}

// Driver Method
public static void main(String[] args)
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};

    bubbleSort(arr, arr.length);

    System.out.println("Sorted array : ");
    System.out.println(Arrays.toString(arr));
}
}
```

Python3

```
# Python Program for implementation of
# Recursive Bubble sort
```

```
def bubble_sort(listt):
    for i, num in enumerate(listt):
        try:
            if listt[i+1] < num:
                listt[i] = listt[i+1]
                listt[i+1] = num
                bubble_sort(listt)
        except IndexError:
            pass
    return listt
```

```
listt = [64, 34, 25, 12, 22, 11, 90]
bubble_sort(listt)
```

```
print("Sorted array:");
for i in range(0, len(listt)):
    print(listt[i], end=' ')
```

```
# Code contributed by Mohit Gupta_OMG
```

C#

```
// C# program for recursive
// implementation of Bubble sort
using System;

class GFG
{
    // A function to implement
    // bubble sort
    static void bubbleSort(int []arr,
                           int n)
    {
        // Base case
        if (n == 1)
            return;

        // One pass of bubble
        // sort. After this pass,
        // the largest element
        // is moved (or bubbled)
        // to end.
        for (int i = 0; i < n - 1; i++)
            if (arr[i] > arr[i + 1])
            {
                // swap arr[i], arr[i+1]
                int temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
            }

        // Largest element is fixed,
        // recur for remaining array
        bubbleSort(arr, n - 1);
    }

    // Driver code
    static void Main()
    {
        int []arr = {64, 34, 25,
                     12, 22, 11, 90};

        bubbleSort(arr, arr.Length);

        Console.WriteLine("Sorted array : ");
        for(int i = 0; i < arr.Length; i++)
            Console.Write(arr[i] + " ");
    }
}
```

```
// This code is contributed  
// by Sam007
```

Output :

Sorted array :
11 12 22 25 34 64 90

Improved By : [Sam007](#)

Source

<https://www.geeksforgeeks.org/recursive-bubble-sort/>

Chapter 125

Recursive Functions

Recursive Functions - GeeksforGeeks

Recursion:

In programming terms a recursive function can be defined as a routine that calls itself directly or indirectly.

Using recursive algorithm, certain problems can be solved quite easily. [Towers of Hanoi \(TOH\)](#) is one such programming exercise. Try to write an *iterative* algorithm for TOH. Moreover, every recursive program can be written using iterative methods (Refer Data Structures by Lipschutz).

Mathematically recursion helps to solve few puzzles easily.

For example, a routine interview question,

In a party of N people, each person will shake her/his hand with each other person only once. On total how many hand-shakes would happen?

Solution:

It can be solved in different ways, graphs, recursion, etc. Let us see, how recursively it can be solved.

There are N persons. Each person shake-hand with each other only once. Considering N -th person, (s)he has to shake-hand with $(N-1)$ persons. Now the problem reduced to small instance of $(N-1)$ persons. Assuming T_N as total shake-hands, it can be formulated recursively.

$T_N = (N-1) + T_{N-1}$ [$T_1 = 0$, i.e. the last person have already shook-hand with every one]

Solving it recursively yields an arithmetic series, which can be evaluated to $N(N-1)/2$.

Exercise: In a party of N couples, only one gender (either male or female) can shake hand with every one. How many shake-hands would happen?

Usually recursive programs results in poor time complexities. An example is Fibonacci series. The time complexity of calculating n -th Fibonacci number using recursion is ap-

proximately 1.6^n . It means the same computer takes almost 60% more time for next Fibonacci number. Recursive Fibonacci algorithm has overlapped subproblems. There are other techniques like *dynamic programming* to improve such overlapped algorithms.

However, few algorithms, (e.g. merge sort, quick sort, etc...) results in optimal time complexity using recursion.

Base Case:

One critical requirement of recursive functions is termination point or base case. Every recursive program must have base case to make sure that the function will terminate. Missing base case results in unexpected behaviour.

Different Ways of Writing Recursive Functions

Function calling itself: (Direct way)

Most of us aware atleast two different ways of writing recursive programs. Given below is towers of Hanoi code. It is an example of direct calling.

CPP

```
// Assuming n-th disk is bottom disk (count down)
void tower(int n, char sourcePole, char destinationPole, char auxiliaryPole)
{
    // Base case (termination condition)
    if(0 == n)
        return;

    // Move first n-1 disks from source pole
    // to auxiliary pole using destination as
    // temporary pole
    tower(n-1, sourcePole, auxiliaryPole,
        destinationPole);

    // Move the remaining disk from source
    // pole to destination pole
    printf("Move the disk %d from %c to %c\n", n,
        sourcePole, destinationPole);

    // Move the n-1 disks from auxiliary (now source)
    // pole to destination pole using source pole as
    // temporary (auxiliary) pole
    tower(n-1, auxiliaryPole, destinationPole,
        sourcePole);
}

void main()
{
    tower(3, 'S', 'D', 'A');
}
```

Java

```
// Assuming n-th disk is
// bottom disk (count down)
class GFG {

static void tower(int n, char sourcePole,
                  char destinationPole, char auxiliaryPole)
{
    // Base case (termination condition)
    if (0 == n)
        return;

    // Move first n-1 disks from source pole
    // to auxiliary pole using destination as
    // temporary pole
    tower(n - 1, sourcePole, auxiliaryPole,
          destinationPole);

    // Move the remaining disk from source
    // pole to destination pole
    System.out.printf("Move the disk %d from %c to %c\n",
                      n, sourcePole, destinationPole);

    // Move the n-1 disks from auxiliary (now source)
    // pole to destination pole using source pole as
    // temporary (auxiliary) pole
    tower(n - 1, auxiliaryPole, destinationPole, sourcePole);
}

public static void main(String[] args)
{
    tower(3, 'S', 'D', 'A');
}
}

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

Python3

```
# Assuming n-th disk is
# bottom disk (count down)
def tower(n, sourcePole, destinationPole, auxiliaryPole):

    # Base case (termination condition)
    if(0 == n):
        return
```

```
# Move first n-1 disks
# from source pole
# to auxiliary pole
# using destination as
# temporary pole
tower(n-1, sourcePole, auxiliaryPole, destinationPole)

# Move the remaining
# disk from source
# pole to destination pole
print("Move the disk",sourcePole,"from",sourcePole,"to",destinationPole)

# Move the n-1 disks from
# auxiliary (now source)
# pole to destination pole
# using source pole as
# temporary (auxiliary) pole
tower(n-1, auxiliaryPole, destinationPole,sourcePole)

# Driver code
tower(3, 'S', 'D', 'A')
```

C#

```
// Assuming n-th disk is bottom disk
// (count down)
using System;

class GFG {

    static void tower(int n, char sourcePole,
                      char destinationPole,
                      char auxiliaryPole)
    {

        // Base case (termination condition)
        if (0 == n)
            return;

        // Move first n-1 disks from source
        // pole to auxiliary pole using
        // destination as temporary pole
        tower(n - 1, sourcePole, auxiliaryPole,
              destinationPole);

        // Move the remaining disk from source
```

```
// pole to destination pole
Console.WriteLine("Move the disk " + n
    + "from " + sourcePole + "to "
    + destinationPole);

// Move the n-1 disks from auxiliary
// (now source) pole to destination
// pole using source pole as temporary
// (auxiliary) pole
tower(n - 1, auxiliaryPole,
    destinationPole, sourcePole);
}

// Driver code
public static void Main()
{
    tower(3, 'S', 'D', 'A');
}

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

PHP

```
<?php
// Assuming n-th disk is
// bottom disk (count down)

function tower($n, $sourcePole,
    $destinationPole,
    $auxiliaryPole)
{
    // Base case
    // (termination condition)
    if(0 == $n)
        return;

    // Move first n-1 disks
    // from source pole to
    // auxiliary pole using
    // destination as temporary
    // pole
    tower($n - 1, $sourcePole,
        $auxiliaryPole,
        $destinationPole);

    // Move the remaining
```



```
// disk from source
// pole to destination pole
echo "Move the disk ", $n,
    " from ", $sourcePole,
    " to ", $destinationPole,
    "\n";

// Move the n-1 disks from
// auxiliary (now source)
// pole to destination pole
// using source pole as
// temporary (auxiliary) pole
tower($n - 1, $auxiliaryPole,
    $destinationPole,
    $sourcePole);
}

// Driver Code
tower(3, 'S', 'D', 'A');

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

Output :

```
Move the disk 1 from S to D
Move the disk 2 from S to A
Move the disk 1 from D to A
Move the disk 3 from S to D
Move the disk 1 from A to S
Move the disk 2 from A to D
Move the disk 1 from S to D
```

The time complexity of TOH can be calculated by formulating number of moves.

We need to move the first N-1 disks from Source to Auxiliary and from Auxiliary to Destination, i.e. the first N-1 disks requires two moves. One more move of last disk from Source to Destination. Mathematically it can be defined recursively.

$$M_N = 2M_{N-1} + 1.$$

We can easily solve the above recursive relation ($2^N - 1$), which is exponential.

Recursion using mutual function call: (Indirect way)

Indirect calling. Though least practical, a function [funA()] can call another function

[funB()] which inturn calls [funA()] former function. In this case both the functions should have the base case.

Defensive Programming:

We can combine defensive coding techniques with recursion for graceful functionality of application. Usually recursive programming is not allowed in safety critical applications, such as flight controls, health monitoring, etc. However, one can use a static count technique to avoid uncontrolled calls (NOT in safety critical systems, may be used in soft real time systems).

```
void recursive(int data)
{
    static callDepth;
    if(callDepth > MAX_DEPTH)
        return;

    // Increase call depth
    callDepth++;

    // do other processing
    recursive(data);

    // do other processing
    // Decrease call depth
    callDepth--;
}
```

callDepth depends on function stack frame size and maximum stack size.

Recursion using function pointers: (Indirect way)

Recursion can also implemented with function pointers. An example is signal handler in POSIX complaint systems. If the handler causes to trigger same event due to which the handler being called, the function will reenter.

Related Articles:

- [Iterative solution to TOH puzzle](#)
- [Tail recursion](#)
- [Quiz on Recursion](#)

Thanks to Venki for writing the above post. 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/recursive-functions/>

Chapter 126

Recursive Implementation of atoi()

Recursive Implementation of atoi() - GeeksforGeeks

The [atoi\(\)](#) function takes a string (which represents an integer) as an argument and returns its value.

We have discussed [iterative implementation of atoi\(\)](#). How to compute recursively?

We strongly recommend you to minimize your browser and try this yourself first

The idea is separate the last digit, recursively compute result for remaining n-1 digits, multiply the result with 10 and add the obtained value to last digit.

Below is C implementation of the idea.

```
// Recursive C program to compute atoi()
#include <stdio.h>
#include <string.h>

// Recursive function to compute atoi()
int myAtoiRecursive(char *str, int n)
{
    // Base case (Only one digit)
    if (n == 1)
        return *str - '0';

    // If more than 1 digits, recur for (n-1), multiplu result with 10
    // and add last digit
    return (10 * myAtoiRecursive(str, n - 1) + str[n-1] - '0');
}

// Driver Program
```

```
int main(void)
{
    char str[] = "112";
    int n = strlen(str);
    printf("%d", myAtoiRecursive(str, n));
    return 0;
}
```

Output:

112

This article is contributed by **Narendra Kangralkar**. 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/recursive-implementation-of-atoi/>

Chapter 127

Recursive Insertion Sort

Recursive Insertion Sort - GeeksforGeeks

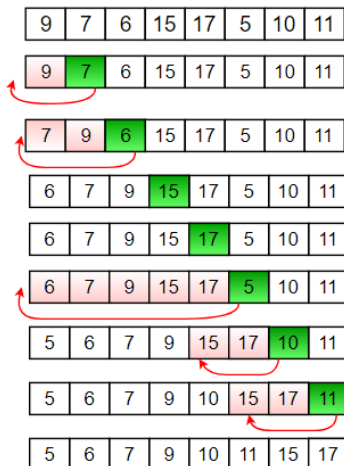
Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Below is an iterative algorithm for insertion sort

Algorithm

```
// Sort an arr[] of size n
insertionSort(arr, n)
    Loop from i = 1 to n-1.
        a) Pick element arr[i] and insert
           it into sorted sequence arr[0..i-1]
```

Example:



Refer [Insertion Sort](#) for more details.

How to implement it recursively?

Recursive Insertion Sort has no performance/implementation advantages, but can be a good question to check one's understanding of Insertion Sort and recursion.

If we take a closer look at Insertion Sort algorithm, we keep processed elements sorted and insert new elements one by one in the inserted array.

Recursion Idea.

1. Base Case: If array size is 1 or smaller, return.
2. Recursively sort first n-1 elements.
3. Insert last element at its correct position in sorted array.

Below is implementation of above idea.

C/C++

```
// Recursive C++ program for insertion sort
#include <iostream>
using namespace std;

// Recursive function to sort an array using
// insertion sort
void insertionSortRecursive(int arr[], int n)
{
    // Base case
    if (n <= 1)
        return;

    // Sort first n-1 elements
    insertionSortRecursive( arr, n-1 );

    // Insert last element at its correct position
    // in sorted array.
    int last = arr[n-1];
    int j = n-2;

    /* Move elements of arr[0..i-1], that are
       greater than key, to one position ahead
       of their current position */
    while (j >= 0 && arr[j] > last)
    {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = last;
}
```

```
// A utility function to print an array of size n
void printArray(int arr[], int n)
{
    for (int i=0; i < n; i++)
        cout << arr[i] <<" ";
}

/* Driver program to test insertion sort */
int main()
{
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr)/sizeof(arr[0]);

    insertionSortRecursive(arr, n);
    printArray(arr, n);

    return 0;
}
```

Java

```
// Recursive Java program for insertion sort

import java.util.Arrays;

public class GFG
{
    // Recursive function to sort an array using
    // insertion sort
    static void insertionSortRecursive(int arr[], int n)
    {
        // Base case
        if (n <= 1)
            return;

        // Sort first n-1 elements
        insertionSortRecursive( arr, n-1 );

        // Insert last element at its correct position
        // in sorted array.
        int last = arr[n-1];
        int j = n-2;

        /* Move elements of arr[0..i-1], that are
        greater than key, to one position ahead
        of their current position */
        while (j >= 0 && arr[j] > last)
```



```
        {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = last;
    }

    // Driver Method
    public static void main(String[] args)
    {
        int arr[] = {12, 11, 13, 5, 6};

        insertionSortRecursive(arr, arr.length);

        System.out.println(Arrays.toString(arr));
    }
}
```

Python

```
# Recursive Python program for insertion sort
# Recursive function to sort an array using insertion sort

def insertionSortRecursive(arr,n):
    # base case
    if n<=1:
        return

    # Sort first n-1 elements
    insertionSortRecursive(arr,n-1)
    '''Insert last element at its correct position
    in sorted array.'''
    last = arr[n-1]
    j = n-2

    # Move elements of arr[0..i-1], that are
    # greater than key, to one position ahead
    # of their current position
    while (j>=0 and arr[j]>last):
        arr[j+1] = arr[j]
        j = j-1

    arr[j+1]=last

# A utility function to print an array of size n
def printArray(arr,n):
    for i in range(n):
        print arr[i],
```

```
# Driver program to test insertion sort
arr = [12,11,13,5,6]
n = len(arr)
insertionSortRecursive(arr, n)
printArray(arr, n)
```

Contributed by Harsh Valecha

C#

```
// Recursive C# program
// for insertion sort
using System;

class GFG
{
    // Recursive function to sort
    // an array using insertion sort
    static void insertionSortRecursive(int []arr,
                                       int n)
    {
        // Base case
        if (n <= 1)
            return;

        // Sort first n-1 elements
        insertionSortRecursive(arr, n - 1);

        // Insert last element at
        // its correct position
        // in sorted array.
        int last = arr[n - 1];
        int j = n - 2;

        /* Move elements of arr[0..i-1],
        that are greater than key, to
        one position ahead of their
        current position */
        while (j >= 0 && arr[j] > last)
        {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = last;
    }
}
```

```
//Driver Code
static void Main()
{
    int []arr = {12, 11, 13, 5, 6};

    insertionSortRecursive(arr, arr.Length);

    for(int i = 0; i < arr.Length; i++)
        Console.Write(arr[i] + " ");
}

// This code is contributed by Sam007
```

PHP

```
<?php
// Recursive PHP program for insertion sort

// Recursive function to sort an
// array using insertion sort
function insertionSortRecursive(&$arr, $n)
{
    // Base case
    if ($n <= 1)
        return;

    // Sort first n-1 elements
    insertionSortRecursive($arr, $n - 1);

    // Insert last element at its correct
    // position in sorted array.
    $last = $arr[$n - 1];
    $j = $n - 2;

    // Move elements of arr[0..i-1], that are
    // greater than key, to one position ahead
    // of their current position
    while ($j >= 0 && $arr[$j] > $last)
    {
        $arr[$j + 1] = $arr[$j];
        $j--;
    }
    $arr[$j + 1] = $last;
}

// A utility function to
```

```
// print an array of size n
function printArray(&$arr, $n)
{
    for ($i = 0; $i < $n; $i++)
        echo $arr[$i]. " ";
}

// Driver Code
$arr = array(12, 11, 13, 5, 6);
$n = sizeof($arr);

insertionSortRecursive($arr, $n);
printArray($arr, $n);

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

Output :

5 6 11 12 13

Improved By : [Sam007](#), [ChitraNayal](#)

Source

<https://www.geeksforgeeks.org/recursive-insertion-sort/>

Chapter 128

Recursive Practice Problems with Solutions

Recursive Practice Problems with Solutions - GeeksforGeeks

- [Recursion](#)
- [Recursive Functions](#)
- [Tail Recursion](#)

- [Given a string, print all possible palindromic partitions](#)
- [Check if a number is Palindrome](#)
- [Print all possible strings of length k that can be formed from a set of n characters](#)
- [Recursive Implementation of atoi\(\)](#)
- [Find all even length binary sequences with same sum of first and second half bits](#)
- [Print all possible expressions that evaluate to a target](#)
- [String with additive sequence](#)
- [Generate all binary strings without consecutive 1's](#)
- [Recursive solution to count substrings with same first and last characters](#)
- [All possible binary numbers of length n with equal sum in both halves](#)
- [Combinations in a String of Digits](#)
- [Count consonants in a string \(Iterative and recursive methods\)](#)
- [Program for length of a string using recursion](#)
- [First uppercase letter in a string \(Iterative and Recursive\)](#)
- [Partition given string in such manner that i'th substring is sum of \(i-1\)'th and \(i-2\)'th substring](#)
- [Power Set in Lexicographic order](#)
- [Function to copy string \(Iterative and Recursive\)](#)

- [Print all possible combinations of r elements in a given array of size n](#)
- [Print all increasing sequences of length k from first n natural numbers](#)
- [Generate all possible sorted arrays from alternate elements of two given sorted arrays](#)
- [Program to find the minimum \(or maximum\) element of an array](#)

- Sum triangle from array
- Reverse a stack using recursion
- Sort a stack using recursion
- Recursive function to delete k-th node from linked list
- Recursive insertion and traversal linked list
- Reverse a Doubly linked list using recursion
- Delete a linked list using recursion
- Print alternate nodes of a linked list using recursion
- Recursive approach for alternating split of Linked List
- Find middle of singly linked list Recursively
- Practice questions for Linked List and Recursion
- Print all leaf nodes of a Binary Tree from left to right
- Leaf nodes from Preorder of a Binary Search Tree (Using Recursion)
- Print all longest common sub-sequences in lexicographical order
- Recursive Tower of Hanoi using 4 pegs / rods
- Time Complexity Analysis | Tower Of Hanoi (Recursion)
- Recursive Bubble Sort
- Recursive Insertion Sort
- Print a pattern without using any loop
- Print all non-increasing sequences of sum equal to a given number x
- Print all n-digit strictly increasing numbers
- Print sums of all subsets of a given set
- Find ways an Integer can be expressed as sum of n-th power of unique natural numbers
- Recaman's sequence
- 1 to n bit numbers with no consecutive 1s in binary representation
- Program for Sum the digits of a given number
- Count ways to express a number as sum of powers
- Find m-th summation of first n natural numbers
- Print N-bit binary numbers having more 1's than 0's in all prefixes
- Generate all passwords from given character set
- Minimum tiles of sizes in powers of two to cover whole area
- Alexander Bogomolny's UnOrdered Permutation Algorithm
- Sum of natural numbers using recursion
- Decimal to binary number using recursion
- Sum of digit of a number using recursion
- Binary to Gray code using recursion
- Number of non-negative integral solutions of sum equation
- Product of 2 Numbers using Recursion
- Print all combinations of factors (Ways to factorize)
- Recursive program for prime number

- Program for Chocolate and Wrapper Puzzle
- N Queen in $O(n)$ space

- Mutual Recursion with example of Hofstadter Female and Male sequences
- Check if a destination is reachable from source with two movements allowed
- Minimum steps to reach a destination
- Identify all Grand-Parent Nodes of each Node in a Map
- C++ program to implement Collatz Conjecture
- Practice Questions for Recursion | Set 1
- Practice Questions for Recursion | Set 2
- Practice Questions for Recursion | Set 3
- Practice Questions for Recursion | Set 4
- Practice Questions for Recursion | Set 5
- Practice Questions for Recursion | Set 6
- Practice Questions for Recursion | Set 7

Chapter 129

Category Archives: Recursion (Recent articles based on Recursion)

Chapter 130

Practice Problems on Geeks for Geeks!

Source

<https://www.geeksforgeeks.org/recursion-practice-problems-solutions/>

Chapter 131

Recursive Tower of Hanoi using 4 pegs / rods

Recursive Tower of Hanoi using 4 pegs / rods - GeeksforGeeks

Tower of Hanoi is a mathematical puzzle. **Traditionally**, It consists of three poles and a number of disks of different sizes which can slide onto any poles. The puzzle starts with the disk in a neat stack in ascending order of size in one pole, the smallest at the top thus making a conical shape. The objective of the puzzle is to move all the disks from one pole (say 'source pole') to another pole (say 'destination pole') with the help of third pole (say auxiliary pole).

The puzzle has the following two rules:

1. You can't place a larger disk onto smaller disk
2. Only one disk can be moved at a time

We've already discussed [recursive solution for Tower of Hanoi](#) with time complexity $O(2^n)$. Using 4 rods, same approach shows significant decrease in time complexity.

Examples:

Input : 3

Output :

Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 3 from rod A to rod D
Move disk 2 from rod C to rod D
Move disk 1 from rod B to rod D

Input : 5

Output :

Move disk 1 from rod A to rod C

```
Move disk 2 from rod A to rod D
Move disk 3 from rod A to rod B
Move disk 2 from rod D to rod B
Move disk 1 from rod C to rod B
Move disk 4 from rod A to rod C
Move disk 5 from rod A to rod D
Move disk 4 from rod C to rod D
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 3 from rod B to rod D
Move disk 2 from rod C to rod D
Move disk 1 from rod A to rod D
```

C/C++

```
// Recursive program for Tower of Hanoi
#include <stdio.h>

// Recursive function to solve Tower
// of Hanoi puzzle
void towerOfHanoi(int n, char from_rod, char to_rod,
                  char aux_rod1, char aux_rod2)
{
    if (n == 0)
        return;
    if (n == 1) {
        printf("\n Move disk %d from rod %c to rod %c",
               n, from_rod, to_rod);
        return;
    }

    towerOfHanoi(n - 2, from_rod, aux_rod1, aux_rod2,
                  to_rod);
    printf("\n Move disk %d from rod %c to rod %c ",
           n - 1, from_rod, aux_rod2);
    printf("\n Move disk %d from rod %c to rod %c ",
           n, from_rod, to_rod);
    printf("\n Move disk %d from rod %c to rod %c ",
           n - 1, aux_rod2, to_rod);
    towerOfHanoi(n - 2, aux_rod1, to_rod, from_rod,
                  aux_rod2);
}

// driver program
int main()
{
    int n = 4; // Number of disks
```

```
// A, B, C and D are names of rods
towerOfHanoi(n, 'A', 'D', 'B', 'C');
return 0;
}
```

Java

```
// Recursive program for Tower of Hanoi
public class GFG {

    // recursive function to solve
    // Tower of Hanoi puzzle
    static void towerOfHanoi(int n, char from_rod,
                             char to_rod, char aux_rod1,
                             char aux_rod2)
    {
        if (n == 0)
            return;
        if (n == 1) {
            System.out.println("Move disk " + n +
                               " from rod " + from_rod +
                               " to rod " + to_rod);
            return;
        }

        towerOfHanoi(n - 2, from_rod, aux_rod1, aux_rod2,
                      to_rod);
        System.out.println("Move disk " + (n - 1) +
                           " from rod " + from_rod +
                           " to rod " + aux_rod2);
        System.out.println("Move disk " + n +
                           " from rod " + from_rod +
                           " to rod " + to_rod);
        System.out.println("Move disk " + (n - 1) +
                           " from rod " + aux_rod2 +
                           " to rod " + to_rod);
        towerOfHanoi(n - 2, aux_rod1, to_rod, from_rod,
                      aux_rod2);
    }

    // Driver method
    public static void main(String args[])
    {
        int n = 4; // Number of disks

        // A, B, C and D are names of rods
        towerOfHanoi(n, 'A', 'D', 'B', 'C');
    }
}
```

```
}
```

Python 3

```
# Recursive program for Tower of Hanoi

# Recursive function to solve Tower
# of Hanoi puzzle
def towerOfHanoi(n, from_rod, to_rod, aux_rod1,
                aux_rod2):

    if (n == 0):
        return
    if (n == 1):
        print("Move disk", n, "from rod",
              from_rod, "c to rod", to_rod)
        return

    towerOfHanoi(n - 2, from_rod, aux_rod1,
                aux_rod2, to_rod)
    print("Move disk", n-1, "from rod", from_rod,
          "c to rod", aux_rod2)

    print("Move disk", n, "from rod", from_rod,
          "c to rod", to_rod)

    print("Move disk", n-1, "from rod", aux_rod2,
          "c to rod", to_rod)

    towerOfHanoi(n - 2, aux_rod1, to_rod,
                from_rod, aux_rod2)

# driver program
n = 4 # Number of disks

# A, B, C and D are names of rods
towerOfHanoi(n, 'A', 'D', 'B', 'C')

# This code is contributed by Smitha.
```

C#

```
// Recursive program for Tower of Hanoi
using System;

public class GFG {
```

```
// recursive function to solve
// Tower of Hanoi puzzle
static void towerOfHanoi(int n, char from_rod,
                        char to_rod, char aux_rod1,
                        char aux_rod2)
{
    if (n == 0)
        return;
    if (n == 1) {
        Console.WriteLine("Move disk " + n +
                          " from rod " + from_rod +
                          " to rod " + to_rod);
        return;
    }

    towerOfHanoi(n - 2, from_rod, aux_rod1,
                aux_rod2, to_rod);
    Console.WriteLine("Move disk " + (n - 1)
                    + " from rod " + from_rod
                    + " to rod " + aux_rod2);
    Console.WriteLine("Move disk " + n +
                    " from rod " + from_rod
                    + " to rod " + to_rod);
    Console.WriteLine("Move disk " + (n - 1)
                    + " from rod " + aux_rod2
                    + " to rod " + to_rod);
    towerOfHanoi(n - 2, aux_rod1, to_rod,
                from_rod, aux_rod2);
}

// Driver method
public static void Main()
{
    int n = 4; // Number of disks

    // A, B, C and D are names of rods
    towerOfHanoi(n, 'A', 'D', 'B', 'C');
}
}
```

// This code is contributed by Anant Agarwal.

PHP

```
<?php
// Recursive PHP program -
// Tower of Hanoi
```

```
// Recursive function to solve
// Tower of Hanoi puzzle
function towerOfHanoi($n, $from_rod, $to_rod,
                    $aux_rod1, $aux_rod2)
{
    if ($n == 0)
        return;
    if ($n == 1)
    {
        echo"\n", "Move disk", $n
            , " from rod ", $from_rod
            , " to rod", $to_rod;

        return;
    }

    towerOfHanoi($n - 2, $from_rod,
                $aux_rod1, $aux_rod2,
                $to_rod);
    echo "\n Move disk ", $n - 1,
        " from rod ", $from_rod,
        " to rod ", $aux_rod2;

    echo "\n Move disk ", $n,
        " from rod ", $from_rod,
        " to rod ", $to_rod ;

    echo "\n Move disk ", $n-1,
        " from rod ", $aux_rod2,
        " to rod ", $to_rod;

    towerOfHanoi($n - 2, $aux_rod1,
                $to_rod, $from_rod,
                $aux_rod2);
}

// Driver Code
// Number of disks
$n = 4;

// A, B, C and D are
// names of rods
towerOfHanoi($n, 'A', 'D', 'B', 'C');

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

Output:

```
Move disk 1 from rod A to rod D
Move disk 2 from rod A to rod B
Move disk 1 from rod D to rod B
Move disk 3 from rod A to rod C
Move disk 4 from rod A to rod D
Move disk 3 from rod C to rod D
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod D
Move disk 1 from rod C to rod D
```

Time Complexity: $O(2^{(N/2)})$

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

Source

<https://www.geeksforgeeks.org/recursive-tower-hanoi-using-4-pegs-rods/>

Chapter 132

Recursive approach for alternating split of Linked List

Recursive approach for alternating split of Linked List - GeeksforGeeks

Given a linked list, split the linked list into two with alternate nodes.

Examples:

```
Input : 1 2 3 4 5 6 7
Output : 1 3 5 7
         2 4 6
```

```
Input : 1 4 5 6
Output : 1 5
         4 6
```

We have discussed [Iterative splitting of linked list](#).

The idea is to begin from two nodes first and second. Let us call these nodes as 'a' and 'b'.

We recurs

C++

```
// C++ code to split linked list
#include <bits/stdc++.h>
using namespace std;

// Node structure
struct Node {
    int data;
    struct Node* next;
};
```

```
// Function to push nodes
// into linked list
void push(Node** head, int new_data)
{
    Node* new_node = new Node;
    new_node->data = new_data;
    new_node->next = (*head);
    (*head) = new_node;
}

// We basically remove link between 'a'
// and its next. Similarly we remove link
// between 'b' and its next. Then we recur
// for remaining lists.
void moveNode(Node* a, Node* b)
{
    if (b == NULL || a == NULL)
        return;

    if (a->next != NULL)
        a->next = a->next->next;

    if (b->next != NULL)
        b->next = b->next->next;

    moveNode(a->next, b->next);
}

// function to split linked list
void alternateSplitLinkedList(Node* head, Node** aRef,
                             Node** bRef)
{
    Node* curr = head;
    *aRef = curr;
    *bRef = curr->next;
    moveNode(*aRef, *bRef);
}

void display(Node* head)
{
    Node* curr = head;
    while (curr != NULL) {
        printf("%d ", curr->data);
        curr = curr->next;
    }
}
```

```
// Driver code
int main()
{
    Node* head = NULL;
    Node *a = NULL, *b = NULL;

    push(&head, 7);
    push(&head, 6);
    push(&head, 5);
    push(&head, 4);
    push(&head, 3);
    push(&head, 2);
    push(&head, 1);

    alternateSplitLinkedList(head, &a, &b);

    printf("a : ");
    display(a);
    printf("\nb : ");
    display(b);

    return 0;
}
```

Output:

```
a : 1 3 5 7
b : 2 4 6
```

Source

<https://www.geeksforgeeks.org/recursive-approach-alternating-split-linked-list/>

Chapter 133

Recursive function to delete k-th node from linked list

Recursive function to delete k-th node from linked list - GeeksforGeeks

Given a singly linked list delete node at k-th position without using loop.

Examples:

Input : list = 9->8->3->5->2->1
k = 4

Output : 9->8->3->2->1

Input : list = 0->0->1->6->2->3
k = 3

Output : 0->0->6->2->3

We recursively reduce value of k. When k reaches 1, we delete current node and return next of current node as new node. When function returns, we link the returned node as next of previous node.

```
// Recursive CPP program to delete k-th node
// of a linked list
#include <bits/stdc++.h>
using namespace std;

struct Node {
    int data;
    struct Node* next;
};

// Deletes k-th node and returns new header.
```

```
Node* deleteNode(Node* start, int k)
{
    // If invalid k
    if (k < 1)
        return start;

    // If linked list is empty
    if (start == NULL)
        return NULL;

    // Base case (start needs to be deleted)
    if (k == 1)
    {
        Node *res = start->next;
        delete(start);
        return res;
    }

    start->next = deleteNode(start->next, k-1);
    return start;
}

/* Utility function to insert a node at the beginning */
void push(struct Node **head_ref, int new_data)
{
    struct Node *new_node = new Node;
    new_node->data = new_data;
    new_node->next = *head_ref;
    *head_ref = new_node;
}

/* Utility function to print a linked list */
void printList(struct Node *head)
{
    while (head!=NULL)
    {
        cout << head->data << " ";
        head = head->next;
    }
    printf("\n");
}

/* Driver program to test above functions */
int main()
{
    struct Node *head = NULL;

    /* Create following linked list
```

```
    12->15->10->11->5->6->2->3 */
    push(&head,3);
    push(&head,2);
    push(&head,6);
    push(&head,5);
    push(&head,11);
    push(&head,10);
    push(&head,15);
    push(&head,12);

    int k = 3;
    head = deleteNode(head, k);

    printf("\nModified Linked List: ");
    printList(head);

    return 0;
}
```

Output:

Modified Linked List: 12 15 11 5 6 2 3

Source

<https://www.geeksforgeeks.org/recursive-function-delete-k-th-node-linked-list/>

Chapter 134

Recursive insertion and traversal linked list

Recursive insertion and traversal linked list - GeeksforGeeks

We have discussed different methods of [linked list insertion](#). How to recursively create a linked list?

Recursively inserting at the end:

To create a Linked list using recursion follow these steps. Below steps insert a new node recursively at the end of linked list.

```
// Function to insert a new node at the
// end of linked list using recursion.
Node* insertEnd(Node* head, int data)
{
    // If linked list is empty, create a
    // new node (Assuming newNode() allocates
    // a new node with given data)
    if (head == NULL)
        return newNode(data);

    // If we have not reached end, keep traversing
    // recursively.
    else
        head->next = insertEnd(head->next, data);
    return head;
}
```

Recursively traversing the list:

The idea is simple, we print current node and recur for remaining list.

```
void traverse(Node* head)
{
    if (head == NULL)
        return;

    // If head is not NULL, print current node
    // and recur for remaining list
    cout << head->data << " ";

    traverse(head->next);
}
```

Complete Program:

Below is complete program to demonstrate working of insert and traverse a linked list.

```
// Recursive CPP program to recursively insert
// a node and recursively print the list.
#include <bits/stdc++.h>
using namespace std;
struct Node {
    int data;
    Node* next;
};

// Allocates a new node with given data
Node *newNode(int data)
{
    Node *new_node = new Node;
    new_node->data = data;
    new_node->next = NULL;
    return new_node;
}

// Function to insert a new node at the
// end of linked list using recursion.
Node* insertEnd(Node* head, int data)
{
    // If linked list is empty, create a
    // new node (Assuming newNode() allocates
    // a new node with given data)
    if (head == NULL)
        return newNode(data);

    // If we have not reached end, keep traversing
    // recursively.
    else
```



```
        head->next = insertEnd(head->next, data);
    return head;
}

void traverse(Node* head)
{
    if (head == NULL)
        return;

    // If head is not NULL, print current node
    // and recur for remaining list
    cout << head->data << " ";

    traverse(head->next);
}

// Driver code
int main()
{
    Node* head = NULL;
    head = insertEnd(head, 6);
    head = insertEnd(head, 8);
    head = insertEnd(head, 10);
    head = insertEnd(head, 12);
    head = insertEnd(head, 14);
    traverse(head);
}
```

Output:

6 8 10 12 14

Source

<https://www.geeksforgeeks.org/recursive-insertion-and-traversal-linked-list/>

Chapter 135

Recursive program for prime number

Recursive program for prime number - GeeksforGeeks

Given a number n, check whether it's [prime number](#) or not using recursion.

Examples:

Input : n = 11

Output : Yes

Input : n = 15

Output : No

The idea is based on [school method to check for prime numbers](#).

C++

```
// CPP Program to find whether a Number
// is Prime or Not using Recursion
#include <bits/stdc++.h>
using namespace std;

// Returns true if n is prime, else
// return false.
// i is current divisor to check.
bool isPrime(int n, int i = 2)
{
    // Base cases
    if (n <= 2)
        return (n == 2) ? true : false;
```

```
    if (n % i == 0)
        return false;
    if (i * i > n)
        return true;

    // Check for next divisor
    return isPrime(n, i + 1);
}
```

```
// Driver Program
int main()
{
    int n = 15;
    if (isPrime(n))
        cout << "Yes";
    else
        cout << "No";

    return 0;
}
```

Java

```
// java Program to find whether a Number
// is Prime or Not using Recursion
import java.util.*;

class GFG {

    // Returns true if n is prime, else
    // return false.
    // i is current divisor to check.
    static boolean isPrime(int n, int i)
    {

        // Base cases
        if (n <= 2)
            return (n == 2) ? true : false;
        if (n % i == 0)
            return false;
        if (i * i > n)
            return true;

        // Check for next divisor
        return isPrime(n, i + 1);
    }

    // Driver program to test above function
```

```
public static void main(String[] args)
{
    int n = 15;

    if (isPrime(n, 2))
        System.out.println("Yes");
    else
        System.out.println("No");
}

// This code is contributed by Sam007.
```

Python3

```
# Python 3 Program to find whether
# a Number is Prime or Not using
# Recursion

# Returns true if n is prime, else
# return false.
# i is current divisor to check.
def isPrime(n, i = 2):

    # Base cases
    if (n <= 2):
        return True if(n == 2) else False
    if (n % i == 0):
        return False
    if (i * i > n):
        return true

    # Check for next divisor
    return isPrime(n, i + 1)

# Driver Program
n = 15
if (isPrime(n)):
    print("Yes")
else:
    print("No")

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

C#

```
// C# Program to find whether a Number
// is Prime or Not using Recursion
using System;

class GFG
{
    // Returns true if n is prime, else
    // return false.
    // i is current divisor to check.
    static bool isPrime(int n, int i)
    {
        // Base cases
        if (n <= 2)
            return (n == 2) ? true : false;
        if (n % i == 0)
            return false;
        if (i * i > n)
            return true;

        // Check for next divisor
        return isPrime(n, i + 1);
    }

    // Driver code
    static void Main()
    {
        int n = 15;

        if (isPrime(n, 2))
            Console.Write("Yes");
        else
            Console.Write("No");
    }
}

// This code is contributed by Sam007
```

PHP

```
<?php
// PHP Program to find whether a Number
// is Prime or Not using Recursion

// Returns true if n is prime, else
// return false.
```

```
// i is current divisor to check.
function isPrime($n, $i = 2)
{
    // Base cases
    if ($n <= 2)
        return ($n == 2) ? true : false;
    if ($n % $i == 0)
        return false;
    if ($i * $i > $n)
        return true;

    // Check for next divisor
    return isPrime($n, $i + 1);
}

// Driver Code
$n = 15;
if (isPrime($n))
    echo("Yes");
else
    echo("No");

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

Output:

No

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

Source

<https://www.geeksforgeeks.org/recursive-program-prime-number/>

Chapter 136

Recursive program to generate power set

Recursive program to generate power set - GeeksforGeeks

Given a set represented as string, write a recursive code to print all subsets of it. The subsets can be printed in any order.

Examples:

```
Input : set = "abc"
Output : "" "a", "b", "c", "ab", "ac", "bc", "abc"
```

```
Input : set = "abcd"
Output : "" "a" "ab" "abc" "abcd" "abd" "ac" "acd"
         "ad" "b" "bc" "bcd" "bd" "c" "cd" "d"
```

Method 1 : The idea is to fix a prefix, generate all subsets beginning with current prefix. After all subsets with a prefix are generated, replace last character with one of the remaining characters.

```
// CPP program to generate power set
#include <bits/stdc++.h>
using namespace std;

// str : Stores input string
// curr : Stores current subset
// index : Index in current subset, curr
void powerSet(string str, int index = -1,
              string curr = "")
{
    int n = str.length();
```

```
// base case
if (index == n)
    return;

// First print current subset
cout << curr << "\n";

// Try appending remaining characters
// to current subset
for (int i = index + 1; i < n; i++) {

    curr += str[i];
    powerSet(str, i, curr);

    // Once all subsets beginning with
    // initial "curr" are printed, remove
    // last character to consider a different
    // prefix of subsets.
    curr.erase(curr.size() - 1);
}
return;
}

// Driver code
int main()
{
    string str = "abc";
    powerSet(str);
    return 0;
}
```

Output:

```
a
ab
abc
ac
b
bc
c
```

Method 2 : The idea is to consider two cases for every character. (i) Consider current character as part of current subset (ii) Do not consider current character as part of current subset.


```
// CPP program to generate power set
#include <bits/stdc++.h>
using namespace std;

// str : Stores input string
// curr : Stores current subset
// index : Index in current subset, curr
void powerSet(string str, int index = 0,
              string curr = "")
{
    int n = str.length();

    // base case
    if (index == n)
    {
        cout << curr << endl;
        return;
    }

    // Two cases for every character
    // (i) We consider the character
    //     as part of current subset
    // (ii) We do not consider current
    //      character as part of current
    //      subset
    powerSet(str, index+1, curr+str[index]);
    powerSet(str, index+1, curr);
}

// Driver code
int main()
{
    string str = "abc";
    powerSet(str);
    return 0;
}
```

Output:

```
abc
ab
ac
a
bc
b
c
```

Iterative program for power set.

Source

<https://www.geeksforgeeks.org/recursive-program-to-generate-power-set/>

Chapter 137

Recursive program to print formula for GCD of n integers

Recursive program to print formula for GCD of n integers - GeeksforGeeks

Given a function **gcd(a, b)** to find GCD (Greatest Common Divisor) of two number. It is also known that GCD of three elements can be found by `gcd(a, gcd(b, c))`, similarly for four element it can find the GCD by `gcd(a, gcd(b, gcd(c, d)))`. Given a positive integer **n**. The task is to print the formula to find the GCD of n integer using given `gcd()` function.

Examples:

Input : n = 3
Output : `gcd(int, gcd(int, int))`

Input : n = 5
Output : `gcd(int, gcd(int, gcd(int, gcd(int, int))))`

Approach: The idea is to use recursion to print the single line command. Now, to write a recursive function, say `recursiveFun(n)`, the required string is composed of `gcd(int, + recursiveFun(n - 1) +)`. This means that the `recursiveFun(n)` should return a string that contains a call to itself and in order to evaluate that value, the recursive function will begin again for `n - 1`. This will, in turn, return another string with a call to `n - 1` and so until `n == 1` and the recursive function instead returns the string "int".

Below is implementation of the above approach:

C++

```
// CPP Program to print single line command
// to find the GCD of n integers
#include <bits/stdc++.h>
```

```
using namespace std;

// Function to print single line command
// to find GCD of n elements.
string recursiveFun(int n)
{
    // base case
    if (n == 1)
        return "int";

    // Recursive Step
    return "gcd(int, " + recursiveFun(n - 1) + ")";
}

// Driver Program
int main()
{
    int n = 5;

    cout << recursiveFun(n) << endl;

    return 0;
}
```

Java

```
// Java Program to print
// single line command to
// find the GCD of n integers
class GFG
{
    // Function to print single
    // line command to find GCD
    // of n elements.
    static String recursiveFun(int n)
    {
        // base case
        if (n == 1)
            return "int";

        // Recursive Step
        return "gcd(int, " +
            recursiveFun(n - 1) + ")";
    }

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

```
{  
    int n = 5;  
  
    System.out.println(recursiveFun(n));  
}  
}
```

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

C#

```
// C# Program to print single  
// line command to find the  
// GCD of n integers  
using System;  
class GFG  
{  
  
    // Function to print single  
    // line command to find GCD  
    // of n elements.  
    static String recursiveFun(int n)  
    {  
        // base case  
        if (n == 1)  
            return "int";  
  
        // Recursive Step  
        return "gcd(int, " +  
            recursiveFun(n - 1) + " )";  
    }  
}
```

```
// Driver Code  
public static void Main()  
{  
    int n = 5;  
  
    Console.Write(recursiveFun(n));  
}  
}
```

```
// This code is contributed  
// by smitha
```

Output:

```
gcd(int, gcd(int, gcd(int, gcd(int, int))))
```

Time Complexity: $O(N)$, where N is the given number.

Improved By : [Smitha Dinesh Semwal](#)

Source

<https://www.geeksforgeeks.org/single-line-command-gcd-n-integers/>

Chapter 138

Recursive solution to count substrings with same first and last characters

Recursive solution to count substrings with same first and last characters - GeeksforGeeks

We are given a string S, we need to find count of all contiguous substrings starting and ending with same character.

Examples :

Input : S = "abcab"

Output : 7

There are 15 substrings of "abcab"

a, ab, abc, abca, abcab, b, bc, bca
bcab, c, ca, cab, a, ab, b

Out of the above substrings, there
are 7 substrings : a, abca, b, bcab,
c, a and b.

Input : S = "aba"

Output : 4

The substrings are a, b, a and aba

We have discussed different solutions in below post.

[Count substrings with same first and last characters](#)

In this article, a simple recursive solution is discussed.

C++

```
// c++ program to count substrings with same
// first and last characters
#include <iostream>
#include <string>
using namespace std;

/* Function to count subtrings with same first and
last characters*/
int countSubstrs(string str, int i, int j, int n)
{
    // base cases
    if (n == 1)
        return 1;
    if (n <= 0)
        return 0;

    int res = countSubstrs(str, i + 1, j, n - 1) +
              countSubstrs(str, i, j - 1, n - 1) -
              countSubstrs(str, i + 1, j - 1, n - 2);

    if (str[i] == str[j])
        res++;

    return res;
}

// driver code
int main()
{
    string str = "abcab";
    int n = str.length();
    cout << countSubstrs(str, 0, n - 1, n);
}
```

Java

```
// Java program to count substrings
// with same first and last characters

class GFG
{
    // Function to count subtrings
    // with same first and
    // last characters
    static int countSubstrs(String str, int i,
                           int j, int n)
    {
        // base cases
```



```
        if (n == 1)
            return 1;
        if (n <= 0)
            return 0;

        int res = countSubstrs(str, i + 1, j, n - 1) +
                  countSubstrs(str, i, j - 1, n - 1) -
                  countSubstrs(str, i + 1, j - 1, n - 2);

        if (str.charAt(i) == str.charAt(j))
            res++;

        return res;
    }

    // Driver code
    public static void main (String[] args)
    {
        String str = "abcaab";
        int n = str.length();
        System.out.print(countSubstrs(str, 0, n - 1, n));
    }
}

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

Python 3

```
# Python 3 program to count substrings with same
# first and last characters

# Function to count subtrings with same first and
# last characters
def countSubstrs(str, i, j, n):

    # base cases
    if (n == 1):
        return 1
    if (n <= 0):
        return 0

    res = (countSubstrs(str, i + 1, j, n - 1)
           + countSubstrs(str, i, j - 1, n - 1)
           - countSubstrs(str, i + 1, j - 1, n - 2))

    if (str[i] == str[j]):
        res += 1
```

```
        return res

# driver code
str = "abcbab"
n = len(str)
print(countSubstrs(str, 0, n - 1, n))

# This code is contributed by Smitha

C#

// C# program to count substrings
// with same first and last characters
using System;

class GFG {

    // Function to count substrings
    // with same first and
    // last characters
    static int countSubstrs(string str, int i,
                           int j, int n)
    {

        // base cases
        if (n == 1)
            return 1;
        if (n <= 0)
            return 0;

        int res = countSubstrs(str, i + 1, j, n - 1)
                  + countSubstrs(str, i, j - 1, n - 1)
                  - countSubstrs(str, i + 1, j - 1, n - 2);

        if (str[i] == str[j])
            res++;

        return res;
    }

    // Driver code
    public static void Main ()
    {
        string str = "abcbab";
        int n = str.Length;

        Console.WriteLine(
            countSubstrs(str, 0, n - 1, n));
    }
}
```

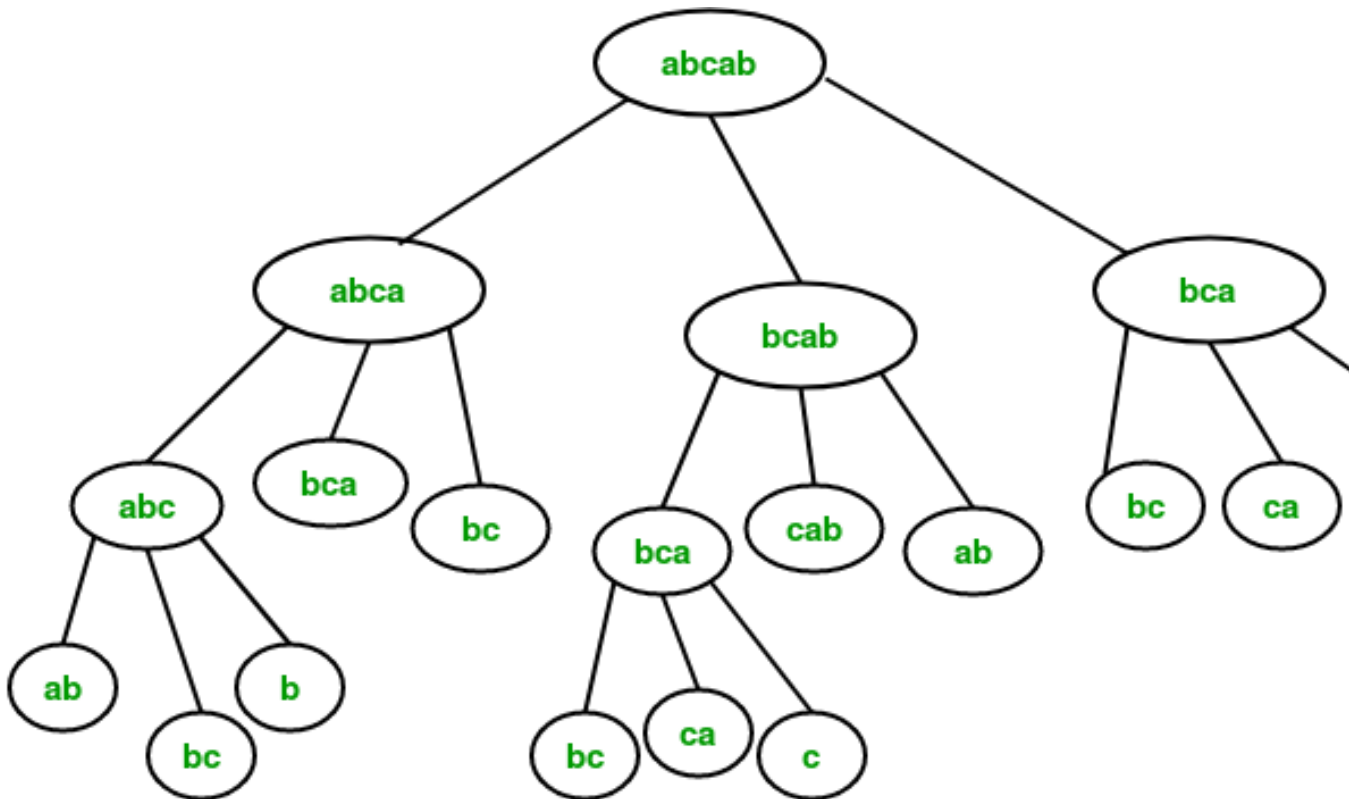
```
    }  
}  
  
// This code is contributed by vt_m.
```

PHP

```
<?php  
// PHP program to count  
// substrings with same  
// first and last characters  
  
//Function to count subtrings  
// with same first and  
// last characters  
function countSubstrs($str, $i,  
                      $j, $n)  
{  
  
    // base cases  
    if ($n == 1)  
        return 1;  
    if ($n <= 0)  
        return 0;  
  
    $res = countSubstrs($str, $i + 1, $j, $n - 1) +  
           countSubstrs($str, $i, $j - 1, $n - 1) -  
           countSubstrs($str, $i + 1, $j - 1, $n - 2);  
  
    if ($str[$i] == $str[$j])  
        $res++;  
  
    return $res;  
}  
  
// Driver Code  
$str = "abcbab";  
$n = strlen($str);  
echo(countSubstrs($str, 0, $n - 1, $n));  
  
// This code is contributed by Ajit.  
?>
```

Output:

The time complexity of above solution is exponential. In Worst case, we may end up doing $O(3^n)$ operations.



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

Source

<https://www.geeksforgeeks.org/recursive-solution-count-substrings-first-last-characters/>

Chapter 139

Recursively Reversing a linked list (A simple implementation)

Recursively Reversing a linked list (A simple implementation) - GeeksforGeeks

Given pointer to the head node of a linked list, the task is to recursively reverse the linked list. We need to reverse the list by changing links between nodes.

Examples:

Input : Head of following linked list

1->2->3->4->NULL

Output : Linked list should be changed to,

4->3->2->1->NULL

Input : Head of following linked list

1->2->3->4->5->NULL

Output : Linked list should be changed to,

5->4->3->2->1->NULL

Input : NULL

Output : NULL

Input : 1->NULL

Output : 1->NULL

We have discussed an iterative and two recursive approaches in [previous post on reverse a linked list](#).

In this approach of reversing a linked list by passing a single pointer what we are trying to do is that we are making the previous node of the current node as his next node to reverse the linked list.

1. We return the pointer of next node to his previous(current) node and then make the previous node as the next node of returned node and then returning the current node.
2. We first traverse till the last node and making the last node as the head node of reversed linked list and then applying the above procedure in the recursive manner.

```
// Recursive C++ program to reverse
// a linked list
#include <iostream>
using namespace std;

/* Link list node */
struct Node {
    int data;
    struct Node* next;
    Node(int data)
    {
        this->data = data;
        next = NULL;
    }
};

struct LinkedList {
    Node* head;
    LinkedList()
    {
        head = NULL;
    }

    /* Function to reverse the linked list */
    Node* reverse(Node* node)
    {
        if (node == NULL)
            return NULL;
        if (node->next == NULL) {
            head = node;
            return node;
        }
        Node* node1 = reverse(node->next);
        node1->next = node;
        node->next = NULL;
        return node;
    }

    /* Function to print linked list */
    void print()
    {
        struct Node* temp = head;
        while (temp != NULL) {
```

```
        cout << temp->data << " ";
        temp = temp->next;
    }
}

void push(int data)
{
    Node* temp = new Node(data);
    temp->next = head;
    head = temp;
}

};

/* Driver program to test above function*/
int main()
{
    /* Start with the empty list */
    LinkedList ll;
    ll.push(20);
    ll.push(4);
    ll.push(15);
    ll.push(85);

    cout << "Given linked list\n";
    ll.print();

    ll.reverse(ll.head);

    cout << "\nReversed Linked list \n";
    ll.print();
    return 0;
}
```

Output:

```
Given linked list
85 15 4 20
Reversed Linked list
20 4 15 85
```

Source

<https://www.geeksforgeeks.org/recursively-reversing-a-linked-list-a-simple-implementation/>

Chapter 140

Recursively remove all adjacent duplicates

Recursively remove all adjacent duplicates - GeeksforGeeks

Given a string, recursively remove adjacent duplicate characters from string. The output string should not have any adjacent duplicates. See following examples.

Input: azxxzy
Output: ay
First "azxxzy" is reduced to "azzy".
The string "azzy" contains duplicates,
so it is further reduced to "ay".

Input: geeksforgeeg
Output: gksfor
First "geeksforgeeg" is reduced to
"gksforgg". The string "gksforgg"
contains duplicates, so it is further
reduced to "gksfor".

Input: caaabbbbaacdddd
Output: Empty String

Input: acaaabbbaacdddd
Output: acac

A simple approach would be to run the input string through multiple passes. In every pass remove all adjacent duplicates from left to right. Stop running passes when there are no duplicates. The worst time complexity of this method would be $O(n^2)$.

```
// C/C++ program to remove all adjacent duplicates
```



```
// from a string
#include <iostream>
#include <string.h>
using namespace std;

// This function recursively removes duplicates
// and returns modified string
char* removeDup(char * str, int n)
{
    int len = strlen(str);
    int k = 0; // To store index of result

    // Start from second character and add
    // unique ones
    for (int i=1; i< len; i++)
    {
        // If different, increment k and add
        // previous character
        if (str[i-1] != str[i])
            str[k++] = str[i-1];

        else
            // Keep skipping (removing) characters
            // while they are same.
            while (str[i-1] == str[i])
                i++;
    }

    // Add last character and terminator
    str[k++] = str[i-1];
    str[k] = '\0';

    // Recur for string if there were some removed
    // character
    if (k != n)
        removeDup(str, k); // Shlemial Painter's Algorithm

    // If all characters were unique
    else return str;
}

int main()
{
    char str1[] = "geeksforgeeg";
    cout << removeDup(str1, strlen(str1)) << endl;

    char str2[] = "azxxxzy";
    cout << removeDup(str2, strlen(str2)) << endl;
```

```
char str3[] = "caaabbbaac";
cout << removeDup(str3, strlen(str3)) << endl;

char str4[] = "gghhg";
cout << removeDup(str4, strlen(str4)) << endl;

char str5[] = "aaaacddddcapp";
cout << removeDup(str5, strlen(str5)) << endl;

char str6[] = "aaaaaaaaa";
cout << removeDup(str6, strlen(str6)) << endl;

char str7[] = "qpaaaaadaaaaadprq";
cout << removeDup(str7, strlen(str7)) << endl;

char str8[] = "acaaabbbacdddd";
cout << removeDup(str8, strlen(str8)) << endl;

char str9[] = "acbbcddc";
cout << removeDup(str9, strlen(str9)) << endl;
}
// This code is contributed by Aditya Goel.
```

Output:

```
gksfor
ay

g
a

qrq
acac
a
```

We can remove all duplicates in $O(n)$ time.

- 1) Start from the leftmost character and remove duplicates at left corner if there are any.
- 2) The first character must be different from its adjacent now. Recur for string of length $n-1$ (string without first character).
- 3) Let the string obtained after reducing right substring of length $n-1$ be *rem_str*. There are three possible cases
 -a) If first character of *rem_str* matches with the first character of original string, remove the first character from *rem_str*.
 -b) Else if the last removed character in recursive calls is same as the first character of the original string. Ignore the first character of original string and return *rem_str*.

.....c) Else, append the first character of the original string at the beginning of *rem_str*.

4) Return *rem_str*.

Following are C++ and Python implementations of the above algorithm.

C++

```
// C/C++ program to remove all adjacent duplicates from a string
#include <iostream>
#include <string.h>
using namespace std;

// Recursively removes adjacent duplicates from str and returns
// new string. las_removed is a pointer to last_removed character
char* removeUtil(char *str, char *last_removed)
{
    // If length of string is 1 or 0
    if (str[0] == '\0' || str[1] == '\0')
        return str;

    // Remove leftmost same characters and recur for remaining
    // string
    if (str[0] == str[1])
    {
        *last_removed = str[0];
        while (str[1] && str[0] == str[1])
            str++;
        str++;
        return removeUtil(str, last_removed);
    }

    // At this point, the first character is definitely different
    // from its adjacent. Ignore first character and recursively
    // remove characters from remaining string
    char* rem_str = removeUtil(str+1, last_removed);

    // Check if the first character of the rem_string matches with
    // the first character of the original string
    if (rem_str[0] && rem_str[0] == str[0])
    {
        *last_removed = str[0];
        return (rem_str+1); // Remove first character
    }

    // If remaining string becomes empty and last removed character
    // is same as first character of original string. This is needed
    // for a string like "acbbcdcd"
    if (rem_str[0] == '\0' && *last_removed == str[0])
        return rem_str;
```

```
// If the two first characters of str and rem_str don't match,
// append first character of str before the first character of
// rem_str.
rem_str--;
rem_str[0] = str[0];
return rem_str;
}

char *remove(char *str)
{
    char last_removed = '\0';
    return removeUtil(str, &last_removed);
}

// Driver program to test above functions
int main()
{
    char str1[] = "geeksforgeeg";
    cout << remove(str1) << endl;

    char str2[] = "azxxxzy";
    cout << remove(str2) << endl;

    char str3[] = "caaabbbaac";
    cout << remove(str3) << endl;

    char str4[] = "gghhg";
    cout << remove(str4) << endl;

    char str5[] = "aaaacdddcapp";
    cout << remove(str5) << endl;

    char str6[] = "aaaaaaaaa";
    cout << remove(str6) << endl;

    char str7[] = "qpaaaaadaaadprq";
    cout << remove(str7) << endl;

    char str8[] = "acaaabbacddd";
    cout << remove(str8) << endl;

    char str9[] = "acbbcdc";
    cout << remove(str9) << endl;

    return 0;
}
```

Python

```
# Python program to remove all adjacent duplicates from a string

# Recursively removes adjacent duplicates from str and returns
# new string. last_removed is a pointer to last_removed character
def removeUtil(string, last_removed):

    # If length of string is 1 or 0
    if len(string) == 0 or len(string) == 1:
        return string

    # Remove leftmost same characters and recur for remaining
    # string
    if string[0] == string[1]:
        last_removed = ord(string[0])
        while len(string) > 1 and string[0] == string[1]:
            string = string[1:]
            string = string[1:]

        return removeUtil(string, last_removed)

    # At this point, the first character is definitely different
    # from its adjacent. Ignore first character and recursively
    # remove characters from remaining string
    rem_str = removeUtil(string[1:], last_removed)

    # Check if the first character of the rem_string matches
    # with the first character of the original string
    if len(rem_str) != 0 and rem_str[0] == string[0]:
        last_removed = ord(string[0])
        return (rem_str[1:])

    # If remaining string becomes empty and last removed character
    # is same as first character of original string. This is needed
    # for a string like "acbbcdcd"
    if len(rem_str) == 0 and last_removed == ord(string[0]):
        return rem_str

    # If the two first characters of str and rem_str don't match,
    # append first character of str before the first character of
    # rem_str.
    return ([string[0]] + rem_str)

def remove(string):
    last_removed = 0
    return toString(removeUtil(toList(string), last_removed))
```

```
# Utility functions
def toList(string):
    x = []
    for i in string:
        x.append(i)
    return x

def toString(x):
    return ''.join(x)

# Driver program
string1 = "geeksforgeeg"
print remove(string1)

string2 = "azxxxzy"
print remove(string2)

string3 = "caaabbbaac"
print remove(string3)

string4 = "gghhg"
print remove(string4)

string5 = "aaaacdddcapp"
print remove(string5)

string6 = "aaaaaaaaa"
print remove(string6)

string7 = "qpaaaaadaaaaadprq"
print remove(string7)

string8 = "acaaabbbacddd"
print remove(string8)

string9 = "acbbcddc"
print remove(string9)

# This code is contributed by BHAVYA JAIN
```

Output:

```
gksfor
ay

g
a
```

```
qrq  
acac  
a
```

Time Complexity: The time complexity of the solution can be written as $T(n) = T(n-k) + O(k)$ where n is length of the input string and k is the number of first characters which are same. Solution of the recurrence is $O(n)$

Thanks to **Prachi Bodke** for suggesting this problem and initial solution. 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/recursively-remove-adjacent-duplicates-given-string/>

Chapter 141

Reduce a number to 1 by performing given operations

Reduce a number to 1 by performing given operations - GeeksforGeeks

Given a number N. The task is to reduce the given number N to 1 in the minimum number of steps. You can perform any one of the below operations in each step.

- **Operation 1:** If the number is even then you can divide the number by 2.
- **Operation 2:** If the number is odd then you are allowed to perform either $(n+1)$ or $(n-1)$.

You need to print the minimum number of steps required to reduce the number N to 1 by performing the above operations.

Examples:

```
Input : n = 15
Output : 5
15 is odd 15+1=16
16 is even 16/2=8
8 is even 8/2=4
4 is even 4/2=2
2 is even 2/2=1
```

```
Input : n = 7
Output : 4
7->6
6->3
3->2
2->1
```


The idea is to recursively compute the minimum number of steps required.

- If the number is even, then we are allowed to only divide the number by 2.
- But, when the number is Odd, we can either increment or decrement it by 1. So, we will use recursion for both $n-1$ and $n+1$ and return the one with the minimum number of operations.

Below is the implementation of above approach:

C++

```
// CPP program to count minimum
// steps to reduce a number
#include <iostream>
#include <cmath>

using namespace std;

int countways(int n)
{
    if (n == 1)
        return 0;
    else if (n % 2 == 0)
        return 1 + countways(n / 2);
    else
        return 1 + min(countways(n - 1),
                        countways(n + 1));
}

// Driver code
int main()
{
    int n = 15;

    cout << countways(n) << "\n";

    return 0;
}
```

Java

```
// Java program to count minimum
// steps to reduce a number
class Geeks {

    static int countways(int n)
```

```
{
    if (n == 1)
        return 0;
    else if (n % 2 == 0)
        return 1 + countways(n / 2);
    else
        return 1 + Math.min(countways(n - 1),
                             countways(n + 1));
}

// Driver code
public static void main(String args[])
{
    int n = 15;

    System.out.println(countways(n));
}
}
```

// This code is contributed by ankita_saini

Output:

5

Improved By : [ankita_saini](#)

Source

<https://www.geeksforgeeks.org/reduce-a-number-to-1-by-performing-given-operations/>

Chapter 142

Remove duplicates from a sorted linked list using recursion

Remove duplicates from a sorted linked list using recursion - GeeksforGeeks

Write a `removeDuplicates()` function which takes a list sorted in non-decreasing order and deletes any duplicate nodes from the list. The list should only be traversed once.

For example if the linked list is 11->11->11->21->43->43->60 then `removeDuplicates()` should convert the list to 11->21->43->60.

Algorithm:

Traverse the list recursively from the head (or start) to end and after completion of recursion calls, compare the next node(returned node) and current node(head). If data of both nodes are equal then return the next (**head-> next**) node else return the current **node(head)**.

Implementation:

Functions other than `removeDuplicates()` are just to create a linked linked list and test `removeDuplicates()`.

```
/* C Program to remove duplicates
   from a sorted linked list */
#include <bits/stdc++.h>
#include <stdlib.h>

/* Link list node */
struct Node {
    int data;
    struct Node* next;
};

/* The function removes duplicates from a sorted list */
struct Node* removeDuplicates(struct Node* head)
{
```

```
/* if head is null then return*/
if (head == NULL)
    return NULL;

/* Remove duplicates from list after head */
head->next = removeDuplicates(head->next);

// Check if head itself is duplicate
if (head->next != NULL &&
    head->next->data == head->data) {

    Node* res = head->next;
    delete head;
    return res;
}

return head;
}

/* UTILITY FUNCTIONS */
/* Function to insert a node at
the beginning of the linked list */
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node = new Node;
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

/* Function to print nodes in a given linked list */
void printList(struct Node* node)
{
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
}

/* Drier program to test above functions*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;

    /* Let us create a sorted linked list to test the functions
Created linked list will be 11->11->11->13->13->20 */
    push(&head, 20);
```

```
push(&head, 13);
push(&head, 13);
push(&head, 11);
push(&head, 11);
push(&head, 11);

printf("\n Linked list before duplicate removal ");
printList(head);

/* Remove duplicates from linked list */
struct Node* h = removeDuplicates(head);

printf("\n Linked list after duplicate removal ");
printList(h);

return 0;
}
```

Output:

```
Linked list before duplicate removal 11 11 11 13 13 20
Linked list after duplicate removal 11 13 20
```

Source

<https://www.geeksforgeeks.org/remove-duplicates-sorted-linked-list-using-recursion/>

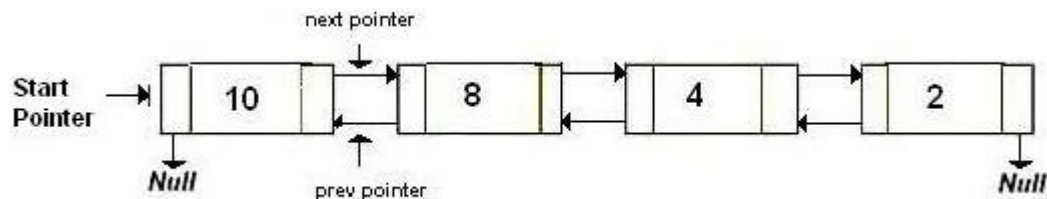
Chapter 143

Reverse a Doubly linked list using recursion

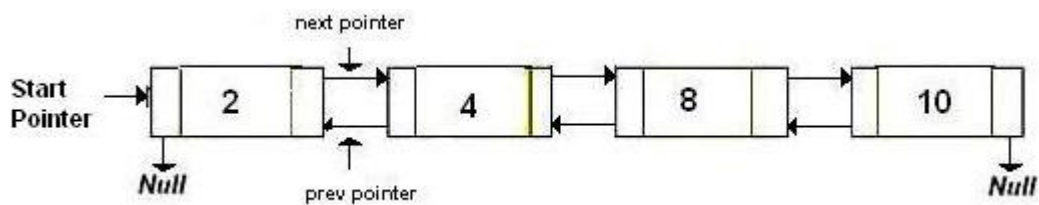
Reverse a Doubly linked list using recursion - GeeksforGeeks

Given a doubly linked list. Reverse it using recursion.

Original Doubly linked list



Reversed Doubly linked list



We have discussed

[Iterative solution to reverse a Doubly Linked List](#)

Algorithm

- 1) If list is empty, return
- 2) Reverse head by swapping head->prev and head->next
- 3) If prev = NULL it means that list is fully reversed. Else reverse(head->prev)

```
// C++ implementation to reverse a doubly
// linked list using recursion
#include <bits/stdc++.h>
using namespace std;

// a node of the doubly linked list
struct Node {
    int data;
    Node *next, *prev;
};

// function to get a new node
Node* getNode(int data)
{
    // allocate space
    Node* new_node = new Node;
    new_node->data = data;
    new_node->next = new_node->prev = NULL;
    return new_node;
}

// function to insert a node at the beginning
// of the Doubly Linked List
void push(Node** head_ref, Node* new_node)
{
    // since we are adding at the beginning,
    // prev is always NULL
    new_node->prev = NULL;

    // link the old list off the new node
    new_node->next = (*head_ref);

    // change prev of head node to new node
    if ((*head_ref) != NULL)
        (*head_ref)->prev = new_node;

    // move the head to point to the new node
    (*head_ref) = new_node;
}

// function to reverse a doubly linked list
Node* Reverse(Node* node)
{
    // If empty list, return
    if (!node)
        return NULL;

    // Otherwise, swap the next and prev
```

```
Node* temp = node->next;
node->next = node->prev;
node->prev = temp;

// If the prev is now NULL, the list
// has been fully reversed
if (!node->prev)
    return node;

// Otherwise, keep going
return Reverse(node->prev);
}

// Function to print nodes in a given doubly
// linked list
void printList(Node* head)
{
    while (head != NULL) {
        cout << head->data << " ";
        head = head->next;
    }
}

// Driver program to test above
int main()
{
    // Start with the empty list
    Node* head = NULL;

    // Create doubly linked: 10<->8<->4<->2 */
    push(&head, getNode(2));
    push(&head, getNode(4));
    push(&head, getNode(8));
    push(&head, getNode(10));
    cout << "Original list: ";
    printList(head);

    // Reverse doubly linked list
    head = Reverse(head);
    cout << "\nReversed list: ";
    printList(head);
    return 0;
}
```

Output:

```
Original list: 10 8 4 2
Reversed list: 2 4 8 10
```


Source

<https://www.geeksforgeeks.org/reverse-doubly-linked-list-using-recursion/>

Chapter 144

Reverse a stack using recursion

Reverse a stack using recursion - GeeksforGeeks

Write a program to reverse a stack using recursion. You are not allowed to use loop constructs like while, for..etc, and you can only use the following ADT functions on Stack S:

isEmpty(S)

push(S)

pop(S)

The idea of the solution is to hold all values in Function Call Stack until the stack becomes empty. When the stack becomes empty, insert all held items one by one at the bottom of the stack.

For example, let the input stack be

```
1 <-- top
2
3
4
```

First 4 is inserted at the bottom.

```
4 <-- top
```

Then 3 is inserted at the bottom

```
4 <-- top
3
```

Then 2 is inserted at the bottom

```
4 <-- top
3
2
```

Then 1 is inserted at the bottom

```
4 <-- top
3
2
1
```

So we need a function that inserts at the bottom of a stack using the above given basic stack function.

void insertAtBottom(): First pops all stack items and stores the popped item in function call stack using recursion. And when stack becomes empty, pushes new item and all items stored in call stack.

void reverse(): This function mainly uses insertAtBottom() to pop all items one by one and insert the popped items at the bottom.

C

```
// C program to reverse a
// stack using recursion
#include<stdio.h>
#include<stdlib.h>
#define bool int

// structure of a stack node
struct sNode
{
    char data;
    struct sNode *next;
};

// Function Prototypes
void push(struct sNode** top_ref,
          int new_data);
int pop(struct sNode** top_ref);
bool isEmpty(struct sNode* top);
void print(struct sNode* top);

// Below is a recursive function
// that inserts an element
// at the bottom of a stack.
void insertAtBottom(struct sNode** top_ref,
                    int item)
{
    if (isEmpty(*top_ref))
        push(top_ref, item);
    else
    {
```

```
        // Hold all items in Function Call
        // Stack until we reach end of the
        // stack. When the stack becomes
        // empty, the isEmpty(*top_ref) becomes
        // true, the above if part is executed
        // and the item is inserted at the bottom
        int temp = pop(top_ref);
        insertAtBottom(top_ref, item);

        // Once the item is inserted
        // at the bottom, push all
        // the items held in Function
        // Call Stack
        push(top_ref, temp);
    }
}

// Below is the function that
// reverses the given stack using
// insertAtBottom()
void reverse(struct sNode** top_ref)
{
    if (!isEmpty(*top_ref))
    {
        // Hold all items in Function
        // Call Stack until we
        // reach end of the stack
        int temp = pop(top_ref);
        reverse(top_ref);

        // Insert all the items (held in
        // Function Call Stack)
        // one by one from the bottom
        // to top. Every item is
        // inserted at the bottom
        insertAtBottom(top_ref, temp);
    }
}

// Driver Code
int main()
{
    struct sNode *s = NULL;
    push(&s, 4);
    push(&s, 3);
    push(&s, 2);
    push(&s, 1);
```

```
    printf("\n Original Stack ");
    print(s);
    reverse(&s);
    printf("\n Reversed Stack ");
    print(s);
    return 0;
}

// Function to check if
// the stack is empty
bool isEmpty(struct sNode* top)
{
    return (top == NULL)? 1 : 0;
}

// Function to push an item to stack
void push(struct sNode** top_ref,
          int new_data)
{
    // allocate node
    struct sNode* new_node =
        (struct sNode*) malloc(sizeof(struct sNode));

    if (new_node == NULL)
    {
        printf("Stack overflow \n");
        exit(0);
    }

    // put in the data
    new_node->data = new_data;

    // link the old list
    // off the new node
    new_node->next = (*top_ref);

    // move the head to
    // point to the new node
    (*top_ref) = new_node;
}

// Function to pop an item from stack
int pop(struct sNode** top_ref)
{
    char res;
    struct sNode *top;
```

```
// If stack is empty then error
if (*top_ref == NULL)
{
    printf("Stack overflow \n");
    exit(0);
}
else
{
    top = *top_ref;
    res = top->data;
    *top_ref = top->next;
    free(top);
    return res;
}
}
```

```
// Function to print a
// linked list
void print(struct sNode* top)
{
    printf("\n");
    while (top != NULL)
    {
        printf(" %d ", top->data);
        top = top->next;
    }
}
```

C++

```
// C++ code to reverse a
// stack using recursion
#include<bits/stdc++.h>
using namespace std;

// using std::stack for
// stack implementation
stack<char> st;

// intializing a string to store
// result of reversed stack
string ns;

// Below is a recursive function
// that inserts an element
// at the bottom of a stack.
char insert_at_bottom(char x)
{

```

```
if(st.size() == 0)
    st.push(x);

else
{
    // All items are held in Function Call
    // Stack until we reach end of the stack
    // When the stack becomes empty, the
    // st.size() becomes 0, the above if
    // part is executed and the item is
    // inserted at the bottom

    char a = st.top();
    st.pop();
    insert_at_bottom(x);

    // push all the items held in
    // Function Call Stack
    // once the item is inserted
    // at the bottom
    st.push(a);
}
}

// Below is the function that
// reverses the given stack using
// insert_at_bottom()
char reverse()
{
    if(st.size() > 0)
    {
        // Hold all items in Function
        // Call Stack until we
        // reach end of the stack
        char x = st.top();
        st.pop();
        reverse();

        // Insert all the items held
        // in Function Call Stack
        // one by one from the bottom
        // to top. Every item is
        // inserted at the bottom
        insert_at_bottom(x);
    }
}
```

```
}

// Driver Code
int main()
{
    // push elements into
    // the stack
    st.push('1');
    st.push('2');
    st.push('3');
    st.push('4');

    cout<<"Original Stack"<<endl;

    // print the elements
    // of original stack
    cout<<"1"<<" "<<"2"<<" "
        <<"3"<<" "<<"4"
        <<endl;

    // function to reverse
    // the stack
    reverse();
    cout<<"Reversed Stack"
        <<endl;

    // storing values of reversed
    // stack into a string for display
    while(!st.empty())
    {
        char p=st.top();
        st.pop();
        ns+=p;
    }

    //display of reversed stack
    cout<<ns[3]<<" "<<ns[2]<<" "
        <<ns[1]<<" "<<ns[0]<<endl;
    return 0;
}

// This code is contributed by Gautam Singh
```

Java

```
// Java code to reverse a
// stack using recursion
```



```
import java.util.Stack;

class Test {

    // using Stack class for
    // stack implementation
    static Stack<Character> st = new Stack<>();

    // Below is a recursive function
    // that inserts an element
    // at the bottom of a stack.
    static void insert_at_bottom(char x)
    {

        if(st.isEmpty())
            st.push(x);

        else
        {

            // All items are held in Function
            // Call Stack until we reach end
            // of the stack. When the stack becomes
            // empty, the st.size() becomes 0, the
            // above if part is executed and
            // the item is inserted at the bottom
            char a = st.peek();
            st.pop();
            insert_at_bottom(x);

            // push all the items held
            // in Function Call Stack
            // once the item is inserted
            // at the bottom
            st.push(a);
        }
    }

    // Below is the function that
    // reverses the given stack using
    // insert_at_bottom()
    static void reverse()
    {
        if(st.size() > 0)
        {

            // Hold all items in Function
            // Call Stack until we
```

```
        // reach end of the stack
        char x = st.peek();
        st.pop();
        reverse();

        // Insert all the items held
        // in Function Call Stack
        // one by one from the bottom
        // to top. Every item is
        // inserted at the bottom
        insert_at_bottom(x);
    }
}

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

    // push elements into
    // the stack
    st.push('1');
    st.push('2');
    st.push('3');
    st.push('4');

    System.out.println("Original Stack");

    System.out.println(st);

    // function to reverse
    // the stack
    reverse();

    System.out.println("Reversed Stack");

    System.out.println(st);
}
}
```

Python3

```
# Python program to reverse a
# stack using recursion

# Below is a recursive function
# that inserts an element
# at the bottom of a stack.
def insertAtBottom(stack, item):
```

```
    if isEmpty(stack):
        push(stack, item)
    else:
        temp = pop(stack)
        insertAtBottom(stack, item)
        push(stack, temp)

# Below is the function that
# reverses the given stack
# using insertAtBottom()
def reverse(stack):
    if not isEmpty(stack):
        temp = pop(stack)
        reverse(stack)
        insertAtBottom(stack, temp)

# Below is a complete running
# program for testing above
# functions.

# Function to create a stack.
# It initializes size of stack
# as 0
def createStack():
    stack = []
    return stack

# Function to check if
# the stack is empty
def isEmpty( stack ):
    return len(stack) == 0

# Function to push an
# item to stack
def push( stack, item ):
    stack.append( item )

# Function to pop an
# item from stack
def pop( stack ):

    # If stack is empty
    # then error
    if(isEmpty( stack )):
        print("Stack Underflow ")
        exit(1)

    return stack.pop()
```

```
# Function to print the stack
def prints(stack):
    for i in range(len(stack)-1, -1, -1):
        print(stack[i], end = ' ')
    print()
```

Driver Code

```
stack = createStack()
push( stack, str(4) )
push( stack, str(3) )
push( stack, str(2) )
push( stack, str(1) )
print("Original Stack ")
prints(stack)
```

```
reverse(stack)
```

```
print("Reversed Stack ")
prints(stack)
```

This code is contributed by Sunny Karira

Output:

```
Original Stack
1 2 3 4
Reversed Stack
4 3 2 1
```

Improved By : [SBanzal](#)

Source

<https://www.geeksforgeeks.org/reverse-a-stack-using-recursion/>

Chapter 145

Reversing a queue using recursion

Reversing a queue using recursion - GeeksforGeeks

Given a queue, write a recursive function to reverse it.

Standard operations allowed :

enqueue(x) : Add an item x to rear of queue.

dequeue() : Remove an item from front of queue.

empty() : Checks if a queue is empty or not.

Examples :

Input : Q = [5, 24, 9, 6, 8, 4, 1, 8, 3, 6]

Output : Q = [6, 3, 8, 1, 4, 8, 6, 9, 24, 5]

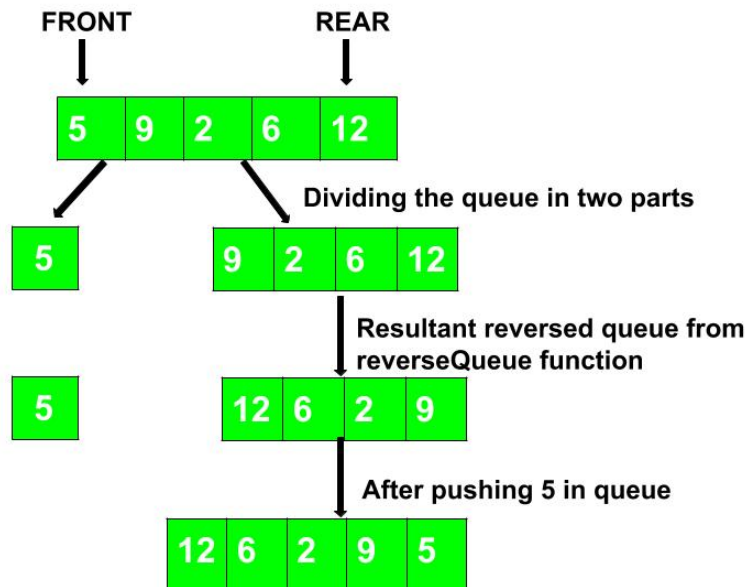
Explanation : Output queue is the reverse of the input queue.

Input : Q = [8, 7, 2, 5, 1]

Output : Q = [1, 5, 2, 7, 8]

Recursive Algorithm :

- 1) Pop element from the queue if the queue has elements otherwise return empty queue.
- 2) Call reverseQueue function for the remaining queue.
- 3) Push the popped element in the resultant reversed queue.



Pseudo Code :

```
queue reverseFunction(queue)
{
    if (queue is empty)
        return queue;
    else {
        data = queue.front()
        queue.pop()
        queue = reverseFunction(queue);
        q.push(data);
        return queue;
    }
}
```

C++

```
// C++ code for reversing a queue
#include <bits/stdc++.h>
using namespace std;

// Utility function to print the queue
void printQueue(queue<long long int> Queue)
{
    while (!Queue.empty()) {
        cout << Queue.front() << " ";
        Queue.pop();
    }
}
```

```
}

// Recursive function to reverse the queue
void reverseQueue(queue<long long int>& q)
{
    // Base case
    if (q.empty())
        return;

    // Dequeue current item (from front)
    long long int data = q.front();
    q.pop();

    // Reverse remaining queue
    reverseQueue(q);

    // Enqueue current item (to rear)
    q.push(data);
}

// Driver code
int main()
{
    queue<long long int> Queue;
    Queue.push(56);
    Queue.push(27);
    Queue.push(30);
    Queue.push(45);
    Queue.push(85);
    Queue.push(92);
    Queue.push(58);
    Queue.push(80);
    Queue.push(90);
    Queue.push(100);
    reverseQueue(Queue);
    printQueue(Queue);
}
```

Java

```
// Java program to reverse a Queue by recursion
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

// Java program to reverse a queue recursively
public class Queue_reverse {
```

```
static Queue<Integer> queue;

// Utility function to print the queue
static void Print()
{
    while (!queue.isEmpty())
    {
        System.out.print(queue.peek() + " ");
        queue.remove();
    }
}

// Recursive function to reverse the queue
static Queue<Integer> reverseQueue(Queue<Integer> q)
{
    // Base case
    if (q.isEmpty())
        return q;

    // Dequeue current item (from front)
    int data = q.peek();
    q.remove();

    // Reverse remaining queue
    q = reverseQueue(q);

    // Enqueue current item (to rear)
    q.add(data);

    return q;
}

// Driver code
public static void main(String args[])
{
    queue = new LinkedList<Integer>();
    queue.add(56);
    queue.add(27);
    queue.add(30);
    queue.add(45);
    queue.add(85);
    queue.add(92);
    queue.add(58);
    queue.add(80);
    queue.add(90);
    queue.add(100);
    queue = reverseQueue(queue);
    Print();
}
```



```
}  
}
```

Python3

```
# Queue Class  
class Queue:  
    def __init__(self):  
        self.items = []  
  
    def isEmpty(self):  
        return self.items == []  
  
    def add(self, item):  
        self.items.append(item)  
  
    def pop(self):  
        return self.items.pop(0)  
  
    def front(self):  
        return self.items[0]  
  
    def printQueue(self):  
        for i in self.items:  
            print(i, end = " ")  
        print("")  
  
# Recursive Function to reverse the queue  
def reverseQueue(q):  
  
    # Base case  
    if (q.isEmpty()):  
        return  
  
    # Dequeue current item (from front)  
    data = q.front();  
    q.pop();  
  
    # Reverse remaining queue  
    reverseQueue(q)  
  
    # Enqueue current item (to rear)  
    q.add(data)  
  
# Driver Code
```

```
q = Queue()
q.add(56)
q.add(27)
q.add(30)
q.add(45)
q.add(85)
q.add(92)
q.add(58)
q.add(80)
q.add(90)
q.add(100)
reverseQueue(q)
q.printQueue()
```

C#

```
// C# code for reversing a queue
using System;
using System.Collections.Generic;

class GFG
{
    // Utility function
    // to print the queue
    static void printQueue(Queue<long> queue)
    {
        while (queue.Count != 0)
        {
            Console.Write(queue.Peek() + " ");
            queue.Dequeue();
        }
    }

    // Recursive function
    // to reverse the queue
    static void reverseQueue(ref Queue<long> q)
    {
        // Base case
        if (q.Count == 0)
            return;

        // Dequeue current
        // item (from front)
        long data = q.Peek();
        q.Dequeue();

        // Reverse remaining queue
        reverseQueue(ref q);
    }
}
```

```
        // Enqueue current
        // item (to rear)
        q.Enqueue(data);
    }

    // Driver code
    static void Main()
    {
        Queue<long> queue = new Queue<long>();
        queue.Enqueue(56);
        queue.Enqueue(27);
        queue.Enqueue(30);
        queue.Enqueue(45);
        queue.Enqueue(85);
        queue.Enqueue(92);
        queue.Enqueue(58);
        queue.Enqueue(80);
        queue.Enqueue(90);
        queue.Enqueue(100);
        reverseQueue(ref queue);
        printQueue(queue);
    }
}

// This code is contributed by
// Manish Shaw(manishshaw1)
```

Output:

100 90 80 58 92 85 45 30 27 56

Time Complexity : $O(n)$.

Improved By : [manishshaw1](#)

Source

<https://www.geeksforgeeks.org/reversing-queue-using-recursion/>

Chapter 146

Shuffle $2n$ integers in format $\{a_1, b_1, a_2, b_2, a_3, b_3, \dots, a_n, b_n\}$ without using extra space

Shuffle $2n$ integers in format $\{a_1, b_1, a_2, b_2, a_3, b_3, \dots, a_n, b_n\}$ without using extra space
- GeeksforGeeks

Given an array of $2n$ elements in the following format $\{a_1, a_2, a_3, a_4, \dots, a_n, b_1, b_2, b_3, b_4, \dots, b_n\}$. The task is shuffle the array to $\{a_1, b_1, a_2, b_2, a_3, b_3, \dots, a_n, b_n\}$ without using extra space.

Examples:

Input : `arr[] = { 1, 2, 9, 15 }`
Output : `1 9 2 15`

Input : `arr[] = { 1, 2, 3, 4, 5, 6 }`
Output : `1 4 2 5 3 6`

Method 1: Brute Force

A brute force solution involves two nested loops to rotate the elements in the second half of the array to the left. The first loop runs n times to cover all elements in the second half of the array. The second loop rotates the elements to the left. Note that the start index in the second loop depends on which element we are rotating and the end index depends on how many positions we need to move to the left.

Below is implementation of this approach:

C++

```
// C++ Naive program to shuffle an array of size 2n
```

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

// function to shuffle an array of size 2n
void shuffleArray(int a[], int n)
{
    // Rotate the element to the left
    for (int i = 0, q = 1, k = n; i < n; i++, k++, q++)
        for (int j = k; j > i + q; j--)
            swap(a[j-1], a[j]);
}

// Driven Program
int main()
{
    int a[] = { 1, 3, 5, 7, 2, 4, 6, 8 };
    int n = sizeof(a) / sizeof(a[0]);

    shuffleArray(a, n/2);

    for (int i = 0; i < n; i++)
        cout << a[i] << " ";

    return 0;
}
```

Java

```
// Java Naive program to shuffle an array of size 2n

import java.util.Arrays;

public class GFG
{
    // method to shuffle an array of size 2n
    static void shuffleArray(int a[], int n)
    {
        // Rotate the element to the left
        for (int i = 0, q = 1, k = n; i < n; i++, k++, q++)
            for (int j = k; j > i + q; j--){
                // swap a[j-1], a[j]
                int temp = a[j-1];
                a[j-1] = a[j];
                a[j] = temp;
            }
    }

    // Driver Method
}
```

```
public static void main(String[] args)
{
    int a[] = { 1, 3, 5, 7, 2, 4, 6, 8 };

    shuffleArray(a, a.length/2);

    System.out.println(Arrays.toString(a));
}
}
```

Python3

```
# Python3 Naive program to
# shuffle an array of size 2n

# Function to shuffle an array of size 2n
def shuffleArray(a, n):

    # Rotate the element to the left
    i, q, k = 0, 1, n
    while(i < n):
        j = k
        while(j > i + q):
            a[j - 1], a[j] = a[j], a[j - 1]
            j -= 1
        i += 1
        k += 1
        q += 1

# Driver Code
a = [1, 3, 5, 7, 2, 4, 6, 8]
n = len(a)
shuffleArray(a, int(n / 2))
for i in range(0, n):
    print(a[i], end = " ")

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

Output:

1 2 3 4 5 6 7 8

Time Complexity: $O(n^2)$

Method 2: (Divide and Conquer)

The idea is to use Divide and Conquer Technique. Divide the given array into half (say

arr1[] and arr2[]) and swap second half element of arr1[] with first half element of arr2[]. Recursively do this for arr1 and arr2.

Let us explain with the help of an example.

1. Let the array be a1, a2, a3, a4, b1, b2, b3, b4
2. Split the array into two halves: a1, a2, a3, a4 : b1, b2, b3, b4
3. Exchange element around the center: exchange a3, a4 with b1, b2 correspondingly.
you get: a1, a2, b1, b2, a3, a4, b3, b4
4. Recursively split a1, a2, b1, b2 into a1, a2 : b1, b2
then split a3, a4, b3, b4 into a3, a4 : b3, b4.
5. Exchange elements around the center for each subarray we get:
a1, b1, a2, b2 and a3, b3, a4, b4.

Note: This solution only handles the case when $n = 2^i$ where $i = 0, 1, 2, \dots$ etc.

Below is implementation of this approach:

C++

```
// C++ Effective program to shuffle an array of size 2n

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

// function to shuffle an array of size 2n
void shuffleArray(int a[], int f, int l)
{
    // If only 2 element, return
    if (l - f == 1)
        return;

    // finding mid to divide the array
    int mid = (f + l) / 2;

    // using temp for swapping first half of second array
    int temp = mid + 1;

    // mmid is use for swapping second half for first array
    int mmid = (f + mid) / 2;

    // Swapping the element
    for (int i = mmid + 1; i <= mid; i++)
        swap(a[i], a[temp++]);

    // Recursively doing for first half and second half
    shuffleArray(a, f, mid);
    shuffleArray(a, mid + 1, l);
}
```

```
// Driven Program
int main()
{
    int a[] = { 1, 3, 5, 7, 2, 4, 6, 8 };
    int n = sizeof(a) / sizeof(a[0]);

    shuffleArray(a, 0, n - 1);

    for (int i = 0; i < n; i++)
        cout << a[i] << " ";

    return 0;
}
```

Java

```
// Java Effective program to shuffle an array of size 2n

import java.util.Arrays;

public class GFG
{
    // method to shuffle an array of size 2n
    static void shuffleArray(int a[], int f, int l)
    {
        // If only 2 element, return
        if (l - f == 1)
            return;

        // finding mid to divide the array
        int mid = (f + l) / 2;

        // using temp for swapping first half of second array
        int temp = mid + 1;

        // mmid is use for swapping second half for first array
        int mmid = (f + mid) / 2;

        // Swapping the element
        for (int i = mmid + 1; i <= mid; i++)
        {
            // swap a[i], a[temp++]
            int temp1 = a[i];
            a[i] = a[temp];
            a[temp++] = temp1;
        }
    }
}
```



```
        // Recursively doing for first half and second half
        shuffleArray(a, f, mid);
        shuffleArray(a, mid + 1, l);
    }

    // Driver Method
    public static void main(String[] args)
    {
        int a[] = { 1, 3, 5, 7, 2, 4, 6, 8 };

        shuffleArray(a, 0, a.length - 1);

        System.out.println(Arrays.toString(a));
    }
}
```

Python3

```
# Python3 effective program to
# shuffle an array of size 2n

# Function to shuffle an array of size 2n
def shuffleArray(a, f, l):

    # If only 2 element, return
    if (l - f == 1):
        return

    # Finding mid to divide the array
    mid = int((f + l) / 2)

    # Using temp for swapping first
    # half of second array
    temp = mid + 1

    # Mid is use for swapping second
    # half for first array
    mmid = int((f + mid) / 2)

    # Swapping the element
    for i in range(mmid + 1, mid + 1):
        (a[i], a[temp]) = (a[temp], a[i])
        temp += 1

    # Recursively doing for first
    # half and second half
    shuffleArray(a, f, mid)
    shuffleArray(a, mid + 1, l)
```

```
# Driver Code
a = [1, 3, 5, 7, 2, 4, 6, 8]
n = len(a)
shuffleArray(a, 0, n - 1)

for i in range(0, n):
    print(a[i], end = " ")

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

Output:

1 2 3 4 5 6 7 8

Time Complexity: $O(n \log n)$

Linear time solution

Source

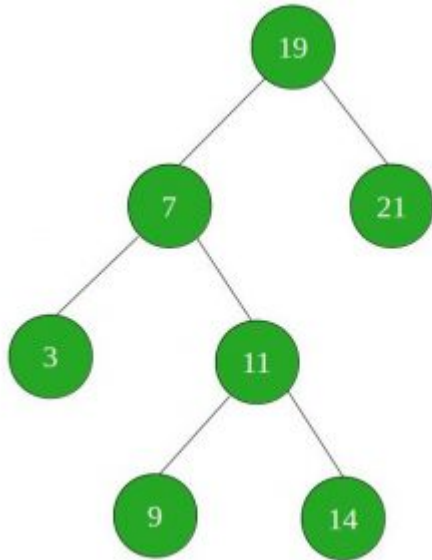
<https://www.geeksforgeeks.org/shuffle-2n-integers-format-a1-b1-a2-b2-a3-b3-bn-without-using-extra-space/>

Chapter 147

Smallest number in BST which is greater than or equal to N

Smallest number in BST which is greater than or equal to N - GeeksforGeeks

Given a [Binary Search Tree](#) and a number N, the task is to find the smallest number in the binary search tree that is greater than or equal to N. Print the value of the element if it exists otherwise print -1.



Examples:

Input: N = 20

Output: 21

Explanation: 21 is the smallest element greater than 20.

Input: N = 18

Output: 19

Explanation: 19 is the smallest element greater than 18.

Approach:

The idea is to follow the recursive approach for solving the problem i.e. start searching for the element from the root.

- If there is a leaf node having a value less than N, then element doesn't exist and return -1.
- Otherwise, if node's value is greater than or equal to N and left child is NULL or less than N then return the node value.
- Else if node's value is less than N, then search for the element in the right subtree.
- Else search for the element in the left subtree by calling the function recursively according to the left or right value.

```
// C++ program to find the smallest value
// greater than or equal to N
#include <bits/stdc++.h>
using namespace std;

struct Node {
    int data;
    Node *left, *right;
};

// To create new BST Node
Node* createNode(int item)
{
    Node* temp = new Node;
    temp->data = item;
    temp->left = temp->right = NULL;

    return temp;
}

// To add a new node in BST
Node* add(Node* node, int key)
{
    // if tree is empty return new node
    if (node == NULL)
        return createNode(key);

    // if key is less then or grater then
    // node value then recur down the tree
    if (key < node->data)
        node->left = add(node->left, key);
```

```
    else if (key > node->data)
        node->right = add(node->right, key);

    // return the (unchanged) node pointer
    return node;
}

// function to find min value less than N
int findMinforN(Node* root, int N)
{
    // If leaf node reached and is smaller than N
    if (root->left == NULL && root->right == NULL
        && root->data < N)
        return -1;

    // If node's value is greater than N and left value
    // is NULL or smaller then return the node value
    if ((root->data >= N && root->left == NULL)
        || (root->data >= N && root->left->data < N))
        return root->data;

    // if node value is smaller than N search in the
    // right subtree
    if (root->data <= N)
        return findMinforN(root->right, N);

    // if node value is greater than N search in the
    // left subtree
    else
        return findMinforN(root->left, N);
}

// Drivers code
int main()
{
    /*      19
       /    \
      7      21
     /  \
    3    11
     /  \
    9    14
    */

    Node* root = NULL;
    root = add(root, 19);
    root = add(root, 7);
    root = add(root, 3);
```

```
    root = add(root, 11);
    root = add(root, 9);
    root = add(root, 13);
    root = add(root, 21);

    int N = 18;
    cout << findMinforN(root, N) << endl;

    return 0;
}
```

Output:

19

Source

<https://www.geeksforgeeks.org/smallest-number-in-bst-which-is-greater-than-or-equal-to-n/>

Chapter 148

Solve the Crossword Puzzle

Solve the Crossword Puzzle - GeeksforGeeks

A **10 x 10 Crossword grid** is provided, along with a set of words (or names of places) which need to be filled into the grid. The cells in the grid are initially, either + signs or – signs. Cells marked with a ‘+’ have to be **left** as they are. Cells marked with a ‘-’ need to be **filled** up with an appropriate character.

You are also given an array of words that need to be filled in Crossword grid.

Example :

Input :

```
+++++++--
-+++++--
-----+-
-+++++--
-+++++--
-+++++--
-+++++--
-+++++--
-+++++--
-+++++--
+-----
++++++++
```

Output :

```
+++++++C
P+++++H
HISTORY++E
Y+++++M
S+++++I
I++++MATHS
CIVICS+++T
S+++++R
+GEOGRAPHY
```

+++++++

The approach behind this is to recursively check for each word in the vertical position and in the horizontal position. Then fill the word in the matrix that can be the best fit in the corresponding position of the grid, then update the crossword grid by filling the gap with that word.

```
// CPP code to fill the crossword puzzle
#include <bits/stdc++.h>
using namespace std;

// ways are to calculate the number of
// possible ways to fill the grid
int ways = 0;

// this function is used to print
// the resultant matrix
void printMatrix(vector<string>& matrix, int n)
{
    for (int i = 0; i < n; i++)
        cout << matrix[i] << endl;
}

// this function checks for the current word
// if it can be placed horizontally or not
// x -> it represent index of row
// y -> it represent index of column
// currentWord -> it represent the
// current word in word array
vector<string> checkHorizontal(int x, int y,
                              vector<string> matrix,
                              string currentWord)
{
    int n = currentWord.length();

    for (int i = 0; i < n; i++) {
        if (matrix[x][y + i] == '#' ||
            matrix[x][y + i] == currentWord[i]) {
            matrix[x][y + i] = currentWord[i];
        }
        else {
            // this shows that word cannot
            // be placed horizontally
            matrix[0][0] = '@';
            return matrix;
        }
    }
}
```



```
    return matrix;
}

// this function checks for the current word
// if it can be placed vertically or not
// x -> it represent index of row
// y -> it represent index of column
// currentWord -> it represent the
// current word in word array
vector<string> checkVertical(int x, int y,
                           vector<string> matrix,
                           string currentWord)
{
    int n = currentWord.length();

    for (int i = 0; i < n; i++) {
        if (matrix[x + i][y] == '#' ||
            matrix[x + i][y] == currentWord[i]) {
            matrix[x + i][y] = currentWord[i];
        }
        else {
            // this shows that word
            // cannot be placed vertically
            matrix[0][0] = '@';
            return matrix;
        }
    }
    return matrix;
}

// this function recursively checks for every
// word that can align vertically in one loop
// and in another loop it checks for those words
// that can align horizontally words -> it
// contains all the words to fill in a crossword
// puzzle matrix -> it contain the current
// state of crossword index -> it represent
// the index of current word n -> it represent
// the length of row or column of the square matrix
void solvePuzzle(vector<string>& words,
                vector<string> matrix,
                int index, int n)
{
    if (index < words.size()) {
        string currentWord = words[index];
        int maxLen = n - currentWord.length();
```

```
// loop to check the words that can align vertically.
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= maxLen; j++) {
        vector<string> temp = checkVertical(j, i,
                                           matrix, currentWord);

        if (temp[0][0] != '@') {
            solvePuzzle(words, temp, index + 1, n);
        }
    }
}

// loop to check the words that can align horizontally.
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= maxLen; j++) {
        vector<string> temp = checkHorizontal(i, j,
                                           matrix, currentWord);

        if (temp[0][0] != '@') {
            solvePuzzle(words, temp, index + 1, n);
        }
    }
}
}
else {
    // calling of print function to
    // print the crossword puzzle
    cout << (ways + 1) << " way to solve the puzzle "
         << endl;
    printMatrix(matrix, n);
    cout << endl;

    // increase the ways
    ways++;
    return;
}
}

// Driver Code
int main()
{
    // length of grid
    int n1 = 10;

    // matrix to hold the grid of puzzle
    vector<string> matrix;
```

```

// take input of puzzle in matrix
// input of grid of size n1 x n1
matrix.push_back("#####");
matrix.push_back("#####");
matrix.push_back("#####");
matrix.push_back("#####");
matrix.push_back("#####");
matrix.push_back("#####");
matrix.push_back("#####");
matrix.push_back("#####");
matrix.push_back("#####");
matrix.push_back("#####");
matrix.push_back("#####");

vector<string> words;

// the words matrix will hold all
// the words need to be filled in the grid
words.push_back("PUNJAB");
words.push_back("JHARKHAND");
words.push_back("MIZORAM");
words.push_back("MUMBAI");

// initialize the number of ways
// to solve the puzzle to zero
ways = 0;

// recursive function to solve the puzzle
// Here 0 is the initial index of words array
// n1 is length of grid
solvePuzzle(words, matrix, 0, n1);
cout << "Number of ways to fill the grid is "
      << ways << endl;

return 0;
}

```

Output:

```

1 way to solve the puzzle
*J*****
*H*****
*A***P***
*R#***U***
*K***N***
*H***J***
*A***A***
*N*MUMBAI*

```

```
*D*****  
***MIZORAM
```

Number of ways to fill the grid is 1

Source

<https://www.geeksforgeeks.org/solve-crossword-puzzle/>

Chapter 149

Sort a stack using a temporary stack

Sort a stack using a temporary stack - GeeksforGeeks

Given a stack of integers, sort it in ascending order using another temporary stack.

Examples:

Input : [34, 3, 31, 98, 92, 23]
Output : [3, 23, 31, 34, 92, 98]

Input : [3, 5, 1, 4, 2, 8]
Output : [1, 2, 3, 4, 5, 8]

We follow this algorithm.

1. Create a temporary stack say **tmpStack**.
2. While input stack is NOT empty do this:
 - Pop an element from input stack call it **temp**
 - while temporary stack is NOT empty and top of temporary stack is greater than temp,
pop from temporary stack and push it to the input stack
 - push **temp** in temporary stack
3. The sorted numbers are in tmpStack

Here is a dry run of above pseudo code.

input: [34, 3, 31, 98, 92, 23]

```
Element taken out: 23
input: [34, 3, 31, 98, 92]
tmpStack: [23]

Element taken out: 92
input: [34, 3, 31, 98]
tmpStack: [23, 92]

Element taken out: 98
input: [34, 3, 31]
tmpStack: [23, 92, 98]

Element taken out: 31
input: [34, 3, 98, 92]
tmpStack: [23, 31]

Element taken out: 92
input: [34, 3, 98]
tmpStack: [23, 31, 92]

Element taken out: 98
input: [34, 3]
tmpStack: [23, 31, 92, 98]

Element taken out: 3
input: [34, 98, 92, 31, 23]
tmpStack: [3]

Element taken out: 23
input: [34, 98, 92, 31]
tmpStack: [3, 23]

Element taken out: 31
input: [34, 98, 92]
tmpStack: [3, 23, 31]

Element taken out: 92
input: [34, 98]
tmpStack: [3, 23, 31, 92]

Element taken out: 98
input: [34]
tmpStack: [3, 23, 31, 92, 98]

Element taken out: 34
input: [98, 92]
tmpStack: [3, 23, 31, 34]
```

```
Element taken out: 92
input: [98]
tmpStack: [3, 23, 31, 34, 92]

Element taken out: 98
input: []
tmpStack: [3, 23, 31, 34, 92, 98]

final sorted list: [3, 23, 31, 34, 92, 98]
```

C++

```
// C++ program to sort a stack using an
// auxiliary stack.
#include <bits/stdc++.h>
using namespace std;

// This function return the sorted stack
stack<int> sortStack(stack<int> &input)
{
    stack<int> tmpStack;

    while (!input.empty())
    {
        // pop out the first element
        int tmp = input.top();
        input.pop();

        // while temporary stack is not empty and top
        // of stack is greater than temp
        while (!tmpStack.empty() && tmpStack.top() > tmp)
        {
            // pop from temporary stack and push
            // it to the input stack
            input.push(tmpStack.top());
            tmpStack.pop();
        }

        // push temp in temporary of stack
        tmpStack.push(tmp);
    }

    return tmpStack;
}

// main function
```

```
int main()
{
    stack<int> input;
    input.push(34);
    input.push(3);
    input.push(31);
    input.push(98);
    input.push(92);
    input.push(23);

    // This is the temporary stack
    stack<int> tmpStack = sortStack(input);
    cout << "Sorted numbers are:\n";

    while (!tmpStack.empty())
    {
        cout << tmpStack.top() << " ";
        tmpStack.pop();
    }
}
```

Java

```
// Java program to sort a stack using
// a auxiliary stack.
import java.util.*;

class SortStack
{
    // This function return the sorted stack
    public static Stack<Integer> sortstack(Stack<Integer>
                                           input)
    {
        Stack<Integer> tmpStack = new Stack<Integer>();
        while(!input.isEmpty())
        {
            // pop out the first element
            int tmp = input.pop();

            // while temporary stack is not empty and
            // top of stack is greater than tmp
            while(!tmpStack.isEmpty() && tmpStack.peek()
                  > tmp)
            {
                // pop from temporary stack and
                // push it to the input stack
                input.push(tmpStack.pop());
            }
        }
    }
}
```



```
        // push temp in temporary of stack
        tmpStack.push(tmp);
    }
    return tmpStack;
}

// Driver Code
public static void main(String args[])
{
    Stack<Integer> input = new Stack<Integer>();
    input.add(34);
    input.add(3);
    input.add(31);
    input.add(98);
    input.add(92);
    input.add(23);

    // This is the temporary stack
    Stack<Integer> tmpStack=sortstack(input);
    System.out.println("Sorted numbers are:");

    while (!tmpStack.empty())
    {
        System.out.print(tmpStack.pop()+" ");
    }
}
// This code is contributed by Danish Kaleem
```

Python3

```
# Python program to sort a
# stack using auxiliary stack.

# This function return the sorted stack
def sortStack ( stack ):
    tmpStack = createStack()
    while(isEmpty(stack) == False):

        # pop out the first element
        tmp = top(stack)
        pop(stack)

        # while temporary stack is not
        # empty and top of stack is
        # greater than temp
        while(isEmpty(tmpStack) == False and
```

```
        int(top(tmpStack)) > int(tmp)):

        # pop from temporary stack and
        # push it to the input stack
        push(stack,top(tmpStack))
        pop(tmpStack)

        # push temp in tempory of stack
        push(tmpStack,tmp)

    return tmpStack

# Below is a complete running
# program for testing above
# function.

# Function to create a stack.
# It initializes size of stack
# as 0
def createStack():
    stack = []
    return stack

# Function to check if
# the stack is empty
def isEmpty( stack ):
    return len(stack) == 0

# Function to push an
# item to stack
def push( stack, item ):
    stack.append( item )

# Function to get top
# item of stack
def top( stack ):
    p = len(stack)
    return stack[p-1]

# Function to pop an
# item from stack
def pop( stack ):

    # If stack is empty
    # then error
    if(isEmpty( stack )):
        print("Stack Underflow ")
        exit(1)
```

```
    return stack.pop()

# Function to print the stack
def prints(stack):
    for i in range(len(stack)-1, -1, -1):
        print(stack[i], end = ' ')
    print()

# Driver Code
stack = createStack()
push( stack, str(34) )
push( stack, str(3) )
push( stack, str(31) )
push( stack, str(98) )
push( stack, str(92) )
push( stack, str(23) )

print("Sorted numbers are: ")
sortedst = sortStack ( stack )
prints(sortedst)

# This code is contributed by
# Prasad Kshirsagar
```

Output:

```
Sorted numbers are:
98 92 34 31 23 3
```

Microsoft

Improved By : [programmer2k17](#), [Prasad_Kshirsagar](#)

Source

<https://www.geeksforgeeks.org/sort-stack-using-temporary-stack/>

Chapter 150

Sort a stack using recursion

Sort a stack using recursion - GeeksforGeeks

Given a stack, sort it using recursion. Use of any loop constructs like while, for..etc is not allowed. We can only use the following ADT functions on Stack S:

```
is_empty(S) : Tests whether stack is empty or not.
push(S)      : Adds new element to the stack.
pop(S)       : Removes top element from the stack.
top(S)       : Returns value of the top element. Note that this
              function does not remove element from the stack.
```

Example:

Input: -3

This problem is mainly a variant of Reverse stack using recursion.

The idea of the solution is to hold all values in Function Call Stack until the stack becomes empty.

Algorithm

We can use below algorithm to sort stack elements:

```
sortStack(stack S)
```

```
if stack is not empty:
    temp = pop(S);
    sortStack(S);
    sortedInsert(S, temp);
```

Below algorithm is to insert element in sorted order:

```
sortedInsert(Stack S, element)
    if stack is empty OR element > top element
        push(S, elem)
    else
        temp = pop(S)
        sortedInsert(S, element)
        push(S, temp)
```

Illustration:

Let given stack be

-3

Let us illustrate sorting of stack using above example:

First pop all the elements from the stack and store popped element in variable 'temp'. After pop

```
temp = -3    --> stack frame #1
temp = 14    --> stack frame #2
temp = 18    --> stack frame #3
temp = -5    --> stack frame #4
temp = 30    --> stack frame #5
```

Now stack is empty and 'insert_in_sorted_order()' function is called and it inserts 30 (from stack frame #5) at the bottom of the stack. Now stack looks like below:

30

Now next element i.e. -5 (from stack frame #4) is picked. Since $-5 < 30$, -5 is inserted at the top

30 -5

Next 18 (from stack frame #3) is picked. Since $18 < 30$, 18 is inserted below 30. Now stack becomes:

30 18
-5

Next 14 (from stack frame #2) is picked. Since $14 < 30$ and $14 < 18$, it is inserted below 18. Now stack becomes:

```
30    14
-5
```

Now -3 (from stack frame #1) is picked, as $-3 < 30$ and $-3 < 18$ and $-3 < 14$, it is inserted below 14. Now stack becomes:

```
30    -3
-5
```

Implementation:

Below is C and Java implementation of above algorithm.

C

```
// C program to sort a stack using recursion
#include <stdio.h>
#include <stdlib.h>

// Stack is represented using linked list
struct stack
{
    int data;
    struct stack *next;
};

// Utility function to initialize stack
void initStack(struct stack **s)
{
    *s = NULL;
}

// Utility function to check if stack is empty
int isEmpty(struct stack *s)
{
    if (s == NULL)
        return 1;
    return 0;
}

// Utility function to push an item to stack
```

```
void push(struct stack **s, int x)
{
    struct stack *p = (struct stack *)malloc(sizeof(*p));

    if (p == NULL)
    {
        fprintf(stderr, "Memory allocation failed.\n");
        return;
    }

    p->data = x;
    p->next = *s;
    *s = p;
}

// Utility function to remove an item from stack
int pop(struct stack **s)
{
    int x;
    struct stack *temp;

    x = (*s)->data;
    temp = *s;
    (*s) = (*s)->next;
    free(temp);

    return x;
}

// Function to find top item
int top(struct stack *s)
{
    return (s->data);
}

// Recursive function to insert an item x in sorted way
void sortedInsert(struct stack **s, int x)
{
    // Base case: Either stack is empty or newly inserted
    // item is greater than top (more than all existing)
    if (isEmpty(*s) || x > top(*s))
    {
        push(s, x);
        return;
    }

    // If top is greater, remove the top item and recur
    int temp = pop(s);
```

```
sortedInsert(s, x);

// Put back the top item removed earlier
push(s, temp);
}

// Function to sort stack
void sortStack(struct stack **s)
{
    // If stack is not empty
    if (!isEmpty(*s))
    {
        // Remove the top item
        int x = pop(s);

        // Sort remaining stack
        sortStack(s);

        // Push the top item back in sorted stack
        sortedInsert(s, x);
    }
}

// Utility function to print contents of stack
void printStack(struct stack *s)
{
    while (s)
    {
        printf("%d ", s->data);
        s = s->next;
    }
    printf("\n");
}

// Driver Program
int main(void)
{
    struct stack *top;

    initStack(&top);
    push(&top, 30);
    push(&top, -5);
    push(&top, 18);
    push(&top, 14);
    push(&top, -3);

    printf("Stack elements before sorting:\n");
    printStack(top);
}
```



```
    sortStack(&top);
    printf("\n\n");

    printf("Stack elements after sorting:\n");
    printStack(top);

    return 0;
}
```

Java

```
// Java program to sort a Stack using recursion
// Note that here predefined Stack class is used
// for stack operation

import java.util.ListIterator;
import java.util.Stack;

class Test
{
    // Recursive Method to insert an item x in sorted way
    static void sortedInsert(Stack<Integer> s, int x)
    {
        // Base case: Either stack is empty or newly inserted
        // item is greater than top (more than all existing)
        if (s.isEmpty() || x > s.peek())
        {
            s.push(x);
            return;
        }

        // If top is greater, remove the top item and recur
        int temp = s.pop();
        sortedInsert(s, x);

        // Put back the top item removed earlier
        s.push(temp);
    }

    // Method to sort stack
    static void sortStack(Stack<Integer> s)
    {
        // If stack is not empty
        if (!s.isEmpty())
        {
            // Remove the top item
            int x = s.pop();
```

```
        // Sort remaining stack
        sortStack(s);

        // Push the top item back in sorted stack
        sortedInsert(s, x);
    }
}

// Utility Method to print contents of stack
static void printStack(Stack<Integer> s)
{
    ListIterator<Integer> lt = s.listIterator();

    // forwarding
    while(lt.hasNext())
        lt.next();

    // printing from top to bottom
    while(lt.hasPrevious())
        System.out.print(lt.previous()+" ");
}

// Driver method
public static void main(String[] args)
{
    Stack<Integer> s = new Stack<>();
    s.push(30);
    s.push(-5);
    s.push(18);
    s.push(14);
    s.push(-3);

    System.out.println("Stack elements before sorting: ");
    printStack(s);

    sortStack(s);

    System.out.println(" \n\nStack elements after sorting:");
    printStack(s);
}
}
```

Output:

Stack elements before sorting:

-3 14 18 -5 30

Stack elements after sorting:

30 18 14 -3 -5

Exercise: Modify above code to reverse stack in descending order.

This article is contributed by **Narendra Kangralkar**. 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/sort-a-stack-using-recursion/>

Chapter 151

String with additive sequence

String with additive sequence - GeeksforGeeks

Given a string, the task is to find whether it contains an additive sequence or not. A string contains an additive sequence if its digits can make a sequence of numbers in which every number is addition of previous two numbers. A valid string should contain at least three digit to make one additive sequence.

Examples:

```
Input : s = "235813"
Output : true
2 + 3 = 5, 3 + 5 = 8, 5 + 8 = 13
```

```
Input : s = "199100199"
Output : true
1 + 99 = 100, 99 + 100 = 199
```

```
Input : s = "12345678"
Output : false
```

This problem can be solved recursively, note that number of digits in added value can't be smaller than digits in any of its operand that is why we will loop till (length of string)/2 for first number and (length of string – first number's length)/ 2 for second number to ignore invalid result.

Next thing to note is, first and second number can't start with 0, which is checked in below code by isValid method. When we call recursively, we check that sum of first and second number is exactly equal to rest of string. If yes then direct return the result else check that sum string is prefix of rest of string or not, If yes then call recursively with second number, sum string and rest of string after removing sum string from rest of string and if sum string is not prefix of rest of string then no solution is available.

Below is C++ implementation.

```
// C++ program to check whether a string
// makes an additive sequence or not
#include <bits/stdc++.h>
using namespace std;

// Checks whether num is valid or not, by
// checking first character and size
bool isValid(string num)
{
    if (num.size() > 1 && num[0] == '0')
        return false;
    return true;
}

// returns int value at pos string, if pos is
// out of bound then returns 0
int val(string a, int pos)
{
    if (pos >= a.length())
        return 0;

    // converting character to integer
    return (a[pos] - '0');
}

// add two number in string form and return
// result as a string
string addString(string a, string b)
{
    string sum = "";
    int i = a.length() - 1;
    int j = b.length() - 1;
    int carry = 0;

    // loop untill both string get processed
    while (i >= 0 || j >= 0)
    {
        int t = val(a, i) + val(b, j) + carry;
        sum += (t % 10 + '0');
        carry = t / 10;
        i--; j--;
    }
    if (carry)
        sum += (carry + '0');
    reverse(sum.begin(), sum.end());
    return sum;
}
```

```

// Recursive method to check c = a + b
bool checkAddition(list<string>& res, string a,
                  string b, string c)
{
    // both first and second number should be valid
    if (!isValid(a) || !isValid(b))
        return false;
    string sum = addString(a, b);

    // if sum is same as c then direct return
    if (sum == c)
    {
        res.push_back(sum);
        return true;
    }

    /* if sum size is greater than c, then no
       possible sequence further OR if c is not
       prefix of sum string, then no possible
       sequence further */
    if (c.size() <= sum.size() ||
        sum != c.substr(0, sum.size()))
        return false;
    else
    {
        res.push_back(sum);

        // next recursive call will have b as first
        // number, sum as second number and string
        // c as third number after removing prefix
        // sum string from c
        return checkAddition(res, b, sum,
                             c.substr(sum.size()));
    }
}

// Method returns additive sequence from string as
// a list
list<string> additiveSequence(string num)
{
    list<string> res;
    int l = num.length();

    // loop untill l/2 only, because if first
    // number is larger, then no possible sequence
    // later
    for (int i = 1; i <= l/2; i++)
    {

```

```
        for (int j = 1; j <= (l - i)/2; j++)
        {
            if (checkAddition(res, num.substr(0, i),
                               num.substr(i, j),
                               num.substr(i + j)))
            {
                // adding first and second number at
                // front of result list
                res.push_front(num.substr(i, j));
                res.push_front(num.substr(0, i));
                return res;
            }
        }
    }

    // If code execution reaches here, then string
    // doesn't have any additive sequence
    res.clear();
    return res;
}

// Method to print result list
void printResult(list<string> res)
{
    for (auto it = res.begin(); it != res.end(); it++)
        cout << *it << " ";
    cout << endl;
}

// Driver code to test above methods
int main()
{
    string num = "235813";
    list<string> res = additiveSequence(num);
    printResult(res);

    num = "199100199";
    res = additiveSequence(num);
    printResult(res);
    return 0;
}
```

Output:

```
2 3 5 8 13
1 99 100 199
```

Source

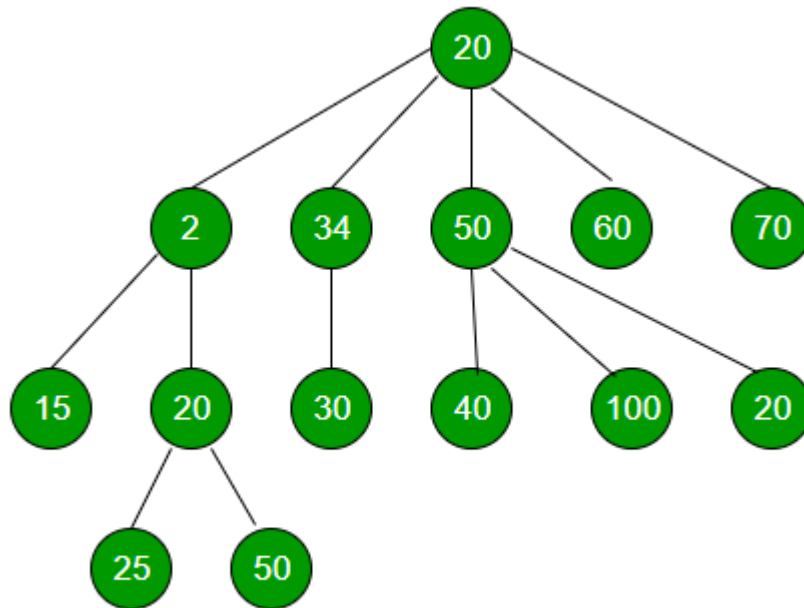
<https://www.geeksforgeeks.org/string-with-additive-sequence/>

Chapter 152

Sum of all elements of N-ary Tree

Sum of all elements of N-ary Tree - GeeksforGeeks

Given an N-ary tree, find sum of all elements in it.



Example :

Input : Above tree

Output : Sum is 536

Approach : The approach used is similar to [Level Order traversal in a binary tree](#). Start by pushing the root node in the queue. And for each node, while popping it from queue, add the value of this node in the **sum** variable and push the children of the popped element in the queue. In case of a generic tree store child nodes in a vector. Thus, put all elements of the vector in the queue.

Below is the implementation of the above idea :

```
// C++ program to find sum of all
// elements in generic tree
#include <bits/stdc++.h>
using namespace std;

// Represents a node of an n-ary tree
struct Node {
    int key;
    vector<Node*> child;
};

// Utility function to create a new tree node
Node* newNode(int key)
{
    Node* temp = new Node;
    temp->key = key;
    return temp;
}

// Function to compute the sum
// of all elements in generic tree
int sumNodes(Node* root)
{
    // initialize the sum variable
    int sum = 0;

    if (root == NULL)
        return 0;

    // Creating a queue and pushing the root
    queue<Node*> q;
    q.push(root);

    while (!q.empty()) {
        int n = q.size();

        // If this node has children
        while (n > 0) {
```

```
        // Dequeue an item from queue and
        // add it to variable "sum"
        Node* p = q.front();
        q.pop();
        sum += p->key;

        // Enqueue all children of the dequeued item
        for (int i = 0; i < p->child.size(); i++)
            q.push(p->child[i]);
        n--;
    }
}
return sum;
}

// Driver program
int main()
{
    // Creating a generic tree
    Node* root = newNode(20);
    (root->child).push_back(newNode(2));
    (root->child).push_back(newNode(34));
    (root->child).push_back(newNode(50));
    (root->child).push_back(newNode(60));
    (root->child).push_back(newNode(70));
    (root->child[0]->child).push_back(newNode(15));
    (root->child[0]->child).push_back(newNode(20));
    (root->child[1]->child).push_back(newNode(30));
    (root->child[2]->child).push_back(newNode(40));
    (root->child[2]->child).push_back(newNode(100));
    (root->child[2]->child).push_back(newNode(20));
    (root->child[0]->child[1]->child).push_back(newNode(25));
    (root->child[0]->child[1]->child).push_back(newNode(50));

    cout << sumNodes(root) << endl;

    return 0;
}
```

Output:

536

Time Complexity : $O(N)$, where N is the number of nodes in tree.

Auxiliary Space : $O(N)$, where N is the number of nodes in tree.

Source

<https://www.geeksforgeeks.org/sum-elements-n-ary-tree/>

Chapter 153

Sum of digit of a number using recursion

Sum of digit of a number using recursion - GeeksforGeeks

Given a number, we need to find sum of its digits using recursion.

Examples:

Input : 12345

Output : 15

Input : 45632

Output : 20

Step by step process for better understanding of how the algorithm works.

Let number be 12345.

Step 1-> $12345 \% 10$ which is equal-too 5 + (send $12345/10$ to next step)

Step 2-> $1234 \% 10$ which is equal-too 4 + (send $1234/10$ to next step)

Step 3-> $123 \% 10$ which is equal-too 3 + (send $123/10$ to next step)

Step 4-> $12 \% 10$ which is equal-too 2 + (send $12/10$ to next step)

Step 5-> $1 \% 10$ which is equal-too 1 + (send $1/10$ to next step)

Step 6-> 0 algorithm stops

following diagram will illustrate the process of recursion

return 15



15

C

```
// Recursive C program to find sum of digits
// of a number
#include <stdio.h>

// Function to check sum of digit using recursion
int sum_of_digit(int n)
{
    if (n == 0)
        return 0;
    return (n % 10 + sum_of_digit(n / 10));
}

// Driven Program to check above
int main()
{
    int num = 12345;
    int result = sum_of_digit(num);
    printf("Sum of digits in %d is %d\n", num, result);
    return 0;
}
```

Java

```
// Recursive java program to
// find sum of digits of a number
import java.io.*;

class sum_of_digits
{
    // Function to check sum
    // of digit using recursion
    static int sum_of_digit(int n)
    {
        if (n == 0)
            return 0;
        return (n % 10 + sum_of_digit(n / 10));
    }

    // Driven Program to check above
    public static void main(String args[])
    {
        int num = 12345;
        int result = sum_of_digit(num);
        System.out.println("Sum of digits in " +
                           num + " is " + result);
    }
}
```

```
    }  
}  
  
// This code is contributed by Anshika Goyal.
```

Python3

```
# Recursive Python3 program to  
# find sum of digits of a number  
  
# Function to check sum of  
# digit using recursion  
def sum_of_digit( n ):  
    if n == 0:  
        return 0  
    return (n % 10 + sum_of_digit(int(n / 10)))  
  
# Driven code to check above  
num = 12345  
result = sum_of_digit(num)  
print("Sum of digits in",num,"is", result)  
  
# This code is contributed by "Sharad_Bhardwaj".
```

C#

```
// Recursive C# program to  
// find sum of digits of a number  
using System;  
  
class GFG {  
  
    // Function to check sum  
    // of digit using recursion  
    static int sum_of_digit(int n)  
    {  
        if (n == 0)  
            return 0;  
  
        return (n % 10 + sum_of_digit(n / 10));  
    }  
  
    // Driven Program to check above  
    public static void Main()  
    {  
        int num = 12345;  
        int result = sum_of_digit(num);  
    }  
}
```



```
        Console.WriteLine("Sum of digits in " +
                           num + " is " + result);
    }
}

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

PHP

```
<?php
// Recursive PHP program
// to find sum of digits
// of a number

// Function to check sum of
// digit using recursion
function sum_of_digit($n)
{
    if ($n == 0)
        return 0;
    return ($n % 10 +
            sum_of_digit($n / 10));
}

// Driven Code
$num = 12345;
$result = sum_of_digit($num);
echo("Sum of digits in " . $num . " is " . $result);

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

Output:

Sum of digits in 12345 is 15

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/sum-digit-number-using-recursion/>

Chapter 154

Sum of elements of all partitions of number such that no element is less than K

Sum of elements of all partitions of number such that no element is less than K - Geeks-forGeeks

Given an integer N, the task is to find an aggregate sum of all integer partitions of this number such that each partition does not contain any integer less than K.

Examples:

Input: N = 6 and K = 2

Output: 24

In this case, there are 4 valid partitions.

- 1) {6}
- 2) {4, 2}
- 3) {3, 3}
- 4) {2, 2, 2}

Therefore, aggregate sum would be

$$6 + 4 + 2 + 3 + 3 + 2 + 2 + 2 = 24$$

Input: N = 10 and K = 3

Output: 50

Here, 5 valid partitions are:

- 1) {10}
- 2) {7, 3}
- 3) {6, 4}
- 4) {5, 5}
- 5) {3, 3, 4}

Aggregate sum in this case would be

$$10 + 7 + 3 + 6 + 4 + 5 + 5 + 3 + 3 + 4 = 50$$

Approach: This problem has a simple **recursive** solution. First, we need to count the total number of valid partitions of number N such that each partition contains integers greater than or equal to K. So we will iteratively apply our recursive solution to find valid partitions that have the minimum integer K, K+1, K+2, ..., N. Our final answer would be **N * no of valid partitions** because each valid partition has a sum equal to N.

Following are some key ideas for designing recursive function to find total number of valid partitions.

- If $N < K$ then no partition is possible.
- If $N < 2K$ then only one partition is possible and that is the number N itself.
- We can find number partitions in a recursive manner that contains integers at least equal to 'i' ('i' can be from K to N) and add them all to get final answer.

Pseudo code for recursive function to find number of valid partitions:

```
f(N,K):
    if N < K
        return 0
    if N < 2K
        return 1
    Initialize answer = 1
    FOR i from K to N
        answer = answer + f(N-i,i)
    return answer
```

Below is the **Dynamic Programming** solution:

C++

```
// C++ implementation of above approach
#include <bits/stdc++.h>
using namespace std;

// Function that returns total number of valid
// partitions of integer N
long long int countPartitions(int n, int k)
{

    // Global declaration of 2D dp array
    // which will be later used for memoization
    long long int dp[201][201];

    // initializing 2D dp array with -1
```

```
// we will use this 2D array for memoization
for (int i = 0; i < n + 1; i++) {
    for (int j = 0; j < n + 1; j++) {
        dp[i][j] = -1;
    }
}

// if this subproblem is already previously
// calculated, then directly return that answer
if (dp[n][k] >= 0)
    return dp[n][k];

// if N < K, then no valid
// partition is possible
if (n < k)
    return 0;

// if N is between K to 2*K then
// there is only one
// partition and that is the number N itself
if (n < 2 * k)
    return 1;

// Initialize answer with 1 as
// the number N itself
// is always a valid partition
long long int answer = 1;

// for loop to iterate over K to N
// and find number of
// possible valid partitions recursively.
for (int i = k; i < n; i++)
    answer = answer + countPartitions(n - i, i);

// memoization is done by storing
// this calculated answer
dp[n][k] = answer;

// returning number of valid partitions
return answer;
}

// Driver code
int main()
{
    int n = 10, k = 3;

    // Printing total number of valid partitions
```

```
    cout << "Total Aggregate sum of all Valid Partitions: "  
        << countPartitions(n, k) * n;  
  
    return 0;  
}
```

Java

```
// Java implementation of  
// above approach  
class GFG  
{  
    // Function that returns  
    // total number of valid  
    // partitions of integer N  
    static long countPartitions(int n, int k)  
    {  
  
        // Global declaration of 2D  
        // dp array which will be  
        // later used for memoization  
        long[][] dp = new long[201][201];  
  
        // initializing 2D dp array  
        // with -1 we will use this  
        // 2D array for memoization  
        for (int i = 0; i < n + 1; i++)  
        {  
            for (int j = 0; j < n + 1; j++)  
            {  
                dp[i][j] = -1;  
            }  
        }  
  
        // if this subproblem is already  
        // previously calculated, then  
        // directly return that answer  
        if (dp[n][k] >= 0)  
            return dp[n][k];  
  
        // if N < K, then no valid  
        // partition is possible  
        if (n < k)  
            return 0;  
  
        // if N is between K to 2*K  
        // then there is only one  
        // partition and that is
```

```
// the number N itself
if (n < 2 * k)
    return 1;

// Initialize answer with 1
// as the number N itself
// is always a valid partition
long answer = 1;

// for loop to iterate over
// K to N and find number of
// possible valid partitions
// recursively.
for (int i = k; i < n; i++)
    answer = answer +
        countPartitions(n - i, i);

// memoization is done by storing
// this calculated answer
dp[n][k] = answer;

// returning number of
// valid partitions
return answer;
}

// Driver code
public static void main(String[] args)
{
    int n = 10, k = 3;

    // Printing total number
    // of valid partitions
    System.out.println("Total Aggregate sum of " +
        "all Valid Partitions: " +
        countPartitions(n, k) * n);
}
}
```

// This code is contributed by mits

Output:

Total Aggregate sum of all Valid Partitions: 50

Time Complexity: $O(N^2)$

Improved By : [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/sum-of-elements-of-all-partitions-of-number-such-that-no-element-is-less-than-k/>

Chapter 155

Sum of natural numbers using recursion

Sum of natural numbers using recursion - GeeksforGeeks

Given a number n , find sum of first n natural numbers. To calculate the sum, we will use a recursive function `recur_sum()`.

Examples :

Input : 3
Output : 6
Explanation : $1 + 2 + 3 = 6$

Input : 5
Output : 15
Explanation : $1 + 2 + 3 + 4 + 5 = 15$

Below is code to find the sum of natural numbers up to n using recursion :

C++

```
// C++ program to find the
// sum of natural numbers up
// to n using recursion
#include <iostream>
using namespace std;

// Returns sum of first
// n natural numbers
int recurSum(int n)
```



```
{
    if (n <= 1)
        return n;
    return n + recurSum(n - 1);
}

// Driver code
int main()
{
    int n = 5;
    cout << recurSum(n);
    return 0;
}
```

Java

```
// Java program to find the
// sum of natural numbers up
// to n using recursion
import java.util.*;
import java.lang.*;

class GFG
{
    // Returns sum of first
    // n natural numbers
    public static int recurSum(int n)
    {
        if (n <= 1)
            return n;
        return n + recurSum(n - 1);
    }

    // Driver code
    public static void main(String args[])
    {
        int n = 5;
        System.out.println(recurSum(n));
    }
}

// This code is contributed by Sachin Bisht
```

Python

```
# Python code to find sum
```

```
# of natural numbers upto
# n using recursion

# Returns sum of first
# n natural numbers
def recurSum(n):
    if n <= 1:
        return n
    return n + recurSum(n - 1)

# Driver code
n = 5
print(recurSum(n))
```

C#

```
// C# program to find the
// sum of natural numbers
// up to n using recursion
using System;

class GFG
{
    // Returns sum of first
    // n natural numbers
    public static int recurSum(int n)
    {
        if (n <= 1)
            return n;
        return n + recurSum(n - 1);
    }

    // Driver code
    public static void Main()
    {
        int n = 5;
        Console.WriteLine(recurSum(n));
    }
}

// This code is contributed by vt_m
```

PHP

```
<?php
// PHP program to find the
```

```
// sum of natural numbers
// up to n using recursion

// Returns sum of first
// n natural numbers
function recurSum($n)
{
    if ($n <= 1)
        return $n;
    return $n + recurSum($n - 1);
}

// Driver code
$n = 5;
echo(recurSum($n));

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

Output :

15

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/sum-of-natural-numbers-using-recursion/>

Chapter 156

Sum triangle from array

Sum triangle from array - GeeksforGeeks

Given an array of integers, print a sum triangle from it such that the first level has all array elements. From then, at each level number of elements is one less than the previous level and elements at the level is be the Sum of consecutive two elements in the previous level.

Example :

Input : A = {1, 2, 3, 4, 5}

Output : [48]

[20, 28]

[8, 12, 16]

[3, 5, 7, 9]

[1, 2, 3, 4, 5]

Explanation :

Here, [48]

[20, 28] -->(20 + 28 = 48)

[8, 12, 16] -->(8 + 12 = 20, 12 + 16 = 28)

[3, 5, 7, 9] -->(3 + 5 = 8, 5 + 7 = 12, 7 + 9 = 16)

[1, 2, 3, 4, 5] -->(1 + 2 = 3, 2 + 3 = 5, 3 + 4 = 7, 4 + 5 = 9)

Approach :

1. Recursion is the key. At each iteration create a new array which contains the Sum of consecutive elements in the array passes as parameter.
2. Make a recursive call and pass the newly created array in the previous step.
3. While back tracking print the array (for printing in reverse order).

Source

<https://www.geeksforgeeks.org/sum-triangle-from-array/>

Chapter 157

Tail Recursion

Tail Recursion - GeeksforGeeks

What is tail recursion?

A recursive function is tail recursive when recursive call is the last thing executed by the function. For example the following C++ function print() is tail recursive.

```
// An example of tail recursive function
void print(int n)
{
    if (n < 0) return;
    cout << " " << n;

    // The last executed statement is recursive call
    print(n-1);
}
```

Why do we care?

The tail recursive functions considered better than non tail recursive functions as tail-recursion can be optimized by compiler. The idea used by compilers to optimize tail-recursive functions is simple, since the recursive call is the last statement, there is nothing left to do in the current function, so saving the current function's stack frame is of no use (See [this](#) for more details).

Can a non-tail recursive function be written as tail-recursive to optimize it?

Consider the following function to calculate factorial of n. It is a non-tail-recursive function. Although it looks like a tail recursive at first look. If we take a closer look, we can see that the value returned by fact(n-1) is used in fact(n), so the call to fact(n-1) is not the last thing done by fact(n)

C++

```
#include<iostream>
```

```
using namespace std;

// A NON-tail-recursive function. The function is not tail
// recursive because the value returned by fact(n-1) is used in
// fact(n) and call to fact(n-1) is not the last thing done by fact(n)
unsigned int fact(unsigned int n)
{
    if (n == 0) return 1;

    return n*fact(n-1);
}

// Driver program to test above function
int main()
{
    cout << fact(5);
    return 0;
}
```

Java

```
class GFG {

    // A NON-tail-recursive function.
    // The function is not tail
    // recursive because the value
    // returned by fact(n-1) is used
    // in fact(n) and call to fact(n-1)
    // is not the last thing done by
    // fact(n)
    static int fact(int n)
    {
        if (n == 0) return 1;

        return n*fact(n-1);
    }

    // Driver program
    public static void main(String[] args)
    {
        System.out.println(fact(5));
    }
}

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

Python 3

```
# A NON-tail-recursive function.
# The function is not tail
# recursive because the value
# returned by fact(n-1) is used
# in fact(n) and call to fact(n-1)
# is not the last thing done by
# fact(n)
def fact(n):

    if (n == 0):
        return 1

    return n * fact(n-1)

# Driver program to test
# above function
print(fact(5))
# This code is contributed by Smitha.
```

C#

```
using System;

class GFG {

    // A NON-tail-recursive function.
    // The function is not tail
    // recursive because the value
    // returned by fact(n-1) is used
    // in fact(n) and call to fact(n-1)
    // is not the last thing done by
    // fact(n)
    static int fact(int n)
    {
        if (n == 0)
            return 1;

        return n * fact(n-1);
    }

    // Driver program to test
    // above function
    public static void Main()
    {
        Console.Write(fact(5));
    }
}
```

// This code is contributed by Smitha

PHP

```
<?php
// A NON-tail-recursive function.
// The function is not tail
// recursive because the value
// returned by fact(n-1) is used in
// fact(n) and call to fact(n-1) is
// not the last thing done by fact(n)

function fact( $n)
{
    if ($n == 0) return 1;

    return $n * fact($n - 1);
}

// Driver Code
echo fact(5);

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

Output :

120

The above function can be written as a tail recursive function. The idea is to use one more argument and accumulate the factorial value in second argument. When n reaches 0, return the accumulated value.

C++

```
#include<iostream>
using namespace std;

// A tail recursive function to calculate factorial
unsigned factTR(unsigned int n, unsigned int a)
{
    if (n == 0) return a;

    return factTR(n-1, n*a);
}
```



```
// A wrapper over factTR
unsigned int fact(unsigned int n)
{
    return factTR(n, 1);
}

// Driver program to test above function
int main()
{
    cout << fact(5);
    return 0;
}
```

Java

```
// Java Code for Tail Recursion

class GFG {

    // A tail recursive function
    // to calculate factorial
    static int factTR(int n, int a)
    {
        if (n == 0)
            return a;

        return factTR(n - 1, n * a);
    }

    // A wrapper over factTR
    static int fact(int n)
    {
        return factTR(n, 1);
    }

    // Driver code
    static public void main (String[] args)
    {
        System.out.println(fact(5));
    }
}

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

Python 3

```
# A tail recursive function
```

```
# to calculate factorial
def factTR(n, a):

    if (n == 0):
        return a

    return factTR(n - 1, n * a)

# A wrapper over factTR
def fact(n):
    return factTR(n, 1)

# Driver program to test
# above function
print(fact(5))

# This code is contributed
# by Smitha
```

C#

```
// C# Code for Tail Recursion
using System;

class GFG {

    // A tail recursive function
    // to calculate factorial
    static int factTR(int n, int a)
    {
        if (n == 0)
            return a;

        return factTR(n - 1, n * a);
    }

    // A wrapper over factTR
    static int fact(int n)
    {
        return factTR(n, 1);
    }

    // Driver code
    static public void Main ()
    {
        Console.WriteLine(fact(5));
    }
}
```

// This code is contributed by Ajit.

PHP

```
<?php
// A tail recursive function
// to calculate factorial
function factTR($n, $a)
{
    if ($n == 0) return $a;

    return factTR($n - 1, $n * $a);
}

// A wrapper over factTR
function fact($n)
{
    return factTR($n, 1);
}

// Driver program to test
// above function
echo fact(5);

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

Output :

120

Next articles on this topic:

[Tail Call Elimination](#)

[QuickSort Tail Call Optimization \(Reducing worst case space to Log n \)](#)

References:

http://en.wikipedia.org/wiki/Tail_call

<http://c2.com/cgi/wiki?TailRecursion>

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

Source

<https://www.geeksforgeeks.org/tail-recursion/>

Chapter 158

Tail recursion to calculate sum of array elements.

Tail recursion to calculate sum of array elements. - GeeksforGeeks

Given an array A[] we need to find sum of its elements using [Tail Recursion Method](#). We generally want to achieve tail recursion (a recursive function where recursive call is the last thing that function does) so that compilers can optimize the code. Basically if recursive call is last statement, compiler do not need to save state of parent call.

Examples:

Input : A[] = {1, 8, 9}

Output : 18

Input : A[] = {2, 55, 1, 7}

Output : 65

For Linear Recursion Method refer : <https://www.geeksforgeeks.org/sum-array-elements-using-recursion/>

Logic : Here the key to tail recursion is whatever operation is applied with function call, maintain it as a separate function parameter.

So, keep the sum of last elements K elements as a function parameter and return sum when K=0.

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

// Tail recursive function
int arrSum(int* array, int size, int sum = 0)
{
```

```
// Base Case
if (size == 0)
    return sum;

// Function Call Observe sum+array[size-1]
// to maintain sum of elements
return arrSum(array, size - 1, sum + array[size - 1]);
}

int main()
{
    int array[] = { 2, 55, 1, 7 };
    int size = sizeof(array) / sizeof(array[0]);
    cout << arrSum(array, size);
    return 0;
}
```

Output:

65

Time Complexity : $O(n)$

Source

<https://www.geeksforgeeks.org/tail-recursion-to-calculate-sum-of-array-elements/>

Chapter 159

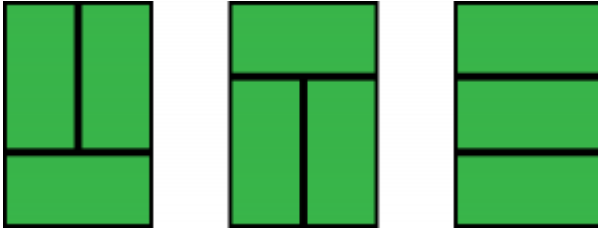
Tiling with Dominoes

Tiling with Dominoes - GeeksforGeeks

Given a $3 \times n$ board, find the number of ways to fill it with 2×1 dominoes.

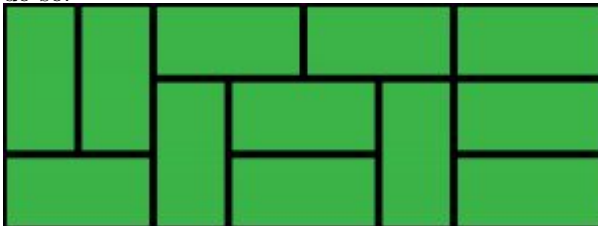
Example 1

Following are all the **3** possible ways to fill up a **3 x 2** board.



Example 2

Here is one possible way of filling a 3 x 8 board. You have to find all the possible ways to do so.



Examples :

Input : 2

Output : 3

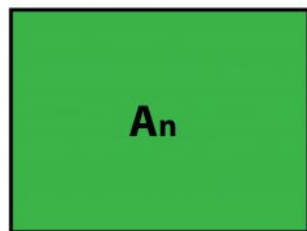
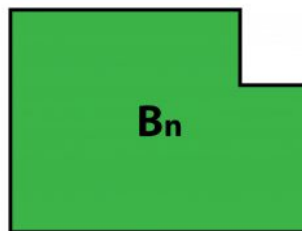
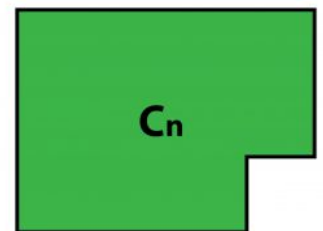
Input : 8

Output : 153

Input : 12
Output : 2131

Defining Subproblems:

At any point while filling the board, there are three possible states that the last column can be in:

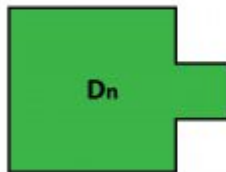
Possible states of the last column**Completely Filled****Top Corner Empty****Bottom Corner Empty**

A_n = No. of ways to completely fill a $3 \times n$ board. (We need to find this)

B_n = No. of ways to fill a $3 \times n$ board with top corner in last column not filled.

C_n = No. of ways to fill a $3 \times n$ board with bottom corner in last column not filled.

Note: The following states are impossible to reach:

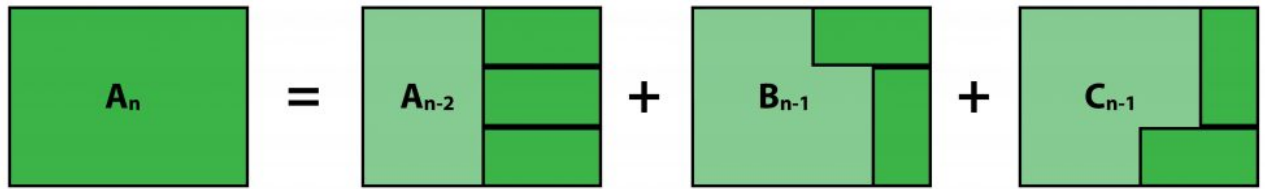
Following States are NOT possible**Finding Recurrences**

Note: Even though B_n and C_n are different states, they will be equal for same ' n '. i.e

$$B_n = C_n$$

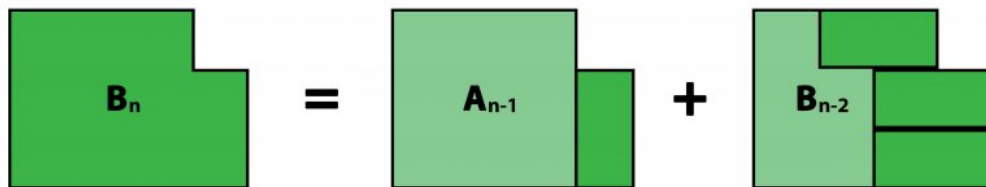
Hence, we only need to calculate one of them.

Calculating A_n :



$$A_n = A_{n-2} + 2 * (B_{n-1})$$

Calculating B_n:



$$B_n = A_{n-1} + B_{n-2}$$

Final Recursive Relations are:

Base Cases:

C++

```
// C++ program to find no. of ways
// to fill a 3xn board with 2x1 dominoes.
#include <iostream>
using namespace std;

int countWays(int n)
{
    int A[n + 1], B[n + 1];
```



```
A[0] = 1, A[1] = 0, B[0] = 0, B[1] = 1;
for (int i = 2; i <= n; i++) {
    A[i] = A[i - 2] + 2 * B[i - 1];
    B[i] = A[i - 1] + B[i - 2];
}

return A[n];
}

int main()
{
    int n = 8;
    cout << countWays(n);
    return 0;
}
```

Java

```
// Java program to find no. of ways
// to fill a 3xn board with 2x1 dominoes.
import java.io.*;

class GFG {

    static int countWays(int n)
    {
        int []A = new int[n+1];
        int []B = new int[n+1];
        A[0] = 1; A[1] = 0;
        B[0] = 0; B[1] = 1;
        for (int i = 2; i <= n; i++)
        {
            A[i] = A[i - 2] + 2 * B[i - 1];
            B[i] = A[i - 1] + B[i - 2];
        }

        return A[n];
    }

    // Driver code
    public static void main (String[] args)
    {
        int n = 8;
        System.out.println(countWays(n));
    }
}

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

Python 3

```
# Python 3 program to find no. of ways
# to fill a 3xn board with 2x1 dominoes.
```

```
def countWays(n):

    A = [0] * (n + 1)
    B = [0] * (n + 1)
    A[0] = 1
    A[1] = 0
    B[0] = 0
    B[1] = 1
    for i in range(2, n+1):
        A[i] = A[i - 2] + 2 * B[i - 1]
        B[i] = A[i - 1] + B[i - 2]

    return A[n]

n = 8
print(countWays(n))
```

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

C#

```
// C# program to find no. of ways
// to fill a 3xn board with 2x1 dominoes.
using System;
```

```
class GFG {

    static int countWays(int n)
    {
        int []A = new int[n+1];
        int []B = new int[n+1];
        A[0] = 1; A[1] = 0;
        B[0] = 0; B[1] = 1;
        for (int i = 2; i <= n; i++)
        {
            A[i] = A[i - 2] + 2 * B[i - 1];
            B[i] = A[i - 1] + B[i - 2];
        }

        return A[n];
    }
}
```

```
// Driver code
public static void Main ()
{
    int n = 8;
    Console.WriteLine(countWays(n));
}

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

PHP

```
<?php
// PHP program to find no. of ways
// to fill a 3xn board with 2x1 dominoes.

function countWays($n)
{
    $A = array();
    $B = array();
    $A[0] = 1; $A[1] = 0;
    $B[0] = 0; $B[1] = 1;
    for ( $i = 2; $i <= $n; $i++)
    {
        $A[$i] = $A[$i - 2] + 2 *
                $B[$i - 1];
        $B[$i] = $A[$i - 1] +
                $B[$i - 2];
    }

    return $A[$n];
}

// Driver Code
$n = 8;
echo countWays($n);

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

Output :

153

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

Source

<https://www.geeksforgeeks.org/tiling-with-dominoes/>

Chapter 160

Time Complexity Analysis | Tower Of Hanoi (Recursion)

Time Complexity Analysis | Tower Of Hanoi (Recursion) - GeeksforGeeks

[Tower of Hanoi](#) is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

- 1) Only one disk can be moved at a time.
- 2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- 3) No disk may be placed on top of a smaller disk.

Pseudo Code

```
TOH(n, x, y, z)
{
    if (n >= 1)
    {
        // put (n-1) disk to z by using y
        TOH((n-1), x, z, y)

        // move larger disk to right place
        move:x-->y

        // put (n-1) disk to right place
        TOH((n-1), z, y, x)
    }
}
```

Analysis of Recursion

Recursive Equation : $T(n) = 2T(n-1) + 1$ —equation-1

Solving it by BackSubstitution :

$$T(n-1) = 2T(n-2) + 1 \text{-----equation-2}$$

$$T(n-2) = 2T(n-3) + 1 \text{-----equation-3}$$

Put value of T(n-2) in equation-2 with help of equation-3

$$T(n-1) = 2(2T(n-3) + 1) + 1 \text{-----equation-4}$$

Put value of T(n-1) in equation-1 with help of equation-4

$$T(n) = 2(2(2T(n-3) + 1) + 1) + 1$$

$$T(n) = 2^3T(n-3) + 2^2 + 2^1 + 1$$

After Generalization :

$$T(n) = 2^kT(n-k) + 2^{k-1} + 2^{k-2} + \dots + 2^1 + 1$$

Base condition $T(0) == 1$

$n - k = 0$

$n = k$;

put, $k = n$

$$T(n) = 2^nT(0) + 2^{n-1} + 2^{n-2} + \dots + 2^1 + 1$$

It is GP series, and sum is $\frac{2^{n+1}}{2} - 1$

$$T(n) = 2^n(2^{n+1} - 1), \text{ or you can say } O(2^n) \text{ which is exponential}$$

Source

<https://www.geeksforgeeks.org/time-complexity-analysis-tower-hanoi-recursion/>

Chapter 161

Two Dimensional Segment Tree | Sub-Matrix Sum

Two Dimensional Segment Tree | Sub-Matrix Sum - GeeksforGeeks

Given a rectangular matrix $M[0...n-1][0...m-1]$, and queries are asked to find the sum / minimum / maximum on some sub-rectangles $M[a...b][e...f]$, as well as queries for modification of individual matrix elements (i.e $M[x][y] = p$).

We can also answer sub-matrix queries using [Two Dimensional Binary Indexed Tree](#).

In this article, We will focus on solving sub-matrix queries using two dimensional segment tree. Two dimensional segment tree is nothing but segment tree of segment trees.

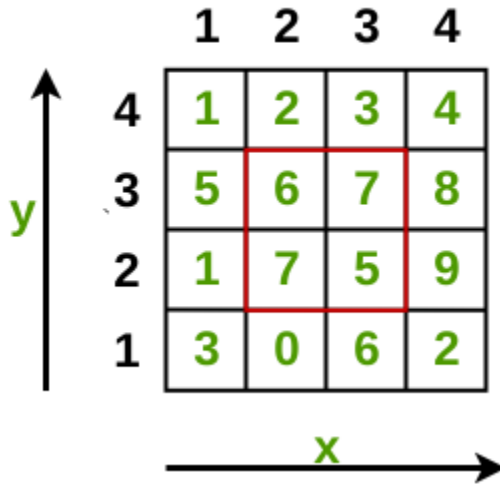
Prerequisite : [Segment Tree – Sum of given range](#)

Algorithm :

We will build a two-dimensional tree of segments by the following principle:

- 1 . In First step, We will construct an ordinary one-dimensional segment tree, working only with the first coordinate say 'x' and 'y' as constant. Here, we will not write number in inside the node as in the one-dimensional segment tree, but an entire tree of segments.
2. The second step is to combine the values of segmented trees. Assume that in second step instead of combining the elements we are combining the segment trees obtained from the step first.

Consider the below example. Suppose we have to find the sum of all numbers inside the highlighted red area



Step 1 : We will first create the segment tree of each strip of y- axis. We represent the segment tree here as an array where child node is $2n$ and $2n+1$ where $n > 0$.

Segment Tree for strip $y=1$



Segment Tree for Strip $y = 2$



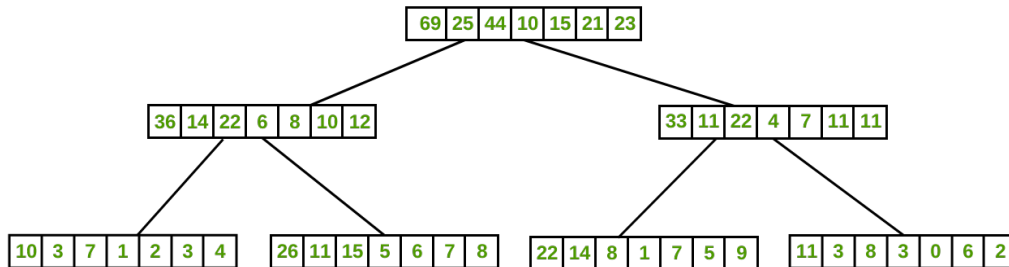
Segment Tree for Strip $y = 3$



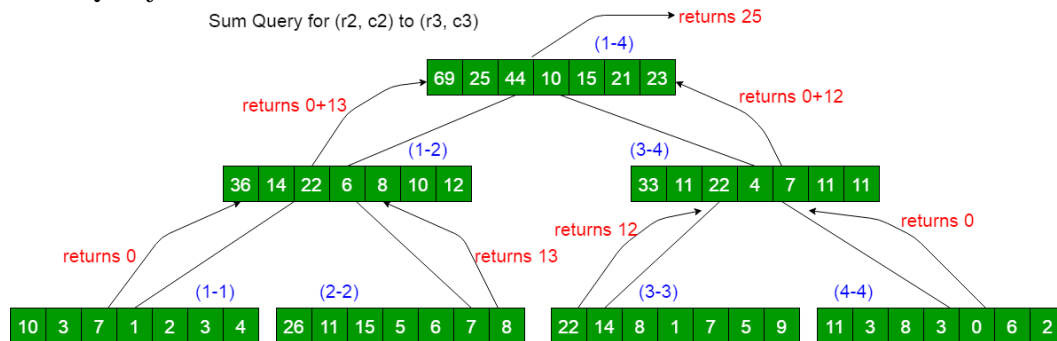
Segment Tree for Strip $y = 4$



Step 2: In this step, we create the segment tree for the rectangular matrix where the base node are the strips of y-axis given above. The task is to merge above segment trees.



Sum Query :



Thanks to **Sahil Bansal** for contributing this image.

Processing Query :

We will respond to the two-dimensional query by the following principle: first to break the query on the first coordinate, and then, when we reached some vertex of the tree of segments with the first coordinate and then we call the corresponding tree of segments on the second coordinate.

This function works in time $O(\log n * \log m)$, because it first descends the tree in the first coordinate, and for each traversed vertex of that tree, it makes a query from the usual tree of segments along the second coordinate.

Modification Query :

We want to learn how to modify the tree of segments in accordance with the change in the value of an element $M[x][y] = p$. It is clear that the changes will occur only in those vertices of the first tree of segments that cover the coordinate x , and for the trees of the segments corresponding to them, the changes will only occur in those vertices that cover the coordinate y . Therefore, the implementation of the modification request will not be very different from the one-dimensional case, only now we first descend the first coordinate, and then the second.

Output for the highlighted area will be 25.

Below is the implementation of above approach :

```
// C++ program for implementation
// of 2D segment tree.
#include <bits/stdc++.h>
using namespace std;

// Base node of segment tree.
int ini_seg[1000][1000] = { 0 };

// final 2d-segment tree.
int fin_seg[1000][1000] = { 0 };

// Rectangular matrix.
int rect[4][4] = {
    { 1, 2, 3, 4 },
    { 5, 6, 7, 8 },
    { 1, 7, 5, 9 },
    { 3, 0, 6, 2 },
};

// size of x coordinate.
int size = 4;

/*
 * A recursive function that constructs
 * Initial Segment Tree for array rect[][] = { }.
 * 'pos' is index of current node in segment
 * tree seg[]. 'strip' is the enumeration
 * for the y-axis.
 */

int segment(int low, int high,
            int pos, int strip)
{
    if (high == low) {
        ini_seg[strip][pos] = rect[strip][low];
    }
    else {
        int mid = (low + high) / 2;
        segment(low, mid, 2 * pos, strip);
        segment(mid + 1, high, 2 * pos + 1, strip);
        ini_seg[strip][pos] = ini_seg[strip][2 * pos] +
                               ini_seg[strip][2 * pos + 1];
    }
}

/*
 * A recursive function that constructs
 * Final Segment Tree for array ini_seg[][] = { }.
 */
```

```
*/
int finalSegment(int low, int high, int pos)
{
    if (high == low) {

        for (int i = 1; i < 2 * size; i++)
            fin_seg[pos][i] = ini_seg[low][i];
    }
    else {
        int mid = (low + high) / 2;
        finalSegment(low, mid, 2 * pos);
        finalSegment(mid + 1, high, 2 * pos + 1);

        for (int i = 1; i < 2 * size; i++)
            fin_seg[pos][i] = fin_seg[2 * pos][i] +
                               fin_seg[2 * pos + 1][i];
    }
}

/*
 * Return sum of elements in range from index
 * x1 to x2 . It uses the final_seg[][] array
 * created using finalsegment() function.
 * 'pos' is index of current node in
 * segment tree fin_seg[][].
 */
int finalQuery(int pos, int start, int end,
               int x1, int x2, int node)
{
    if (x2 < start || end < x1) {
        return 0;
    }

    if (x1 <= start && end <= x2) {
        return fin_seg[node][pos];
    }

    int mid = (start + end) / 2;
    int p1 = finalQuery(2 * pos, start, mid,
                        x1, x2, node);

    int p2 = finalQuery(2 * pos + 1, mid + 1,
                        end, x1, x2, node);

    return (p1 + p2);
}

/*
```

```
* This fuction calls the finalQuery fuction
* for elements in range from index x1 to x2 .
* This fuction queries the yth coordinate.
*/
int query(int pos, int start, int end,
          int y1, int y2, int x1, int x2)
{
    if (y2 < start || end < y1) {
        return 0;
    }

    if (y1 <= start && end <= y2) {
        return (finalQuery(1, 1, 4, x1, x2, pos));
    }

    int mid = (start + end) / 2;
    int p1 = query(2 * pos, start,
                  mid, y1, y2, x1, x2);
    int p2 = query(2 * pos + 1, mid + 1,
                  end, y1, y2, x1, x2);

    return (p1 + p2);
}

/* A recursive function to update the nodes
   which for the given index. The following
   are parameters : pos --> index of current
   node in segment tree fin_seg[] []. x ->
   index of the element to be updated. val -->
   Value to be change at node idx
*/
int finalUpdate(int pos, int low, int high,
               int x, int val, int node)
{
    if (low == high) {
        fin_seg[node][pos] = val;
    }
    else {
        int mid = (low + high) / 2;

        if (low <= x && x <= mid) {
            finalUpdate(2 * pos, low, mid, x, val, node);
        }
        else {
            finalUpdate(2 * pos + 1, mid + 1, high,
                      x, val, node);
        }
    }
}
```

```
        fin_seg[node][pos] = fin_seg[node][2 * pos] +
                           fin_seg[node][2 * pos + 1];
    }
}

/*
This funtion call the final update function after
visiting the yth coordinate in the segment tree fin_seg[][].
*/
int update(int pos, int low, int high, int x, int y, int val)
{
    if (low == high) {
        finalUpdate(1, 1, 4, x, val, pos);
    }
    else {
        int mid = (low + high) / 2;

        if (low <= y && y <= mid) {
            update(2 * pos, low, mid, x, y, val);
        }
        else {
            update(2 * pos + 1, mid + 1, high, x, y, val);
        }

        for (int i = 1; i < size; i++)
            fin_seg[pos][i] = fin_seg[2 * pos][i] +
                             fin_seg[2 * pos + 1][i];
    }
}

// Driver program to test above functions
int main()
{
    int pos = 1;
    int low = 0;
    int high = 3;

    // Call the ini_segment() to create the
    // inital segment tree on x- coordinate
    for (int strip = 0; strip < 4; strip++)
        segment(low, high, 1, strip);

    // Call the final function to built the 2d segment tree.
    finalSegment(low, high, 1);

    /*
Query:
* To request the query for sub-rectangle y1, y2=(2, 3) x1, x2=(2, 3)
*/
}
```

```
* update the value of index (3, 3)=100;
* To request the query for sub-rectangle y1, y2=(2, 3) x1, x2=(2, 3)
*/
    cout << "The sum of the submatrix (y1, y2)->(2, 3), "
          << " (x1, x2)->(2, 3) is "
          << query(1, 1, 4, 2, 3, 2, 3) << endl;

    // Function to update the value
    update(1, 1, 4, 2, 3, 100);

    cout << "The sum of the submatrix (y1, y2)->(2, 3), "
          << "(x1, x2)->(2, 3) is "
          << query(1, 1, 4, 2, 3, 2, 3) << endl;

    return 0;
}
```

Output:

```
The sum of the submatrix (y1, y2)->(2, 3), (x1, x2)->(2, 3) is 25
The sum of the submatrix (y1, y2)->(2, 3), (x1, x2)->(2, 3) is 118
```

Time complexity :

Processing Query : $O(\log n * \log m)$

Modification Query: $O(2 * n * \log n * \log m)$

Space Complexity : $O(4 * m * n)$

Source

<https://www.geeksforgeeks.org/two-dimensional-segment-tree-sub-matrix-sum/>

Chapter 162

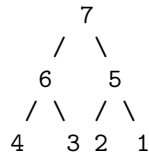
Vertical width of Binary tree | Set 2

Vertical width of Binary tree | Set 2 - GeeksforGeeks

Given a binary tree, find the vertical width of the binary tree. Width of a binary tree is the number of vertical paths.

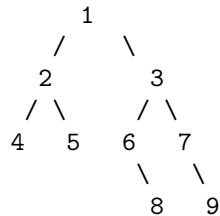
Examples:

Input :



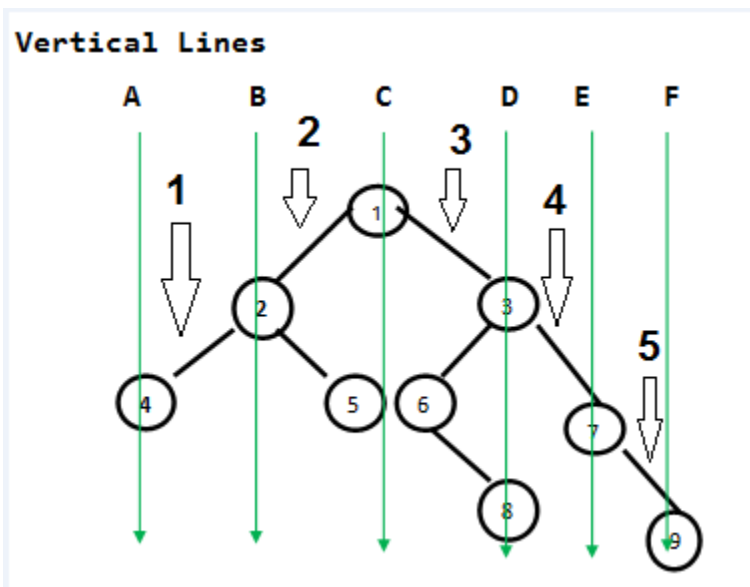
Output : 5

Input :



Output : 6

Prerequisite : [Print Binary Tree in Vertical order](#)



In this image, the tree contains 6 vertical lines which is the required width of tree.

Approach : In this Approach, we use the approach for printing vertical View of binary tree. Store the horizontal distances in a set and return $1 + \text{highest horizontal distance} - \text{lowest horizontal distance}$. 1 is added to consider horizontal distance 0 as well. While going left, do $hd - 1$ and for right do $hd + 1$. We insert all possible distances in a hash table and finally return size of the hash table.

```
// CPP code to find vertical
// width of a binary tree
#include <bits/stdc++.h>
using namespace std;

// Tree class
class Node
{
public :
    int data;
    Node *left, *right;

    // Constructor
    Node(int data_new)
    {
        data = data_new;
        left = right = NULL;
    }
};

// Function to fill hd in set.
void fillSet(Node* root, unordered_set<int>& s,
```



```
int hd)
{
    if (!root)
        return;

    fillSet(root->left, s, hd - 1);
    s.insert(hd);
    fillSet(root->right, s, hd + 1);
}

int verticalWidth(Node* root)
{
    unordered_set<int> s;

    // Third parameter is horizontal
    // distance
    fillSet(root, s, 0);

    return s.size();
}

int main()
{
    Node* root = NULL;

    // Creating the above tree
    root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->left = new Node(6);
    root->right->right = new Node(7);
    root->right->left->right = new Node(8);
    root->right->right->right = new Node(9);

    cout << verticalWidth(root) << "\n";

    return 0;
}
```

Output:

Source

<https://www.geeksforgeeks.org/vertical-width-binary-tree-set-2/>

Chapter 163

Water Jug Problem using Memoization

Water Jug Problem using Memoization - GeeksforGeeks

Given two jugs with the maximum capacity of **m** and **n** liters respectively. The jugs don't have markings on them which can help us to measure smaller quantities. The task is to measure **d** liters of water using these two jugs. Hence our goal is to reach from initial state (m, n) to final state (0, d) or (d, 0).

Examples:

Input: 4 3 2

Output: (0, 0) -> (4, 0) -> (4, 3) -> (0, 3) -> (3, 0) -> (3, 3) -> (4, 2) -> (0, 2)

Input: 5 2 4

Output: (0, 0) -> (5, 0) -> (5, 2) -> (0, 2) -> (2, 0) -> (2, 2) -> (4, 0)

Approach: An approach using [BFS](#) has been discussed in the previous [post](#). In this post an approach using [memoization](#) and [recursion](#) has been discussed. At any point, there can be a total of six possibilities:

- Empty the first jug completely
- Empty the second jug completely
- Fill the first jug
- Fill the second jug
- Fill the water from the second jug into the first jug until the first jug is full or the second jug has no water left
- Fill the water from the first jug into the second jug until the second jug is full or the first jug has no water left

Approach: Using [Recursion](#), visit all the six possible moves one by one until one of them returns True. Since there can be repetitions of same recursive calls, hence every return value

is stored using [memoization](#) to avoid calling the recursive function again and returning the stored value.

Below is the implementation of the above approach:

```
# This function is used to initialize the
# dictionary elements with a default value.
from collections import defaultdict

# jug1 and jug2 contain the value
# for max capacity in respective jugs
# and aim is the amount of water to be measured.
jug1, jug2, aim = 4, 3, 2

# Initialize dictionary with
# default value as false.
visited = defaultdict(lambda: False)

# Recursive function which prints the
# intermediate steps to reach the final
# solution and return boolean value
# (True if solution is possible, otherwise False).
# amt1 and amt2 are the amount of water present
# in both jugs at a certain point of time.
def waterJugSolver(amt1, amt2):

    # Checks for our goal and
    # returns true if achieved.
    if (amt1 == aim and amt2 == 0) or (amt2 == aim and amt1 == 0):
        print(amt1, amt2)
        return True

    # Checks if we have already visited the
    # combination or not. If not, then it proceeds further.
    if visited[(amt1, amt2)] == False:
        print(amt1, amt2)

        # Changes the boolean value of
        # the combination as it is visited.
        visited[(amt1, amt2)] = True

        # Check for all the 6 possibilities and
        # see if a solution is found in any one of them.
        return (waterJugSolver(0, amt2) or
                waterJugSolver(amt1, 0) or
                waterJugSolver(jug1, amt2) or
                waterJugSolver(amt1, jug2) or
                waterJugSolver(amt1 + min(amt2, (jug1-amt1)),
                                amt2 - min(amt2, (jug1-amt1))) or
```

```
        waterJugSolver(amt1 - min(amt1, (jug2-amt2)),
                        amt2 + min(amt1, (jug2-amt2)))

    # Return False if the combination is
    # already visited to avoid repetition otherwise
    # recursion will enter an infinite loop.
    else:
        return False

print("Steps: ")

# Call the function and pass the
# initial amount of water present in both jugs.
waterJugSolver(0, 0)
```

Output:

```
Steps:
0 0
4 0
4 3
0 3
3 0
3 3
4 2
0 2
```

Time complexity: $O(M * N)$

Auxiliary Space: $O(M * N)$

Source

<https://www.geeksforgeeks.org/water-jug-problem-using-memoization/>

Chapter 164

Word Break Problem using Backtracking

Word Break Problem using Backtracking - GeeksforGeeks

Given a valid sentence without any spaces between the words and a dictionary of valid English words, find all possible ways to break the sentence in individual dictionary words.

Example

Consider the following dictionary

```
{ i, like, sam, sung, samsung, mobile, ice,
  cream, icecream, man, go, mango }
```

Input: "ilikesamsungmobile"

Output: i like sam sung mobile
 i like samsung mobile

Input: "ilikeicecreamandmango"

Output: i like ice cream and man go
 i like ice cream and mango
 i like icecream and man go
 i like icecream and mango

We have discussed a Dynamic Programming solution in below post.

[Dynamic Programming | Set 32 \(Word Break Problem\)](#)

The Dynamic Programming solution only finds whether it is possible to break a word or not. Here we need to print all possible word breaks.

We start scanning the sentence from left. As we find a valid word, we need to check whether rest of the sentence can make valid words or not. Because in some situations the first found word from left side can leave a remaining portion which is not further separable. So in that case we should come back and leave the current found word and keep on searching for the

next word. And this process is recursive because to find out whether the right portion is separable or not, we need the same logic. So we will use recursion and backtracking to solve this problem. To keep track of the found words we will use a stack. Whenever the right portion of the string does not make valid words, we pop the top string from stack and continue finding.

```
// A recursive program to print all possible
// partitions of a given string into dictionary
// words
#include <iostream>
using namespace std;

/* A utility function to check whether a word
   is present in dictionary or not. An array of
   strings is used for dictionary. Using array
   of strings for dictionary is definitely not
   a good idea. We have used for simplicity of
   the program*/
int dictionaryContains(string &word)
{
    string dictionary[] = {"mobile","samsung","sam","sung",
                           "man","mango", "icecream","and",
                           "go","i","love","ice","cream"};
    int n = sizeof(dictionary)/sizeof(dictionary[0]);
    for (int i = 0; i < n; i++)
        if (dictionary[i].compare(word) == 0)
            return true;
    return false;
}

//prototype of wordBreakUtil
void wordBreakUtil(string str, int size, string result);

// Prints all possible word breaks of given string
void wordBreak(string str)
{
    // last argument is prefix
    wordBreakUtil(str, str.size(), "");
}

// result store the current prefix with spaces
// between words
void wordBreakUtil(string str, int n, string result)
{
    //Process all prefixes one by one
    for (int i=1; i<=n; i++)
    {
        //extract substring from 0 to i in prefix
```

```
string prefix = str.substr(0, i);

// if dictionary contains this prefix, then
// we check for remaining string. Otherwise
// we ignore this prefix (there is no else for
// this if) and try next
if (dictionaryContains(prefix))
{
    // if no more elements are there, print it
    if (i == n)
    {
        // add this element to previous prefix
        result += prefix;
        cout << result << endl; //print result
        return;
    }
    wordBreakUtil(str.substr(i, n-i), n-i,
                  result + prefix + " ");
}
} //end for
} //end function

int main()
{
    cout << "First Test:\n";
    wordBreak("iloveicecreamandmango");

    cout << "\nSecond Test:\n";
    wordBreak("ilovesamsungmobile");
    return 0;
}
```

Output:

```
First Test:
i love ice cream and man go
i love ice cream and mango
i love icecream and man go
i love icecream and mango
```

```
Second Test:
i love sam sung mobile
i love samsung mobile
```

Source

<https://www.geeksforgeeks.org/word-break-problem-using-backtracking/>

Chapter 165

Write a program to print all permutations of a given string

Write a program to print all permutations of a given string - GeeksforGeeks

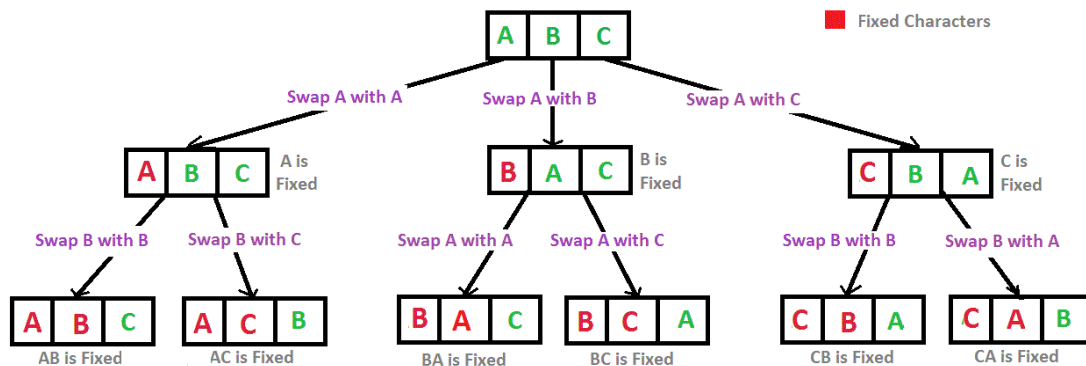
A permutation, also called an “arrangement number” or “order,” is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself. A string of length n has $n!$ permutation.

Source: Mathworld(<http://mathworld.wolfram.com/Permutation.html>)

Below are the permutations of string ABC.

ABC ACB BAC BCA CBA CAB

Here is a solution that is used as a basis in backtracking.



Recursion Tree for Permutations of String "ABC"

C/C++

```
// C program to print all permutations with duplicates allowed
#include <stdio.h>
```

```
#include <string.h>

/* Function to swap values at two pointers */
void swap(char *x, char *y)
{
    char temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

/* Function to print permutations of string
   This function takes three parameters:
   1. String
   2. Starting index of the string
   3. Ending index of the string. */
void permute(char *a, int l, int r)
{
    int i;
    if (l == r)
        printf("%s\n", a);
    else
    {
        for (i = l; i <= r; i++)
        {
            swap((a+l), (a+i));
            permute(a, l+1, r);
            swap((a+l), (a+i)); //backtrack
        }
    }
}

/* Driver program to test above functions */
int main()
{
    char str[] = "ABC";
    int n = strlen(str);
    permute(str, 0, n-1);
    return 0;
}
```

Java

```
// Java program to print all permutations of a
// given string.
public class Permutation
{
    public static void main(String[] args)
```

```
{
    String str = "ABC";
    int n = str.length();
    Permutation permutation = new Permutation();
    permutation.permute(str, 0, n-1);
}

/**
 * permutation function
 * @param str string to calculate permutation for
 * @param l starting index
 * @param r end index
 */
private void permute(String str, int l, int r)
{
    if (l == r)
        System.out.println(str);
    else
    {
        for (int i = l; i <= r; i++)
        {
            str = swap(str,l,i);
            permute(str, l+1, r);
            str = swap(str,l,i);
        }
    }
}

/**
 * Swap Characters at position
 * @param a string value
 * @param i position 1
 * @param j position 2
 * @return swapped string
 */
public String swap(String a, int i, int j)
{
    char temp;
    char[] charArray = a.toCharArray();
    temp = charArray[i] ;
    charArray[i] = charArray[j];
    charArray[j] = temp;
    return String.valueOf(charArray);
}

}

// This code is contributed by Mihir Joshi
```

Python

```
# Python program to print all permutations with
# duplicates allowed

def toString(List):
    return ''.join(List)

# Function to print permutations of string
# This function takes three parameters:
# 1. String
# 2. Starting index of the string
# 3. Ending index of the string.
def permute(a, l, r):
    if l==r:
        print toString(a)
    else:
        for i in xrange(l,r+1):
            a[l], a[i] = a[i], a[l]
            permute(a, l+1, r)
            a[l], a[i] = a[i], a[l] # backtrack

# Driver program to test the above function
string = "ABC"
n = len(string)
a = list(string)
permute(a, 0, n-1)

# This code is contributed by Bhavya Jain
```

C#

```
// C# program to print all
// permutations of a given string.
using System;

class GFG
{
    /**
     * permutation function
     * @param str string to
     *       calculate permutation for
     * @param l starting index
     * @param r end index
     */
    private static void permute(String str,
                                int l, int r)
```

```
{
    if (l == r)
        Console.WriteLine(str);
    else
    {
        for (int i = l; i <= r; i++)
        {
            str = swap(str, l, i);
            permute(str, l + 1, r);
            str = swap(str, l, i);
        }
    }
}

/**
 * Swap Characters at position
 * @param a string value
 * @param i position 1
 * @param j position 2
 * @return swapped string
 */
public static String swap(String a,
                           int i, int j)
{
    char temp;
    char[] charArray = a.ToCharArray();
    temp = charArray[i] ;
    charArray[i] = charArray[j];
    charArray[j] = temp;
    string s = new string(charArray);
    return s;
}

// Driver Code
public static void Main()
{
    String str = "ABC";
    int n = str.Length;
    permute(str, 0, n-1);
}
}
```

// This code is contributed by mits

PHP

```
<?php
// PHP program to print all
```

```
// permutations of a given string.

/**
 * permutation function
 * @param str string to
 * calculate permutation for
 * @param l starting index
 * @param r end index
 */
function permute($str, $l, $r)
{
    if ($l == $r)
        echo $str. "\n";
    else
    {
        for ($i = $l; $i <= $r; $i++)
        {
            $str = swap($str, $l, $i);
            permute($str, $l + 1, $r);
            $str = swap($str, $l, $i);
        }
    }
}

/**
 * Swap Characters at position
 * @param a string value
 * @param i position 1
 * @param j position 2
 * @return swapped string
 */
function swap($a, $i, $j)
{
    $temp;
    $charArray = str_split($a);
    $temp = $charArray[$i] ;
    $charArray[$i] = $charArray[$j];
    $charArray[$j] = $temp;
    return implode($charArray);
}

// Driver Code
$str = "ABC";
$n = strlen($str);
permute($str, 0, $n - 1);

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

?>

Output:

ABC
ACB
BAC
BCA
CBA
CAB

Algorithm Paradigm: Backtracking

Time Complexity: $O(n \cdot n!)$ Note that there are $n!$ permutations and it requires $O(n)$ time to print a permutation.

Note : The above solution prints duplicate permutations if there are repeating characters in input string. Please see below link for a solution that prints only distinct permutations even if there are duplicates in input.

[Print all distinct permutations of a given string with duplicates.](#)

[Permutations of a given string using STL](#)

Improved By : [Mithun Kumar](#)

Source

<https://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/>

Chapter 166

Write a program to reverse digits of a number

Write a program to reverse digits of a number - GeeksforGeeks

Write a program to reverse digits of an integer.

Examples :

Input : num = 12345

Output : 54321

Input : num = 876

Output : 678

ITERATIVE WAY

Algorithm:

Input: num

(1) Initialize rev_num = 0

(2) Loop while num > 0

(a) Multiply rev_num by 10 and add remainder of num
divide by 10 to rev_num

rev_num = rev_num*10 + num%10;

(b) Divide num by 10

(3) Return rev_num

Example:

num = 4562

rev_num = 0


```
rev_num = rev_num *10 + num%10 = 2
num = num/10 = 456

rev_num = rev_num *10 + num%10 = 20 + 6 = 26
num = num/10 = 45

rev_num = rev_num *10 + num%10 = 260 + 5 = 265
num = num/10 = 4

rev_num = rev_num *10 + num%10 = 265 + 4 = 2654
num = num/10 = 0
```

Program:

C

```
#include <stdio.h>

/* Iterative function to reverse digits of num*/
int reversDigits(int num)
{
    int rev_num = 0;
    while(num > 0)
    {
        rev_num = rev_num*10 + num%10;
        num = num/10;
    }
    return rev_num;
}

/*Driver program to test reversDigits*/
int main()
{
    int num = 4562;
    printf("Reverse of no. is %d", reversDigits(num));

    getchar();
    return 0;
}
```

Java

```
// Java program to reverse a number

class GFG
{
    /* Iterative function to reverse
    digits of num*/
    static int reversDigits(int num)
```

```
{
    int rev_num = 0;
    while(num > 0)
    {
        rev_num = rev_num * 10 + num % 10;
        num = num / 10;
    }
    return rev_num;
}

// Driver code
public static void main (String[] args)
{
    int num = 4562;
    System.out.println("Reverse of no. is "
                       + reversDigits(num));
}
}
```

// This code is contributed by Anant Agarwal.

Python

```
# Python program to reverse a number

n = 4562;
rev = 0

while(n > 0):
    a = n % 10
    rev = rev * 10 + a
    n = n / 10

print(rev)

# This code is contributed by Shariq Raza
```

C#

```
// C# program to reverse a number
using System;

class GFG
{
    // Iterative function to
    // reverse digits of num
    static int reversDigits(int num)
```

```
{
    int rev_num = 0;
    while(num > 0)
    {
        rev_num = rev_num * 10 + num % 10;
        num = num / 10;
    }
    return rev_num;
}

// Driver code
public static void Main()
{
    int num = 4562;
    Console.WriteLine("Reverse of no. is "
        + reversDigits(num));
}
}

// This code is contributed by Sam007
```

PHP

```
<?php
// Iterative function to
// reverse digits of num
function reversDigits($num)
{
    $rev_num = 0;
    while($num > 1)
    {
        $rev_num = $rev_num * 10 +
            $num % 10;
        $num = (int)$num / 10;
    }
    return $rev_num;
}

// Driver Code
$num = 4562;
echo "Reverse of no. is ",
    reversDigits($num);

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

Time Complexity: $O(\log(n))$ where n is the input number.
Output:

2654

RECURSIVE WAY

Thanks to Raj for adding this to the original post.

C

```
#include <stdio.h>;

/* Recursive function to reverse digits of num*/
int reversDigits(int num)
{
    static int rev_num = 0;
    static int base_pos = 1;
    if(num > 0)
    {
        reversDigits(num/10);
        rev_num += (num%10)*base_pos;
        base_pos *= 10;
    }
    return rev_num;
}

/*Driver program to test reversDigits*/
int main()
{
    int num = 4562;
    printf("Reverse of no. is %d", reversDigits(num));

    getchar();
    return 0;
}
```

Time Complexity: $O(\log(n))$ where n is the input number.

[Reverse digits of an integer with overflow handled](#)

Note that above program doesn't consider leading zeroes. For example, for 100 program will print 1. If you want to print 001 then see this comment from Maheshwar.

Try extensions of above functions that should also work for floating point numbers.

Improved By : [jit_t](#)

Source

<https://www.geeksforgeeks.org/write-a-program-to-reverse-digits-of-a-number/>

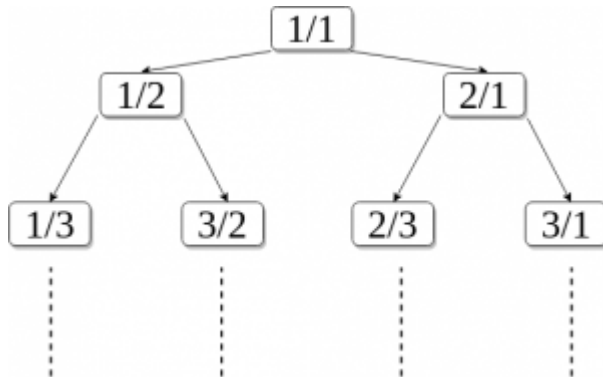
Chapter 167

nth Rational number in Calkin-Wilf sequence

nth Rational number in Calkin-Wilf sequence - GeeksforGeeks

What is Calkin Wilf Sequence?

A Calkin-Wilf tree (or sequence) is a special binary tree which is obtained by starting with the fraction $1/1$ and adding $a/(a+b)$ and $(a+b)/b$ iteratively below each fraction a/b . This tree generates every rational number. Writing out the terms in a sequence gives $1/1, 1/2, 2/1, 1/3, 3/2, 2/3, 3/1, 1/4, 4/3, 3/5, 5/2, 2/5, 5/3, 3/4, 4/1, \dots$ The sequence has the property that each denominator is the next numerator.



The image above is the Calkin-Wilf Tree where all the rational numbers are listed. The children of a node a/b is calculated as $a/(a+b)$ and $(a+b)/b$.

The task is to find the nth rational number in breadth first traversal of this tree.

Examples:

Input : 13

Output : [5, 3]

Input : 5
Output : [3, 2]

Explanation: This tree is a Perfect Binary Search tree and we need $\text{floor}(\log(n))$ steps to compute n th rational number. The concept is similar to searching in a binary search tree. Given n we keep dividing it by 2 until we get 0. We return fraction at each stage in the following manner:-

```
if n%2 == 0
    update frac[1]+=frac[0]
else
    update frac[0]+=frac[1]
```

Below is the program to find the n th number in Calkin Wilf sequence:

C++

```
// C++ program to find the
// nth number in Calkin
// Wilf sequence:
# include<bits/stdc++.h>
using namespace std;

int frac[] = {0, 1};

// returns 1x2 int array
// which contains the nth
// rational number
int nthRational(int n)
{
    if (n > 0)
        nthRational(n / 2);

    // ~n&1 is equivalent to
    // !n%2?1:0 and n&1 is
    // equivalent to n%2
    frac[~n & 1] += frac[n & 1];
}

// Driver Code
int main()
{
    int n = 13; // testing for n=13

    // converting array
    // to string format
```

```
    nthRational(n);
    cout << "[" << frac[0] << ","
        << frac[1] << "]" << endl;
    return 0;
}
```

```
// This code is contributed
// by Harshit Saini
```

Java

```
// Java program to find the nth number
// in Calkin Wilf sequence:
import java.util.*;

public class GFG {
    static int[] frac = { 0, 1 };

    public static void main(String args[])
    {
        int n = 13; // testing for n=13

        // converting array to string format
        System.out.println(Arrays.toString(nthRational(n)));
    }

    // returns 1x2 int array which
    // contains the nth rational number
    static int[] nthRational(int n)
    {
        if (n > 0)
            nthRational(n / 2);

        // ~n&1 is equivalent to !n%2?1:0
        // and n&1 is equivalent to n%2
        frac[~n & 1] += frac[n & 1];

        return frac;
    }
}
```

Python3

```
# Python program to find
# the nth number in Calkin
# Wilf sequence:
```



```
frac = [0, 1]

# returns 1x2 int array
# which contains the nth
# rational number
def nthRational(n):
    if n > 0:
        nthRational(int(n / 2))

    # ~n&1 is equivalent to
    # !n%2?1:0 and n&1 is
    # equivalent to n%2
    frac[~n & 1] += frac[n & 1]

    return frac

# Driver code
if __name__ == "__main__":

    n = 13 # testing for n=13

    # converting array
    # to string format
    print(nthRational(n))

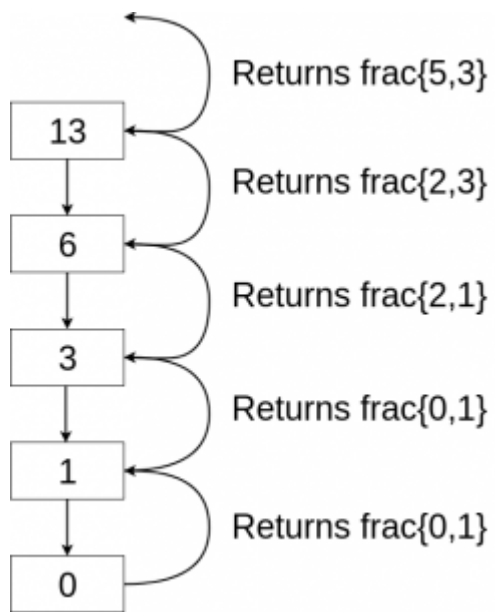
# This code is contributed
# by Harshit Saini
```

Output:

[5, 3]

Explanation:

For $n = 13$,



Improved By : [Harshit Saini](#)

Source

<https://www.geeksforgeeks.org/nth-rational-number-in-calkin-wilf-sequence/>