Dangling Pointer:

It's a type of pointer that points to a memory location that is not valid now. (initially it might be valid)

ex:
```
int *ptr = NULL;
{
  int x = 10;
  ptr = &x;
}
cout<<ptr;
```
→ o/p: 0x16b7c7364 (i.e. it points to that variable which doesn't exist)

Void Pointer:

- These are special pointers that can point to any datatype value.
- Void pointers cannot be de-referenced, but using typecasting we can!

ex:
```
float f = 11.3;
int x = 10;
void *ptr = &f;
ptr = &x;
int *intPtr = (int *)ptr;
cout<< *intPtr;
```
→ o/p: 10

RECURSION | RECURSION CONCEPT AND PROBLEMS

In recursion, we try to solve a bigger problem by finding out solutions to smaller sub-problems.

We represent these problems in the form of functions & these func. calls itself to solve smaller sub-problems.

ex: Write a func which calcs n!

↳ $5! = 5\times4\times3\times2\times1 = 5\times4! = 5\times4\times3! = 5\times4\times3\times2! = 5\times4\times3\times2\times1$

$f(n) = n * f(n-1)$: $\{n! = n\times(n-1)!\}$

($f(n)$: func. to calc. $n!$; Recursion: function calls itself)

simple soln. {Iterative soln.}
```
ans=1;
for(int i=1; i<=n; i++){
  ans=ans*i;
}
```

Now, how can we think about this? So we should know;

PMI → Principal of Mathematical Induction

Here we do 3 works:
- Base Case
- Assumption
- Self-work

ex: Prove sum of first 'N' natural no.s is $\frac{n(n+1)}{2}$.

Soln: # Base Case → for N=1 → ans will be =1

Assumption → Lets say formula works for: $N=k \rightarrow \frac{(k)(k+1)}{2}$

Self-Work → $(1+2+3+\dots+k+k+1)$

$\frac{(k)(k+1)}{2}+(k+1) = \frac{(k+1)(k+2)}{2}$ → Satisfies for sum upto k+1. Similarly upto, k+2, k+3, ...

Hence Proved

In recursion also we use these 3 cases.

Now lets go back to factorial problem;

$f(n) = n \times f(n-1)$ → (recurrence reln.)

(assump + self-work)

Base Case = Halting condn.

Assumption = $f(n-1)$

Self-Work = multiply n & $f(n-1)$

we wrote a func. f (that returns $n!$)

&

assume func. f works correctly for 'n-1'

⇒ $5! = 5\times4!$ → $4\times3!$ → $3\times2!$ → $2\times(1!)$ → base case $\{1! = 1\}$

(120, 24, 6, 3×2, 2×1)

base case: $n==1 \rightarrow f(n) = 1$

∴ In code.

```
1) int f(int n){
2)     if(n==1){return 1;}
3)     int ans = n*f(n-1);
4)     return ans;
5) }
   int main(){
     int result = f(5);
     cout<< result;
     return 0;
   }
```

Now this prog. converts into process which is loaded into RAM. Now memory to this allocated in call stack way.

We can also write this as,

return n*f(n-1);

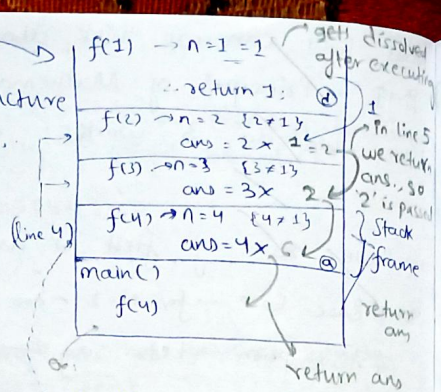
{Better explanation next page}

o/p = 120

Call Stack:

Stack is like a linear data structure which supports last & first func.

```
int main(){
    int i=f(4);
    return 0;
}
```



Whenever a func. call is seen, a new entry in the call stack is made. Now main fnc. will wait until all func. above it completed.

Now ⓐ will also wait until all fnc.'s above it is executed & so on similarly.

Now in ① '1=1', & we return '1' because if we return, the whole stack frame gets dissolved & the final value is returned from where the func. is called.

Complexity Analysis.

Now, as we can see above for int n=4 → 4 stack frames are getting created {inside just variables are passed ~ constant space, so ignored} → so stack frames depends on 'n'.

∴ Space Complexity = O(n) ; Also: space gets introduced due to call stack

(That's why in gen., iterative soln. better as it takes less S_c)

Time → (No. of times func. is called) × (Time consumed in one single fnc. call) (∝ C) ↳ same const. task

(∝n)

= n × C → nC

{ we can see from 1 to 6 all are const. part.
Calling fnc. = const. task
Multiplication = " "

∴ T_c = O(n)

(Not all recursive problems have this approach)

Syntax: (of Recursive func.)

```
methodName (N parameters){
    if (halt condition){
        return result;
    }
    return methodName (N parameters);
}
```

bigger problem → methodName (N parameters)

base case

↳ Assumption + self working

smaller sub-prob.

Q) Program to find n^{th} fibonacci number? (use recursion)

(seeding value)

A) Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

↑ 0th ↑ 1st ↑ 2nd ↑ 3rd ↑ 4th ↑ 5th ↑ 6th ↑ 7th ↑ 8th (fab no.)

Value of any n^{th} term is equals to the sum of prev. 2 terms

i.e. $(n-1)^{th}$ term + $(n-2)^{th}$ term (bigger problem)

Let:

fib(n) → fnc. returns n^{th} fabonacci no.

- Base Case: if (n==0) return 0; if (n==1) return 1;
- Assumption: Lets assume fnc. fib works correctly for n-1 & n-2
  i.e. fib(n-1) ✓ , fib(n-2) ✓
- Self work: Sum of $(n-1)^{th}$ & $(n-2)^{th}$ term

⇒ fib(n) = fib(n-1) + fib(n-2)

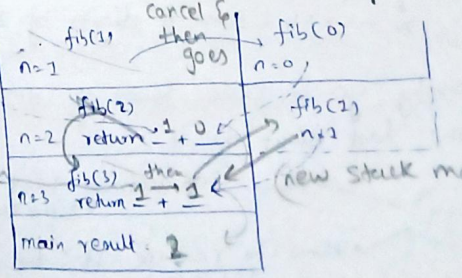
(Assumption) (Self work)

∴ code:

```
1)  int fib(int n){
2)      if(n==0 or n==1){
3)          return n;
4)      }
5)      return fib(n-1)+fib(n-2);
6)  }
```

```
int main(){
    cout<<fib(8);
    return 0;
}
```

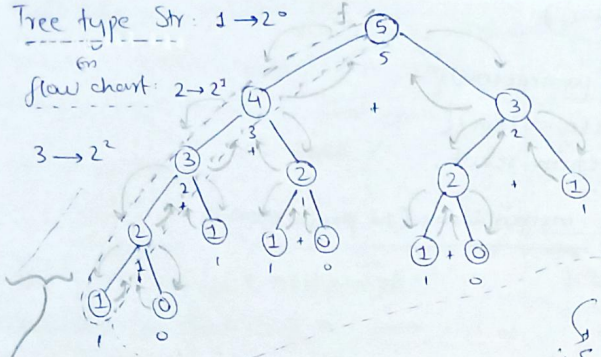
Process:



Tree type Str: $1 \to 2^0$

flow chart: $2 \to 2^1$

$3 \to 2^2$



Space Complexity

↳ max depth of the tree or after getting the value the stack is deleted, so

→ Here for n=5, depth went till 5

$\therefore S_c \to O(n)$

Time Complexity (formula in prev. page)

As we can see → ① to ⑥ are const. func. call

No. of times func. called $= 2^0 + 2^1 + 2^2 + \dots + 2^{n-1}$ → {on an avg, we get approx like this only}

$= \frac{1 \times (2^n - 1)}{2-1} = 2^n - 1$

$\therefore T_c = O(2^n)$

RECURSION PROBLEMS - 1 | SUM OF DIGITS & POWER OF A NUMBER

Q) Given an integer, find out the sum of its digits using recursion?

A) $f(n)$ → this func. returns the sum of digits of n

As usual, we use PMI:

- Base Case → if(n≥0 and n≤9){ return n; }
  ↳ if you have single digit no, then ans. is no. itself.
- Assumption
- Self-work

Now, how do we add all digits

one way → ∵ n = 1234 → 'd' digits

↳ digitsum(123) + 4

$x \to x+4$ = sum of digits of 1234

$f(1234) \to f(123)+4 \to f(12)+3 \to f(1)+2$ (base case)

∴ Recursively go & calc. sum of d-1 digits & add the last digit to it.

1234 → How to extract last digit ↳ (n%10)

1234 → How to reduce it to 123. ↳ (n/10)

$\therefore f(n) = f(n/10) + n\%10$

calc. sum of no. remaining after elimination of last digit

last digit

- Assumption: assume that func. works correctly for n/10
- Self Work: Adding last digit of n i.e. n%10 to the sum of the digits n/10.

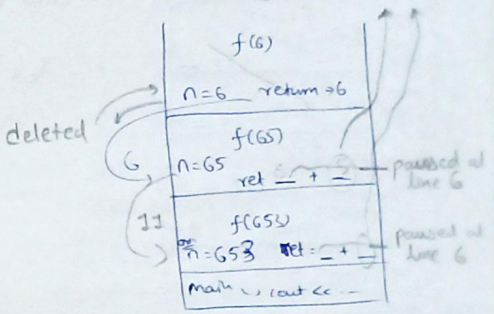
Code

```
1) int f(int n){
2)   if(n<=9 and n>=0){
3)     return n;
4)   }
5)   return f(n/10) + (n%10);
6) }

int main(){
    int n;
    cin>>n;
    cout<<f(n);
    return 0;
}
```

dry run: → (call stack method)

this action will occur until above stack is executed



$\therefore f(1234) \to f(123) \to f(12) \to f(1)$

∴ No. of calls = d (no. of digits)

For T_c → 2 - 5 } const. part = c

↳ $\therefore d \times c \to O(d)$

For S_c → O(d) space is used.

Q) Given 2 numbers p & q, find the value p^q using a recursive function?

A) $f(p,q) = p \times f(p, q-1)$

$f(p,q)$: this func f, calculates 'p' raise to the power q

assuming that for f correctly calculates p^{q-1}

with the result of p^{q-1}, we will multiply p

- Base Case: for q==0, answer will be always 1 : f(p,0){return 1;}
- Assumption: lets assume func. f works correctly for (p,q-1),
- Self-Work: multiply 'p' with f(p,q-1)

∵ $4^3 \to 4^2 \times 4 \Rightarrow f(4,3) = f(4,2) \times 4 = f(4,1) \times 4 = f(4,0) \times 4$

Code:

```
int f(int p, int q){
    if(q==0){
        return 1;
    }
    return p*f(p,q-1);
}
```

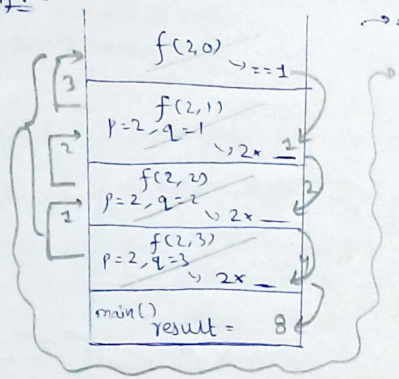
→ const. work in time comp.

```
int main(){
    int result = f(2,3);
    cout<<result;
    return 0;
}
```

→ o/p: 8

Runs via call-stack way.

Dry run:



→ f(p,q) → f(p,q-1) → f(p,q-2) → ..... f(p,0)

Total no. of calls = q

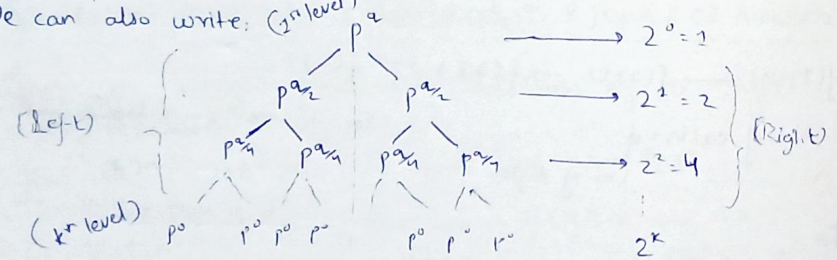
∴ $T_c = q \times c$

$= T_c \to O(q) \leftarrow S_c$

(as q↑'s, no. of stacks also increases proportionally)

Soln. 2 {optimised}

We can also write: (1st level) p^q



⇒ Total levels → $\frac{q}{2^0} \to \frac{q}{2^1} \to \frac{q}{2^2} \to \frac{q}{2^3} \cdots \to \frac{q}{2^k}$

Finally, $\frac{q}{2^k} = 1$ (log) $\Rightarrow \log_2 q = k$

Total no. of func. calls = $2^0 + 2^1 + 2^2 + \cdots 2^{\log q}$

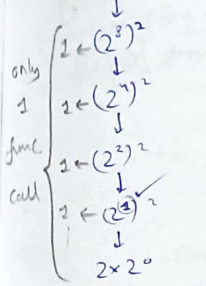
$= \frac{1(2^{(\log_2 q)+1} - 1)}{2-1} = 2 \times 2^{\log_2 q} (-1)^x$

$= 2q \to q$

∴ T_c is still $O(q)$

because the problem arises from right side of tree, it has to calculate all the values which it did from left. Let's look out for other options.

ex: 2^{16}



∴ Here: $q \to \frac{q}{2} \to \frac{q}{4} \cdots \to \frac{q}{2^{k-1}}$

$\frac{q}{2^{k-1}} = 1$ (log) $\Rightarrow \log_2 q = k-1 \Rightarrow k = \log_2 q$

$T_c \to O(\log q)$

{which is very good as previously we were getting total 'q' T_c}

$$\therefore f(p,q) = \begin{cases} (f(p, q/2))^2 & ; q \to \text{even} \\ (f(p, q-1/2))^2 \times p & , q \to \text{odd} \end{cases}$$

In code:

```
int f(int p, int q){
    if(q==0){
        return 1;
    }
    if(q%2==0){ //if q=even
        int result = f(p, q/2);
        return result*result;
    } else { //if q=odd
        int result = f(p,(q-1)/2);
        return p*result*result;
    }
}
```

```
int main(){
    int res = f(3,4);
    cout<<res;
    return 0;
}
```

→ pg 81

Dry run → Via call stack ways.

PROBLEM SOLVING ON RECURSION - 2 | RECURSIVE OPERATIONS ON ARRAYS

Most of the times to solve a problem related to arrays & strings, we definetely need to read them.

idx → idx+1 / idx-1

: f(arr, idx) → f(arr, idx-1)

(given array) (current index we are pointing in the array) (we go to prev. index)

Q) Given an array, print all the elements of the array recursively?

A) $f(arr, idx, n)$ = print (arr[idx])

(before you go to print remaining array, I can print myself)

$f(arr, idx+1, n)$

(recursively prints the elements of array starting from index idx)

(assume that we can correctly print the remaining array using f)

n → length of array

$f(arr,0,5) \to f(arr,1,5) \to f(arr,2,5) \to f(arr,3,5) \to f(arr,4,5)$

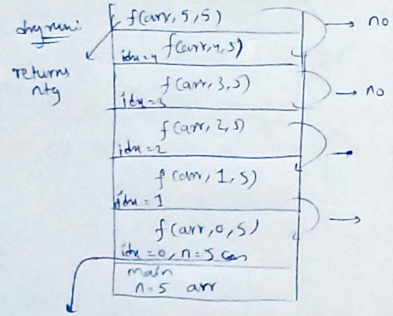
$f(arr,5,5)$ → base case

Code:

```
void f(int *arr, int idx, int n){
    if(idx==n){ //base case
        return; //self-work
    }
    cout<<arr[idx]<<"\n";
    f(arr, idx+1, n); //assumption
}
```

```
int main(){
    int n=5;
    int arr[]={6,1,9,3,4};
    f(arr,0,n);
    return 0;
}
```

dry run:



(print arr[0] & then func. call)

$\therefore T_c = (\text{No. of func. calls}) \times (\text{time taken for 1 func. call})$

$= O(n) = S_c$

Q) Print the max value of the array [3,10,3,2,5]?

A) $f(arr, idx, n) = \max(arr[idx], f(arr, idx+1, n))$

this func. returns max element in the given array starting from index idx

compare current element with max. of remaining array

- Base Case: if array has only 1 element, that is the max
  ∴ (idx == n-1) → return arr[idx]
- Assumption: f' works correctly for f(arr, idx+1, n)
  i.e. it successfully finds out max in remaining array.
- Self Work: Compare the max of remaining array with curr. ele.

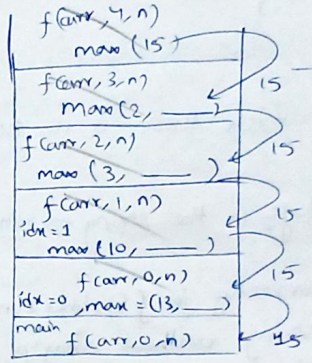
Code:

```
int f(int *arr, int idx, int n){
    if(idx==n-1){ //base case
        //we only have 1 element left, so it's the maximum
        return arr[idx];
    }
    return max(arr[idx], f(arr, idx+1, n));
}
```

```
int main(){
    int arr[]={13,10,3,2,15};
    int n=5;
    cout<<f(arr,0,n);
    return 0;
}
```

o/p: 15

dry run:



→ after returning value, above stack gets deleted

Q) Find the sum of the values of the array [2,3,5,20,1]?

A) Similar like prev. que:

- Base Case: if there is only 1 element in the array, then sum of elements is equal to the given element
- Assumption: assume func. f works for f(arr, idx+1, n) i.e. f can calculate sum of remaining elements
- Self-Work: add arr[idx] to sum of remaining elements

$f(arr, idx, n) = arr[idx] + f(arr, idx+1, n)$

returns sum of elements of array from index idx

sum of remaining elements

eg: [2,3,5,20,1]

$f(arr,0,5) \to f(arr,1,5) \to f(arr,2,5) \to f(arr,3,5) \to f(arr,4,5)$

31 ← 29 ← 26 ← 21 ← 1

Code:

```
int f(int *arr, int idx, int n){
    if(idx==n-1){
        return arr[idx];
    }
    return arr[idx]+f(arr, idx+1, n);
}
```

```
int main(){
    int arr[]={2,3,5,20,1};
    int n=5;
    cout<<f(arr,0,n);
    return 0;
}
```

o/p: 31

Dry run: → like prev, so not writing!

PROBLEM SOLVING ON RECURSION - 3 | STRING PROBLEMS

Q) Remove all the occurences of 'a' from string S = "abcax"

A) Lets look at iterative soln.

$T_c = O(n)$

```
result = "";
for(int i=0; i<n; i++){
    if(s[i] != 'a'){
        result += s[i];
    }
}
```

For recursive soln;

f(s, idx, n) → returns string in which there is no occurence of 'a'.

- Base Case: if there is no char left in the string → return empty string;
- Assumption: assume func. works correctly for f(s, idx+1, n) means, func. will return a string without any 'a', from the remaining chars of 's' from index idx+1.
- Self-Work: if my current character is not 'a', then append current char with ans of f(s, idx+1, n) else don't append anything.

$\therefore f(s, idx, n) = x + f(s, idx+1, n)$ where $x = \begin{cases} "" & \text{if } s[idx] == 'a' \\ s[idx] & \end{cases}$

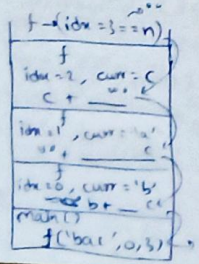
$= (s[idx] == 'a') \; ? \; "" : s[idx]) + f(s, idx+1, n)$

Code:

```
string f(string &s, int idx, int n){
    if(idx==n){ return ""; }
    string curr = "";
    curr += s[idx];
    return ((s[idx]=='a') ? "" : curr) + f(s, idx+1, n);
}
```

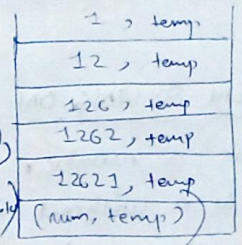
```
int main(){
    string s = "abcax";
    int n = 5;
    cout << f(s, 0, n);
    return 0;
}
```

o/p: bcx



Q) Write a program to check whether a given number is palindrome (or) not ?

A) If solved via iteration → 1) Make no. to string
2) Start 'i' & 'j' from front & back
3) run until $j < i$
4) Check if $i == j$



Now for recursive soln.;

ex: num = 12621
temp = 12621

*temp = *temp / 10 (then reducing temp)

f(num, *temp) = f(num/10, temp) and (num%10 == temp%10)

(check if num is palindrome (or) not by reading num from L→R & temp from R→L)

we make changes in org. temp

checking: num = 12621, temp = 12621 → these both are equal (or) not

- Base Case: Single digit is always a palindrome

Code:

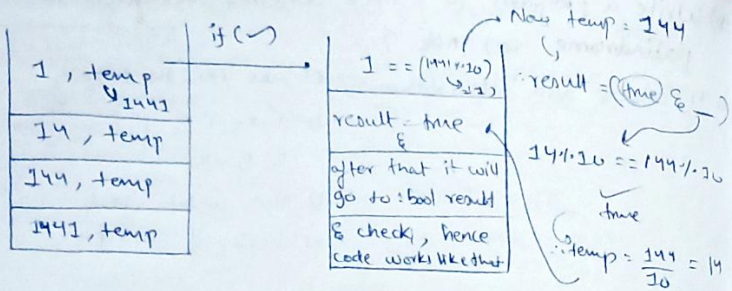
```
bool f(int num, int *temp){
    if(num>=0 and num<=9){
        int lastDigitTemp = (*temp)%10;
        (*temp) /= 10;
        return (num == lastDigitTemp);
    }
    bool result = (f(num/10, temp) and (num%10) == ((*temp)%10));
    (*temp) /= 10;
    return result;
}
```

```
int main(){
    int num = 12621;
    int anotherNum = num;
    int *temp = &anotherNum;
    cout << f(num, temp);
    return 0;
}
```

Here we are passing value of temp using call by reference, so that changes done in temp are original & match the condition.

dry run:

eg: 1441



(only 1 call stack made, but here made kinda 2 for simplicity)

PROBLEM SOLVING ON RECURSION - 4 | BASIC PROBLEMS

Q) Given a number 'n'. Find the increasing sequence from 1 to n without using any loop.

constraint: $0 < n <= 1e^6$; Ip 1 : n = 4 ; Ip 2 : n = 2

Op 1 : 1 2 3 4 Op 2 : 1 2

A) ① My answer:

```
int f(int n,int m){
    if(n-->0){
        cout<<(m-n)<<" ";
    }
    return f(n,m);
}
```

```
int main(){
    int n=12, m=12;
    cout<<f(n,m);
    return 0;
}
```

here we are printing wrt given ip → i.e. 3 → printing 3-2 = 1, 3-1 = 2, 3-0 = 3

② Vid soln.:

f(n) = f(n-1) → print(n) {n=5 ; 1 → n ; 1 2 3 4 5}

(this func. prints the first 'n' natural no.) (assume that func. works correctly for n-1)

; Base Case → if (n<1){ return; }

Code:

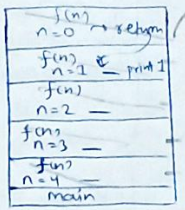
```
1 void f(int n){
2     if(n<1){
3         return;
4     }
5     f(n-1);
6     cout<<n<<" ";
7 }
```

```
int main(){
    f(5);
    return 0;
}
```

Op: 1 2 3 4 5

dry run:

eg: f(4)



(stack removed after returning)

5 → Now this part left at line 5, & as we can see, no other part is left to call, so we go to line 6 → cout<<n;

5 (similarly) 5 5

(then stack gets deleted)

Q) Given a number num & a value 'k'. Print k multiples of num.

Constraints: k > 0 ; Ip 1 : num = 12, k = 5 Op 1 : 12, 24, 36, 48, 60

Ip 2 : num = 3, k = 8 Op 2 : 3, 6, 9, 12, 15, 18, 21, 24

A) Similar to prev. que.

Iteratively →
```
for(int i = 1 ; i<=k ; i++){
    int val = num*i;
    cout<<val<<" ";
}
```

Recursively → f(num, k) = f(num, k-1) → print(num*k)

(print first 'k' multiples of num) (assumption) (selfwork)

Base case → (k==0) → we do nothing & just return

Code:

```
void f(int num, int k){
    if(k==0) return;
    f(num,k-1);
    cout<<(num*k)<<" ";
}
```

```
int main(){
    f(8,5);
    return 0;
}
```

Op: 8 16 24 32 40

(no need to add brackets)

dry run: → like prev. que.

Q) Given a number n. Find the sum of natural no.'s till 'n' but with alternate signs.

That means if n=5 then you have to return 1-2+3-4+5 = 3 as your answer.

Ip 1 : n = 10 Ip 2 : n = 5

Op 1 : -5 Op 2 : 3

A) ⊙ All even no.'s get a (-)ve sign

⊙ All odd no.'s get a (+)ve sign

My thought:

```
int f(int n){
    if(n<1) return 0;
    int res = 0;
    if(n%2==0) res -= n;
    else res += n;
    return res;
}
```

if n = 10 → o/p = -10
reason: result = 0 always, So its printing -10 always.

- If we dont declare value of 'res', we will get garbage ans.

*So always remember 'Ternary Operator' in these cases!

$f(n) = f(n-1) + ((n\%2==0)\ ?(-n):n)$

($f(n)$: sum of first n natural no. with alt. sign) ($f(n-1)$: assume func. works correctly for n-1)

: Base case: n → 0 → return 0;

Code:

```
int f(int n){
    if(n<1) return 0;
    return f(n-1) + ((n%2==0)?(-n):(n));
}
```

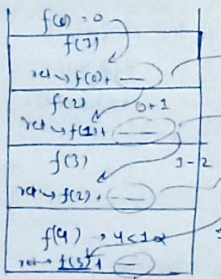
(helps in summing up all the values from before)

```
int main(){
    cout<<f(10);
    return 0;
}
```

o/p: -5

dry run

e.g. f(4) →



cout → 1-2+3-4 = -2

PROBLEM SOLVING ON RECURSION - 5 | FROG JUMP PROBLEM

Q) Given 2 no.s 'x' & 'y'. Find the greatest common divisor of x & y using recursion.

Constraints: $0<=x, y<=10^6$; ip 1: x=4, y=9 ; op 1: 1 ; ip 2: x=12, y=20 ; op 2: 4

e.g. GCD = 12 → 2×2×3
20 → 2×2×5
∴ GCD(12, 20) = 4

Iteratively:

```
int x,y,a,b,z=0;
cin>>x>>y;
a = max(x,y);
b = min(x,y);
for(int i=1; i<b; i++){
    if(a%i==0 and b%i==0){
        z = i;
    }
}
cout<<z;
```

Recursively:

Euclid's Algorithm:

If we subtract a smaller number from a larger one, we can (reduce the large no.) but the gcd will not change

e.g. gcd → 54, 72 ~ 54, 72-54 = 18 ~ 18

And this same theory is correct also incase of division

e.g. gcd → 54, 72%54 ~ 54, 18 → 18, 54%18 → 18, 0

Considering a > b ⇒ gcd(a,b) = gcd(b, a%b)

Now assume gcd(a,b) is 'q'

$a = bq + r \rightarrow \boxed{a - bq = r}$ {bq → is multiple of 'b'}

We can say that 'q' completely divides 'a' & 'b'.

Also: (a-bq) is completely divisibly by q.

∴ gcd(a,b) = gcd(b, a%b) → when a > b → base case: if(b==0){ return a; }
↳ recurrence

DIVISION

e.g. 7) 23 (3 ← quotient (q)
21
2 ← remainder (r)
↑ divisor

a = b×q + r = 23 = 7×3+2

e.g. gcd(48, 40) → gcd(40, 48%40) → gcd(40, 8) → gcd(8, 0)

Code:

```
int gcd(int a, int b){
    if(b>a) return gcd(a,b);
    if(b==0) return a; //base case
    return gcd(b, a%b);
}
```

```
int main(){
    int x = gcd(40,48);
    cout<<x;
    return 0;
}
```

Dry run → via call stack model

Q) Given a number n. Print if it's an armstrong number (or) not.

An armstrong number is a number if the sum of every digit in that number raised to the power of total digits in that number is equal to the number.

ip 1 : 153
op 1 : Yes

153 = $1^3+5^3+3^3$ = 1+125+27 = 153, hence 153 is an armstrong no.

A) $f(n,d) = pow(n\%10, d) + f(n/10, d)$

sum of digits of 'n', raise to power 'd'

Self

Assumption {pow(a,b) = a^b}

d: no. of digits in original number

eg: num = 4162, d = 4

f(4162,4) → f(416,4) → f(41,4) → f(4,4) → f(0,4)

$4^4+1^4+6^4+2^4$; $1^4+4^4+6^4$; 4^4+1^4 ; 0+44

base case ↳ if(n==0) → return

In code:

```
int pow_recursive(int p, int q){
    if (q==0) return 1;
    if (q%2==0){
        int result = pow_recursive(p, q/2);
        return result * result;
    }else {
        int result = pow_recursive (p, (q-1)/2);
        return p * result * result;
    }
}

int f(int n, int d){
    if(n==0) return 0; → base case
    return pow_recursive(n%10, d) + f(n/10, d);
}
```

power func. recur. ↑ done before

```
int main(){
    int n;
    cin>>n;
    int no_of_digits = 0;
    int temp = n;
    while(temp>0){
        temp = temp/10;
        no_of_digits++;
    }
    int result = f(n, no_of_digits);
    if(result == n){
        cout<<"YES";
    }else{
        cout<<"NO";
    }
    return 0;
}
```

eg: n = 153
op → YES

Q) There are 'N' stones, numbered 1,2,...N. For each i (1 ≤ i ≤ N), the height of stone 'i' is 'hi'. There is a frog who is initially on stone 1. He will repeat the following action some number of times to reach stone N:

If the frog is currently on stone i, jump to i+1 stone (or) stone i+2. Here, a cost of |hi - hj| is incurred, where 'j' is the stone to land on.

Find the minimum possible total cost incurred before the frog reaches stone N.

ip n=4 → arr[] = 10 30 40 20 → op = 30

eg: h → 10 30 40 20
i → 1 2 3 4
|30-10| + |40-30| + |20-40|

No. of paths
1 → 2 → 3 → 4 = 20+10+20 = 50
1 → 3 → 4 = 30+20 = 50
1 → 2 → 4 = 20+10 = 30

(i → i+1, i → i+2) (only 2 choices) (frog) |10-30| + x (ith final stone)

|h2-h1|+x ; |h2-h3|+y

(min) (max)

Base Case: i == last_stone (no stones to jump)
i == second_last_stone (only i+1 we can call)

(height array)

$f(h,n,i) = \begin{cases} f(h,n,i+1) + |h_i - h_{i+1}| \\ f(h,n,i+2) + |h_i - h_{i+2}| \end{cases}$

min cost to reach n^{th} stone from i^{th} stone

Code:

```
int f(int *h, int n, int i){
    if(i==n-1) return 0;
    if(i==n-2) return f(h,n,i+1) + abs(h[i] - h[i+1]);
    return min(f(h,n,i+1) + abs(h[i] - h[i+1]), f(h,n,i+2) + abs(h[i] - h[i+2]));
}
int main(){
    int arr = {10,30,40,20};
    int n = 4;
    cout<<f(arr, n, 0);
    return 0;
}
```

op: 30

PROBLEM SOLVING ON RECURSION - 6

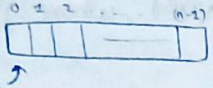
Q) Given an array on 'n' integers & a target value 'x'. Print whether 'x' exists in the array or not?

Constraints: 0<=n<=1e6, -1e8<=x<=1e8, -1e8<=a[i]<=1e8

ip1: n=8, x=14, array = [4,12,54,14,3,8,6,1]
op1: Yes

ip2: n=1, x=9, array = [2]
op2: No

A) $(n \le 10^8)$



x → element to be found
↳ target

⇒ $f(arr, n, i, x) = (arr[i] == x) \;||\; f(arr, n, i+1, x)$ {self-work}

- this func. returns whether element x, is present in the array (arr) from index i to (n-1) or not
- this checks detects whethere element is present on i^{th} index or not
- step of assumption ↓ func. correctly evaluates whether 'x' is present in range [i+1, n-1]

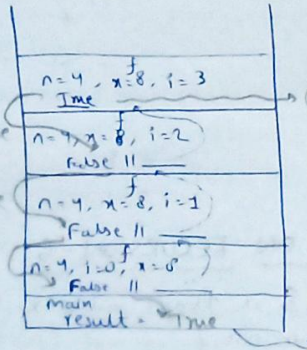
Base Case → if (i == n)
↳ return false.

```
// f represents whether 'x' is present in the range [i, n-1] or not
bool f(int *arr, int n, int i, int x){
    // base case
    if (i==n){
        // array is exhausted
        return false;
    }
    return (arr[i]==x) || f(arr, n, i+1, x);
}
```

```
int main(){
    int arr[] = {5,4,1,8};
    int n = 4;
    int x = 8;
    bool result = f(arr, n, 0, x);
    if (result) cout<<"YES";
    else cout<<"NO";
    return 0;
}
```

o/p: YES

dry run:

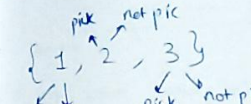


Q) Given an array of integers, print sums of all subsets in it. Output sums can be printed in any order.

i/p: arr[] = {2, 3}
o/p: 0 2 3 5

i/p: arr[] = {2, 4, 5}
o/p: 0 2 4 5 6 7 9 11

Total subset for set of length 'n' = 2^n (kinda derivation)



$2 \times 2 \times 2 = 2^3$ } each element has a choice whether to be picked or not

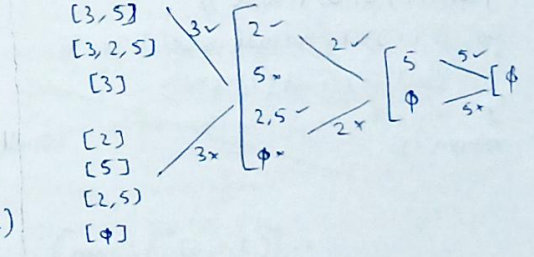
pick → element can be added to the subset
not pick → element cannot be added to the subset

ex: {1, 2, 3}

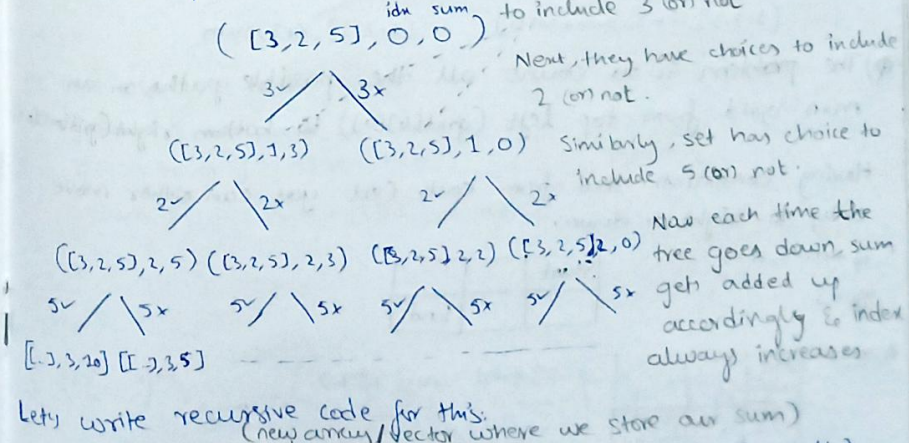
×	×	×	{ϕ}
✓	×	×	{1}
×	✓	×	{2}
×	×	✓	{3}
✓	✓	×	{1,2}
×	✓	✓	{2,3}
✓	×	✓	{1,3}
✓	✓	✓	{1,2,3}

(How subset is calculated)

ex: arr = [3, 2, 5]



for better explanation:



Now set has 2 choice, whether to include 3 (or) not

Next, they have choices to include 2 (or) not.

Similarly, set has choice to include 5 (or) not.

Now each time the tree goes down, sum gets added up accordingly & index always increases

Lets write recursive code for this:

f(arr, n, idx, sum, result) (new array / vector where we store our sum) = f(arr, n, idx+1, sum+arr[idx], result)
↳ f(arr, n, idx, sum, result)

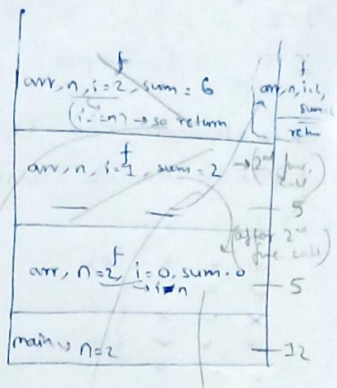
Prints sum of all subsets [idx, n-1]

*Base Case: if (idx == n)
↳ result.push_back(sum);
return;

In code: `#include <vector>`

```
void f(int *arr, int n, int i, int sum, vector<int>&result){
1)  if(i==n){
2)      result.push_back(sum);
3)      return;
4)  }
5)  f(arr, n, i+1, sum+arr[i], result);   // picks ith element
6)  f(arr, n, i+1, sum, result);          // doesn't pick up ith element
}
int main(){
1)  int arr[] = {2,4,5};
2)  int n = 3;
3)  vector<int> result;
4)  f(arr, n, 0, 0, result);
    for(int i=0; i<result.size(); i++){
        cout<<result[i]<<" ";
    }
    return 0;
}
```

dry run on arr = [2,4]



result = [6 2 4 0]

(code goes to line 6 & stack repeats)

∴ ([1,2,3], i, sum)

[1,2,3], i+1, sum+arr[i] [1,2,3], i+1, sum

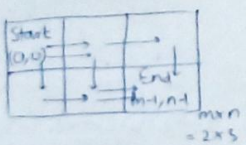
Q) The problem is to count all the possible paths on an m×n grid from top left (grid[0][0]) to bottom right (grid[m-1][n-1]) having constraints that from each cell you can either move only to right or down.

ip → m=2, n=3

op = 3

Start		
		End

A) Ex 1:



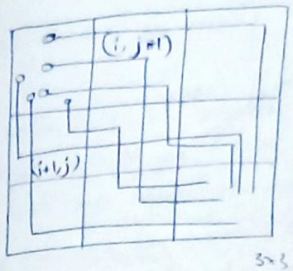
→ total ways: RRD, RDR, DRR } 3 ways (= 3!/2!)

(i,j) → right (i,j+1)

↓ down (i+1,j)

(if we know how to go to 'end' from here, then we easily get the ans.)

Ex 2:



RRDD
RDRD
RDDR
DDRR
DRRD
DRDR
} 6

also = no. of combination = 4!/(2!2!) = 6

If we can calculate no. of steps from (i,j+1) & (i+1,j) to bottom, we can get the ans.

f(i,j,m,n) = f(i,j+1,m,n) + f(i+1,j,m,n)

(self work)

this returns no. of ways to reach m-1, n-1 from i,j if one can only move right & down.

• Assumption:

→ f(i,j+1,m,n) → we assume fun. f works correctly for i,j+1 i.e. it correctly gives no of ways to reach B.R from right cell.

→ f(i+1,j,m,n) → we assume fun. f works correctly for i+1,j i.e. it correctly gives no. of ways to reach B.R from down cell.

• Base case:

```
if(i==m-1 && j==n-1)
    return 1;
if(i>=m || j>=n)
    return 0;
```

(when arrow goes out of grid)

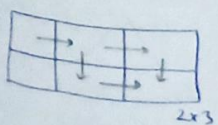
In code:

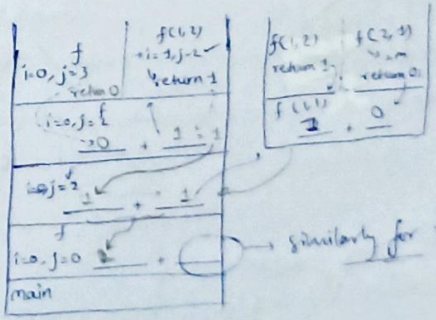
```
int f(int i, int j, int m, int n){
    if(i==m-1 and j==n-1) return 1;
    if(i>=m or j>=n) return 0;
    return f(i,j+1,m,n) + f(i+1,j,m,n);
}
```

```
int main(){
    cout<<f(0,0,2,3);
    return 0;
}
```

op = 3.

dry run





∴ Final → 2+1 = 3

Also, we can solve this ques. using perm. & comb. concept

like → $6 = \frac{4!}{2!\,2!}$ → we can create diff cases.

↳ (when I tried in my VS code, no.s ≤ 3 we have to make one case & no.s > 3 → another case —! so yea we can also solve like this

but recursion code is better as sometimes large i/p can also be. ←

PROBLEM SOLVING ON RECURSION-7 | SUBSEQUENCE CONCEPT

Q) Given a string, we have to find out all its 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.

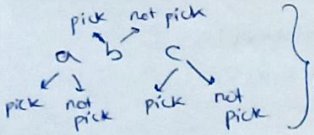
i/p: abc

o/p: a, b, c, ab, bc, ac, abc

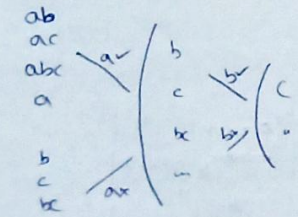
A) (abc)$_3$

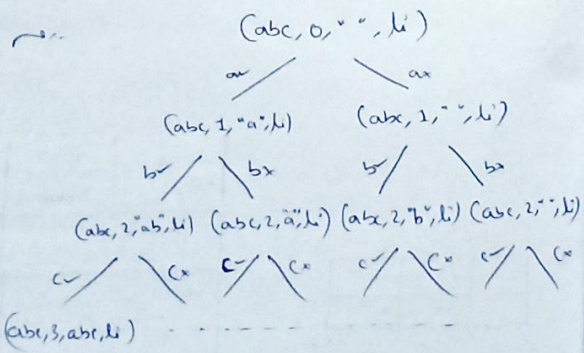
n → length → 2^n subsequences

(means relative order of subset is same
for ex: abc → subset can be ac, ca but subsequence will be ac only as 'c' comes after 'a')



each of them has 2 choices





∴ relation: string, vector string

f(str, i, result, li) = f(str, i+1, result + str[i], li)
f(str, i+1, result, li)

• Base case: if (i == str.length) li.pushback(result)

code

```
void f(string &str, int i, string result, vector<string> &li){
    if(i==str.length()){
        li.pushback(result);
        return;
    }
    f(str, i+1, result+str[i], li);
    f(str, i+1, result, li);
}
```

```
int main(){
    vector<string> res;
    string str = "abc";
    f(str, 0, "", res);
    for(int i=0; i<res.size(); i++){
        cout<<res[i]<<" ";
    }
    return 0;
}
```

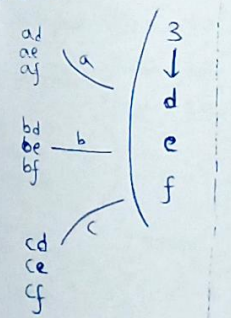
Q) Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order?

i/p: digits = "23"

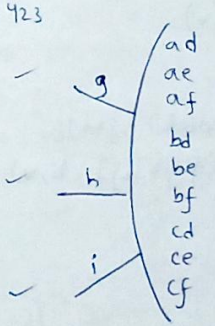
o/p: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"]

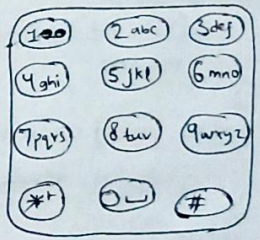
{old phones}

A) ex: 23



ex: 423





Means if we need all comb. of 423, then we make all comb. of 23 & then we attach 4

Similar we used in subsequence concept also. When we need subsequence of "abc", then first we make subsequence of bc & then we → attach 'a' to it. → don't attach 'a' to it. } by keeping result variable

f(str, i, result, li) = f(str, i+1, result + comb[str[i]], li)

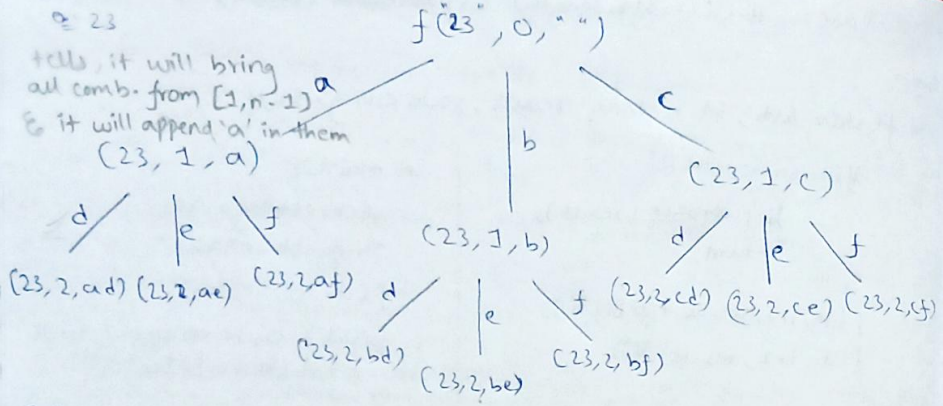
↳ creates all string combination

[i, n-1]

dry run:

n = 23

tells, it will bring all comb. from [1, n-1] & it will append 'a' in them



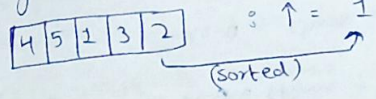
Code:

```
void f(string &str, int i, String result, vector<string> &li, vector<string> &v){
    if( i== str.size()){
        li.push_back(result);
        return;
    }
    int digit = str[i] - '0';
    if(digit <= 1){
        f(str, i+1, result, li, v);
        return;
    }
    for(int j=0; j < v[digit].size(); j++){
        f(str, i+1, result+v[digit][j], li, v);
    }
    return;
}
int main(){
    vector<string> v(10);
    v={"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
    string str = "23";
    vector<string> li;
    f(str, 0, "", li, v);
    for(int i=0; i<li.size(); i++){
        cout<<li[i]<<" ";
    }
    return 0;
}
```

Dry run: ad ae af bd be bf cd ce cf

BUBBLE SORT ALGORITHM | OPTIMIZED BUBBLE SORT

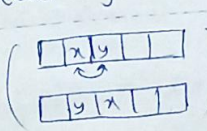
Sorting means ordering of elements in ↑ (or) ↓ order.

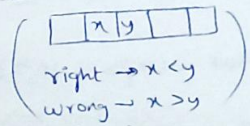
ex: 

Bubble Sort Algorithm:

"Repeatedly swap two adjacent elements if in wrong order"

(until the array is sorted)





(↑ bubbles coming from bottom to top in water)

ex: n=5

0th pass: ① 20 50 40 10 30 (=4)
② 20 40 50 10 30
③ 20 40 10 50 30
④ 20 40 10 30 50
{4 comparisons}

1st pass: ⑤ 20, 40, 10, 30, 50
⑥ 20, 10, 40, 30, 50
⑦ 20, 10, 30, 40, 50
{3 comparisons}
(we didn't compare last element)

2nd pass: ⑧ 20 10 30 40 50
⑨ 10 20 30 40 50
{2 comparisons}
(we didn't compare last 2 elements)

3rd pass: ⑩ 10 20 30 40 50
(we didn't compare last 3 elements)

∴ n' element = n-1 passes

ex: 0th pass = n-1 elements at correct positions

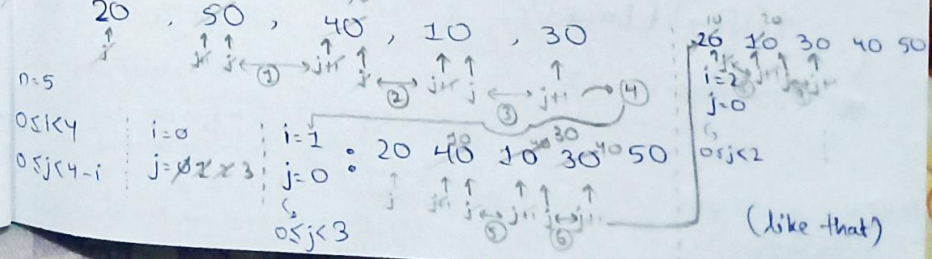
(means last one element also at crt. posn.) → n elements in correct position

(∴ In iᵗʰ pass, we will not compare last i elements)

Code: (array size)

```
for(int i=0; i<n-1; i++){
    for(int j=0; j<n-1-i; j++){
        if(array[j] > array[j+1]){
            swap(array[j], array[j+1]);
        }
    }
}
```

Dry run:



Max. no. of swaps in worst case: (when array in descending order)

1st : 50 40 30 20 10 ; 4 swaps = (n-1) = (5-1)

2nd : 40 30 20 10 50 ; 3 swaps = (n-2)

3rd : 30 20 10 40 50 ; 2 swaps = (n-3)

4th : 20 10 30 40 50 ; 1 swap

10 20 30 40 50

∴ Max swaps = $(n-1)+(n-2)+\cdots+1 = \frac{n(n-1)}{2}$

∴ Time Complexity = $O(n^2)$

∴ Space Complexity = $O(1)$ (since we are re-arranging in same array)

Optimized bubble sort:

ex: 10 20 40 30 50

(1st pass) 10 20 30 40 50 ✓ (Here itself code is sorted)

(2nd pass) → no swap done → so all elements are in right order, so now we can break here itself without going to any further passes

code:

```
for(int i = 0; i < n-1; i++){
    bool flag = false;
    for(int j = 0; j < n-i-1; j++){
        if (array[j] > array[j+1]){
            flag = true;
            swap(array[j], array[j+1]);
        }
    }
    if(!flag) {break;}
}
```

(not false = true)

Stable & Unstable sort: (for identical values)

(does not disturb the order of elements with same value)

ex: 40 30 10 20 30*

Stable: → 10 20 30 30* 40

Unstable: → 10 20 30* 30 40

∴ Bubble sort is a stable sort

SELECTION SORT ALGORITHM

"Repeatedly find min. element in unsorted array & place it at beginning"

(until array is sorted)

ex: 5 8 4 9 2 (swap 5 & 2)

0→ 2 | 8 4 9 5 (2 sorted | unsorted)

1→ 2 4 | 8 9 5 (2 4 sorted | unsorted)

2→ 2 4 5 | 9 8 (2 4 5 sorted | unsorted)

3→ 2 4 5 8 | 9 (Now (n-1) elements are sorted, so obviously 'n' elements got sorted)

Code:

```
for(int i=0; i<n-1; i++){
    // finding min. element in unsorted array
    int min_idx = i;
    for(int j=i+1; j<n; j++){
        if(arr[j] < arr[min_idx]){
            min_idx = j;
        }
    }
    // placing min element at beginning
    if(min_idx != i){
        swap(arr[i], arr[min_idx]);
    }
}
```

for selection sort:

(i^th iter. — selecting element for i^th position)

[0 .... i-1] (sorted) | [i .... n-1] (unsorted)

dry run:

ex: 5 8 4 9 (min_idx) 5

n=4

0≤i<3

i+1≤j<4

i=0, j=1 → swapping '4' & '5' → min_idx = 2

ex: 4 8 5 9

i = 1

j = 2 → min_idx = 5

{min_idx}_i = 8

4 5 8 9

i = 2

j = 3 → sorted

(*Similarly we can do by finding max element & keeping it at (n-i-1)^th position)

For complexity:

i=0, j=1 to n-1 = n-1

i=1, j=2 to n-2 = n-2

...

i=n-2, j=n-1 to n-1 = 1

{n-1+n-2+...+1}

$T_c = O(n^2)$

$= \Omega(n^2)$ in best case ← (as it is comparing min. ele. even when it's sorted)

Space complexity $S_c = O(1)$

Now lets see if this sorting is stable / unstable?

ex: 3 4 3* 2
2 4 3* 3
2 3* 4 3
2 3* 3 4

∴ Selection sort is unstable

- Max sways in selection sort = O(N) { in bubble sort it was O(N²) }

INSERTION SORT ALGORITHM

"Repeatedly take elements from unsorted array & insert in sorted subarray" (correct position)

(until array sorted)

(* Here consider take first element as sorted & rest unsorted ; Now code!)

ex: 11 8 15 9 4 (11 is shifted to next position & 8 is placed in left out position)
(sorted) ← → (unsorted)

8 11 15 9 4 (11, 15 shifted 1 position right & 9 is shifted to left out position)
(sorted) ← → (unsorted)

8 11 15 9 4
(sorted) ← → (unsorted)

8 9 11 15 4
(sorted) ← → (unsorted)

4 8 9 11 15
(sorted)

Code.

```
for(int i=1; i<n ; i++){
    int current = arr[i];
    int j = i-1 ;
    while (j>=0 && arr[j]>current){
        arr[j+1] = arr[j];   // shifting elements
        j--;                 // comparing curr. with prev. element
    }
    arr[j+1] = current;      // inserting current element
}
```

Complexity Analysis

ex: 5 4 3 2 1 # 0 to -1 → 1
4 5 3 2 1 # 1 to -1 → 2
2 3 4 5 1 : n-2 to -1 → n-1

1+2+...+n-1 = n(n-1)/2

∴ Time Complexity = O(n²)
Space Complexity = O(1)

∴ Insertion sort is a stable sorting algorithm.

PROBLEM SOLVING ON SORTING ALGORITHMS - 1

Q) Given an integer array arr, move all 0's to the end of it while maintaining the relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

ip: 0 5 0 3 42
op: 5 3 42 0 0

A) Vid soln.

```
void swapZeroes(vector<int> &v){
    int n = v.size();
    for(int i=n-1; i>=0; i--){
        int j=0;
        bool flag = false;
        while(j!=i){
            if(v[j]==0 && v[j+1]!=0){
                swap(v[j], v[j+1]);
                flag = true;
            }
            j++;
        }
        if(!flag) break;
    }
    return;
}

int main(){
    int n;
    cin>>n;
    vector<int> v(n);
    for(int i=0; i<n; i++){
        cin>>v[i];
    }
    swapZeroes(v);
    for(int i=0; i<n; i++){
        cout<<v[i]<<" ";
    }
    return 0;
}
```

Space Complexity = O(1)
(∵ we are making changes in the same array)

Time Complexity = O(n²)

Explanation

i →						
3	0	2	0	1	5	(n-1)
(0*)						
3	0	2	1	5	0	(n-2)
↑ j						

Sum (1 to n-1) = n(n-1)/2 ~ n²

A) My soln.

```
int main(){
    int n;
    cin>>n;
    int arr[n];
    for(int i=0; i<n; i++){
        cin>>arr[i];
    }
    int K=0;
    for(int i=0; i<n; i++){
        if(arr[i]==0){
            K++;
        }
    }
    for(int i=0; i<K; i++){
        for(int j=i; j<n-i-1; j++){
            if(arr[j]==0 && arr[j+1]!=0){
                swap(arr[j], arr[j+1]);
            }
        }
    }
    for(int i=0; i<n; i++){
        cout<<arr[i]<<" ";
    }
    return 0;
}
```

Q) Given an array of names of fruits, you are supposed to sort it in lexicographical order using the selection sort.

Ip: ["papaya", "lime", "watermelon", "apple", "mango", "kiwi"]

Op: ["apple", "kiwi", "lime", "mango", "papaya", "watermelon"]

A) strcmp(arr1, arr2) = 0 → equal
> 0 → arr1 > arr2
< 0 → arr1 < arr2
(string compare)

strcpy → (copy)

#include<string>

Code:

```
void selectionSort(char fruit[][20], int n){
    for(int i=0; i<n-1; i++){
        int min_idx = i;
        for(int j=i+1; j<n; j++){
            if(strcmp(fruit[min_idx], fruit[j]) > 0){   → comparison & replacement
                min_idx = j;
            }
        }
        if(i != min_idx){
            swap(fruit[i], fruit[min_idx]);
        }
    }
    return;
}

int main(){
    char fruit[][20] = {"papaya", "lime", "watermelon", "apple", "mango", "kiwi"};
    int n = sizeof(fruit)/sizeof(fruit[0]); //no. of rows
    selectionSort(fruit, n);
    for(int i=0; i<n; i++){
        cout << fruit[i] << " ";
    }
    return 0;
}
```

(dynamic array/input)

this depends on no. of input words, that's why in next line we found it's size(no. of inputs)

no. of columns

20 char array

printing

no. of columns = 0 1 2 3 4 5 6 7 8 9 10

papaya\0
lime\0
watermelon\0
apple\0
mango\0
kiwi\0

(here we have taken 20, but we can also take 11 as min. no. & get ans.)

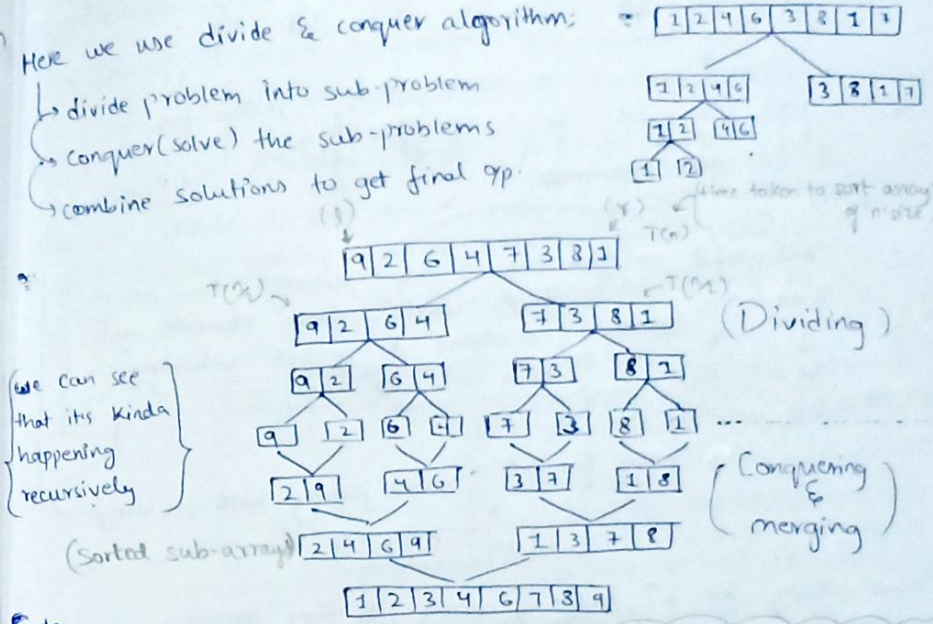
apple kiwi lime mango papaya watermelon

(We can use strings also here, but we created a character array of dynamic size.)

MERGE SORT ALGORITHM

Here we use divide & conquer algorithm:

- divide problem into sub-problem
- conquer (solve) the sub-problems
- combine solutions to get final op.



Code:

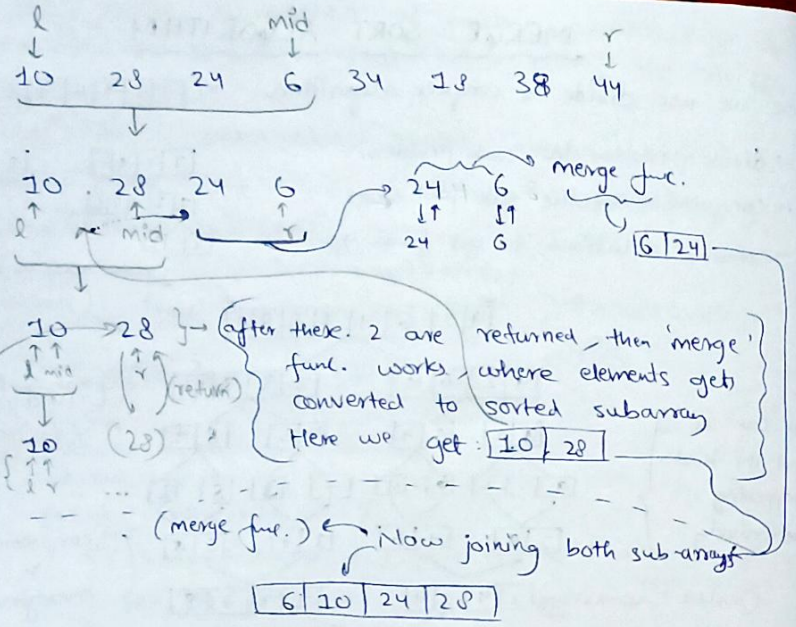
```
void merge(int arr[], int l, int mid, int r){
    int an = mid - l + 1;
    int bn = r - mid;
    int a[an], b[bn]; //created 2 temp. arrays
    for(int i=0; i<an; i++){
        a[i] = arr[l+i];
    }
    for(int j=0; j<bn; j++){
        b[j] = arr[mid+1+j];
    }
    int i=0, j=0; //initial index of array a & b
    int k=l; //initial index of merged subarray
    while(i<an && j<bn){
        if(a[i] < b[j]){arr[k++] = a[i++];}
        else{
            arr[k++] = b[j++];
        }
    while(i<an){
            arr[k++] = a[i++];
    }
    while(j<bn){
            arr[k++] = b[j++];
    }
}
```

```
void mergeSort(int arr[], int l, int r){
    if(l>=r){ //base case
        return;
    }
    int mid = (l+r)/2;
    mergeSort(arr, l, mid);
    mergeSort(arr, mid+1, r);
    merge(arr, l, mid, r);
}

int main(){
    int arr[] = {10,28,24,6,34,18,38,44};
    int n = sizeof(arr)/sizeof(arr[0]);
    mergeSort(arr, 0, n-1);
    for(int i=0; i<n; i++){
        cout<<arr[i]<<" ";
    }
    return 0;
}
```

Op: 6 10 18 24 28 34 38 44

dry run:



Same thing for 2nd half of array & then merging

Complexity Analysis:

T(n) = time taken to sort array of 'n' size

T(n/2) = " " " " " of 'n/2' size.

$\therefore T(n) = T(n/2) + T(n/2) + n$ — ①

$= 2T(n/2) + n$

(time taken to merge) → (as we will traverse through each element)

Master's Theorem:

$T(n) = aT(n/b) + \Theta(n^k \log^p n)$

$(a \geq 1, b > 1, k \geq 0, p = \text{real no.})$

if $\log_b^a > k \Rightarrow T(n) = \Theta(n^{\log_b^a})$

if $\log_b^a = k \Rightarrow$ if $p > -1 \rightarrow \Theta(n^k \log^{p+1} n)$

if $p = -1 \rightarrow \Theta(n^k \log\log n)$

if $p < -1 \rightarrow \Theta(n^k)$

if $\log_b^a < k \Rightarrow$ if $p \geq 0 \rightarrow \Theta(n^k \log^p n)$

if $p < 0 \rightarrow \Theta(n^k)$

From eqn ①

$a = 2, b = 2, k = 1, p = 0$

$\log_b^a = 1 = k$

$\rightarrow \Theta(n^1 \log^1 n)$

$\therefore$ • $T_c = O(n \log n)$ • Merge, temp arrays → N

• $S_c = O(n)$ • Recursive stack → max(log n) calls.

• Merge sort is stable

• Applications: → large data sets ✓ → linked lists

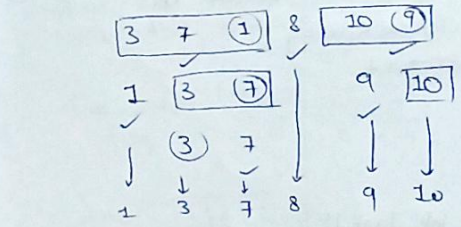
Drawbacks: → slower for smaller tasks → O(n) extra space

→ goes through whole process even if array is sorted.

QUICK SORT ALGORITHM

Here we choose a pivot element – put at correct place – change pivot element around it – put them in correct place.

Ex: 10 3 7 9 1 (8) → let = pivot element



pseudocode:

```
quicksort(arr, first, last){
if(first>=last) return;
pi = partition(arr, first, last);
quicksort(arr, first, pi-1);
quicksort(arr, pi+1, last)
}
```

Choosing the pivot element: 1) last element → (here we will be taking this one)

(diff. types → diff. complexities) 2) first element

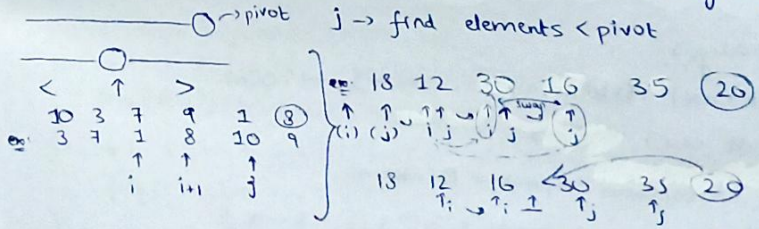
3) Median

4) Random element

Partition Algorithm:

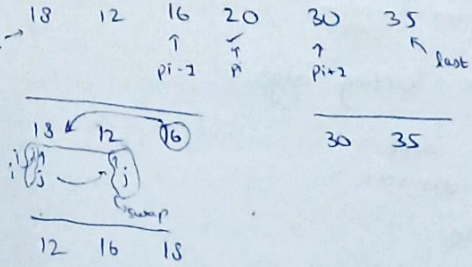
i → insert element at the beginning < pivot

j → find elements < pivot



→ When ele. at j < pivot → do i++ & swap elements (i, j) else j++

→ Now we swap ele. at (i+1)th position & pivot



Code:

```
int partition(int arr[], int first, int last){
    int pivot = arr[last];
    int i = first-1; //for inserting elements < pivot
    int j = first; // for finding elements < pivot
    for(; j<last; j++){
        if(arr[j] < pivot){
            i++;
            swap(arr[i], arr[j]);
        }
    }
    //now 'i' is pointing to the last element < pivot
    //correct position for pivot will be -> i+1
    swap(arr[i+1], arr[last]);
    return i+1;
}

void quickSort(int arr[], int first, int last){
    if(first >= last){ // base case
        return;
    }
    int pi = partition(arr, first, last);
    quickSort(arr, first, pi-1); //recursive case
    quickSort(arr, pi+1, last);
}
```

```
int main(){
    int arr[] = {20,12,35,16,18,30};
    int n = sizeof(arr)/sizeof(arr[0]);

    quickSort(arr, 0, n-1);

    for(int i=0; i<n; i++){
        cout<<arr[i]<<" ";
    }
    return 0;
}
```

Complexity Analysis:

- T_c → (best case) : $\Theta(n \log n)$ — S_c → $O(n)$
  - → (worst case) : $O(n^2)$
- Other ways → Median → pivot ⇒ $\Theta(n \log n)$
  - → Randomised pivot : ex. 1 (2) 3 4 5
    - pivot = first + rand()%(last-first+1);
    - (chooses random value in C++ → random func.)
- Quick Sort is not a stable sorting algo.

Merge Sort	vs	Quick Sort
→ large datasets		→ smaller datasets
→ stable		→ unstable
→ linked list		→ new memory is costly

COUNT SORT ALGORITHM | RADIX SORT ALGORITHM

COUNT SORT

ex: array: | 5 | 2 | 3 | 2 | 1 |

freq./count: | 0 | 1 | 2 | 1 | 0 | 1 | (indices 0 1 2 3 4 5)

(means '1' present until i<1) — (2 present until index <3)

cumulative freq: | 0 | 1 | 3 | 4 | 4 | 5 | (indices 0 1 2 3 4 5)

Now we traverse array from last & we keep decrementing cumulative freq. by '1' and place it in ans array.

Cumulative freq: | 0 | 1 | 3 | 4 | 4 | 5 | (indices 0 1 2 3 4 5)

Ans: | 1 | 2 | 2 | 3 | 5 | (indices 0 1 2 3 4)

∴ Here we are doing 4 things:
- finding max element
- make frequency array
- calculate cumulative freq.
- ans array

Codes:

```
void countSort(vector<int> &v){
    int n = v.size();
    int max_ele = INT_MIN; //finding the max element
    for(int i=0; i<n; i++){
        max_ele = max(v[i], max_ele);
    }                (size)      (initialising value)
    vector<int> freq(max_ele+1, 0); //creating the freq. array
    for(int i=0; i<n; i++){
        freq[v[i]]++;
    }
    for(int i=1; i<=max_ele; i++){ //calculating cumulative freq
        freq[i] += freq[i-1];
    }
                (vector array of size 'n')
    vector<int> ans(n); //calculate the sorted array
    for(int i=n-1; i>=0; i--){
        ans[--freq[v[i]]] = v[i];
    }

    for(int i=0; i<n; i++){ //copy back the ans
        v[i] = ans[i];      // to original array
    }
}
```

```
int main(){
    int n;
    cin>>n;
    vector<int> a(n);
    for(int i=0; i<n; i++){
        cin>>a[i];
    }
    countSort(a);
    for(int i=0; i<n; i++){
        cout<<a[i]<<" ";
    }
    return 0;
}
```

- $T_c \equiv O(n+max)$ → we use count sort when $n >> max$.
  ↳ $\therefore T_c = O(n)$
  ↳ $3n + max$ → calculate cumulative freq.
  find max ele. | create freq array | calculate sorted array

(we can't use count sort when input has floating no.'s)

- $S_c \equiv O(n+k)$
  (sorted array size) (freq. array size)

(count sort is generally stable algo.)

- for (-)ve no.'s → we find the min. no. & subtract it.
  ↳ In end ans array, we add the min no. again

eg: 5 4 -3 2 3 -2

-(-3): 8 7 0 5 6 1

(count sort)

0	1	5	6	7	8
0	1	2	3	4	5

+(-3) ⇒

-3	-2	2	3	4	5
0	1	2	3	4	5

- We also don't use count sort when no.'s have high discrepancies → eg: 2, 4500, 679, 49 instead we use here radix sort

RADIX SORT

eg:

	170	45	75	90	802	2
(1st pass)	170	90	802	02	45	75
(2nd pass)	802	002	045	170	075	090
(3rd pass)	2	45	75	90	170	802

Here we compare the 1's digit, then 10's digit, ...

Radix sort is also like count sort, but in count sort → freq[v[i]]++; here we instead do: freq[(arr(i)/position)%10]++.

(since we are calculating frequency of digits)

Code:

```
void countSort(vector<int> &v, int pos){
    int n = v.size();
    vector<int> freq(10,0); //create freq array   (digits → 0 to 9, so we take 10)
    for(int i=0; i<n; i++){
        freq[(v[i]/pos)%10]++;
    }
    for(int i=1; i<10; i++){ //cumulative freq
        freq[i] += freq[i-1];
    }
    vector<int> ans(n); //ans array
    for(int i=n-1; i>=0; i--){
        ans[--freq[(v[i]/pos)%10]] = v[i];
    }
    for(int i=0; i<n; i++){
        v[i] = ans[i];
    }
}
```

```
void radixSort(vector<int> &v){
    int max_ele = INT_MIN;
    for(auto x:v){ (automatically detects type of variable)
        max_ele = max(x, max_ele);
    }
    for(int pos=1; max_ele/pos>0; pos*=10){
        countSort(v, pos);
    }
}
int main(){
    int n;
    cin>>n;
    vector<int> v(n);
    for(int i=0; i<n; i++){
        cin>>v[i];
    }
    radixSort(v);
    for(int i=0; i<n; i++){
        cout<<v[i]<<" ";
    }
    return 0;
}
```

dry run:
max_ele = 802
pos = 1

freq. | 2 | 0 | 2 | 0 | 0 | 2 | 0 | 0 | 0 | 0 |
(0 1 2 3 4 5 6 7 8 9)

Cum. freq. | 2 | 2 | 4 | 4 | 4 | 6 | 6 | 6 | 6 | 6 |
(0 1 2 3 4 5 6 7 8 9)

ans.

170	90	802	2	45	75
0	1	2	3	4	5

170 45 75 90 802 2

pos = 10 : freq. | 2 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 1 | 0 |
(0 1 2 3 4 5 6 7 8 9)

cf | 2 | 2 | 2 | 3 | 3 | 3 | 5 | 5 | 6 |
(0 1 2 3 4 5 6 7 8 9)

170, 90, 802, 2, 45, 75

ans

802	2	45	170	75	90
0	1	2	3	4	5

pos = 100 : freq. : | 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
(0 1 2 3 4 5 6 7 8 9)

cf : | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 6 | 6 |
(0 1 2 3 4 5 6 7 8 9)

802, 2, 45, 170, 75, 90

ans

2	45	75	90	170	802
0	1	2	3	4	5

- Time complexity = $O(d * n)$
  (d: no. of digits, n: sorted array size)

- $S_c \equiv O(n)$

BUCKET SORT ALGORITHM

ex: 0.13, 0.45, 0.12, 0.89, 0.75, 0.63, 0.85, 0.39

Here we create buckets:

index = arr[i] * (size of array)

(Here → (0.13) × 8 → [0-8))

0	→ 0.12
1	→ 0.13
2	
3	→ 0.39 → 0.45
4	
5	→ 0.63
6	→ 0.75 → 0.85
7	→ 0.89

(bucket)

we then internally arrange this.

Here as we can see, a 2D array is created ✓
↳ (buckets, internal array)

Above is called → scatter & gather approach.

Here we have 4 steps: 1) create buckets of size 'n'
2) insert elements into bucket → n {index = arr[i]*n}
3) sort individual buckets
4) combine all elements

- T_c → worst: $O(n^2)$
  - ↳ avg.: $O(n+k)$ (internal array of elements)

Code:

```
void bucketSort(float arr[], int size){ (initialising, column size not defined)
    vector<vector<float>> bucket(size, vector<float>()); // step 1
    for(int i=0; i<size; i++){ //step 2: inserting elements into bucket
        int index = arr[i]*size;
        bucket[index].push_back(arr[i]);
    }
    //step 3: sorting individual buckets
    for(int i=0; i<size; i++){
        if(!bucket[i].empty()){
            sort(bucket[i].begin(), bucket[i].end());
        }
    }
    //step 4: combining elements from bucket
    int k=0;
    for(int i=0; i<size; i++){
        for(int j=0; j<bucket[i].size(); j++){
            arr[k++] = bucket[i][j];
        }
    }
}
```

```
int main(){
    float arr[] = {0.13, 0.45, 0.12, 0.89, 0.75};
    int size = sizeof(arr)/sizeof(arr[0]);
    bucketSort(arr, size);

    for(int i=0; i<size; i++){
        cout << arr[i] << " ";
    }
    return 0;
}
```

Now if we have, elements > 1.

ex: 6.13, 8.45, 0.12, 1.89, 4.75, 2.63, 7.85, 10.39

In order to arrange these elements into buckets, we find index in terms of range →
- range = (max - min)/size
  (Here = (10.39 - 0.12)/8 = 1.28)
- index = (arr[i] - min)/range
  = (arr[i] - min) * size / (max - min)

- T_c → Avg.: $O(n+k)$
  - → Best: $O(n^2)$ (no. of elements inside a bucket)
- S_c → $O(n+k)$

Code:

```
void bucketSort(float arr[], int size){
    vector<vector<float>> bucket(size, vector<float>()); //step 1
    //finding range
    float max_ele = arr[0];
    float min_ele = arr[0];
    for(int i=1; i<size; i++){
        max_ele = max(max_ele, arr[i]);
        min_ele = min(min_ele, arr[i]);
    }
    float range = (max_ele - min_ele)/size;
    //step 2: inserting elements into bucket
    for(int i=0; i<size; i++){
        int index = (arr[i] - min_ele)/range;
        float diff = (arr[i] - min_ele)/range - index; //boundary elements
        if(diff==0 && arr[i] != min_ele){
            bucket[index-1].push_back(arr[i]);
        }
        else{
            bucket[index].push_back(arr[i]);
        }
    }
    //step 3: sorting individual buckets
    for(int i=0; i<size; i++){
        if(!bucket[i].empty()){
            sort(bucket[i].begin(), bucket[i].end());
        }
    }
    //step 4: combining elements from buckets
    int k=0;
    for(int i=0; i<size; i++){
        for(int j=0; j<bucket[i].size(); j++){
            arr[k++] = bucket[i][j];
        }
    }
}
```

```
int main(){
    float arr[] = {6.13, 8.45, 0.12, 1.89, 10.29};
    int size = sizeof(arr)/sizeof(arr[0]);
    bucketSort(arr, size);
    for(int i=0; i<size; i++){
        cout<<arr[i]<<" ";
    }
    return 0;
}
```

floor → inbuilt func. in C++ that returns integer value ie. <= the argument

PROBLEM SOLVING ON SORTING ALGORITHMS - 2

Q) Given an array where all its elements are sorted in increasing order except 2 swapped elements, sort it in linear time. Assume there are no duplicates in the array. ex: ip: A[] = {3, 8, 6, 7, 5, 9, 10}
op: A[] = {3, 5, 6, 7, 8, 9, 10}

A) 3, 8, 6, 7, 5, 9, 10 ⇝ 3, 8, 6, 7, 5, 9, 10 (x, y, y)

(we can see prev. ele. greater than next ele.)

(We initialise x = -1, y = -1, to differentiate which distortion we are in.)

{When we see 1st distortion, we can keep higher value as 'x' & next value as 'y'.
When we see 2nd distortion, we can update value of 'y' & swap 'x' & 'y'}

Code:

```
void SortArr(int arr[], int n){
    int x=-1, y=-1;
    if(n<=1){ return ; } //edge cases
    for(int i=1; i<n; i++){ //process all adj. ele.
        if(arr[i-1]>arr[i]){
            if(x==-1){ //first conflict
                x=i-1;
                y=i;
            }
            else{ //second conflict
                y=i;
            }
        }
    }
    swap(arr[x], arr[y]);
}
```

```
int main(){
    int arr[]={3,8,6,7,5,9,10};
    int n = sizeof(arr)/sizeof(arr[0]);
    sortArr(arr,n);
    for(int i:arr){
        cout<<i<<" ";
    }
    return 0;
}
```

op: 3 5 6 7 8 9 10

Here we are traversing array only once, so we have linear T_c

Q) Given an array of (+)ve & (-)ve integers, segregate them in linear time & const. space. The op should print all (-)ve no.'s, followed by all (+)ve no.'s.

ip: [19, -20, 7, -4, -13, 11, -5, 3]

op: [-20, -4, -13, -5, 7, 11, 19, 3]

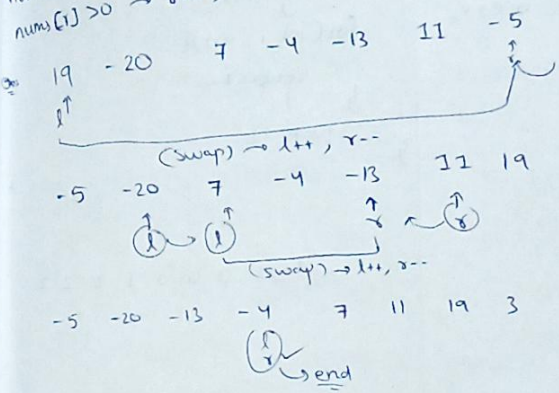
A) We can see that the o/p's (+) & (-) ve no.'s need not be in order.

Here we can use quick sort ⇝ partition method → so that we can seperate.

If we take '0' as our pivot, then → (-)ve on left side, (+)ve on right side

nums[l] < 0 → l++;
nums[r] > 0 → r--;

{0 → imaginary pivot}



Code:

```
void partition(int arr[], int n){
    int l=0, r=n-1;
    while(l<r){
        while(arr[l]<0){
            l++;
        }
        while(arr[r]>0){
            r--;
        }
        if(l<r){
            swap(arr[l], arr[r]);
        }
    }
}
```

```
int main(){
    int arr[]={19,-20,7,-4,-13,11,-5,3};
    int n = sizeof(arr)/sizeof(arr[0]);
    partition(arr,n);
    for(int i:arr){
        cout<<i<<" ";
    }
    return 0;
}
```

op: -5 -20 -13 -4 7 11 19 3

Q) Given an array of size 'N' containing only 0's, 1's & 2's. Sort the array in ascending order.

ip: n=6
arr[] = {0 2 1 2 0 0}
op: 0 0 0 1 2 2

A) Method-1: We can use quick sort/merge sort → $T_c = O(n \log n)$

Method-2: Using counting no. of 0's, 1's, 2's & placing it in array. → $T_c = O(2n) \to O(n)$ (traversing twice)

Method-3: Game of replacements → $T_c = O(n)$ (traversing only once)

Method - 2 soln:

```
void sorting(int arr[], int n){
    int zero=0, one=0, two=0;
    for(int i=0 ; i<n ; i++){
        if(arr[i]==0)  zero++;
        else if(arr[i]==1)  one++;
        else two++;
    }
    int i=0;
    while(zero>0){
        arr[i++]=0;
        zero--;
    }
    while(one>0){
        arr[i++]=1;
        one--;
    }
    while(two>0){
        arr[i++]=2;
        two--;
    }
}
```

```
int main(){
    int arr[]={0,2,1,2,0,2,1};
    int n=sizeof(arr)/sizeof(arr[0]);
    sorting(arr, n);
    for(int i: arr){
        cout<<i<<" ";
    }
    return 0;
}
```

↳ op: 0 0 1 1 2 2 2

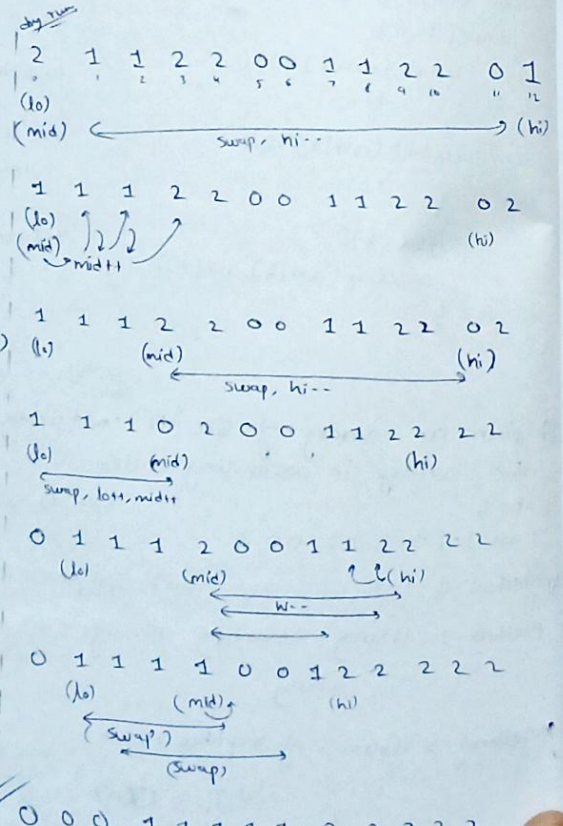
Method - 3 soln: (BEST)

ex Here we have 4 regions:
↳ [0, low-1] → 0
[low, mid-1] → 1
[mid, hi] → unsorted
[hi+1, n-1] → 2

a[mid]=0	a[mid]=1	a[mid]=2

swap(mid, lo)
lo++;
mid++;
	mid++	swap(mid, hi)
		hi--;

Here we are constantly swapping 0's, 1's, 2's as we traverse through array



Code:

```
void sorting(int arr[], int n){
    int low=0, mid=0, high=n-1;
    while(mid<=high){
        if(arr[mid]==0){
            swap(arr[mid], arr[low]);
            mid++;
            low++;
        }
        else if(arr[mid]==1){
            mid++;
        }
        else{
            swap(arr[mid], arr[hi]);
            hi--;
        }
    }
}
```

```
int main(){
    int arr[]={2,1,1,2,2,0,0,1,1,2,2,0,1};
    int n=sizeof(arr)/sizeof(arr[0]);
    sorting(arr,n);
    for(int i: arr){
        cout<<i<<" ";
    }
    return 0;
}
```

PROBLEM SOLVING ON SORTING ALGORITHMS - 3

5) Write a program to find k^{th} smallest element in an array using Quick sort

ip: enter the elements of array
3 5 2 1 4 7 8 6
enter value of k
5

op: k^{th} smallest element is 5

A) ex: 3 5 2 1 4 7 8 6 (position) pivot ; ans. is at $(k-1)^{th}$ index

[1 2 3 4 5] 6 [7 8]
<6 >6

if 'pos' = k-1 → ✓

if pos > k-1 → pivot ele. lies ahead of $(k-1)^{th}$ pos
↳ so we call func. recursively from (l, pos-1, k)

if pos < k-1
(pivot ele. lies behind $(k-1)^{th}$ ele.)
↳ so we call func. recursively from (pos+1, r, k)

ex: K=7 ↳ 6^{th} index
↳ [3 5 2 1 4] 6 [7 8]
pos=5 l r

k = k - (pos - l + 1)
= 7 - 5 + 0 - 1
= 1

(means we need first ele. in the sub-array)

Code:

```
int partition(int arr[], int l, int r){
    int pivot = arr[r];
    int i = l;
    for(int j = l; j<r; j++){
        if(arr[j] <pivot){
            swap(arr[i], arr[j]);
            i++;
        }
    }
    swap(arr[i], arr[r]);
    return i;
}

int main(){
    int arr[] = {3,5,2,1,4,7,8,6};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 7;
    cout<< kthsmall(arr,0,n-1,k);
    return 0;
}
```

```
int kthsmall(int arr[], int l, int r, int k){
    if(k>0 && k<=r-l+1){
        //position of pivot elt
        int pos = partition(arr, l, r);

        if(pos-l == k-1){
            return arr[pos];
        }
        else if(pos-l > k-1){
            return kthsmall(arr, l, pos-1, k);
        }
        else{
            return kthsmall(arr, pos+1, r, k-pos+l-1);
        }
    }
    return -1;
}
```

↳ op 7

dry run.

3 5 2 1 4 7 8 6

i j (if ✓) — i j — i j — i j — j — Piv

i (swap) piv

= 3 5 2 1 4 6 8 7 (pos = 5)

pos - l = 5-0 } k-1 > pos-l

k - 1 = 6

G (8 7)

l r , k = 7 - 5 + 0 - 1 = 1

we find position of partition → last swap → [7, 8]

return i = 0

Now pos-l > k-1 → 0-0 = 0

1-1 = 0 ∴ arr[0] = 7

∴ T_c → Worst Case = $O(n^2)$

→ Avg Case = $O(n)$

Q) Given 2 sorted arrays, write a program to merge them in a sorted manner.

eg: num1[] = {5, 8, 10} , num2[] = {2, 7, 8} ; o/p num3[] = {2, 5, 7, 8, 8, 10}

A Basically similar to merge sort

Code:

```
void merge(int arr1[], int n1, int arr2[], int n2, int arr3[]){
    int a=0, b=0, c=0;
    while(a<n1 && b<n2){
        if(arr1[a] < arr2[b]){
            arr3[c++] = arr1[a++];
        }
        else{
            arr3[c++] = arr2[b++];
        }
    }
    while(a<n1){
        arr3[c++] = arr1[a++];
    }
    while(a<n2){
        arr3[c++] = arr[b++];
    }
}
```

```
int main(){
    int arr1[] = {5, 8, 10, 11, 12};
    int arr2[] = {2, 7, 8};
    int n1 = sizeof(arr1)/sizeof(arr1[0]);
    int n2 = sizeof(arr2)/sizeof(arr2[0]);
    int n3 = n1+n2;
    int arr3[n3];
    merge(arr1, n1, arr2, n2, arr3);
    for(int i=0; i<n3; i++){
        cout<<arr3[i]<<" ";
    }
    return 0;
}
```

↳ o/p 2 5 7 8 8 10 11 12

T_c → $O(n_1+n_2)$

BINARY SEARCH ALGORITHM

Linear Search: we go to every element & match with target

eg: 1, 6, 9, 3, -1, 8, 2 ← (searchspace)

target = 8

size of searches: (N) → (N-1) → (N-2) - - 3 → 2 → 1

∴ $T_c = O(n)$

Code:

```
for(int i=0; i<n; i++){
    if(arr[i] == target){
        return i;
    }
}
return false;
```

This search is a brute-force search.

Binary Search: here we divide our (sorted) array equally into 2 parts.

1	9	13	15	18	22	35	40

(lo) (mid) (high)

target = 13

→ must there in right side of mid

So we do → high = mid - 1

(& further divide array if needed)

Iterative code:

```
int binarySearch(vector<int> &v, int target){
    //define search space
    int lo = 0; //start of search space
    int hi = v.size()-1; //end of search space
    while(lo<=hi){
        //calc. midpt. for search space
        int mid = (lo+hi)/2;
        if(v[mid]==target) return mid;
        else if(v[mid]<target){
            //discarding left of mid
            lo = mid+1;
        }
        else{
            //discarding right of mid
            hi = mid-1;
        }
    }
    return -1;
}
```

```
int main(){
    int n;
    cin>>n;
    vector<int> v;
    for(int i=0; i<n; i++){
        int x;
        cin>>x;
        v.push_back(x);
    }
    int target;
    cin>>target;
    cout<<binarySearch(v,target);
    return 0;
}
```

instead of declaring vector i/p, we can also write it as:
`vector<int> v {2,4,6,9,12,30,70};`

Complexity Analysis

$N \to \frac{N}{2} \to \frac{N}{4} \to \cdots \to 1$

$\frac{N}{2^0} \to \frac{N}{2^1} \to \frac{N}{2^2} \to \cdots \to \frac{N}{2^k}$

$\therefore \frac{N}{2^k} = 1 \Rightarrow \log_2 N = \log_2 2^k$

$\therefore$ Total iterations $= K = \log N$

$\therefore T_c = O(\log N)$

$\therefore S_c = O(1)$

Applying recursively: $f(\text{input}, \text{target}, lo, hi) =$ mid = (lo+hi)/2; if(v[mid]==target) return mid; else if(v[mid]<target) $f(v, \text{target}, mid+1, hi)$; else $f(v, \text{target}, lo, mid-1)$;

Recursive code:

```
int binSearRec(vector<int> &v, int target, int lo, int hi){
    if(lo>hi) return -1;
    int mid = (lo+hi)/2;
    if(v[mid]==target) return mid;
    if(v[mid]<target){
        return binSearRec(v, target, mid+1, hi);
    }
    else{
        return binSearRec(v, target, lo, mid-1);
    }
}
```

```
int main(){
    vector<int> v {2,4,6,8,14,29,30};
    int n = v.size();
    int target;
    cin>>target;
    cout<<binSearRec(v,target,0,n);
    return 0;
}
```

- $T_c = O(\log n)$
- $S_c = O(\log N)$

Small implementation flaw in Binary Search:

```
int mid = (lo+hi)/2;
```

if $lo \to INT\_MAX$, $hi \to INT\_MAX$ } Adding both means value might go out of integer range. So it will be overflowed

So we can also write as:

$mid = \frac{lo+hi}{2} = \frac{lo+hi+lo-lo}{2} = \frac{(lo+lo)+(hi-lo)}{2} = \frac{2*lo+(hi-lo)}{2}$

$\therefore mid = lo + \frac{(hi-lo)}{2}$ → { max + very small value (since subtraction) → okay ✓ }

we can replace this in our binarysearch code

Q) Find the first occurrence of a given element x, given that the given array is sorted. If no occurrence is found then return -1.

eg: arr = [2, 5, 5, 5, 6, 6, 8, 9, 9] x = 5 | o/p : 1

Brute force → linear search → `if(arr[i]==x) return i;`

O(n) ↑ Tc | O(1) ↑ Sc

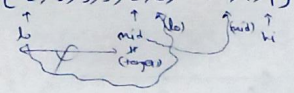
Using binSea:

ex [2, 5, 5, 5, 6, 6, 8, 9, 9, 9] : target = 5

if 'm' ↳ arr[m] = target ↳ index 'm' might be a possible ans

(thus 'm' might be our answer or we can find something better than this by checking its previous element.)

ex: target → 9 : [2, 5, 5, 5, 6, 6, 9, 9, 9, 9]



→ [6, 9, 9, 9, 9]

lo, mid, high → ∴ ans = mid; hi = mid-1;

→ [6, 9] → lo = mid+1;

→ [9] → index returned.

: arr(mid) == target ↳ ans = mid; hi = mid-1;

arr(mid) > target ↳ hi = mid-1;

arr(mid) < target ↳ lo = mid+1;

- $T_c = O(\log n)$
- $S_c = O(1)$

Code:

```
int firstOccurence(vector<int> &v, int target){
    int lo = 0, ans = -1, hi = v.size()-1;
    while(lo<=hi){
        int mid = lo + (hi-lo)/2;
        if(v[mid] == target){
            ans = mid;
            hi = mid-1;
        }
        else if(v[mid] > target) hi = mid-1;
        else lo = mid+1;
    }
    return ans;
}
```

```
int main(){
    vector<int> v{2,5,5,6,6,7,9};
    int n = v.size();
    int target;
    cin>>target;
    cout<<firstOccurence(v,target);
    return 0;
}
```

Q) Find the square root of the given non-ve integer/value x. Round it off to nearest floor integer value.

ip: x=4 | op: 2

ip: x=11 | op: 3

A) $\sqrt{x} \rightarrow [1, x]$ → within this range find the biggest value whose square is less than or equal to 'x'.

ex: x = 40

$\sqrt{40} \rightarrow [1, 40]$

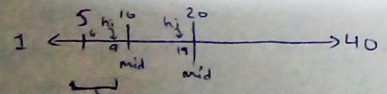
$2^2 \leq 40$
$3^2 \leq 40$
$4^2 \leq 40$
$5^2 \leq 40$
$6^2 \leq 40$ but $7^2 \nleq 40$

```
ans = -1;
for(int i=1; i<=x; i++){
    if(i*i <= x){
        ans = i;
    }
    else{
        break;
    }
}
```

$T_c = O(\sqrt{x})$

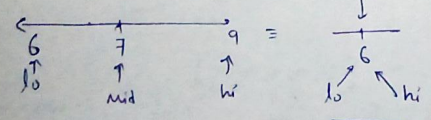
This kind of problems we call → Binary Search on Answer

because we can apply binary search on answers.




```
if(mid*mid <= x){
    ans = mid;
    lo = mid+1;
}
else{
    hi = mid-1;
}
```

(6*5 <= 40)



=

∴ ans = 6

$\therefore T_c = O(\log x)$, $S_c = O(1)$

{We can use direct 'sqrt' func also but here, doing in other way}

Code:

```
int sqrt(int x){
    int lo = 1, hi = x, ans = -1;
    while(lo<=hi){
        int mid = lo + (hi-lo)/2;
        if(mid*mid <= x){
            ans = mid;
            lo = mid+1;
        }else{
            hi = mid-1;
        }
    }
    return ans;
}
```

```
int main(){
    cout<<sqrt(40);
    return 0;
}
```

↳ 6

BINARY SEARCH PROBLEMS - 1

Q) Given an array of integers 'a' that is sorted in non-decreasing order. Find the first & the last position of the given 'target' element in the sorted array. Follow 0-based indexing. If 'target' is not found in the array, return [-1, -1].

ip: Array: [1,2,3,3,3,5,11], target = 3 | op: [2,4]

A) 1) Linear Search: traversing array from starting → first index
" " " ending → last index

$T_c = O(n)$
$S_c = O(1)$

2) Binary Search: Here we divide our prob. into 2 sub-prob. i.e we will first find here → "first index" & then "last index"

first occurence of number → (done previously) ✓

To get last index → a) we can find the index of it's next big no. & then do '-1'.

b) Similar to (first occ.), discarding left side.

ex. [-2,-1,3,4,4,4,5,5,5]

```
if(a[mid] <= target)
    lo = mid+1
if(a[mid] > target)
    hi = mid-1;
    ans = mid;
```

{all ele. right side of mid are > target, so we reduce upper value to get the first ele. > target}

```
if(a[mid] <= target)
    lo = mid+1
    if(a[mid] == target)
        ans = mid;
else
    hi = mid-1;
```

```
int lowerbound(vector<int> &input, int target){
    int lo=0, ans=-1, hi=input.size()-1;
    while(lo<=hi){
        int mid = lo+(hi-lo)/2;
        if(input[mid]>=target){
            ans=mid;
            hi=mid-1;
        }else{
            lo=mid+1;
        }
    }
    return ans;
}
```

```
int upperbound(vector<int> &input, int target){
    int lo=0, ans=-1, hi=input.size()-1;
    while(lo<=hi){
        int mid = lo+(hi-lo)/2;
        if(input[mid]==target){
            ans=mid;
            lo=mid+1;
        }else if(input[mid]>target){
            hi=mid-1;
        }else{
            lo=mid+1;
        }
    }
    return ans;
}
```

```
int upperbound(vector<int> &input, int target){
{
    int lo=0,ans=-1, hi=input.size()-1;
    while(lo<=hi){
        int mid = lo+(hi-lo)/2;
        if(input[mid]>target){
            ans=mid;
            hi=mid-1;
        }else{
            lo=mid+1;
        }
    }
    return ans.
}
```

```
int main(){
    int n;
    cin>>n;
    vector<int> input;
    for(int i=0;i<n;i++){
        int x; cin>>x;
        input.push_back(x);
    }
    int target; cin>>target;
    vector<int> result;
    int lb = lowerbound(input, target);
    if(input[lb]!=target){
        result.push_back(-1); result.push_back(-1);
    }else{
        int ub = upperbound(input, target);
        result.push_back(lb);
        result.push_back(ub-1);
    }
    cout<<result[0]<<" "<<result[1];
    return 0;
}
```

Q) A rotated sorted array is a sorted array on which rotation operation has been performed some no. of times. Given a rotated sorted array, find the index of the min. ele. in the array. Follow 0-based indexing. It's guranteed that all the elements in the array are unique.

ip: [3,4,5,1,2]

op: 3

A) ex: 1,2,3,4,5 → 5,1,2,3,4 → 4,5,1,2,3 (ex. of sorted rotated array)

↓ (sorted asc. array)

if → a[0] < a[n-1] (if a[0] > a[n-1])

Now: ex: 7, 2, 3, 4, 5, 6 (lo, mid, hi)

if → a[lo] > a[hi]

if (a[mid] > a[lo]) lo = mid+1

if (a[mid] < a[lo]) hi = mid-1

(to check if mid is ans) terminating condn.

a[mid] > a[mid+1] → ans

a[mid] < a[mid-1] → ans

Code:

```
int minSortRotated(vector<int> &v){
    if(v.size()==1) return v[0];
    int lo=0, hi=v.size()-1;
    if(v[lo]<v[hi]) return lo; //sorted array
    while(lo<=hi){
        int mid = lo+(hi-lo)/2;
        if(v[mid]>v[mid+1]) return mid+1;
        if(v[mid]<v[mid-1]) return mid;
        if(v[mid]>v[lo]){
            lo=mid+1;
        }else{
            hi=mid-1;
        }
    }
    return -1;
}
```

```
int main(){
    int n;
    cin>>n;
    vector<int> v;
    for(int i=0;i<n;i++){
        int x;
        cin>>x;
        v.push_back(x);
    }
    cout<<minSortRotated(v);
    return 0;
}
```

- $T_c = O(\log n)$, $S_c = O(1)$

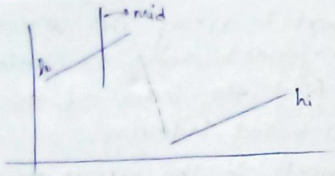
Q) Given the rotated sorted array of integers, which contains distinct elements, & an integer target, return the index of target if it is in the array. Otherwise return -1.

ip: array = [3,4,5,1,2] , target = 4 | o/p = 1

A) eg: (6 7 8) (1) (2 3 4 5) → we can find min. ele & then apply binary search to its left & right arrays.

min. ele.

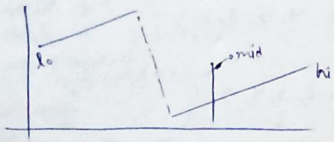
Now:



When: a[lo] > a[hi]

```
if (target >= a[lo] and target <= a[mid])
        hi = mid - 1;
else
        lo = mid + 1;
```

When: a[lo] > a[mid]




```
if (target >= a[mid] and target <= a[hi])
        lo = mid + 1;
else
        hi = mid - 1;
```

- $T_c = O(\log N)$, $S_c = O(1)$

Code:

```
int binSeaSortRotate(vector<int> &v, int target){
    int lo = 0, hi = v.size() - 1;
    while(lo <= hi){
        int mid = lo + (hi - lo)/2;
        if(v[mid] == target) return mid;
        if(v[mid] >= v[lo]){
            if(target >= v[lo] and target <= v[mid]){
                hi = mid - 1;
            }else{
                lo = mid + 1;
            }
        }else{
            if(target >= v[mid] and target <= v[hi]){
                lo = mid + 1;
            }else{
                hi = mid - 1;
            }
        }
    }
    return -1;
}
```

```
int main(){
    int n;
    cin >> n;
    vector<int> v;
    for(int i=0; i<n; i++){
        int x;
        cin >> x;
        v.push_back(x);
    }
    int target;
    cin >> target;
    cout << binSeaSortRotate;
    return 0;
}
```

2) Search element in rotated sorted array with duplicate elements. Return 1 if the element is found else return -1.

ip: [0,0,0,1,1,1,2,0,0,0] , target = 2 op: 1

A) Code

```
int main(){
    int n; cin >> n;
    vector<int> v;
    for(int i=0; i<n; i++){
        int x; cin >> x;
        v.push_back(x); }
    int target; cin >> target;
    cout << binSeaSort(v,
}  return 0;
```

```
int binSeaSort(vector<int> &v, int target){
    int lo = 0, hi = v.size() - 1;
    while(lo <= hi){
        int mid = lo + (hi - lo)/2;
        if(v[mid] == target){ // if we found our target element
            return 1;
        }
        if(v[lo] == v[mid] and v[mid] == v[hi]){ // happens in case of duplicates
            lo++;                                  (see que. → example)
            hi--;
        }
        if(v[lo] <= v[mid]){ // this means array is sorted from lo to mid
            if(target >= v[lo] and target <= v[mid]){
                hi = mid - 1; // checking if target ele. lies in left half or not
            }
            else{ // this means our target lies in other half of array
                lo = mid + 1; // so we shift lo to mid+1 to search in
            }     right half
        }
        else{ // if arr → lo to mid is not sorted, then arr → mid to hi must be sorted subarray
            if(target >= v[mid] and target <= v[hi]){
                lo = mid + 1; // checking if ele. there in right half or not
            }
            else{ // means our target is in left half
                hi = mid - 1;
            }
        }
    }
    return -1; // if target element is not present
}
```

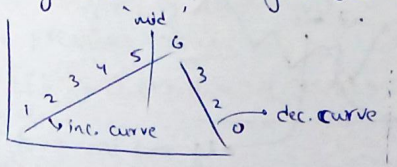
BINARY SEARCH PROBLEMS - 2

1) Given a mountain array 'a' of length greater than 3, return the index 'i' such that arr[0] < arr[1] < arr[2] ... < arr[i-1] < arr[i] > arr[i+1] ---> arr[arr.length - 1]. This index is known as peak index.

ip: array = [0,4,1,0] → op: 1

A) Basically we need to find out max. element

eg



: [1 2 3 4 5 6 3 2 0]

lo, hi, mid

↳ ans = mid & lo = mid+1

ans = 14 ; we check if mid is on ↑ing or ↓ing curve.

∴ Using binary search → $T_c = O(\log N)$

Using linear search → $T_c = O(N)$

Code

```
int peakMountainArr(vector<int> &v){
    int lo=0, high=v.size()-1;
    int ans=-1;
    while(lo<=hi){
        int mid = lo+(hi-lo)/2;
        if(v[mid] > v[mid-1]){   // increasing curve
            ans = max(ans, mid);
            lo=mid+1;
        }else{
            hi=mid-1;
        }
    }
    return ans;
}
```

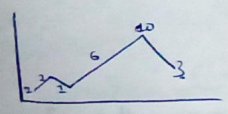
can also write as: ans=mid;

```
int main(){
    vector<int> v;
    int n;
    cin>>n;
    while(n--){
        int x;
        cin>>x;
        v.push_back(x);
    }
    cout<<peakMountainArr(v);
    return 0;
}
```

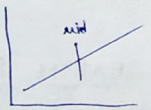
Q) A peak element is an element that is strictly greater than its neighbours. Given a 0-indexed integer array nums, find a peak element, & return its index. If the array contains multiple peaks, return the index to any of the peaks. You may imagine that nums[-1] = nums[n] = $-\infty$. In other words, an element is always considered to be strictly greater than a neighbour that is outside the array.

i/p: Array = [1,2,1,2,6,10,3] | o/p: Either index 1 (or) index 5 are the correct output. At index 1, 2 is the peak ele. & at index 5, 10 is the peak element.

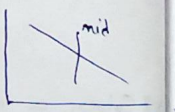
A)



Here we have multiple peaks & we need to return one of them. ∴ Let's make cases



If mid lies on ↑ curve, then we can discard left side & can tell there is min. 1 peak on right side.



If mid lies on ↓ curve, then we can discard right side & can tell there is min. 1 peak on left side.

If mid lies on ↑ curve, then we ignore left side (even though it has peaks) & we move on to right side as there is surety of min. 1 peak

$T_c = O(\log n)$

If mid lies on ↓ curve, then we ignore right side (even though it has peaks) & we move on to left side as there is surety of min. 1 peak

* This shows that we can apply binary search even on unsorted arrays, if the condition demands

Code

```
int findPeak(vector<int> &v){
    int n = v.size()-1;
    int lo=0, hi=n-1;
    while(lo<=hi){
        int mid = lo+(hi-lo)/2;
        if(mid==0){   // on end of ↑ curve, first idx checking (\)
            if(v[mid] > v[mid+1]){   (\)
                return 0;
            }else{
                return 1;
            }
        }
        else if(mid==n-1){   // on end of ↓ curve, last index checking (/)
            if(v[mid] > v[mid-1]){
                return n-1;
            }else{
                return n-2;
            }
        }
        else{   // means peak lies in middle of array
            if(v[mid]>v[mid+1] and v[mid]>v[mid-1]){
                return mid;
            }else if(v[mid] > v[mid-1]){   // we are on ↑ curve, so we have to go right side
                lo = mid+1;
            }else{   // we are on ↓ curve, so we have to go left side
                hi=mid-1;
            }
        }
    }
    return -1;
}
```

```
int main(){
    vector<int> v;
    int n;
    cin>>n;
    while(n--){
        int x;
        cin>>x;
        v.push_back(x);
    }
    cout<<findPeak(v);
    return 0;
}
```

Q) Search the 'target' value in a 2d integer matrix of dimensions n×m & return 1 if found, else return 0. This matrix has the following properties:

→ Integers in each row are sorted left to right.

→ The first integer of each row is greater than the last integer of the previous row.

1	3	5	7
10	11	16	20
23	30	34	60

i/p: Matrix: [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3

o/p: 1

A) Brute force Soln.: We check each element → $T_c = O(nm)$

(optimize)

↳ Let's apply binary search on each row → $T_c = O(n \log m)$ (n rows)

(this soln. would be good if we had only 1st cond., but here we have 2) → optimize

Since we have 2 conditions, we can optimize our code more

1	3	5	7
10	11	16	20
23	30	34	50

Since 1st ele. of each row > last ele. of prev. row

∴ on converting to 1D array → 1 3 5 7 10 11 16 20 23 30 34 50

(it became ↑ing sorted array)

now we can apply here binary search.

∴ $S_c = O(nm)$

$T_c = O(\log(nm))$

But *, we can still optimise the code! without taking any space

1	3	5	7
10	11	16	20
23	30	34	50

(3×4)

mid = 0+11/2 = 5 → to get it's co-orclinates

x = mid/m → eg: 5/4 = 1

y = mid%m → eg: 5%4 = 1

∴ (no. of cols)

∴ $T_c = O(\log(nm))$

$S_c = O(1)$

Code:

```
bool searchMatrix (vector<vector<int>>&V, int target){
    int n= V.size(); //no. of rows
    int m = v[0].size(); //no. of columns
    int lo=0, hi= n*m-1;
    while (lo<=hi){
        int mid = lo + (hi-lo)/2;
        int x= mid/m;
        int y = mid%m;
        if(v[x][y]== mid) {
            return true;
        } else if (v[x][y]<target){
            lo=mid+1;
        }else {
            hi=mid-1;
        }
    }
    return false;
}
```

```
int main(){
    vector(vector<int>> v={{1,3,5,7},
                           {10,11,16,20},
                           {23,30,34,60}};
    int target=23;
    cout<< searchMatrix (v, target);
    return 0;
}
```

This que. → model ⇒ Binary Search on 2D Matrix.

BINARY SEARCH PROBLEMS - 3

Q) You have n (n<=10^5) boxes of chocolate. Each box contains a[i] (a[i]<10000) chocolates. You need to distribute these boxes among 'm' students such that the max. no. of chocolates allocated to a student is minimum.

a) One box will be allocated to exactly 1 student

b) All the boxes should be allocated.

c) Each student has to be allocated at least one box.

d) Allotment should be in contiguous order, for instance, a student cannot be allocated box 1 & box 3, skipping box 2

Calc. & return the min. possible num.

Assume that it's always possible to distribute the chocolates

Input:

The first line of ip will contain the value of n, the no. of boxes.

The second line of ip will contain the 'n' num's. denoting the no. of chocolates in each box.

The third line will contain the m, no. of students

ip:	o/p
4 12 34 67 90 2	113

A) B_1 B_2 B_3 B_4 , S = 2

12, 34, 67, 90

$S_1 = B_1 + B_2 = 46$ } 157

$S_2 = B_3 + B_4 = 157$

$S_1 = B_1 + B_2 + B_3 = 113$ } 113

$S_2 = B_4 = 90$

$S_1 = B_1 = 12$ } 191

$S_2 = B_2 + B_3 + B_4 = 191$

min. → 113 → ans.

(mid → at most a student can have these many choco.)

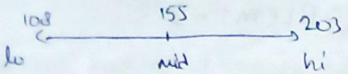
Explanation:

lo (min. choco.) 12 — mid 107 — hi 203 (total choco.)

∴ $S_1 = B_1 + B_2$

$S_2 = B_3$ (one box left)

Since there's one box left at mid = 107, this means when mid < 107 → definitely we won't get our ans, so mid > 107 → ∴ lo = mid + 1



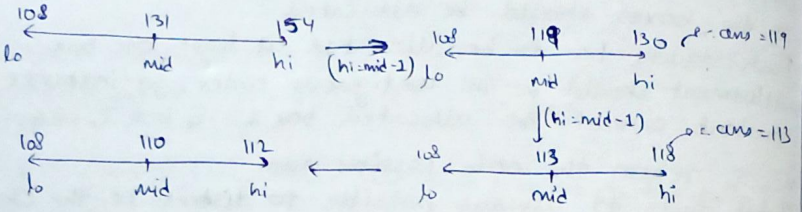
$\therefore S_1 = B_1 B_2 B_3$ ✓

$S_2 = B_4$ ✓

(since all boxes filled) { maybe this can be our ans

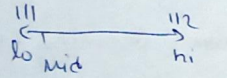
Now >mid → all will satisfy but we need min., so we do hi = mid − 1

$\therefore$ ans = 155



$B_1+B_2+B_3 > 110$; $B_1+B_2 < 110$

Now condn's not satisfying <110, so we do lo = mid+1.



→ condn won't come

∴ finally ans = 113

$\therefore T_c = n \log(\Sigma a_i)$

(no. of boxes) (total no. of chocolates)

Code:

```
bool canDistChoco(vector<int> &v, int mid, int s){
    int n = v.size();
    int studentReq = 1, currSum = 0;
    for(int i=0; i<n; i++){
        if(v[i] > mid){
            return false;
        }
        if(currSum + v[i] > mid){
            studentReq++;
            currSum = v[i];
            if(studentReq > s){
                return false;
            }
        }
        else{
            currSum += v[i];
        }
    }
    return true;
}
```

distribute

```
int distChoco(vector<int> &v, int s){
    int n = v.size();
    int lo = v[0], hi = 0;
    for(int i=0; i<v.size(); i++){
        hi += v[i];
    }
    int ans = -1;
    while(lo <= hi){
        int mid = lo + (hi-lo)/2;
        if(canDistChoco(v, mid, s)){
            ans = mid;
            hi = mid-1;
        }
        else{
            lo = mid+1;
        }
    }
    return ans;
}
int main(){
    int n; cin>>n;
    vector<int> v;
    for(int i=0; i<n; i++){
        int x; cin>>x;
        v.push_back(x);
    }
    int s; cin>>s;
    cout<< distChoco(v, s);
    return 0;
}
```

means checking if atmost 'mid' no. of choco. can be distributed among students or not.

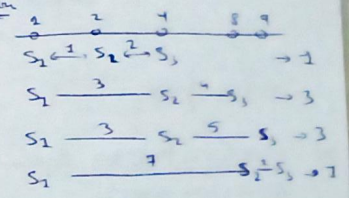
Q) A new racing track for kids is being built in India with 'n' starting spots. The spots are located along a straight line at positions $x_1, x_2, \dots x_n$ ($x_i <= 10^9$). For each 'i', $x_{i+1} > x_i$. At a time only 'm' children are allowed to enter the race. Since the race track is for kids, they may run into each other while running. To prevent this, we want to choose the starting spots such that min. distance btw any 2 of them is as large as possible. What is the largest min. distance?

I/p:
The first line of i/p will contain the value of n, the no. of starting spots.
The second line of i/p will contain the 'n' no.s denoting the location of each spot.
The third line will contain the value of m, no. of children.

i/p:	o/p
5 1 2 4 8 9 3	3

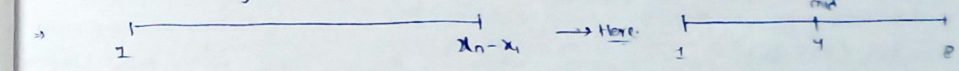
(minimise the maximum)

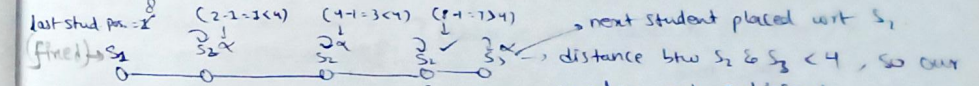
explanation



A) Thinking:

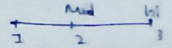
We will try to maintain atleast 'mid' dist. btw adjacent student.

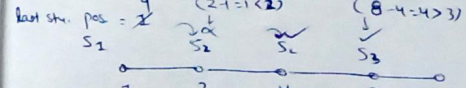




→ next student placed with S_3, distance btw S_2 & S_3 < 4, so our condn. not satisfied

Now we couldn't maintain '4-mid' distance btw students, so we can't maintain >4 distance also → ∴ hi = mid − 1 = 2 (here)

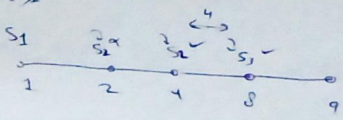




→ ∴ ans = 2 atleast (mean obvio <2 also we can have)

∴ lo = mid+1

(lo, mid, hi)



∴ 3 better ans than 2

↳ ans = 3

for more better ans. → lo = mid+1 > hi → condn false

$\therefore T_c = n \log(x_n - x_1)$ → (we are applying binary search to find max. distance for students)

(we are checking through each spot for 1 student, so 'n' there ↑ total spots)

$\therefore S = O(1)$

Code:

```
bool canPlaceStudents(vector<int> &pos, int s, int mid){
    int studentsReq = 1;
    int lastPlaced = pos[0];
    for(int i=1; i<pos.size(); i++){
        if(pos[i] - lastPlaced >= mid){
            studentsReq++;
            lastPlaced=pos[i];
            if(studentsReq == s){
                return true;
            }
        }
    }
    return false;
}

int race(vector<int> &pos, int s){
    int n = pos.size();
    int lo = 1, hi = pos[n-1] - pos[0];
    int ans = -1;
    while(lo<=hi){
        int mid = lo+(hi-lo)/2;
        if(canPlaceStudents(pos, s, mid)){   // checking if we can place 's' students with atleast 'mid' distance on 'pos'.
            ans = mid;
            lo = mid+1;
        }else{
            hi = mid-1;
        }
    }
    return ans;
}

int main(){
    int n; cin>>n;
    vector<int> pos;
    while(n--){
        int x; cin>>x;
        pos.push_back(x);
    }
    int s;
    cin>>s;
    cout<<race(pos,s);
    return 0;
}
```

COMPLEXITIES OF SORTING ALGO.'s (so far)

Algorithm (n = no. of ele. in array)	Time Complexity: Best	Average	Worst	Space Complexity: Worst
Bubble Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Merge Sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
Quick Sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(\log(n))$
Count Sort	$\Omega(n+k)$ (sorted array size) (freq. array size)	$\Theta(n+k)$	$O(n+k)$	$O(k)$
Radix Sort	$\Omega(nk)$	$\Theta(nk)$	$O(nk)$	$O(n+k)$
Bucket Sort	$\Omega(n+k)$ (bucket size) (internal array of ele.)	$\Theta(n+k)$	$O(n^2)$	$O(n)$

* We can do sorting of array by using 'sort' property in C++.

STRINGS

Strings are: objects of a class std::string in C++ & it's used to represent sequence of characters

Syntax / ex: string (stringname) = "abcd" ; Header file: #include<string>

ex:
```
string str = "college";
string str1("wallah");
cout<<str<<" "<<str1;
```
→ op: college wallah

■ How to take input:

ex:
```
string str;
cin>>str;
cout<<str<<endl;
```
ip: college wallah → op: college

(terminates at " ")

```
getline(cin, str);
cout<<str<<endl;
```
→ op: college wallah

getline(): used to store strings which has " "

Indexing of characters in a String

↳ similar to arrays : ex: str = C O L L E G E \0 (n = 7)

0 1 2 3 4 5 6 7

→ null character shows that string has ended.

cout<<str[3]
↳ op: L

ASCII values

Every character has a numeric value (a-z, A-Z, *, $, +, -, ...)

ex: A - 65, B - 66, ..., Z - 90 ; a - 97, b - 98, ..., z - 122

```
char ch = 'A';
cout << int(ch);
```
↳ op: 65

String vs Character Array

String	Character Array
→ String is a class. String variables - obj. of this class	→ array of char datatype
→ declaration: string str_name;	→ declaration: char array_name[size] (need to give size before)
→ dynamic memory allocation	→ static memory allocation
→ no pre-allocated memory (unless until value given)	→ unused allocated memory is wasted.
→ have inbuilt functions	→ implementation is faster.
→ ease to implement	

Commonly used inbuilt func.'s

- **reverse()**: → reverses a string from starting ptr (pointer) to end ptr
  - → syntax: reverse(ptr1, ptr2)
  - → Time Complexity - O(length of string)
  - → ex:
```
string str = "abcd";
reverse(str.begin(), str.end());
cout<<str;
```
↳ op: dcba

- **substr()**: → used to find substring of a given string
  - → syntax: stringname.substr(position, length);
  - → T_c = O(length)
  - → ex:
```
string str = "collage";
cout << str.substr(0,3) << " " << str.substr(3);
```
(prints 3 indexes starting from index 0; prints until end of str starting from index 3)

of: col lage

- **The "+" operator**: → concatenates strings
  - → ex:
```
string s1 = "college";
string s2 = "wallah";
cout << s1+s2;
```
op: collegewallah

```
string s1 = "college";
string s2 = "wallah";
s1 += s2 // s1 = s1+s2
cout<<s1;
```
op: collegewallah

→ But s1 += s2 & s1 = s1+s2 are not same.

s1 += s2: s2 is getting appended after s1.

s1 = s1+s2: extra space is getting used to create copy of s1.

- **strcat()**: → used to concatenate character arrays
  - → ex:
```
char s1[20] = "college";
char s2[20] = "wallah";
strcat(s1, s2); //appends s2 after s1
cout<<s1;
```
↳ op: collegewallah

- **push_back()**: → used to insert char. at the end of string
  - → syntax: str.push_back(char) (string name)
  - → ex:
```
string s1 = "abcd";
char ch = 'e';
s1.push_back(ch);
cout<<s1;
```
↳ op: abcde

- **size()**: → used to calculate String size
  - → syntax: string_name.size() or string_name.length() } T_c = O(1)
  - → If we want to calculate for character array, then we use "strlen" → syntax: strlen(char_array_name); → T_c = O(n)
  - → ex:
```
string s1 = "abcde";
cout<<s1.size()<<endl;
char ch[20] = "abcde";
cout << strlen(ch);
```
} op: 5 5

- **to_string()**: → used to convert numeric/integer value to string
  - → syntax: to_string(integer_variable_name)
  - → ex:
```
int num = 432;
string str = to_string(num);
str += "1";
cout << str << " " << str[2] << " " << num;
```
op: 4321 2 432

Q) Given a strings str, sort the given string.

Constraints: The string will contain only alphabetical characters from a-z.

ip 1: "coding wallah" ip 2: "star"

op 1: "aacdghillnow" op 2: "arst"

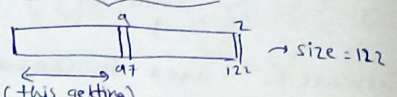
A) We can apply count sort, since it has low $T_c = O(n)$

eg: array: | 4 | 3 | 4 | 1 |

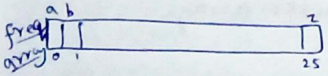
freq. arr: | 0 | 1 | 0 | 1 | 2 |
0 1 2 3 4

: Here: ASCII values

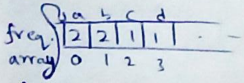
a-z → 97-122



(this getting wasted) → so we subtract 'a' ASCII value & create array of 26 size



eg: "adbcab"



Now when we print, then we do 'i+'a' → getting alphabets

index of 'a' = 'a' - 'a' = 0

index of 'b' = 'b' - 'a' = 1

index of str[i] = str[i] - 'a'

while (freq[i]--){
str[j] = i+'a';
}
j++;

- T_c = O(length of string)
- S_c = O(26) ~ const.

Code:

```
#include<iostream>
#include<string>
#include<vector>
using namespace std;
string countSort(string str){
    vector<int> freq(26,0);
    //storing freq. of every character in string
    for(int i=0; i<str.length(); i++){
        int index = str[i] - 'a';
        freq[index]++;
    }

    int j=0; //creating our sorted array
    for(int i=0; i<26; i++){
        while(freq[i]--){
            str[j++] = i+'a';
        }
    }
    return str;
}
int main(){
    string str;
    cin>>str;
    cout<<countSort(str);
    return 0;
}
```

Q) Given 2 strings s & t, return true if t is an anagram of s, & false otherwise.

Constraints: String s & t will only contain lowercase alphabetical characters.

ip 1: s = "anagram", t = "nagaram"

op 1: yes

ip 2: s = "bank", t = "atm"

op 2: no

A) Anagram - rearranging letters of a word to get another word.

1) same letters
2) same count of letters
i.e. same freq. of letters

Now how can we solve this:

a) Use double for loops checking character of str1 & traversing through str2

b) Making 2 freq. arrays for both ip strings & checking if they both are equal or not

c) To manage S_c → we can solve in 1 freq. array

eg: s1 [b o k o] [b o o k] s2
(+1) (-1)

freq1[ ]: | 1 | 0 | -1 |
b k o

while going through s1, we can do freq1++ & traversing via s2, we can do freq1--. If freq1 = 0 → anagram ✓

- T_c = O(length of string)
- S_c = O(26)

Code:

```
bool isAnagram(string s1, string s2){
    //create freq array
    vector<int> freq(26,0);
    //if lengths are different for s1,s2, then
    if(s1.length() != s2.length()){ return false
        return false;
    }
    //store freq. of characters of s1 & s2
    for(int i=0; i<s1.length(); i++){
        freq[s1[i]-'a']++; //for s1, we are ↑ freq. of char
        freq[s2[i]-'a']--; //for s2, we are ↓ freq. of char
    }
    //checking if freq. of every character is 0
    for(int i=0; i<26; i++){
        if(freq[i] != 0){ //not angram
            return false;
        }
    }
    return true;
}
int main(){
    string s1, s2;
    cin>>s1>>s2;
    if(isAnagram(s1, s2)) cout<<"Strings are anagrams";
    else cout<<"Strings are not anagrams";
    return 0;
}
```

Q) Given 2 strings s & t, determine if they are isomorphic.

ex: s = "egg", t = "add"

ex: yes

A) 2 strings are considered isomorphic if we can replace characters in one string with characters in the other string in such a way that the 2 strings becomes identical.

ex: egg → 'e' mapped to 'a'; means 'e' can't be mapped to anyone else
add

Code:

```
bool isIsomorphic(string s1, string s2){
    vector<int> v1(128,-1);//vector size min. 123 as ASCII of z=122
    vector<int> v2(128,-1);//v1,v2 stores mapping of characters s1 into s2
    if(s1.size()!=s2.size()){ //checking string length, since
        return false;          strings of different length can't be isomorphic.
    }
    for(int i=0;i<s1.size();i++){ //the code then enters a loop that iterates through each character of strings.
        if(v1[s1[i]]!=v2[s2[i]]){ →For each charac, it checks whether
            return false;mappings stored in v1 & v2 (of s1 & s2) match or not.
        }                        If they don't match, it means current charac
        v1[s1[i]]=v2[s2[i]]=i; violates the isomo. condn. & returns
    }                          false.
    return true; →If mappings match, it updates mappings in both v1 & v2
}                  with current index 'i'.
int main(){
    string s1,s2;
    cin>>s1>>s2;
    if(isIsomorphic(s1,s2)){
        cout<<"They're isomorphic";
    }
    else cout<<"They're not isomorphic";
    return 0;
}
```

dry run/ex.: (egg, add)

s1[0]='e' < s2[0]='a'
s1[1]='g' < s2[1]='d'
s1[2]='g' < s2[2]='d'

vector initialised by -1

v1[s1[0]]=v1[e]=v1[101]=-1→0
v1[s1[1]]=v1[g]=v1[103]=-1→1
v2[s2[0]]=v2[a]=v2[97]=-1→0
v2[s2[1]]=v2[d]=v2[100]=-1→1

(Similarly you can check for non-iso. strs)

Q) Given an array of strings. WAP to find the longest common prefix string amongst an array of strings

ex: arr=["flower","flight","flask"]

ex: "fl"

A) Means longest common word: ex: ant, anti, anthem → common = ant = 3

Now suppose we sort the arrays:

ex: [ant, apple, add, anthem, anti] → [add, ant, anthem, anti, apple]

then thing to be noticed is; the first ele. & last ele. will have max difference between each.

& suppose we find common word in first & last word, then obvio. we will have it common amongst the middle elements.

∴ comparing: add, apple → stored in "ans" = 'a'

common → prefix

Code:

```
string LongestCmnPref(vector<string> &str){
    //sorting array of strings
    sort(str.begin(), str.end());
    string s1 = str[0];//first string
    int i=0;
    string s2 = str[str.size()-1];//last string
    int j=0;
    string ans="";
    while(i<s1.size() and j<s2.size()){
        if(s1[i]==s2[j]){
            ans+=s1[i];
            i++; j++;
        }
        else{
            break;
        }
    }
    return ans;
}
```

```
int main(){
    int n;
    cout<<"Enter no. of strings";
    cin>>n;
    cout<<"Enter strings";
    vector<string> str(n);
    for(int i=0;i<n;i++){
        cin>>str[i];
    }
    cout<<LongestCmnPref(str);
    return 0;
}
```

• $S_c = O(1)$

• T_c = O(nlogn * m) + O(min(s[0].size(), s[n-1].size()))

(sorting; no. of strings; general length of strings; while loop)

But here T_c is little more!, so lets think of another solution.

Soln. 2: We take 1st string as const. & compare it with others & accordingly update our "ans" → (min. of common pref. & ans.)

Then we cout of ans → by → string1.substr(0, ans_length);

Code:

```
string LongestCmnPref(vector<string> str){
    //Keeping 1st string as const. & comparing it with all other strings.
    string s1 = str[0];
    int ans_length = s1.size();
    for(int i=1;i<str.size();i++){
        int j=0; →//finding the common prefix string length
        while(j<s1.size() and j<str[i].size() and s1[j]==str[i][j]){
            j++;
        } →//checking the next character in strings
        ans_length = min(ans_length, j);//updating length of ans. string
    }
    string ans = s1.substr(0, ans_length);
    return ans;
}
```

```
int main(){
    //same as above one
}
```

• $S_c = O(1)$

• T_c = O(n × m²) (no. of strings; general length of strings)

Q) An encoded string (s) is given, & the task is to decode it. The encoding pattern is that the occurence of the string is given at the starting of the string & each string is enclosed by square brackets.

Note: The occurence of a single string is less than 1000.

ip 1:	s = 1[b]	ip 2:	s = 3[b2[ca]]
op 1:	b	op 2:	bcacabcacabcaca

Ex: 1[a[2[b[c]]]] → we need to start from inner means [b3[c]] decoding depends on 3[c] decoding, so we go from in to out

⇒ 1[a2[bccc]] ⇒ 1[abcccbccc] ⇒ abcccbccc

∴ Process:

```
for(int i=0; i<s.length(); i++){
    if(s[i]!=']')
        insert into result
    else
        ① Extract str from result till '[' found.
        ② Reverse str
        ③ Remove last char (i.e removing '[')
        ④ Extract digit/num from result till s[i]>='0' and <='9'
        ⑤ Reverse num
        ⑥ Convert num string to int → {'stoi' property of c++}
        ⑦ Insert str in result, int_num times
```

(can be >1 digits as well, so check within their ASCII range)

(we get char. in num. form)

Ex: s = 3[b2[ca]] → result = "3b2[ca"

str = "ac" → rev ⇒ str = "ca"

num = "2" → rev. ⇒ 2 → stoi(num)

{stoi → String to integer}

int_num = stoi(num)

Now 3[b2[ca]] → again we get ']' so we repeat the process.

∴ Final result

↳ "bcacabcacabcaca"

Code:

```
string decodedString(string s){
    string result="";
    //traversing the encoded string
    for(int i=0; i<s.length(); i++){
        if(s[i]!=']'){
            result.push_back(s[i]);
        }
        else{
            //extract str from result
            string str="";
            while(!result.empty() && result.back()!='['){
                str.push_back(result.back());
                result.pop_back();
            }
            //reversing the str
            reverse(str.begin(), str.end());
            //remove last char. from result which is '['
            result.pop_back();
            //extract num from result
            string num="";
            while(!result.empty() && (result.back()>='0' and result.back()<='9')){
                num.push_back(result.back());
                result.pop_back();
            }
            //reversing the num string
            reverse(num.begin(), num.end());
            //convert string to integer
            int int_num=stoi(num);
            //inserting str in result int_num times
            while(int_num>0){
                result+=str; //concatinating str into result
                int_num--;
            }
        }
    }
    return result;
}
```

(until 'result' doesn't get empty, it runs (!false=true); when result gets empty (!true=false), loop ends)

(gives last charac. of the string: stringname.back())

(removes last charac. of the string: stringname.pop_back())

```
int main(){
    string str;
    cin>>str;
    cout<<decodeString(str);
    return 0;
}
```

- $S_c = O(n)$ → length of decoded string = "result" string
- T_c = Considering ①, ②, ③, ④ → (then adding into result string int_num times)
  - ① traversing ip string
  - ② extracting string
  - ③ extracting num
  - ④ ∴ length of result string** is our time complexity.

∴ $T_c = O(n)$, n = length of decoded string

Q) Given a binary string and an integer k, return the max. no. of consecutive 1's in the string if you can flip atmost k 0's.

eg: 0001101011 , k=2 | o/p: 7

Here: 0001101011 : We need to find a substring which has atmost 'k' zeroes present.

0001101011 (>2 zeroes) (can be) (can be)

So brute force approach, we can make a pointer traversing from start to end

0001101011 → next time 0001101011 ....

∴ We need to apply nested loop : for i=0 to i<s.length()
for j=0 to j<s.length
if s[i..j] → 0's <=k ✓
storing: length i..j

$T_c = O(n^2)$

Better Approach:

Sliding Window Approach: * (imp for interview)

helps to get largest/shortest sequence with some given cond^n.

eg: 3 size of "abcdef" (we are not making all possible windows here)

windows: abc, bcd, cde, def

(If we were making all windows → above soln.)

Here: "0001101011" k=2

zero_count = 0 (initially) 1 2 (updations) 1 2

max_length = 0 1 2 3 4 5 6 7

(count<k) s e s e s e

```
while(zero_count > k){
    if(str[s]=='0')
        zero_count--;
    s++;
}
```

- $T_c = O(n)$ → length of input string
- $S_c = O(1)$

Code: → func. prototype

```
int longestOnes(string, int);

int main(){
    string str;
    cout<<"Enter binary string";
    cin>>str;
    int k;
    cout<<"Enter max flips";
    cin>>k;
    cout<<longestOnes(str, k);
    return 0;
}
```

```
int longestOnes(string str, int k){
    int start=0, end=0, zero_count=0, max_length=0;
    for(;end<str.size();end++){
        if(str[end]=='0'){
            zero_count++;
        }
        while(zero_count>k){
            if(str[start]=='0'){
                zero_count++;
                start++; //contracting our window
            }
        }
        max_length=max(max_length, end-start+1); //zero_count<=k
    }
    return max_length;
}
```

(end-start+1 → length of current window)