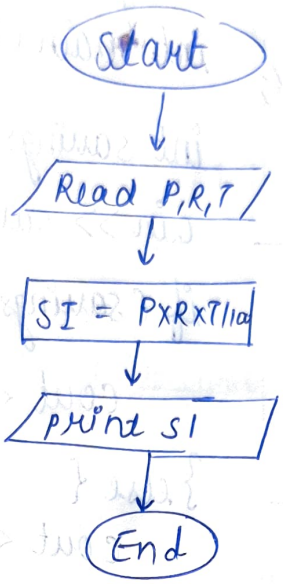
1.2.24

Data Structure And Algorithm

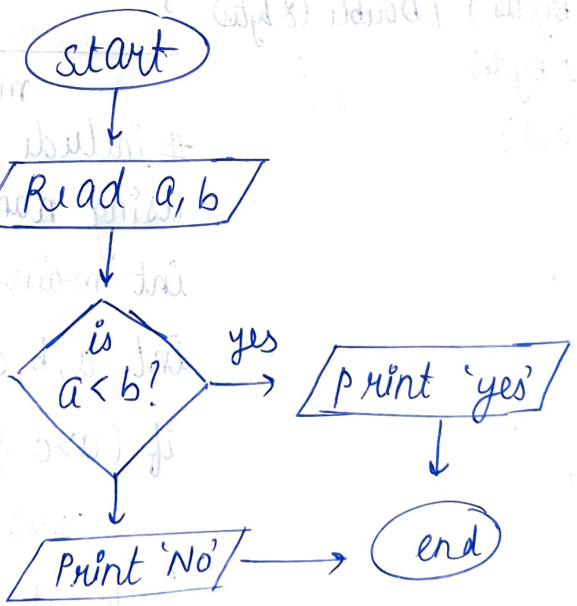
LOVE BABBAR

Flowchart

①



② Check a < b



cpp

```
#include <iostream>
using namespace std;
int main(){
    cout << "Hello World" << endl;
}
```

end line

To get output

```
g++ filename.cpp
.\a.exe
```

Data types and variables

1 Byte = 8 bits

•) Primitive
- → Integer (4 Bytes)
- → Float (4 Bytes) / Double (8 bytes)
- → Character (1 Byte)
- → Boolean (0,1)

•) Derived
- → function
- → Array
- → Pointer
- → Refrence

•) User Defined
- → Class
- → structure
- → Union
- → Enum

```
#include<iostream>
int main(){
std::cout << "Hello\n";
return 0;
}
```

If /else

```
#include <iostream>
using namespace std;
int main(){
int savings;
cin >> savings;
if (savings > 5000) {
    cout << "shopping";
}else {
    cout << "movie";
    }
return 0;
}
```

---

max 3.cpp

```
#include <iostream>
using namespace std;
int main(){
int a,b,c;
if (a>c && a>b){
    cout << a << endl;
}else if (b>a && b>c){
    cout << b << endl;
}else {
    cout<< c << endl;
    }
return 0;
}
```

LOOPS

```
for (initialisation; condtn; updat){
    task;
}
```

entry controlled

```
while ( condition is true){
    // do task;
}
```

entry controlled

```
do {
    // do task;
} while (condition);
```

exit controlled

Jumps in loops

> Break
> Continue

Switch Case

Q.) robot says hello in diff languages for diff buttons.

```
#include <iostream>
using namespace std;
int main() {
    int a;
    cin>> a;
    switch (a) {
        case 1;
        | cout<< "Hello";
        break;

        case 2;
        | cout << "Hola";
        break;

        case 3:
        | cout << "Namaste";
        break;

        default:
        | cout << "Ram Ram";
        break;
        }
        return 0;
    }.
```

Operators

① Arithmetic Operators
- ↳ Binary op. (+, -, *, %, /)
- ↳ Unary op. (+=, ++, --)

② Relational Operators
→ return a boolean value (1 or 0)

==, !=, >, <, >=, <=

③ Logical Operators

&& → And
|| → OR
! → Not

④ Bitwise Operators.

And → &
OR → |
NOT → ~
XOR → ^

Examples

a=5 , b=7

a & b

5 → 101
7 → 111
---
101 → 5

a & b = 5

a=5, b=7

a^b

5 → 101
7 → 111
---
ans 010

a^b = 2

a=5 , b=7

a|b = 7

101
111

111

Not (~)

x	y	(z)x	(z)y
0	0	1	1
0	1	1	0
1	0	0	1
1	1	0	0

a = 2 → 00.... 010

~a = -3 ~a ⇒ 1111...101

↓
-ve no.

twice complement

⇒ 00000 10
+1

...00000 11

ans = -3 //

XOR → dono same mat dena.

x	y	z
1	1	0
1	0	1
0	1	1
0	0	0

• Left shift (<<)

int a = 5;

a << 1

shift bits of 'a' by 1.

a = 000.... 0101

a<<1 = 000...01010

a = 10

similarly for right shift

a = 5 } 000...101

a>>2 } 000...001

a = 1

majorly (not always true)

→ 1 left shift ⇒ x2

→ 1 right shift ⇒ ÷2

↳ if no. is too big → left shift will make it negative

e.g 01000...0101.

fibonacci

```
#include <iostream>
using namespace std;
int main() {
    int nt, n0=0, n1=1, n=10;
    cout << n0 << endl;
    cout << n1 << endl;
    for (int i=0; i<=n; i++){
        nt = n0 + n1;
        n0 = n1;
        n1 = nt;
        cout << nt << endl;
    }
    return 0;
}
```

out 0 1 1 2 3 5 8 13 21 34

Q.) Find if number given by input is prime or not.

```
int main() {
    int n;
    cout << "Enter number" << en
    cin >> n;
    bool isprime = 1;
    for (int i=2; i<n; i++) {
        if (n%2 == 0) {
            is prime = 0;
            break;
        }
    }
    if (isprime == 0) {
        cout << "Not a prime number";
    }else { cout << " its a prime no.";}
    return 0;
}
```

---

break → gets you out of loop

continue → skips particular iteration.

e.g.

```
for (int i=0; i<5; i++) {
    cout << "Hi" << endl;
    cout << "Hey" << endl;
    continue;
    cout << "Reply me" << endl;
}
```

o/p

Hi
Hey
Hi
Hey
Hi
Hey ... for 5 times.

Types of Variables

d.

Types of Variables

1) Global variables : Accessible throughout entire program

2) Local Variables: Declared within functions or block of program. (reducing naming conflict)

3) Instance Variables : Associated with individual objects, defining their properties or attributes

4) Class Variables : Shared by all instances of a class, representing class level data

3, 4) → Object oriented Prog.

Q.) Given an integer, return its difference between product and sum of it's digit.

e.g. n = 234

n % 10 → 4

n / 10 → 23

23 % 10 → 3

23 / 10 → 2

stop when n = 0;

code

```
int main () {
    int prod = 1;
    int sum = 0;
    int n;
    cout << Enter n << endl;
    cin >> n;
    while (n != 0) {
        int digit = n % 10;
        sum = sum + digit;
        prod = prod * digit;
        n = n / 10;
    }
    cout << prod - sum << endl;
    return 0;
}
```

Q.) Write code that takes unsigned integer and returns the number of '1' bits it has

```
int main () {
    while (n != 0) {
        if (n & 1) {
            count++;
        }
        n = n >> 1;
    }
    return ~~0~~ count;
}
```

123

Q.) WAP to reverse an integer.

logic

ans = (ans x 10) + digit

basically shifts the number left adding '0' to the rightmost place. Then adding digit

321

32 x 10 + 1 = 321

---

code

```
int main () {
    int n, digit, ans = 0;
    while (n != 0) {
        digit = n % 10;
        ans = (ans * 10) + digit;
    }     n = n / 10;
```

```
    cout << ans << endl;
    return 0;
}
```

---

compliment

e.g. n = 5 ⟶ 1 0 1
compliment → 0 1 0

n = 2 | compliment of 5

---

e.g. n = 7 ⟶ 1 1 1
0 0 0

n = 0

n = 10 $\xrightarrow{com}$ n = 5

Q.) WAP to find compliment of a given number.

```
int main() {
    int m = n;
    int mask = 0;
// edge case
if (n==0)
    return 1;
while (m!=0) {
    mask = (mask << 1) | 1;
    m = m>>1;
}
int ans = (~n) & mask;
return ans;
}.
```

Q.) Given an integer n., return true, if it is a power of 2 else return false.

```
ispowerof two (int n) {
    for (int i=0; i<=30; i++){
    int ans = pow (2, i);
    if (ans == n) {
        return 1;
    }
    }
    return 0;
}
```

SWITCH CASE

```
switch ( exp_) {
    case (const 1) : ≈≈
        break
    case ( const 2) : ≈≈
        break
    default : ≈≈
}.
```

---

break → use to come out of loop or a block of function when our desired task is finished.

c

If switch is inside infinite loop (stop using exit)

→ use exit instead of break

```
while (1) {
    case 0: print;
            exit(0);
    case 1; print;
            exit(0);
    default: print;
             exit(0);
}
```

---

exit (int_status)

int_status = 0 { no errors}

≠0 { indicates error}

---

continue → skips 1 iteration (index)

use of continue in switch case is not valid.

---

continue is designed to skip a particular iteration in while, do while, for loops.

Since switch case doesn't contain any iterations we can use continue

⇒ Give syntax Error **

FUNCTIONS

→ well defined task

Q.) Write a function to calculate ^nC_r of given n & r.

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

```
#include <iostream>
using namespace std;
int fact(int f) {
    int ans = 1;
    for (int i=1; i<=f; i++){
        ans = ans * i;
    }
    return ans;
}
int nCr(int n, int r) {
    int ans = fact(n) / (fact(r) *
                         fact(n-r));
    return ans;
}
int main() {
    int n,r;
    cin >> n,r;
    cout << "nCr = " << nCr(n,r);
    return 0;
}
```

function-call stack for previous question

last in first out

main calls nCr call fact

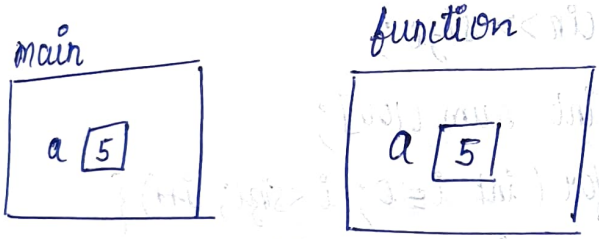
fact gives ans to nCr

nCr gives ans to main

main displays ans.

---

pass by Value



a++;

a = 4

but a = 5

value of 'a' remains 5 in main function

Q.) void update (int a){

a = a/2;

}

int main(){

int a = 10;

update (a);

cout << a << endl;

}

Ans = 10

Q.) find out put

int dummy (int a){

a -= 5;

return a;

}

int main(){

int a = 15;

dummy (a);

cout << a;

}

Ans = 15

---

if in main function

→ a = dummy (a);

for previous question then

Ans = 10

this changes "a" of main f^n

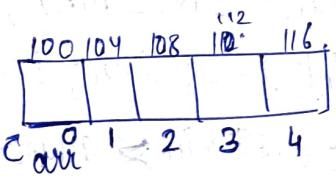
Arrays

→ stores similar type of data

→ Contiguous addresses of locations

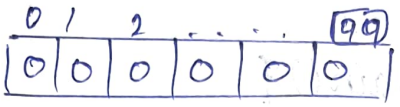
→ accessible through index.

int arr [5];



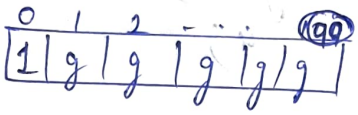
array (n size) index (0 to n-1)

```
int main (){
    int arr[100] = {0};
return 0;
}
```



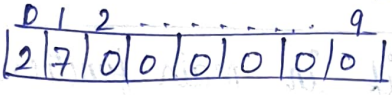
---

if `int arr[100] = {1};`



g = 0

`int arr[10] = {2,7}`



```
void printArray (int arr[], int size){
    for (int i=0; i<=size; i++){
        cout << arr[i] << " " << endl;
    }
```

Q.) Find max/min element of array.

```
#include <iostream>
using namespace std;
int min (int sum[], int n){
    int min = INT_MAX;
    for (int i=0; i<n; i++){
        if ( min > sum[i]){
            min = num[i];
    } }
    return min;
}
int max (int num[], int n) {
    int max = INT_MIN;
    for (int i=0; i<n; i++){
        if (max < num[i]) {
            max = num[i];
        }
    }
    return max;
}
int main (){
    int size;
    cin >> size;
    int num[100];
    for (int i=0; i<size; i++) {
        cin >> num[i];
    }
cout << "Max element : " << max(num, size);
cout << "Min element : " << min(num, size) << endl;
    return 0;
}
```

NOTE

arr[10000] is better than arr[variable]

e.g. num[size];

* in case of arr passing in f^n any changes done in function to given array results in actual change in main function

(This happens because name of arr is ptr to first element in array)

Linear Search

```
#include <iostream>;
using namespace std;
int linearSearch (int arr[], int size,
bool
        int key){
    for ( int i=0; i<size; i++){
        if ( arr[i] == key ) {
            return 1;
        }
    }
    return 0;
}
int main () {

int arr[5] = { 1, 5, 7, 11, 6 };

cout << "Enter element to find:" <<endl
int key;
cin >> key;

bool found = linearSearch(
            arr[], 5, key);

if (found) {
cout << "Element" << key << " was
found at" << i << "index";

else {
    cout << " not found" << endl;
}
return 0;
}
```

Reverse An Array

input : 1, 5, 7, 9, 3

output : 3, 9, 7, 5, 1.

```
#include <iostream>
using namespace std;
int temp;
void reverse (int arr[], int size){
 for ( int i=0; i<(size/2); i++){
     temp = arr[i];
     arr[i] = arr[size-i-1];
     arr[size-i-1] = temp;
 }
}
int main() {
    int arr[10] = { ___ };
    reverse ( arr, 10);
} return 0;
```

Questions on Arrays

Q.1) Swap alternate elements of array

- arr [5] = {1, 2, 7, 8, 5}

o/p = {2, 1, 8, 7, 5}

arr [4] = {1, 2, 3, 4}

o/p = {2, 1, 4, 3}

→

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

void printArray (int Arr[], int n){
  for (int i=0; i<n; i++) {
      cout << Arr[i] << "_";
  } cout << endl;
}

void swap ( int arr[], int k){
  int temp;
  temp = arr[k];
  arr[k] = arr[k+1];
  arr[k+1] = temp;
}

vaid swapAlt [ int A[], int s]{
  for ( int k=0; k<s; k=k+2){
    if (k+1 < s) {
      swapAlt [A, s];
    }
  }
}

int main () {
  int A[10] = {1,2,3,4,5,6,7,8,9,10};
  int s = sizeof (A) / sizeof(int);

  cout << "before Alt swap" << endl;
  printArray (A, s);
  swapAlt [A, s);

  cout << "After Alt swap" << endl;
  printArray (A, s);

  return 0;
}
```

o/p

before

1 2 3 4 5 6 7 8 9 10

After

2 1 4 3 6 5 8 7 10 9

Q.2) Find Unique Number in array (only 1 unique no.)

```
int UniqueNo ( int * arr, int size){
  int ans = 0;
  for ( int i=0; i < size; i++){
      ans = ans ^ arr[i];
  }
  return 0;
}
```

logic (XOR)

6 8 4 8 8 8 6

6^6 = 0 8^8 = 0 0^4 = 4

0^0^0^0^0^4 = 4

(same no. XOR give zero).

Q.3) Unique Number of Occurance

→ Each number in array must be occur @ unique times than others.

E.g

{1,0,1,0} → false

{1,2,3} → false

{1,1,2} ✓

{1,1,2,2,2,3,-3,3,1,1} ✓

Q.4) Find Duplicates in array.

Q.5) Intersection of Array

A = {1,2,3,4}

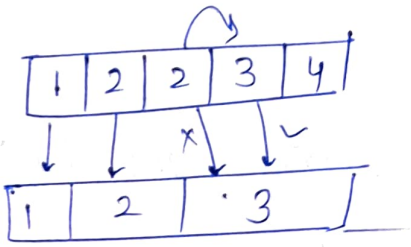
B = {3,3,4,5}

O/P → = {3,4}.

Sorted array is given.

logic

```
vector<int> findArrayIntersec.
(vector<int> & arr1, int n, int m
 vector<int> & arr2) {

 int i=0, j=0;
 while (i<n && j<m) {
  if (arr1[i] < arr2[j]
  if (arr1[i] == arr2[j]) {
   ans.push_back(arr1[i]);
   i++;
   j++;
  }
  else if (arr1[i] < arr2[j]) {
      i++;
  } else {
      j++;
  }
 }
 return ans;
}.
```



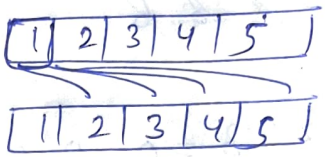
O/P = 1, 2, 3, and.

Q.6) Pair Sum

→ find pairs in arrays which gives sum = S.

```
vector < vector<int>> pairsum (
 vector<int> &arr, int s) {

    vector < vector<int>> ans;

for ( int i=0; i<arr.size(); i++)
{
    for (int j=i+1; j<arr.size(); j++)
    { if (arr[i]+ arr[j]==s)
      {
      vector <int> temp;
      temp.push_back (min (
          arr[i], arr[j]) );
     temp.push_back (max (
          arr[i], arr[j] ));
     ans.push_back (temp);
      }
    }
}

sort (ans.begin(), ans.end());
return ans;
}
```

logic



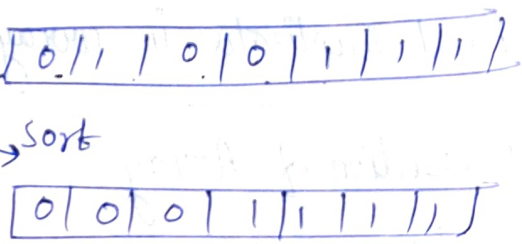
compare i=0 from j=i+1 to j=n-1 where n = size of array.

Q.7) Triplet Sum (H.W)

→ use 3 for loops.

```
for (i → 0 to n-1)
 { for (j → i+1 to n-1)
  { for (k → j+1 to n-1)
   }
  }
}
```

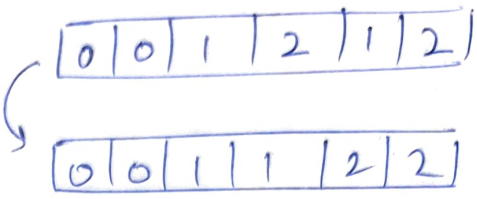
Q.8) Sort 0 1




```
void SortOne (int arr[], int n){
int left = 0;
int right = n-1;

while (left < right) {
```

```
while (arr[left] == 0 && left < right)
{
    left++;
}
while (arr[right] == 1 && left < right)
{
    right--;
}
if (left < right) {
    swap(arr[left], arr[right])
    left++;
    right--;
}
}
}
```

Q.9) Homework (sort 0, 1, 2)



Time & Space complexity

→ Time taken by algorithm to run as a function of length of the input.

Big O notation (worst case)

Theta Θ (avg case)

Omega Ω (lower bound)

Constant — O(1)

linear — O(n)

Quadratic — $O(n^2)$

Cubic — $O(n^3)$

logarithmic — O(log n).

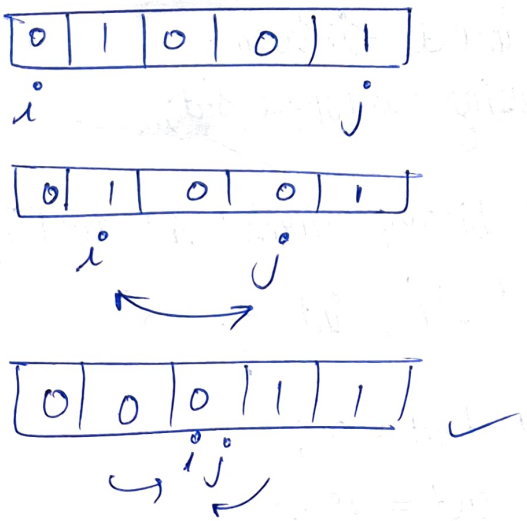
factorial = O(n!)

$2n^2+3n \longrightarrow O(n^2)$

$12001 \longrightarrow O(1)$

$5n^2 + \log n \longrightarrow O(n^2)$

$\frac{n^3}{300} \longrightarrow O(n^3)$



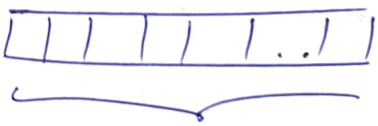
lowest to highest Complexity

1) O(1)
2) O(log n)
3) O(n)
4) O(n log n)
5) O(n²)
6) O(n³)
7) O(2ⁿ)
8) O(n!) → highest time (slowest output)

Print array time Complexity.

FOR(0 → n) → O(n) linear

Reversing an Array



n length

→ $\frac{n}{2}$ swaps → O(n) linear

Eg)
```
FOR(0 → n) {
}
FOR(0 → m) {
}
```
O(n+m) sum

```
for(0-n) {
    for(0→m) {
    }
}
```
O(n×m) Product for nested loops.

Space Complexity

→ memory to be consumed by the algorithm.

int a
int b
int arr[5] } O(1)

Binary Search

→ array should be sorted (monotonic order) either inc↑ or dec↓ order

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

int binary search (int arr[],
int size, int key) {

int start = 0;
int end = size-1;
int mid = (start + end) / 2;
```

```
while (start <= end) {
    if (arr[mid] == key) {
        return mid;
    }
    if (arr[mid] < key) {
        start = mid+1;
    } else {
        end = mid-1;
    }
    mid = (start+end)/2;
}
return -1;
}

int main() {
    int A[10] = {1,2,3,4,5,6,7,8,9,10}
    int size = 10; int key;
    cout << " Enter element to search" << endl;
    cin >> key;
    Int Index = binarysearch(A,10,5);
    cout << " 5 is at Index" << Index
        << endl;
```

o/p = 5 is at Index 4 ;

LIMIT VALUES

mid = (s+e)/2

but what if $s = 2^{31}-1$

$e = 2^{31}-1$

then s+e = out of range.

Soln ⇒

$$mid = s + \frac{(e-s)}{2}$$

$$\boxed{mid = s + (e-s)/2}$$ //

Time Complexity

Linear → O(n)

Binary → O(log n)

Questions

Q.) In given sorted array find leftmost and rightmost occurance of an element.

soln → logic

leftmost

↳ find mid

↳ proceed left subpart of array for every successful iteration of mid

↳ else (mid != element) proceed to right subpart.

Leftmost occurrence

```
int firstOcc (int A[], int n, int key){
    int s=0, e=n-1;
    int m = (s+e)/2;
    int ans=-1;
    while (s<=e) {
        if (A[mid] == key) {
            ans = mid;
            e = mid-1;
        }
        if (key > arr[mid]) {
            s = mid+1;
        }
        else {
            e = mid-1;
        }
        mid = s+(e-s)/2;
    }
    return -1;
}
```

Rightmost occurrence

```
int lastOcc ( ) {
    int s=0, e=n-1;
    int m = (s+e)/2;
    int ans = -1;
    while (s<=e) {
        if (A[m] == key) {
            ans = mid;
            s = mid+1;
        }
        if (key > arr[mid]) {
            s = mid+1;
        }
        else {
            e = mid-1;
        }
        mid = s+(e-s)/2;
    }
    return -1;
}
```

```
pair<int, int> p;
p.first = 5;
p.second = 11;
```

Q.) Find total number of key (occured) in sorted array.

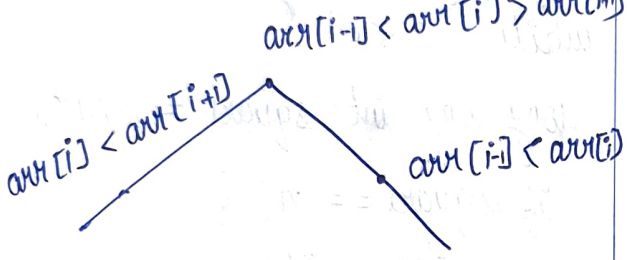
T = Last Index - First Index + 1

Q.3) Find peak index in mountain array.

0	1	2	3	4
0	3	10	5	2

E.g

ans = 2




```
int PeakIndex (arr[], size) {
    int s = 0;
    int e = size-1;
    int m = s + (e-s)/2;
    while (s<e) {
        if (arr[m] < arr[m+1]){
            s = mid+1;
        }
        else {
            e = mid;
        }
        m = s + (e-s)/2;
    }
    return s;
}
```

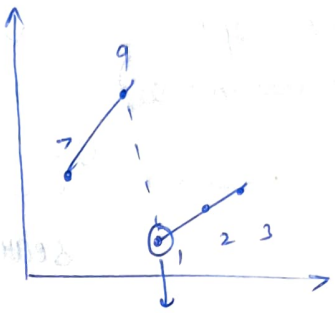
Q.) Find pivot in an Array.

sorted array

1	2	3	7	9

rotated array

7	9	1	2	3



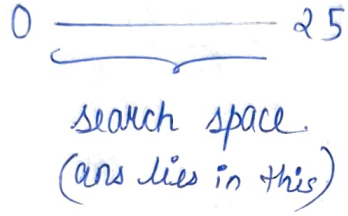
pivot element is 1.

o/p → 1

code -

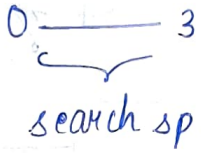
Q) Find square root using binary search (BS).

i/p → n = 25
o/p → ans = 5



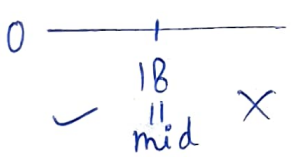
search space
(ans lies in this)

i/p = 3
o/p = 1 7 13 = 2

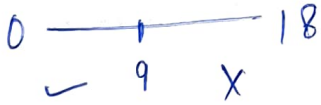


search sp.

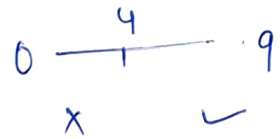
Eg n = 36



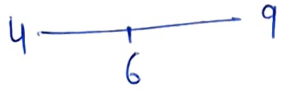
18 * 18 = 324 > 36



9 * 9 = 81 > 36



4 * 4 = 16 < 36



6 * 6 = 36 = 36

return mid = 6

code (for integer part)

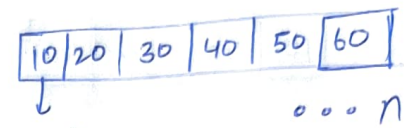
```
class solution {
public:
    long long int binarySearch (int n) {
        int s = 0;
        int e = n;
        long long int mid = s + (e-s)/2;
        long long int ans = -1;
        while (s <= e) {
            long long int square = mid * mid;
            if (square == n) {
                return mid;
            }
            if (square < n) {
                s = mid + 1;
            }
            if (square > n) {
                e = mid - 1;
            }
            mid = s + (e-s)/2;
            ans = mid;
        }
        return ans;
    }
};
```

(for float Part)

```
double morePrecision (int n,
int precision, int tempSol){
double factor = 1;
double ans = tempSol;
for (int i=0; i< precision;
    i++) {
    factor = factor / 10
    for (int j=0; j*j<n; j=j+
                              factor){
        ans j;
    }
}
return ans;
}
int main () {
    int n;
    cout << " Enter the number" <<
    endl;
    cin >> n;
    int tempSol = sqrt Integer (n);
    cout << " Answer is " <<
    morePrecision (n, 3, tempSol)
    << endl;
    return 0;
}
```

Q.) Book Allocation problem

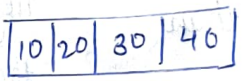
→



arr — pages of books ... n books

- atleast 1 book per student
- no book should left in arr.
- contiguous manner.

Eg



n = 4
m = 2

			max
(10)	10 | 20 30 40	(90)	90
(30)	10 20 | 30 40	(70)	70
(60)	10 20 30 | 40	(40)	60

60 40

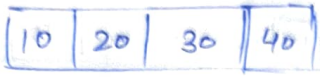
allocate such that the max^m number of pages assigned to a student is minimum.

↳ in all possible way, max pages for one student should be minimised. (90, 70, 60 → 60)

RETURN 60 as ans

NOTE → Basically, when we can neglect the left or right part by finding a possible / NOT possible solut^n on sorted ans search space we use BS binary search.

same case m = 2 n = 4



sum

min = 0 max = 100

search space.

DRY RUN.

I → $\frac{0+100}{2}$ = (50) → 10+20+30

II → 30+40 is this a possible soln

III → 40

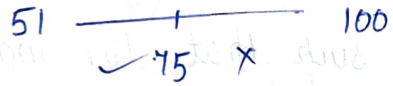
yes NO

neglect more than 50 solutions.

e = mid - 1

neglect less than 50 ans.

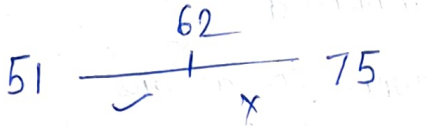
s = mid + 1



10 20 30 | 40

I → 10+20+30+40 → more than 75.

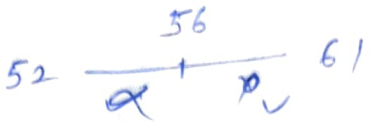
II → 40



mid = $\frac{51+75}{2}$ = 62.

I → 10+20+30+40 → gives more than 62

II → 40

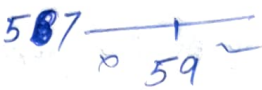


mid = 56.

I = 10+20+30

II = 30+40 X

III = 40



57
61
118
59

I → 10+20+30 X

II → 30+40

III → 40

I → 10+20+30+40

II → 40

(60)

CODE

```
int allocateBooks(vector<int>, int n,
                  int m){
```

```
bool isPossible(vector<int> arr,
                int n, int m, int mid){
```

```
int studentCount = 1;
int pageSum = 0;
for (int i=0; i<n; i++){
  if ( PageSum + arr[i] <= mid)
  {
    pageSum += arr[i];
  }
  else
  {
   studentCount++;
   if ( studentCount > m || arr[i]
                          > mid){
        return false;
   }
   pageSum = arr[i];
  }
}
return true;
}

int bookAllocation (vector<int> arr,
        int n, int m) {
  int s = 0;
  int sum = 0;
  for ( int i = 0; i<n; i++) {
     sum += arr[i];
  }
  int e = sum;
  int ans = -1;
  int mid = s + (e-s)/2
  while (s<=e)
  {
   if (isPossible(arr, n, m, mid))
   {
      ans = mid
      e = mid-1;
   }
   else {
     s = mid+1;
   }
   mid = s + (e-s)/2;
  }
  return ans;
}
```

Q:) Painters Partition Problem
(Same question as book allocation)

give minimum (max) of time from all cases.

K = no of Painters

eg

5	5	5	5

min = 0 max = 20

← search sp →

mid = 10

I → 5+5 ⑩

II → 5+5 ⑩

is mid a possible soln

yes

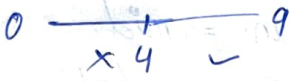
e — m — s

store ans

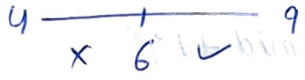
No.

e — m — s

x ✓



I → ~~5~~ ∝



I → 5 + ~~5~~

II → 5

III → 5 ∝ (K=2)

I → 5 + ~~5~~

II → 5 + ~~5~~

III → ∝.

Return largest minimum distance.

Distance bet^w stalls is difference bet^w positions of stalls (elements)

K = 2

e.g.

dist	4	2	1	3	6
2	C_1	C_2			
3	C_1		C_2		
1	C_1			C_2	
2	C_1				C_2
1		C_1	C_2		
1		C_1		C_2	
4		C_1			C_2
2			C_1	C_2	
5			C_1		C_2
3				C_1	C_2

these all are minimum distance from each other.

in which 5 is largest

return 5

Aggressive Cows Problem

Given array length = N

each element denotes position of a stall. K = no. of cows.

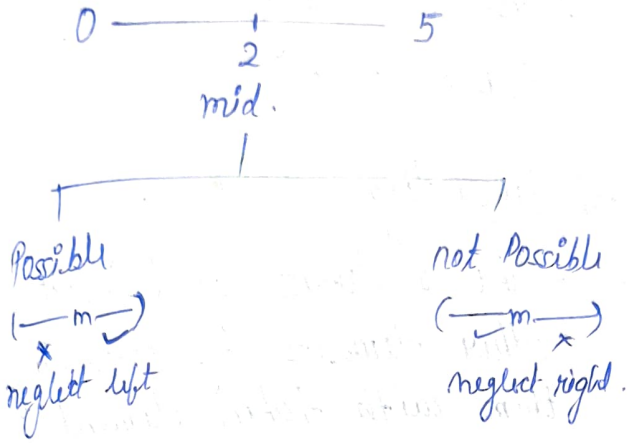
assign cows to stalls such that they have largest distance.

search space

min = 0

max = max posi − min posi

Return largest minimum distance.



SORTING OF ARRAY

① Selection Sort

→ find minimum ~~of~~ element in array and swap with left most part

SELECTION OF MINIMUM

eg | 64 | 25 | 12 | 22 | 11 |

pass I → 11, 25, 12, 22, 64 (min = 11) i=0

pass II → 11, 12, 25, 22, 64 (12) i=1

pass III → 11, 12, 22, 25, 64 (22) i=2

pass IV → 11, 12, 22, 25, 64 (25) no swap i=3

output = | 11 | 12 | 22 | 25 | 64 |

Total passes = n-1

```
void selectionSort (vector<int>& arr, int n) {
  for (int i=0; i<n-1; i++) {
    int minIndex = i;
    for (int =0; i < size-1; i++){
      if (arr[minIndex] > arr[j])
      {
        minIndex = j;
      }
    }
    swap (arr[minIndex], arr[i]);
  }
}
```

---

space complexity.

Best case — $O(n^2)$

worst case — $O(n^2)$.

---

stable sorting → same no. in array stays at same order after sorting

---

Bubble Sort

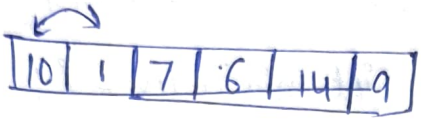
→ | 10 | 1 | 7 | 6 | 14 | 9 |

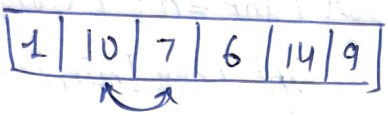
Taking element of left most part and comparing with it's right part.

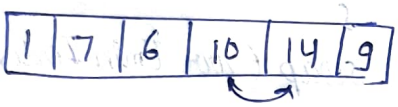
Here comparing 10 & 1

In each round largest gets placed.

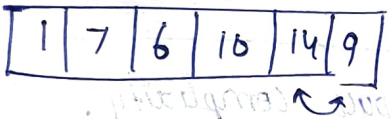
R1.

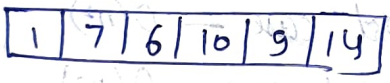






Since 10 < 14
move to ~~greate~~ next elemt

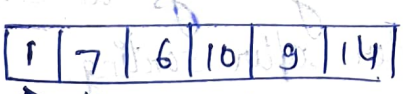




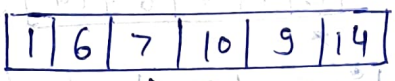
Round 1 completed.

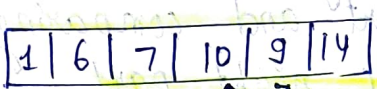
No. of rounds = n-1

R2

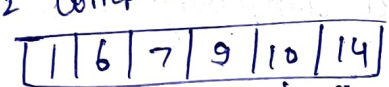


next





Round 2 completed.



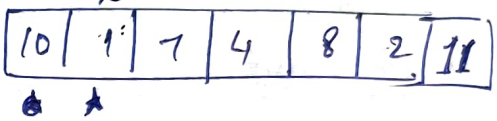
X

space complexity = O(1)
time complexity = $O(n^2)$
stable sort

Insertion Sort

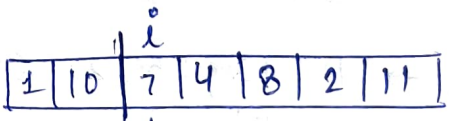
→ Take 1st element and insert other elements by comparing them with taken element

e.g

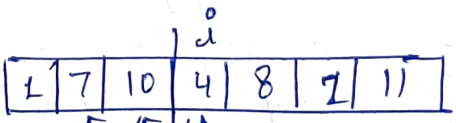


1 (10) _
↑
2 4 7 8 11

1) compare 10 & 1 10 > 1
1 left side of 10.
shift 10 'R' right side by 1



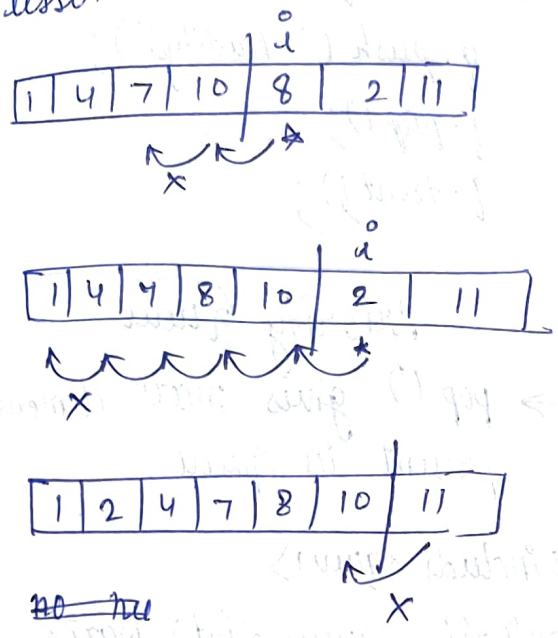
2) now i take 7



shift 10 right

3) Take 4
compare '10' with 4 10 > 4
compare '7' with 4 7 > 4
compare '1' with 4 1 < 4
STOP

shift till taken element is lesser



code on NS code ~

space → O(1)

Time → $O(n^2)$

Best case → already sorted.

worst case → If you want inc↑ order but array is sorted in dec↓ order.

STL in CPP

- Algorithms
  - ↳ binary search
  - ↳ lower / upper bound
  - ↳ min / max
  - ↳ reverse / rotate
  - ↳ sort / swap etc.
- Containers

Containers

- Sequence Containers
  - Array
  - ↓ vector
  - ↓ Deque
  - ↓ List
  - ↓ forward list
- Container Adaptors
  - stack
  - ↓ stack
  - ↓ Queue
  - ↓ Priority Queue.
- Associative Containers
  - Set
  - ↓ Map
  - ↓ Multiset
  - ↓ MultiMap.
- Unordered Associative.
  - Unordered Set
  - ↓ Unordered Map
  - ↓ Unordered Multiset
  - ↓ Unordered MultiMap

vector Data type

→ similar to array but it doubles it's capacity when we try put elements when the vector is full.

```
# inc <iostream>
# include <vector>
using namespace std;
int main() {
 vector <int>
```

```
v.capacity();
v.size();
v.front();
v.back();
v.push_back(1);
v.push_back(387);
v.pop_back(387);
```

V.b

Queue

FIFO

```
#include <queue>
q.push("Aaditya");
q.pop();
q.front();
```

Priority Queue

→ pop() gives max^m element present in queue.

```
#include <queue>
priority_queue<int> maxi;
priority_queue<int, vector<int>, greater<int>> mini;
maxi.push(1);
  -//- (3);
  -//- (2);
  -//- (0);

for{
    pop.maxi();
}
```

o/p = 3 2 1 0

Dequeue (size is const).

```
#include <deque>
int main(){
  deque<int> d;
  d.push_back(1);
  d.push_front(2);
  d.pop_front();
  d.empty(); → erase all elements.
  d.erase(d.begin(), d.begin()+1);
```

List

```
#<list>
  list<int> l;
```

Set

→ Stores only new unique element.

→ returns in sorted way.

```
#inc<stack>
  s.push("love");
  s.push("babbar");
  s.pop();
```

LIFO

o/p → love, babbar (popped)

```
#include <set>
main() {
    set <int> s;
    s.insert (5);
    s.insert (5);
    s.insert (4);
    s.insert (1);
    s.insert (3);
for {
    s.pop() << endl;
}
```

o/p → 1 3 5 7

```
s.count(5);
```

1 → element present

0 → absent.

Map

```
#include <map>
using n....
main() {
    map <int, string> m;
    m[1] = "babbar";
    m[2] = "love";
    m[13] = "kumar";
    for (~~string~~ auto i : m) {
        cout << i.first << endl;
    }
}
```

o/p
1
2
13

```
    for (auto i : m) {
        cout << i.first << " " << i.second
                                << endl;
    }
    m.insert ( {5, "bheem"} );
```

o/p
1 babbar
2 love
5 bheem
13 kumar

Algorithm

```
#include < algorithm>
main() {
    vector <int> V;
    V.push_back (1);
    V.push_back (3);
    ——»—— (6);
    ——»—— (7);
cout << binary_search (V.begin (),
~~V.begin~~ V.end(), 6) << endl;

lower_bound (V.begin(), V.end(),6)
        - V.begin()

upper_bound (V.begin(), V.end(),4)
        - V.begin()

max(a,b)
min (a,b)
swap (a,b)
```

Vector Detail f^n(s)

vector <datatype> vectorname;

* Iterators
1) v.begin()
2) v.end()
3) v.rbegin()
4) v.rend()

2) v.at(g); Return element at
3) v.front();
4) v.back();
5) v.data();
↳ returns direct pointer to the memory array used internally by the vector to store its owned elements.

---

add element(s) in front by:
v.insert(v.begin(), 56);

---

* capacity
1) v.size(); → no. of elements
2) v.max_size();
3) v.capacity(); → capable of holding
4) v.resize(n); → resize to n.
5) v.empty(); → 0 (empty), 1 (not empty)
6) v.shrink_to_fit();
7) v.reserve;

---

* Data / Element access
1) v.reference_operator[g];
↳ returns a refrence to the element at position 'g'.

Modifiers
1) v.assign();
→ assigns new value to the vector elements by replacing old ones
2) push_back();
3) pop_back();
4) v.insert();
→ inserts new elements ~~from a vect~~ before the element at specified position.
5) v.erase();
↳ removes a range of elements
6) swap();
7) emplace();
↳ it extends the container by inserting new elements at position.
8) emplace_back(); adds at end.

```
string abcd = "abcd";
reverse (abcd.begin(), abcd.end());
rotate (v.begin(), v.end());
sort (v.begin(), v.end());
```

sort → quicksort, merge, heapsort.

Character Arrays and string.

char a = 'z'; } stores only 1 charac.

→ 1 dimension character arrays ending with '\0' aka strings

input syntax

↳ char name [20];

i/p cin >> name;

Aaditya

A	a	d	i	t	y	a	\0			...

\0 ↓ terminator (NULL)

ASCII value } 0

o/p → cout << name;

cin >> stops taking input if given space in char array.

Q.) Check if string is Palindrome.

→ if reversing the string does affect string order

e.g जहाज, malayalam

Q.) Convert capital letters to small letters.

'A' . . . 'Z'

'a' . . . . 'z'

e.g char ch = 'B'

ch = ch - 'A' + 'a';

ch = b

small to capital

ch = ch - 'a' + 'A'

Q.) get ASCII value

```
main() {
    char ch = 'A';
cout << "ASCII value of" << ch <<
    " is" << int(ch);
}
```

o/p

ASCII value of A is 65

ASCII value diff betw capital & small letters is 32

'A' - 'a' = 32

'B' - 'b' = 32 etc.

using string to store sentences.

```
#include <stdio.h> <iostream>
using namespace std;
int main(){
    char str[1];
    cin.getline(str,50);
    cout << str << endl;
    return 0;
}
```

Q.) Replace spaces from string with @40

e.g My name is Aaditya

My@40name@40is@40Aaditya

```
string replaceSpaces(string &str){
    string temp = "";
    for(int i=0; i<str.length(); i++){
        if(str[i] == ' '){
            temp.push_back('@');
            temp.push_back('4');
            temp.push_back('0');
        }
        else{
            temp.push_back(str[i]);
        }
    }
    return temp;
}
```

Q.) Remove all occurances of substring

e.g given dababcbc

input abc

o/p dabbc

```
string removeOccurances(string s, string part){
    while(s.length() != 0 && s.find(part) < s.length()){
        s.erase(s.find(part), part.length());
    }
    return s;
}
```

Q.) Permutation of string

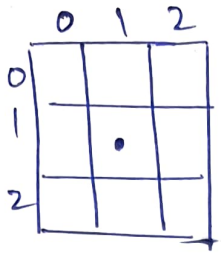
return true if one of s1's permutations is the substring of s2.

e.g 1) "eidbaoo" s2 } True
"ab" s1

2) "eidboao" s2 } false
"ab"

window approach.

2D - Arrays



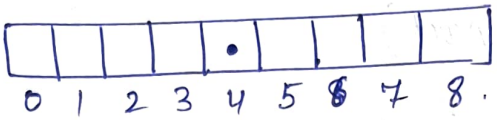
visualisation as 2D arr.

r = rows
c = cols
i = rows
j = cols.

c = 3

index = $ci + j$

3(1) + 1 = 4



← internal memory →

syntax : int arr[3][3];

```
cin >> arr[i][j];
cout << arr[i][j];
```

ROW COLUMN

```
int arr[3][4] = {{1,11,111,1111}, {2,22,222,
    2222}, {3,33,333,3333}};
```

→ this is row wise input

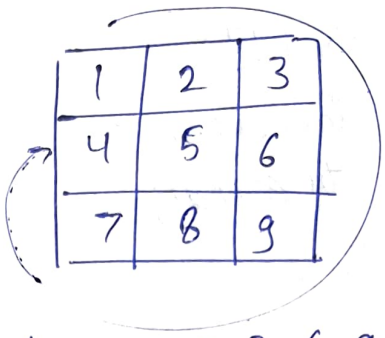
```
o/p   1  11  111  1111
      2  22  222  2222
      3  33  333  3333
```

NOTE → we need to specify the column size when passing 2D array as a parameter because arrays are stored in memory row by row in contiguous block.

Q.) P

vector<vector> → 2D vector

Q.) Spiral print of 2D array or 2D vector.



o/p = 1 2 3 6 9 8 7 4 5

Approach

1) Print 1st Row
2) Print last col
3) Print last row
4) Print starting col.

```
vector<int> spiral(vector<vector
    <int>> &matrix) {
    int row = matrix.size();
    int col = matrix[0].size();
    int count = 0;
// to count how many elements
    are being printed //
    int total = n*m; row * col;

    int startingRow = 0;
    int startingCol = 0;
    int endingRow = row-1;
    int endingCol = col-1;
```

```
while (count < total) {
for (int index = startingCol;
    count < total && index
    <= endingCol ; index++){
    count++;
}
ans.push_back (matrix
    [startingRow][index]);
}
startingRow++;
```

```
for (int index = startingRow;
  count < total && index <=
  endingRow ; index++) {
ans.push_back (matrix[index][
endingCol]);
  count++;
}
endingCol--;
```

```
endingRow--;
```

```
startingCol++;
```

```
return ans.
```

Q.) Binary Search in 2D array.
↳ (sorted).

e.g.

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

→ convert to linear array

Q.) Rotate Matrix

Q.) Search 2D Matrix

Maths for DSA.

1) Prime no. (for (i=2 to n-1))
n%i;
}

#Sieve of Eratosthenes.

```
→ int countPrime (int n) {
    int count = 0;
    vector<bool> prime (n+1, true);
    prime[0] = false;
    prime[1] = true false;
    for (int i=2; i<n; i++) {
        if (prime[i]){
            count++;
        }
        for (int j*2; j<n; j=j+i)
            prime[j] = 0;
        }
    }}
    return count;
  }
};
```

$O(n \log(\log n))$

GCD, greatest common divisor
or HCF

$gcd(a,b) = gcd(a-b, b)$

↓ OR

$gcd(a\%b, b)$

eg $g(72,24) \rightarrow g(48,24) \rightarrow$

$g(24,24) \rightarrow g(0,24)$

↓

if one becomes zero other number is ans.

Fast exponential.

a^b ; → $T(n) \in O(b)$

↳ normal loop

$a^b \begin{cases} \rightarrow (a^{b/2})^2 & b \text{ is even} \\ \rightarrow (a^{b/2})^2 * a & \text{odd.} \end{cases}$

```
int exp(int x, y) {
    int ans = 1;
    while (n>0)
    {
        if (n%2) {
            ans = ans * x;
        }
        x = x * x;
        n = n>>1;
    }
    return ans;
}
```

```
int exp (int x, y) {
```

Pigeon hole → Pigeons > holes

Catalan no.

↳ $C_n = \frac{1}{(n+1)} * {}^{2n}C_n$

or

$C_n = {}^{2n}C_n - {}^{2n}C_{n-1}$

$n \in 0$ to ∞

Pointers in C++

'address of' = &

Pointer stores address

eg `int * ptr = &nums;`

120	080
5	120
nums	ptr

---

eg
```
char ch = 'a';
char * p = &ch;
```

↳ p is pointer to address of ch

'*' value at address gives

```
cout << *p << *ptr;
```

o/p a 5

ch and *p will give same value.

address numbers are in hexadecimal forms.

•) size of pointer is always '8' byte. any datatype

---

same task - two syntax

```
int i=5
int *q = &i;
```

or

```
int *q=0;
q = &i
```

same

Types of Pointers

① Dangling Pointer

→ A pointer pointing to a memory location that has been deleted (or freed) is called a dangling pointer.

→ source of bug.

②



```
free(ptr);
```

③ Void pointer

```
void *ptr
```

→ doesn't have specific data types

→ Void pointers can't be dereferenced

however it can be done using typecasting.

```
int x=4;
```

e.g

```
void *ptr = &x;
printf("%d", *((int*)ptr));
```

op = 4.

---

③ NULL pointer

e.g

```
int *ptr = 0;
```

or

```
int *p = NULL;
```

→ Array name is pointer to array itself.

---

arr[i] = *(arr+i)

internal execution for evaluation

---

i[arr] = arr[i]

are same thing.

---

sizeof (arr) = n x 4 = size of int

↓ no. of int

sizeof (ptr) = 8 {stores add}

Cout f^n for int array and char array are differently implemented

```
char *c = &ch[0];
cout << c << endl;
```

↳ prints entire string.

pointer and functions

1) When we passed pointer in f^n; original value is change (value in main f^n) because it goes to address for modification.

2) When array is passed in f^n, only pointer to first index of array is passed (created or accessed.

↳ benifit: you can send part of array

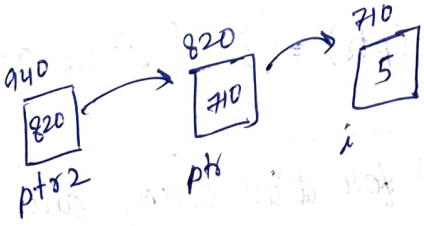
e.g.

function (arr + 3)

↳ argument

Double pointer

```
int i = 5;
int * ptr = &i;
int ** ptr2 = &ptr;
```



you can't change this.

sym table

i → 710
ptr → 820
ptr2 → 940

Double ptr & func.

```
funct (int ** p) {
    **p = **p + 1;
}
```

Reference Variable → (Same memory diff names)

```
int i = 5;
int& j = i;
```

i [5] j
7010

why?
→ to use "pass by reference" in function argument.

update (int& n).

Return by Ref (Bad Practice)

```
int& update (int n)
{
    int a = n;
    int& num = a;
    return num;
}
```

But we don't know whether 'num' is a global or local variable

It will die outside function block (if local). So nothing is being returned.

Arrays

```
int n;
cin >> n;
int arr[n];
```
↳ Bad practice

Size of array should be known at compile time not runtime

---

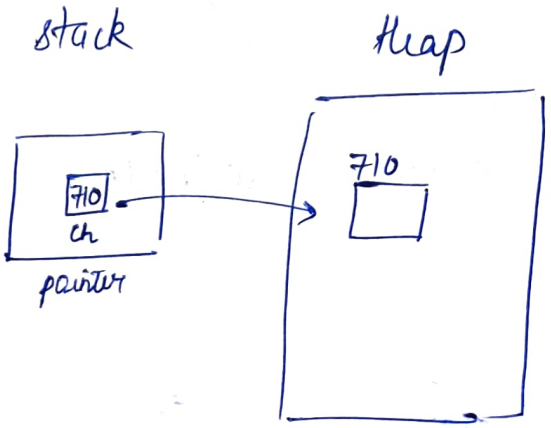
memory allocations.

stack (static)	Heap (dynamic)
eg int a = 5;	'new' keyword
int arr[10];	eg new char;
char c;	↳ address return
	(no name allowed)

Heap

- no variable names allowed.
- let's create a pointer to pointed our dynamically allotted memory.

```
char *ch = new char;
```



dynamic Array

~~new int[5];~~

int *arr = new int[5];

total memory used

① pointer always = 8 → stack

② array = 4×5 = 20 → heap

8 + 20 = 28

NOTE:

allotted memory in heap doesn't get freed after lifespan of var is over.

But in stack allocation, it gets automatically freed.

To free

int *arr = new int[50]

delete []arr;

---

int n

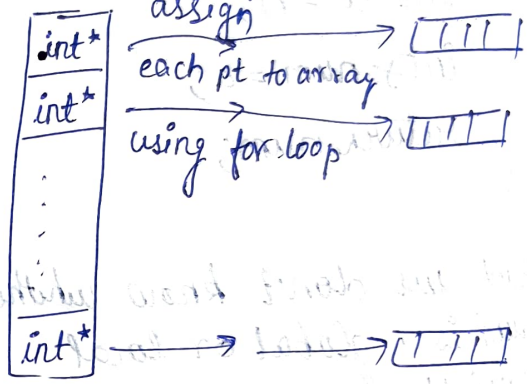
int *arr = new int[n]

2D Array (static) arr[n][n]

int *arr = new int[n]

int **arr = new int*[n];

↳ represents array which contains pointers to multiples dynamic arrays



---

code

```
#include <iostream>
using namespace std;
int main() {
    int n
    cin >> n;
    int ** arr = new int*[n];
    for ( int i=0 ; i<n ; i++) {
                    no. of rows
        arr[i] = new int[n];
    }
```

lec 30

Macros

```
for(int i=0; i<n; i++){
    for (int j=0; j<n; j++){
        cin >> arr[i][j];
    }
}
```

Printing

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

---

* Always free used memory used from Heap, manually.

```
for(int i=0; i<row; i++){
    delete [] arr[i];
}
delete [] arr;

return 0;
}
```

```
#define PI 3.14
```

PI is a Macro (requires no memory)

→ a piece of code in a program that is replaced by value of macro. (before compile time)

```
#define Micro_Name Macro_def.
```

(usually uppercase)

Types

① Object like M (simply replaces)
② Function like M (takes argument)
③ Multiline M (same as function)
④ Chain Macro. ex CLERK to WORKER → CLERK

Global Variables

↪ Do not use G.V for variable sharing.
↪ Any function can change G.V
↪ Use Refrence Var

Inline functions

→ reduces the function calls overhead.
→ if body has 1 line body then use inline function.
→ It replaces f^n call by f^n body just before compilation.

Default Arguments

→ we want to make one argument optional.

syntax

```
void print (int arr[], int size,
            int start = 0) {
    for( ) {
        cout
    }
}
```

NOTE : DA starts from Rightmost to left most arguments.

Recursion lec 31

→ When a function calls itself

→ If question can be divided into multiple of same type of subquestions.

e.g. EXP. $2^n = ?$ $\quad 2^n = 2 \times 2^{n-1}$, $2^{n-1} = 2 \times 2^{n-2}$

$f(n) = 2 \times f(n-1)$

recurrence Relation

e.g. factorial

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$= 5 \times 4!$

$n! = n \times (n-1)!$

$f(n) = n * f(n-1)$

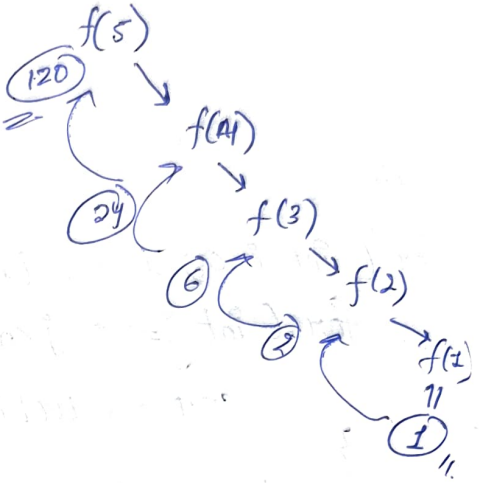
recursive relation

BASE CASE $1! = 1$

→ Must contain a base case / base condition.

8)

Recursion Tree



Q) Print Fibonacci series lec 32

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55

$f_1 = 0$

$f_2 = 1$

$f_3 = f_1 + f_2$

$f_4 = f_3 + f_2$

$f_5 = f_4 + f_3$

$f_n = f_{n-1} + f_{n-2}$

① $T_n = T_{(n-1)} + T_{(n-2)}$

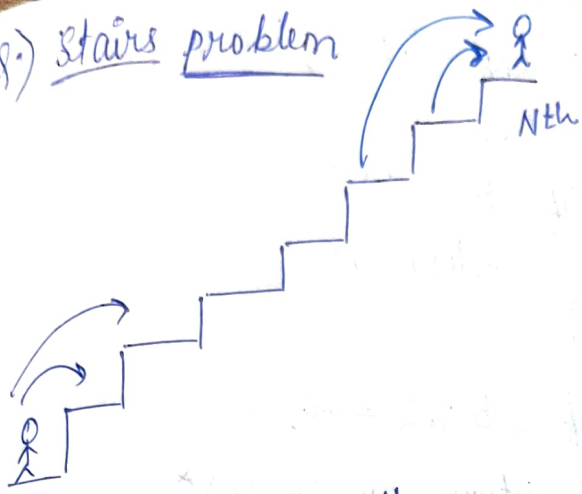
recurrence relat^n

② base case

$f(0) = 0$

$f(1) = 1$

Q.) Stairs problem



If you can go walk one or two steps at a time, then find number of ways to reach N^{th} stair.

→ $f(n) = f(n-1) + f(n-2)$

- $f(n)$: N th stair
- $f(n-1)$: steps till $(N-1)^{th}$ step
- $f(n-2)$: steps till $(N-2)^{th}$ step.

Recursive Function skeleton

```
int function (int a) {
  if (base case) {
      return value;
  }
  recursive relation
  return = f(a-1) + f(a-2)
}
```

Q.) Say digit

i/p → 4 1 2

o/p → four One two

---

lec 33 i) isSorted.

2) BinarySearch using recursion

2	4	6	9	11

H.W

Q) → sum of array using Recur

BinarySearch Using Recursion

```
bool BinarySearch (int *arr, int s,
    int e, int k) {
  # base case
  if (s > e)
      return false;

  int mid = (s+e)/2;

  if ( arr [mid] < k) {
    return BinarySearch (arr, mid+1, e,
                         k);
  } else {
      return BinSearch (arr, s, mid-1, k);
    }
}
```

---

lec 84

Q.) reverse a string. using Recursion

```
void rs (string& s, int i, int j)
if (i > j) { return; }
swap (s[i], s[j]); i++; j++;
return rs (s, i, j);
}
```

NOTE

→ always keep in mind that a function makes a copy of it's pass by value parameters.

→ 1) Pass by reference ↘ OR
2) Print string (copy) ↙ in function itself.

Q.2) Check Palindrome using Recurssion.

```
bool checkP(int i, int j, string s)
    if (i>j) {
        return true;
    }
    if (s[i]!=s[j]) {
        return false;
    }else {
        return checkP(i+1, j-1, s);
    }
}
```

Q.3) a^b optimised way.

b → even $a^b = a^{b/2} \times a^{b/2}$

b → odd $a^b = a(a^{b/2} \times a^{b/2})$

$2^{1024} = (2^{512})^2 = (2^{256})^2$

$(2^4)^2 \rightarrow (2^2)^2 \rightarrow (2^1)^2 \rightarrow (2^{128})^2$

$(2^8)^2 \leftarrow (2^{16})^2 \leftarrow (2^{32})^2 \leftarrow (2^{64})^2$

```
int exp(int a, int b) {
    // base
    if (b==1) {
        return a;
    }
    if (b%2==0) {
        return exp(a, b/2) * exp(a, b/2);
    } else {
        return a * exp(a, b/2) * exp(a, b/2);
    }
}
```

Bubble Sort using Recurssion

```
void sort(int *arr, int n) {
    if (n==1) {
        return;
    }
    for (int i=0; i<n-1; i++) {
        if (arr[i] > arr[i+1]) {
            swap(arr[i], arr[i+1]);
        }
    }
    sort(arr, n-1);
}
```

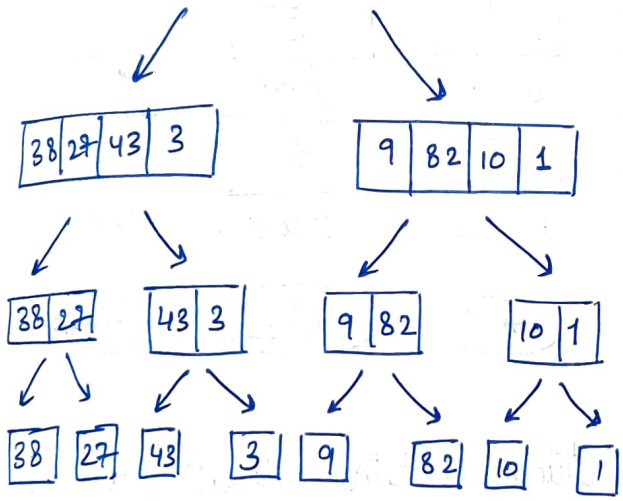
MergeSort Using Recursion

Lec 35

→ merging of two sorted *** arrays. (Algorithm)

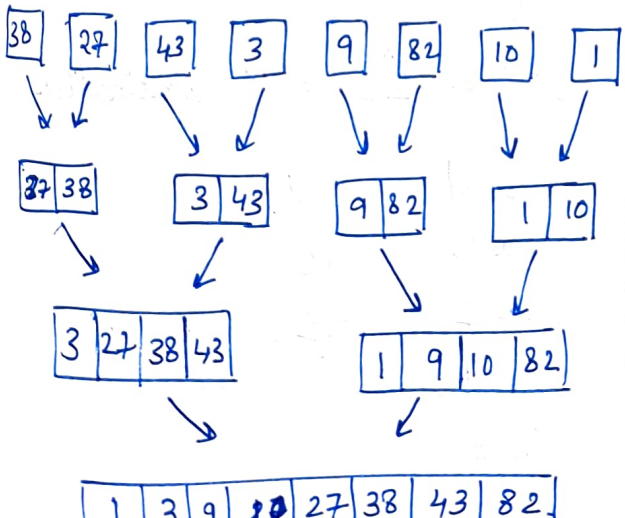
0	1	2	3	4	5	6	7
38	27	43	3	9	82	10	1

$mid = \frac{7}{2} = 3.$



→ ~~0 or~~ Arrays with single element are always sorted.

→ Now merge all sorted arrays one by one.



SORTED ARRAY

Implementation.

→ creating 2 new arrays each time and coping the elements.

---

space Complexity = O(n).

Time Complexity = O(n log n).

Lec 36 Quicksort Using Rec.

1) Take first element and place it in array such that all elements in left should be smaller and all in right shoulder greater.

e.g. Partition 1st

3	5	1	8	2	4

^

1	2	3	5	8	4

1st pass complete

< 3 Pivot element > 3

---

① Take 1st element as pivot

② Count elements less than pivot

③ Place pivot at count.

④ Swap elements using i & j index until all elements less than pivot comes to its left side and all greater elements to its right side.

Dry run

10	80	30	90	40	50	70

pivot = 10

count = 0

s = 0

pivot should be place at = s + c
= 0 + 0
= 0

right sort

80	30	90	40	50	70

count = 4

s = 1

1	2	3	4	5	6
50	30	40	70	80	90

> Quicksort is inplace sorting
> Doesn't require any extra space
> mergesort requires O(N) extra space hence less prefferd.

Both have time O(N log N).

Quicksort

avg = O(n log n)
best = O(n log n)
worst = O(n²)

Subsets & subsequency of string

~~Q1) return powerset of an array~~

lec 38.

Q.) Phone Keypad problem using recursion

1	2	3
e g	a b c	d e f
4	5	6
g h i	j k l	m n o
7	8	9
p q r s	t u v	w x y z
* +	0 ␣	⇧ #

string contains digit from 2 to 9

return all possible combinations of alphabets in any order.

e.g

string = 232 — 3x3x3 = 27 combinations

str = 23 — 3x3 = 9 combinations

a d c d
a e c e
a f c f.
b d
b e
b f

code

```
void solve (string digit, string output, int index, vector<string> &ans, string mapping[]){

    // base
    if (index >= digit.length()){
        ans.push_back(output);
        return;
    }

    int number = digit[index] - '0';
    string value = mapping[number];

    for (int i=0; i<value.length(); i++) {
        output.push_back(value[i]);
        solve (digit, output, index+1, ans, mapping);
        output.pop_back();
    }
}

main () {
    if (digit.length == 0){
        return ans;
    }

    string output;
    int index = 0;

    string mapping[10] = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};

    solve (digits, output, index, ans, mapping);

    return ans;
}
};
```

---

lec 39

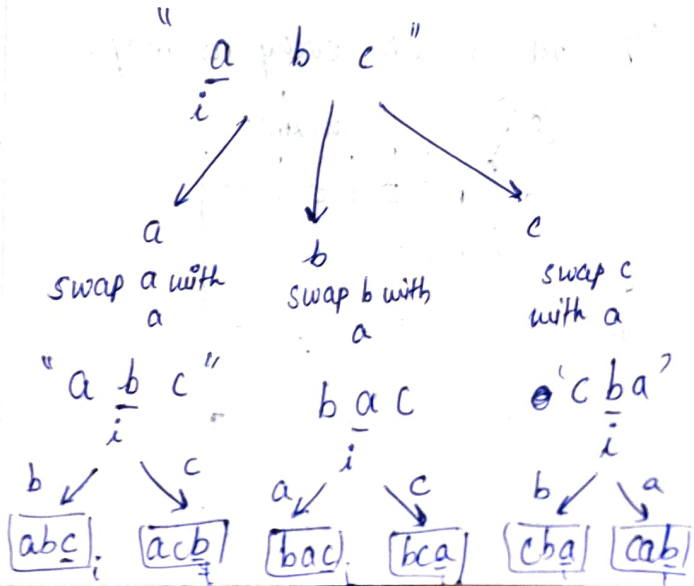
Q.) Permutations of a string

input = string = "abc"

output = abc, acb, bac, bca, cab, cba.

Total permutations = 6 = n!

Approach (Recursion Tree)



NOTE : Do swapping jab tak 'i' bahar na chala jaye.

code

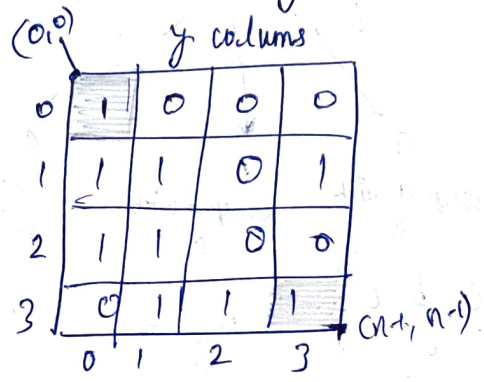
```
void solve (vector<int> nums,
vector<vector<int>> &ans, int index){
    if (index >= nums.size()){
        ans.push_back(nums);
        return;
    }

    for (int j=index; j<nums.size();
                                j++){
        swap(nums[j], nums[index]);
        solve(nums, ans, index+1);
        // backtrack
        swap(nums[j], nums[index]);
    }
}
```

---

lec 40

Q) Rat in a maze problem.

i/p m → 2D array as map.



move in four dir. LRDU

1 → open path

0 → closed path.

starting from → (0,0) → 1

destination → (n-1, n-1) → 1

give all possible solutions

① DDRDRR

② DRDDRR.

---

for x rows & y columns.

up = (x-1, y)

down = (x+1, y) top left (0,0)

left = (x, y-1)

right = (x, y+1). bottom right (n-1, n-1)

---

visited array

V	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

To move Rat to next cell we have to check 3 conditions

① m[i][j] == 1 {path exist}

② V[i][j] == 0 {unvisited}.

③ next cell should be a valid cell (inside m).

when function call returns, make sure u reset visited array.

code

```
void solve ( vector < vector < int >> & m , int n , vector . < string > & ans
             int x , int y , vector < vector < int >> visited , string path)
                                                                        {
    // reached (x , y)
    // base case
    if ( x == n-1 && y == n-1) {
        ans . push_back (path);
        return ;
    }

    visited [x] [y] = 1 ;

    // 4 choices  D, L, R, U  (lexi order)

    // down
    int newx = x+1
    int newy = y
    if ( issafe ( newx , newy , m , visited , n ) ) {
       path . push_back ( 'D' );
       solve ( m , n , ans , newx , newy , visited , path) ;
       path . pop_back ( ) ;
    }
    // Left
    newx = x ;
    newy = y-1;
    if ( issafe ( ... )) {
       path . push_back ( 'L' );
       solve ( ... );
    } pop_back ();

    // Right
    newx = x
    newy = y+1
    if ( is safl ) {
        push
        solve
        pop
    }
```

```
// up
    newx = x-1
    newy = y
    if (isSafe) {
        push
        solve
        pop
    }

    visited[x][y] = 0;
}
```

```
bool isSafe (int x, int y, vector<vector<int>> m, visited, int n){
    if ((x >=0 && x<n) && (y>=0 && y<n) && visited[x][y]==0
        && m[x][y]==1 ) {
            return true;
    } else {
        return false;
    }.
```

```
vector<string> findPath (vector<vector<int>> &m, int n) {
    vector<string> ans;
    if ( m[0][0]==0){
        return ans;
    }
    int srcx=0;
    int srcy=0;
```

```
vector < vector <int> > visited;
    for ( int i = 0; i<n; i++) {
        for ( int j=0; j<n; j++){
            visited [i][j] = 0;
        }
    }

    string path = " ";
    solve ( m, n, ans, srcx, srcy, visited, path);
    sort ( ans.begin(), ans.end());
    return ans;
}
```