LU_SNR_db

Leading University, Sylhet

Contents

</>

Created By: Abu Sufian Rafi

Function:

- char ch = **toupper**('z'); **or**, ch=**('z' ^ 32);** => ch='**Z**';
- char ch = **tolower**('B'); **or**, ch=**('B' | 32);** => ch='**b**';
- **abs(x) = abs(-x)** = x;
- **sqrt(x)**= √x; [**sqrtl**(x) return long double;]
- **pow**(x, y)= x^y**;** [**powl**(x, y) return long double;] **=> round(**pow(x, y)**);**
- **log10(3)** = 0.4771212; **log2(3)** = 1.584962501;
- **round(x.y)** => if **y>=5** then **ans = x+1**;
- **ceil(x.y)** => if **y>=01** then **ans = x+1;** or**, ans=(a+b-1)/b;** [**ceil** symbol=> ⌈x.y⌉= x+1;]
- **floor(x.y)** => if **x.y positive** then **ans=x**; if **x.y negative** then **ans=x-1**;
  ex: floor(-2.3) = -3, floor(3.8) = 3;
- **trunc(x.y)** => if **x.y positive / negative** then **ans = x**; if ex: floor(-2.3)= -2, floor(3.8)= 3;
- **stoi(s)** => convert string to integer; [**stoll(s)->** for long long int value.]
- **to_string(num)** => convert integer to string;
- **getline(cin,** name**)** => input a line. [ignore buffer_use => **fflush(stdin);** or, **cin.ignore();**]
- **s2.substr(s1_pos, s2_len)** => This function generates a new string with its value initialized to a copy of a sub-string of this object. **[** if string s1="abcdef"; then, string s2=s1.substr(1,3); -> s2 = "bcd"**;** string s2 = s1.substr(1); -> s2 = "bcdef"**;]**
- **next_permutation():** It is used to rearrange the elements in the range [first, last) into the next lexicographically greater permutation. {{1,2,3}, {1,3,2}, {2,1,3}, {2,3,1}, {3,1,2}, {3,2,1}};

```
int arr[] = {1, 2, 3};              => O(n*n!)
do{
  //Add any conditions;
  cout << arr[0] << " " << arr[1] << " " << arr[2] << "\n";
} while (next_permutation(arr, arr + 3));
```

STL Function:

- **size()** – Returns the number of elements in the vector.[**v.size(); mp.size(); st.size();**]
- **empty()** – Returns whether the container is empty.If **empty** return **true**(1), if **not empty** return **false**(0). [**v.empty(); mp.empty (); st.empty ();**]
- **front()** – Returns a reference to the first element in the vector. [v.front();]
- **back()** – Returns a reference to the last element in the vector. [v.back();]
- **push_back()** – It push the elements into a vector from the back.
- **pop_back()** – It is used to pop or remove elements from a vector from the back.
- **insert()** – It inserts new elements before the element at the specified position.
  [**v.insert(v.begin(),** value**); mp.insert({key, value}); st.insert(key);**]

- **erase() –** It is used to remove elements from a container from the specified position or range. [**v.erase(v.begin() + position); mp.erase(value); st.erase(value); str.erase(v.begin() + position);**]
- **clear() –** It is used to remove all the elements. **[name.clear();]**
- **mp.count(**K**)** - The function returns the number of times the key K is present in the map/set container. [**st.count(K);**]   => O(logn);
- **max_size()**- Returns the maximum number of elements a set container can hold. [**v.max_size(); mp.max_size(); st.max_size();**]
- **find()** - An iterator to the first element in the range that compares equal to val. If no elements match, the function returns last(**v.end()**). [**find (v.begin(), v.end(),** val**);]** =>O(n);   [**mp.find(**val**); st.find(**val**);**]   => O(log(n))
- **lower_bound() –** Let v={1,5,7}; if we search **1** return **0's** index address, if we search **2** return **1's** index address, if we search **0** return **0's** index address, if we search **7<value** then return **v.end()** address. **(must be sorted)**   => O(logn)
  [**auto x=lower_bound(v.begin(), v.end(), val);] [auto x = mp.lower_bound(val); auto x = st.lower_bound(val);**] →(x->first ; x->second;)
- **upper_bound()** - Let v={1,5,7}; if we search **1** return **1's** index address, if we search **2** return **1's** index address, if we search **0** return **0's** index address, if we search **7<value** then return **v.end()** address. **(must be sorted)**   =>O(logn)
- **rand()**: The rand() function is used in C++ to generate random numbers in the range [0, RAND_MAX). Ex: a=**rand()**; a=(**rand()%10)+1** [=> **a>=1 && a<=10**]; if lb=20 and ub=40 Then, a=**(rand() % (ub - lb + 1)) + lb** [=> **a>=20 && a<=40**];
- **Ordered Set**: The complexity of the **insert** and **erase** functions is O(log n).

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
```

// ***s.find_by_order(k)**: K-th element in a set (counting from zero).
// **s.order_of_key(k)**: Number of items strictly smaller than k. (same as, lower_bound of k)
ordered_set<int> s; // we can change the data type.

Stack & Queue & Deque & Priority_Queue :

- **push():** Adds the element '**x**'. [**s.push(**x**); q.push(**x**); pd.push(x);**]
  - In Deque, you can push both side [**dq.push_back(**x**); dq.push_front(**x**);**]
- **pop():** Deletes the element. [**name.pop();**]
  - In Deque, you can pop both side [**dq.pop_back(**x**); dq.pop_front(**x**);**]
- **top():** Returns a reference to the top most element of the stack. [**s.top(); pq.top();**]
- **front():** Returns a reference to the first element of the queue. [**q.front(); dq.front();**]

- **back():** Returns a reference to the last element of the queue. [**q.back(); dq.back();**]
- **empty():** Returns whether the stack/queue is empty. It return **true** if the stack/queue is empty otherwise returns false. [**name.empty();**]
- **dq.at(**x**):** Returns a reference to the element at position **x** in the deque container object.
  <u>Ex</u>: dq = {10,3,15,20} ; ans= dq.at(2); =>ans=15
- **priority_queue<**int**>max_heapPQ;** => In this queue elements are in non-increasing. [same as **multiset <int, greater<int> > s;**]
- **priority_queue<**int**,** vector<int>, **greater<int>>min_heapPQ;** => In this queue elements are in non-decreasing order. [similar to **multiset**]

<u>Others:</u>

- **binary_search();** Return true(1) or false(0). If found return **1**. Else return **0**;
  [binary_search(start_address**,** end_address**, value_to_find);]**
- **max_element():** Return the maximum value's address of the vector.
  [**int max= *max_element(v.begin(),v.end());**] => O(n)
- **min_element():** Return minimum value's address of the vector.
  [**int min= *min_element(v.begin(),v.end());**] => O(n)
- **accumulate():** Return summation of the vector.
  [**int sum= accumulate(v.begin(),v.end(), 0);**] => O(n)
- **count():** Return count of the 'val' element.
  [**int c = count(v.begin(), v.end(),** val**);**] => O(n)
- **all of()/any_of()/none of():** Are **all of/at least one/none of** the elements **greater than 0?** (you can change this condition) => **return true(1) or, false(0).**
  - **all_of(v.begin(), v.end(), [ ](int x){return x>0;});**
  - **any_of(v.begin(), v.end(), [ ](int x){return x>0;});**
  - **none_of(v.begin(), v.end(), [ ](int x){return x>0;});**
- **iota()**: The algorithm iota() creates a range of sequentially increasing values.
  [iota(a, a+5, 10); => a[5]={10,11,12,13,14};]
- **is_sorted():** Return bool value. If, vector are sorted return 1. Else return 0.
  [**bool x= is_sorted(v.begin(), v.end());**] => O(n)
- **gcd():** Return a and b gcd(**Greatest Common Divisor**) value.
  [int gcd**= __gcd(a,b);**] => O(logn)
- **lcm():** Return a and b lcm(**Least Common Multiple**) value.
  [int lcm**= (a*b)/__gcd(a,b);**] => O(logn)
- **memset():** Initialize a **1D** vector with **-1**(or, **0**): =>O(n)
  memset(vec_name, -1, sizeof(vec_name)); **->** use any time.
- Initialize a **1D** vector with 1(or, any number): =>O(n)
  vector<int> vec**(n , 1);** [n=row, m=col] **->** use only initialize time.

- Initialize a **2D** vector with 1(or, any number): =>O(n)
  vector<vector<int>> vec**(n , vector<int> (m, 1));** [n=row, m=col]

Math:

- p+(p+1)+...+(q - 1)+q = **(q + p)(q – p + 1) / 2;** [Ex: **7**+8+9+10+**11**=(11+7)(11-7+1) / 2 =45]
- 1+2+3+...+(n - 1)+n = **(n * (n + 1)) / 2**; [Ex: 1+2+3+4+5=(5*(5+1))/2 = 15]
- 1+3+5+...+ (2n - 3) + (2n - 1) = $\mathbf{N^2}$; [N-> number of size] [Ex: 1+3+5 = 3^2 = 9]
- 2+4+6+...+ (2n - 2) + 2n **= N * (N + 1)**; [N-> number of size] [Ex: 2+4+6 = 3*(3+1) =12]
- $1^2+2^2+3^2+ \ldots + (n-1)^2 + n^2$ = **n (n + 1) (2n + 1) / 6;** [Ex: 1+4+9 = 3(3+1)(2*3 +1)/6 =14]
- $1^3 + 2^3 + 3^3 + \ldots + (n-1)^3 + n^3$ = $\mathbf{\{n\,(n + 1) / 2\}^2}$; [Ex: 1+8+27= $\{3(3+1) / 2\}^2$ =36]
- $1^2 + 3^2 + 5^2 + \ldots + (2n-3)^2 + (2n-1)^2$ = $\mathbf{N*(4N^2 - 1) / 3}$; [Ex: 1+9+25 = $3*(4*3^2 - 1)/3$ = 35]
- $1^3 + 3^3 + 5^3 + \ldots + (2n-3)^3 + (2n-1)^3$ = $\mathbf{N^2\,(2N^2 - 1)}$; [Ex: 1+27+125 = $3^2\,(2*3^2 - 1)$ = 153]
- $1^4 + 2^4 + 3^4 + \ldots + (n-1)^4 + n^4$ = $\mathbf{n(n+1)\,(2n+1)\,(3n^2 + 3n - 1) / 30}$;
  [Ex: 1+16+81+256 = $4(4+1)\,(2*4+1)\,(3*4^2 + 3*4 - 1) / 30$ = 354]
- $c^a + c^{a+1} + \cdots + c^b$ = $\mathbf{(c^{b+1} - c^a) / (c - 1)}$; [c != 1]
- $2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^{(k-1)}$ = $\mathbf{2^k - 1}$; [Ex: 1+2+4+8+16+32 = $2^6 - 1$ = 63]
- If F(n) = -1 + 2 - 3 + . . .+ $(-1)^n * n$
  - If N **even** number, **ans = N/2;**
  - If N **odd** number, **ans = ((N + 1) / 2) * (-1));**
- N-th **Odd** number = **(2 * N) – 1**;
- N-th **Even** number = **2*N**;
- a + a*k + $a*k^2$ + ... + b = **((b * k) - a) / (k - 1).** [ex: 3 + 6 + 12 + 24 = ((24 * 2) -3) / (2-1) = 45]
- a + (a+4) + (a+2*4) + ... + b= **(n * (a + b)) / 2**. [n-> number of size]
  [ex: 3 + 7 + 11 + 15 = (4 * (3 + 15)) / 2 = 36.]
- even **±** even = **even;** even **±** odd = **odd;** odd **±** odd = **even**;
- even **×** even = **even;** even **×** odd = **even;** odd **×** odd = **odd**;
- Number of digits in N = **floor(log10(**N**)) + 1;**
- Number of trailing **zeros** in **N!** => **while(N) sum+=N/5, N/=5; [**Ex: 10! = 36288**00;]**
- For a grid of size **(N x N)** the total number of **squares** formed: **((n*(n+1)) * (2n+1)) / 6;**
- **5 minutes** Clock Angular Value is **30°**. **[1 min = 6°]**
- Angle between clock minute and hour, **ans = abs ((0.5 * 11 * m) – (30 * h));**
  - For smaller angle, **if (ans >180) ans = 360 – ans;**
- The number of ways of selecting one or more things from N different things is given by $2^N - 1$. (combination)
- Number of possible of N bits = 2^N. [4bits, 2^4 =16 => 0 to 15 number possible with using 4 bits] $(2^n - 1)$ → highest value.
- N = $\mathbf{2^x}$ **=>** x = **log2(N)**. Ex: 64 = 2^6 [log2(64) = **6**].

- **$\log_u(x) = \frac{logk(x)}{logk(u)}$** [k-> any base (2,10)]; **$\log_a(k) = \frac{1}{logk(a)}$**; **$a^x = b$;=> $x = \log_a b$;**
- **(A * B) = ((A % Mod) * (B % Mod)) % Mod;** **<=** [Same As **+,-** Operator]
- **(A / B) = ((A % Mod) * (BinExp(A , Mod-2) % Mod)) % Mod;**

Bits:

- **Bitwise AND(&): (1 & 1)= 1;**
- **Bitwise OR(|): (0 | 1)= 1; (1 | 0)= 1; (1 | 1)= 1;**
- **Bitwise ExOR(^): (0 ^ 1)= 1; (1 ^ 0)=1;**
- **Bitwise NOT(~):** inverts all bits of it. [a = 1001_2 **-> (~a) = 0110**]
- **Right Shift(>>):** right shifting an integer "x" with an integer "y" denoted as '(x>>y)' is equivalent to **dividing** x with **2^y**. Ex: let's take N=32; which is 100000 in Binary Form. Now, if N=(N>>2) then N will become N=N / (2^2). Thus, N=32 / (2^2) = 8 which can be written as 1000. [18 = $(10010)_2$ → **(18>>1) = 01001; (18>>2) = 00100;**]
- **Left Shift(<<):** left shifting an integer "x" with an integer "y" denoted as '(x<<y)' is equivalent to **multiplying** x with **2^y**. Ex: let's take N=22; which is 00010110 in Binary Form. Now, if N=(N<<2) then N will become N=N * (2^2). Thus, N=22 * (2^2) = 88 which can be written as 01011000. [3 = $(11)_2$; => **(3<<1) = 110; (3<<2) = 1100**;]
- **(N&1) == 1 ->** N **odd** number; **(N&1) == 0 ->** N **even** number;
- (**N / 2**) == **(N >> 1);** (**N * 2**) == **(N << 1);**
- (**2^N**) == **(1**LL **<< N);** **=> N = (1LL <<** (long long)**log2**(N) **);**
- A quick way to **swap a** and **b** => [**a ^= b, b ^= a, a ^= b**;]
- **CheckBit(x, k) => (x & (1LL << k));**
- **SetBit(x, k) => (x |= (1LL << k));**
- **ClearBit(x, k) => (x &= ~(1LL << k));**
- **FlipBit(x, k) => (x ^= ~(1<<k));**
- **MSB(mask) => 63 - __builtin_clzll(mask);** [Most Significant Bit position]
- **LSB(mask) => __builtin_ctzll(mask);** [Least Significant Bit position]
- **__builtin_popcount(**x**):** This function is used to count the number of one's(set bits) in an integer(32 bits). Similarly you can use **__builtin_popcountll(**x**)** for **long long** data types (64 bits). Ex: x = **5** (101) => ans=2 ;
- **__builtin_clz(**x**):** It counts number of zeros before the first occurrence of one(set bit) of the integer(**32** bit).(**clz = count leading zero's.**)
  Ex: x= **16** (00000000 00000000 00000000 00010000) => ans = 27
- **__builtin_parity(**x**):** This function returns true(1) if the number has **odd** parity else it returns false(0) for **even** parity.(**parity= count the number of one's**)
  Ex: x = **7** (111) => ans = 1; x = **6** (110) => ans = 0;
- **__builtin_ctz(**x**):** Count number of zeros from **last to first** occurrence of one(set bit) of the given integer.(**ctz = count trailing zeros**;) Ex: x = **16** (00010000) => ans = 4 ;

Bitset Function:

bitset< highest_Bit_number **> name(data);**

- **bitset<64>** b1(val); or**, bitset<**4**>**b2("1011"); => auto-convert to binary;
- **to_ulong():** Converts the contents of the **bitset** to an **unsigned long integer**; [Ex: b1 = 1001, int val = b1.**to_ulong**(); => val = 9;]
- **to_string():** Converts the contents of the **bitset** to a **string**; [Ex: b1 = 1001, s1 = b1.to_string(); => s1= ”1001”;]
- **flip(**position**)**: flip function flips all bits (**1 to 0** and **0 to 1**). [Ex: b1 = 1001; b1.**flip**(1); =>b1 = 1011**;** b1.flip(1); =>b1=1001;]
- **count():** returns the total number of **set bits**(1); [Ex: b1=1001; bit= b1.count(); => bit =2;]
- **any():** function to check if **any of its bits are set or not**; [Ex: b1=1001, any_set = b1.any(); => any_set = 1; b2=0000; any_set = b1.any(); => any_set = 0;]
- **set():** b1.set(**pos**) makes bset[pos] = 1;(i.e. default is 1). b1.set(**pos**, **value**) makes bset[pos] = 0 or, 1; [Ex: b1=1001; b1.set**( )** =>b1 = 1111; b1.set(1)=> b1= 1011; b1.set(0)=> b1=1001; b1.set(3, 0);=> b1=0001; b1.set(2,1)=> b1= 1101;]
- **reset():** reset function makes all bits 0; [Ex: b1 = 1001; b1.reset()=> b1 = 0000; b1.reset(3)=> b1 = 0001;]

Combination And Permutation:

❖ ${}^{n}C_{r} = \frac{nPr}{r!}$ Or, nPr = nCr * r!

Combination(C):

- If, **Order Doesn’t Matter** and **Repetition Allowed** then,

  Possibilities, ${}^{n}C_{r} = \frac{n!}{r!(n-r)!}$

- If, **Order Doesn’t Matter** and **Repetition Not Allowed** then,

  Possibilities, ${}^{n}C_{r} = \frac{(n+r-1)!}{r!(n-1)!}$

- **Properties**: nCr = nC(n-r), nC_0 = nCn = 1, nC1= nC(n-1) = n, nCr + nC(r-1) = (n+1)Cr, nCx = nCy => x = y or, x + y = n.
- **nCr** has **maximum value** if:
  - **r = n/2 ;** when n is **Even.**
  - **r = (n+1)/2;** when n is **Odd.**

- **Short Technique:** $\textbf{nCr} = \frac{n*(n-1)*(n-2)*\ldots r'th\ times}{r*(r-1)*(r-2)*\ldots r'th\ times}$

  Ex: $^{20}C_3 = \frac{20*19*18}{3*2*1} = 1140$, $^{10}C_8 = {}^{10}C_{\mathbf{10-8}} = {}^{10}C_{\mathbf{2}} = \frac{10*9}{2*1} = 45$, $^{20}C_{\mathbf{15}} = {}^{20}C_{\mathbf{20-15}} = {}^{20}C_{\mathbf{5}} = \frac{20*19*18*17*16}{5*4*3*2*1} = 15504$.

Permutation(P):

- If, **Order Matter** and **Repetition Allowed** then, Possibilities = n^r
- If, **Order Matter** and **Repetition Not Allowed** then, Possibilities = $\frac{n!}{(n-r)!}$
- **Properties**: $nP_0 = 1$, $nP1 = n$, $nP(n-1) = n!$, $nPr/nP(r-1) = n - r + 1$.
- **Short Technique:** **nPr = n*(n-1)*(n-2)*... r'th times.**

  Ex: $^{10}P_3 = 10 * (10-1) * (10-2) = 720$, $^{15}P_4 = 15 * 14 * 13 * 12 = 32760$.

Find Combination(nCr): => O(r*log(n))

Ex: 5C2 = 10, 13C5= 1287;

```
void nCr( ll n, ll r)
{
   ll p= 1, k=1, m;
   if (n – r < r)  r = n - r;
   if (r != 0)
   {
      while(r)
      {
         p*=n, k*=r;
         m=__gcd(p, k);
         p/=m, k/=m;
         n--, r--;
      }
   }
   else p=1;
   cout<<p<<endl;
}
```

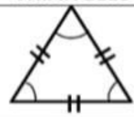
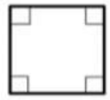
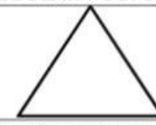
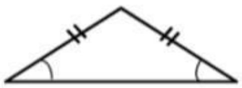
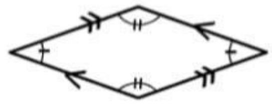
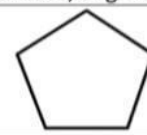
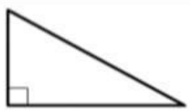
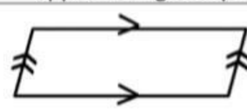
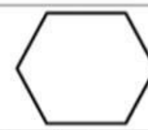
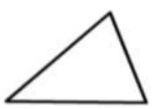
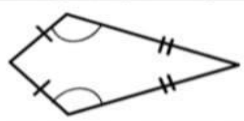
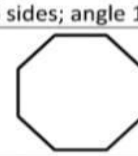
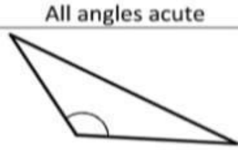
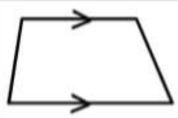
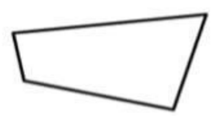
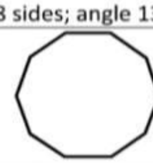
Find Permutation (nPr): => O(n)

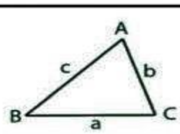
Ex: 5P2= 20, 6P3= 120;

```
ll fact(ll n)
{
    if(n <= 1) return 1;
    return n * fact(n - 1);
}
ll nPr(ll n, ll r)
{
    return fact(n) / fact(n - r);
}
int main()
{
     ll n, r;
     cin>>n>>r;
     cout<<nPr(n, r);
}
```

Geometry:

GEOMETRY QUICK GUIDE 2: 2D SHAPES (UK)

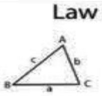
TRIANGLES	QUADRILATERALS	REGULAR POLYGONS
		
Equilateral triangle All sides equal; interior angles 60°	**Square** All sides equal; all angles 90°	**Equilateral triangle** 3 sides; angle 60°
		
Isosceles triangle 2 sides equal; 2 congruent angles	**Rectangle** Opposite sides equal, all angles 90°	**Square** 4 sides; angle 90°
		
Scalene triangle No sides or angles equal	**Rhombus** All sides equal; 2 pairs of parallel lines; opposite angles equal	**Regular Pentagon** 5 sides; angle 108°
		
Right triangle 1 right angle	**Parallelogram** Opposite sides equal, 2 pairs of parallel lines	**Regular Hexagon** 6 sides; angle 120°
		
Acute triangle All angles acute	**Kite** Adjacent sides equal; 2 congruent angles	**Regular Octagon** 8 sides; angle 135°
	 	
Obtuse triangle 1 obtuse angle	**Trapezium** 1 pair of parallel sides **Trapezoid** No pairs of parallel sides	**Regular Decagon** 10 sides; angle 144°



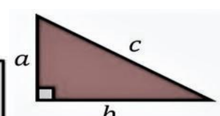
Law of sines

$$\frac{\sin(A)}{a} = \frac{\sin(B)}{b} = \frac{\sin(C)}{c}$$

Law of Cosines

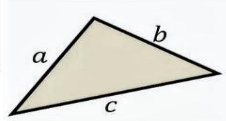


$$c^2 = a^2 + b^2 - 2ab\cos(C)$$
$$a^2 = b^2 + c^2 - 2bc\cos(A)$$
$$b^2 = a^2 + c^2 - 2ac\cos(B)$$



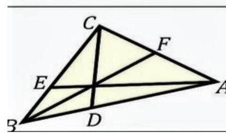
Pythagoras' Theorem

$a^2 + b^2 = c^2$



Heron's Formula

$$Area = \sqrt{s(s-a)(s-b)(s-c)}$$
$$Semiperimeter, s = \frac{a+b+c}{2}$$



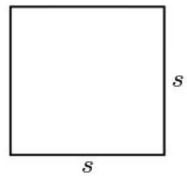
Ceva's Theorem

Given AE, BF & CD concurrent,

$$\frac{AD}{BD} \times \frac{BE}{CE} \times \frac{CF}{AF} = 1$$

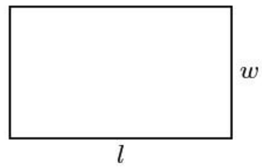
SQUARE

s = side
Area: $A = s^2$
Perimeter: $P = 4s$



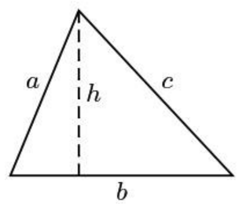
RECTANGLE

l = length, w = width
Area: $A = lw$
Perimeter: $P = 2l + 2w$



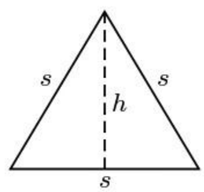
TRIANGLE

b = base, h = height
Area: $A = \frac{1}{2}bh$
Perimeter: $P = a + b + c$



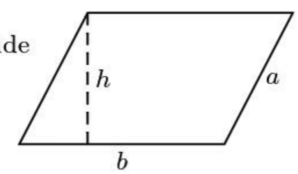
EQUILATERAL TRIANGLE

s = side
Height: $h = \frac{\sqrt{3}}{2}s$
Area: $A = \frac{\sqrt{3}}{4}s^2$



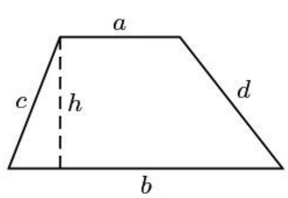
PARALLELOGRAM

b = base, h = height, a = side
Area: $A = bh$
Perimeter: $P = 2a + 2b$



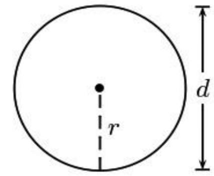
TRAPEZOID

a, b = bases; h = height;
c, d = sides
Area: $A = \frac{1}{2}(a + b)h$
Perimeter:
$P = a + b + c + d$



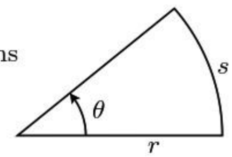
CIRCLE

r = radius, d = diameter
Diameter: $d = 2r$
Area: $A = \pi r^2$
Circumference: $C = 2\pi r = \pi d$



SECTOR OF CIRCLE

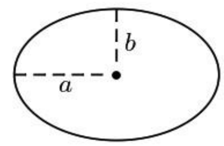
r = radius, θ = angle in radians
Area: $A = \frac{1}{2}\theta r^2$
Arc Length: $s = \theta r$



ELLIPSE

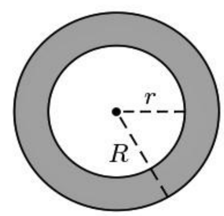
a = semimajor axis
b = semiminor axis
Area: $A = \pi ab$

Circumference:
$C \approx \pi\left(3(a + b) - \sqrt{(a + 3b)(b + 3a)}\right)$



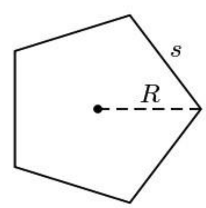
ANNULUS

r = inner radius,
R = outer radius
Average Radius: $\rho = \frac{1}{2}(r + R)$
Width: $w = R - r$
Area: $A = \pi(R^2 - r^2)$
or $A = 2\pi\rho w$



REGULAR POLYGON

s = side length,
n = number of sides
Circumradius: $R = \frac{1}{2}s\csc(\frac{\pi}{n})$
Area: $A = \frac{1}{4}ns^2\cot(\frac{\pi}{n})$
or $A = \frac{1}{2}nR^2\sin(\frac{2\pi}{n})$

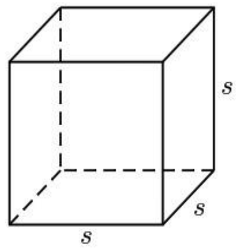


<u>Rhombus</u>: **Area =** (d1 * d2) / 2 **=** s^2 * sin(C)**;** **Perimeter =** 4*s ;
<u>Kite</u>: **Area** = (d1 * d2) / 2; **Perimeter =** 2(s1 + s2);
[d1 and d2= lengths of the diagonals, s = s1= s2 = length of side, C = interior angle;**]**

3D GEOMETRY FORMULAS

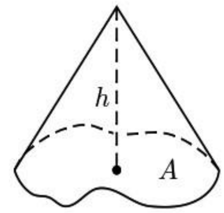
CUBE

s = side
Volume: $V = s^3$
Surface Area: $S = 6s^2$



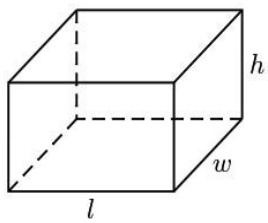
GENERAL CONE OR PYRAMID

A = area of base, h = height
Volume: $V = \frac{1}{3}Ah$



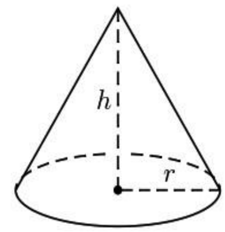
RECTANGULAR SOLID

l = length, w = width,
h = height
Volume: $V = lwh$
Surface Area:
$S = 2lw + 2lh + 2wh$



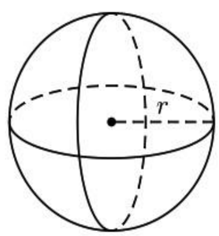
RIGHT CIRCULAR CONE

r = radius, h = height
Volume: $V = \frac{1}{3}\pi r^2 h$
Surface Area:
$S = \pi r\sqrt{r^2 + h^2} + \pi r^2$



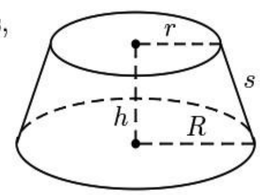
SPHERE

r = radius
Volume: $V = \frac{4}{3}\pi r^3$
Surface Area: $S = 4\pi r^2$



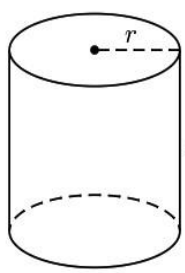
FRUSTUM OF A CONE

r = top radius, R = base radius,
h = height, s = slant height
Volume: $V = \frac{\pi}{3}(r^2 + rR + R^2)h$
Surface Area:
$S = \pi s(R + r) + \pi r^2 + \pi R^2$



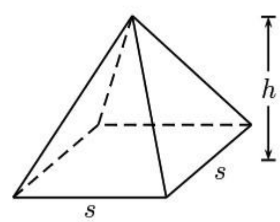
RIGHT CIRCULAR CYLINDER

r = radius, h = height
Volume: $V = \pi r^2 h$
Surface Area: $S = 2\pi rh + 2\pi r^2$



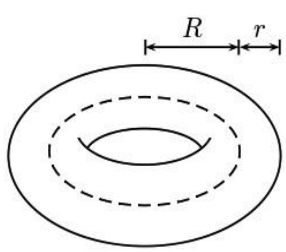
SQUARE PYRAMID

s = side, h = height
Volume: $V = \frac{1}{3}s^2 h$
Surface Area:
$S = s(s + \sqrt{s^2 + 4h^2})$



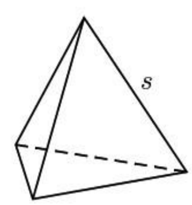
TORUS

r = tube radius,
R = torus radius
Volume: $V = 2\pi^2 r^2 R$
Surface Area: $S = 4\pi^2 rR$



REGULAR TETRAHEDRON

s = side
Volume: $V = \frac{1}{12}\sqrt{2}s^3$
Surface Area: $S = \sqrt{3}s^2$



Algorithm

Binary Search: => O(log(n))

```
void BinarySearch(vector<ll> &v, int n, int target)
{
   int low = 0, high = v.size() - 1, c = 0, mid = 0;
   while (high - low > 1) // or, low<=right
   {
      c++;
      mid = (low + high) >> 1; // or, mid= low+(high - low)/2; or, (low + high)/2;
      if (v[mid] < target) low = mid + 1;
      else high = mid;
   }
   if (v[low] == target)  cout << low << "Found\n";
   else if (v[high] == target)  cout << high << "Found\n";
   else  cout << "Not Found\n";
}
```

Sieve Algorithm (find prime number): =>O(nloglogn)

```
const int N = 1e7 + 10; // N=10^7
vector<bool> isPrime(N, 1);
void sieve()
{
   isPrime[0] = isPrime[1] = false;
   for (int i = 2; i < N; i++)
   {
      if (isPrime[i] == true)
      {
         for (int j = 2 * i; j < N; j += i)
         {
            isPrime[j] = false;
         }
      }
   }
}
```

Prime Factorization (Integer factorization): => O(sqrt(n))

Ex: 36 => 2 2 3 3

```
int main()
{
   int n;
   cin >> n;
   vector<int> prime_factors;
   for (int i = 2; i * i <= n; i++)
   {
      while (n % i == 0)
      {
```

```
            prime_factors.push_back(i);
            n /= i;
        }
    }
    if (n > 1) prime_factors.push_back(n);
    for (auto &prime : prime_factors)
        cout << prime << " ";
}
```

Prime Factorization using Sieve algorithm: => O(log(n))

Ex: 50 => 2 5 5

```
vector<int> spf(N); // SPF : smallest prime factor
void sieve()            // => O( nloglogn)
{
    for (int i = 1; i < N; i++)  spf[i] = i;
    for (int i = 2; i * i < N; i++)
    {
        if (spf[i] == i)
        {
            for (int j = i * i; j < N; j += i)
                if (spf[j] == j) spf[j] = i;
        }
    }
}
int main()
{
    sieve();
    int n;
    cin >> n;
    while (n != 1)
    {
        cout << spf[n] << " ";
        n /= spf[n];
    }
}
```

Find N'th Fibonacci number using Binet's Formula: => O(1)

```
int fib(int n){
    double phi = (sqrt(5) + 1) / 2;
    return round(pow(phi, n) / sqrt(5));
}
```

Binary Exponentiation using Iterative method: **=> O(log(b)).**

Ex: 3^{13} => $3^{(8+4+0+1)}$ => $3^8 * 3^4 * 3^0 * 3^1$ => 1594323; →**(a^b)**

```
const int Mod = 1e9 + 7;
long long BinExpIter(long long a, long long b)
{
   long long ans = 1;
   while (b)
   {
      if (b & 1)
      {
         ans = ans * a;
         // ans=(ans*a) % Mod;
      }
      a = a * a;
      // a=(a*a) % Mod;
      b >>= 1;
   }
   return ans;
}
```

Binary Exponentiation for $N^{1/x}$: **=> O(x*log(N*10^d))**

$3^{1/5}$**=** 1.2457312346;

```
double eps = 1e-6;    // eps=1e-d; =>with d decimal accuracy
double BinExpPow (double n, int x)
{
   double l = 0, r = n, m = (l + r) / 2;
   while (r - l > eps)
   {
      if (pow(m, x) > n) r = m;
      else l = m;
      m = (l + r) / 2;
   }
   return m;
}
```

Sum and Count of Divisor: **=>O(sqrt(n))**

Ex(sum): 20 => 22 (1+2+4+5+10+20). Ex(count): 20 => 6 (1,2,4,5,10,20).

```
int main()
{
   ll n, sum = 0, i, c = 0;
   cin >> n;
   for (i = 1; i * i <= n; i++)
   {
      if (n % i == 0)
      {
```

```
        sum += i, ++c;
        if (i != n / i) sum += n / i, ++c;
      }
    }
    cout <<"Sum = "<< sum <<" Count = "<< c << endl;
}
```

Number of divisors: **=> O(nlog(n))**

Ex: 32 => 2 4 8 16 32

```
const int N = 1e5 + 10;
vector<int> divisor[N];
int main()
{
    for (int i = 2; i < N; i++)
    {
        for (int j = i; j < N; j += i)
            divisor[j].push_back(i);
    }
    int n;  cin >> n;
    for (auto &it : divisor[n]) cout << it << " ";
    cout << endl;
}
```

Graph:

```
const int fx[] = {+0,+0,+1,-1,-1,+1,-1,+1}; // king's move (0 to 3 index => Side Moves)
const int fy[] = {-1,+1,+0,+0,+1,+1,-1,-1}; // king's move (4 to 7 index => Diagonal Moves)

const int kx[] = {-2,-2,-1,-1,+1,+1,+2,+2}; // knight's move
const int ky[] = {-1,+1,-2,+2,-2,+2,-1,+1}; // knight's move
```

Depth First Search(DFS): **=> O(V+E)**

```
const ll N = 1e5 + 10;
vector<ll> g[N], height(N), depth(N);
bool vis[N];
int Par[N];

void dfs(ll vertex, ll par = -1)
{
        /*** Take action on vertex after entering the vertex. ***/
    vis[vertex] = true;
    // bool isLoopExists = false;     //<= Use For Finding Cycle
    Par[vertex] = par;
    for (auto &child : g[vertex])
    {
        /**  Take action on child before entering the child node.  **/
        if (vis[child]) continue;
        dfs(child, vertex);
```

```
        /* => Use for Finding Cycle:
           if(vis[child] && child == par) continue;
           if(vis[child]) return true;
           isLoopExists |= dfs(child, vertex);
        */
        /* => Use for Tree (No need Visited array):
           if(child == par) continue;
           depth[child] = depth[vertex]+1;
           dfs(child, vertex);
           height[vertex] = max(height[vertex], height[child]+1);
        */
        /***  Take action on child after exiting child node.  ***/
   }
     /***  Take action on vertex before exiting the vertex.  ***/
}
```

Breadth-first search (BFS) And 0/1 BFS: => O(V+E)

```
const int N=1e5+10;
vector<int>g[N];             //vector<pair<int, int>> g[N]; =>For 0/1 BFS
bool vis[N];                 // No need vis array for 0/1 BFS.
vector<int> level(N);        //vector<int> level(N, INT_MAX); =>For 0/1 BFS

void bfs(int source)
{
   queue<int>q;         //deque<int>q; =>For 0/1 BFS
   q.push(source);      //q.push_fornt(source); =>For 0/1 BFS
   vis[source]=1;       //level[source] = 0; =>For 0/1 BFS
   while(!q.empty())
   {
      int par=q.front();
      q.pop();          //q.pop_front(); =>For 0/1 BFS
      for(auto &child: g[par])
      {
         if(vis[child]) continue;
         q.push(child);
         vis[child]=1;
         level[child]=level[par]+1;

        /*=> For 0/1 BFS:
          int  u = child.first,  w = child.second ;
          if(level[par] + w < level[u])
          {
             level[u] = level[par] + w;
             if(w==0) q.push_front(u);
             else q.push_back(u);
          }
```

```
      /*
    }
  }
}
```

Dijkstra's Shortest Path Algorithm(Single Source Shortest Path):

```
const int N = 1e5 + 10,  INF = 1e9 + 7;                          => O((V+ E)*log(V))
vector<pair<int, int>> g[N];   //g[u].pb({v,w});
vector<int> dist(N, INF);      //store minimum distance;
vector<bool> vis(N);

void dijkstra(int s)
{
  multiset<pair<int, int>> st;
  st.insert({0, s});
  dist[s] = 0;
  while (st.size())
  {
    int u = (st.begin())->second;
    // int u_w=(st.begin())->first;
    st.erase(st.begin());
    if (vis[u]) continue;
    vis[u] = 1;
    for (auto &child : g[u])
    {
      int v = child.first;
      int v_w = child.second;
      if (dist[u] + v_w < dist[v])
      {
        dist[v] = dist[u] + v_w;
        st.insert({dist[v], v});
      }
    }
  }
}
```

Floyd-Warshall Algorithm(All Pair Shortest Path): =>O(n^3)

=> finding the shortest paths in a weighted graph with positive or negative edge weights (but with **no negative cycles**);

```
const int N=510, INF=1e9+10;
int dp[N][N];
int n, m;
void floyd_warshall()
{
  for (int k = 1; k <= n; ++k)
  {
```

```
    for (int i = 1; i <= n; ++i)
    {
      for (int j = 1; j <= n; ++j)
      {
        if (dp[i][k] < INF && dp[k][j] < INF)
              dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j]);
      }
    }
  }
}
int main()
{
  cin>>n>>m;
  for(int i=1;i<=n; ++i)
  {
    for(int j=1;j<=n; ++j)
    {
      if(i==j)  dp[i][j]=0;
      else  dp[i][j]=INF;
    }
  }
  for(int i=0;i<m;i++)
  {
    int x, y, wt; cin>>x>>y>>wt;
    dp[x][y]=wt;
  }
  floyd_warshall();
  return 0;
}
```

Others:

Sorting pair Using Compare Function: =>O(n*log(n))

If vector<pair<ll, ll>> **vec**{{3, 4}, {1, 2}, {3, 5}, {3, 2}, {6, 1}};

bool **cmp**(pair<ll, ll> a, pair<ll, ll> b)
{
  if (a.first != b.first) return a.first < b.first; // **=>first** value **increasing** order;
  return a.second > b.second; //=> **second** value **descending** order;
}
sort(vec.begin(), vec.end(), **cmp**); //=> **vec**={{1,2}, {3,5}, {3,4}, {3,2}, {6,1}};

Minimum fraction:

If **a/b = c/d** => <u>ex:</u> 12/18 = 2/3

c = **a /__gcd(a,b)**; d = **b /_gcd(a,b);**

Count words in a string using stringstream:

```
#include<sstream>
#include<string>
int countWords(string str)
{
   stringstream sf(str);
   string word;
   int count = 0;
   while (sf >> word)
         count++;        // <= u can change statement
   return count;
}
```

Find SubString of a stirng: => O(n^2)

str = ”abcd” **=>** a, ab, abc, abcd, b, bc, bcd, c, cd, d.

```
for (int i = 0; i < str.length(); i++)
{
      string subStr;
      for (int j = i; j < str.length(); j++)
      {
         subStr += str[j];
         cout << subStr << endl;
      }
}
```

Find SubSequences / SubSet using Iterative: => O(n* 2^n)

{1, 2, 3} => {1}, {2}, {1, 2}, {3}, {2, 3}, {1, 2, 3}, {4}, {1, 4}, {2, 4}, …

```
vector<vector<int>> subsets(vector<int> &nums)
{
   vector<vector<int>> allSubSets;
   int n = nums.size();
  ///=> In Bits SubSets, the nums array is which Bit position you want for SubSets;
   for (int i = 0; i < (1 << n); i++)       //for 2^n possible solution
   {
      vector<int> subset;
      ///int tempA=a, tempB=b; ///=> for Bits SubSets
      for (int j = 0; j < n; j++)   //for nums array
      {
         if (i & (1 << j))
         {
            subset.push_back(nums[j]);
            ///tempA |= (1LL << nums[j]);      ///ON the nums[j] position Bit in tampA
         }
         ///else tempB |= (1LL << nums[j]);  ///ON the nums[j] position Bit in tampB
      }
      allSubSets.push_back(subset);
```

```
    ///ans=max(ans, temA*tempB); //qn needed  operation
  }
  return allSubSets;
}
```

Find SubSequences / SubSet using Recursion: => O(n* 2^n)

s =”abc” **=> subsequences** = { “a”,” b”, “c”, “ab”, “bc”, “ac”, “abc”};

```
vector<string> subsequences;
void AllSubsequences(string &s, string subseq="", int index=0)
{
  if (index == s.length())
  {
    subsequences.push_back(subseq);
    return;
  }
  AllSubsequences(s, subseq, index + 1);
  AllSubsequences(s, subseq + s[index], index + 1);
}
```

Extended Euclid: //=> O(log(min(a, b)))

For this Eq. **(a*x) + (b*y) = gcd(a, b);**

```
ll extended_euclid(ll a, ll b, ll &x, ll &y)
{
  if (b == 0)
  {
    x = 1, y = 0;
    return a;
  }
  ll x1, y1;
  ll gcd = extended_euclid(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return gcd;
}
```

- **Subarrays/Substring**: A subarray is a contiguous part of array and maintains relative ordering of elements. For an array/string of size n, there are **n*(n+1)/2** non-empty subarrays/substrings. **[“1234” => {1,2}, {1,2,3}, {2,3,4} etc.]**
- **Subsequence**: A subsequence maintain relative ordering of elements but may or may not be a contiguous part of an array. For a sequence of size n, we can have (**2^n)-1** non-empty sub-sequences in total. **[“1234” => {1,2,4}, {2,4} etc.]**
- **Subset**: A subset **MAY NOT** maintain relative ordering of elements and can or cannot be a contiguous part of an array. For a set of size n, we can have **(2^n)** sub-sets in total. **[“1234” => {1,3,2}, {4,2,3} etc.]**
- **Co-Prime:** That means a pair of numbers are said to be co-prime when they have their highest common factor as 1. [i.e**: gcd(A, B)=1**;]
- **Lexicographic or Lexicographically:** means sorting in the natural order / dictionary order. [Ex: “a” **<** ”b”**;** “aa” **<** “ab”**;** “aaab” **<** ”ab”**;** “abcd” **<** ”baa”**;**]
- **Parity**: is a term used to refer to the property of being even or odd.
- **Permutations**: are often used to count the **number of ways to arrange** a certain number of objects. The number of permutations of a set of n objects is given by **n!**.
- **MEX:** usually refers to the "minimum excluded value" of a set. Given a set of non-negative integers, the MEX is the smallest non-negative integer that is not present in the set. Ex: {0, 1, 3, 4, 7} => MEX is 2;