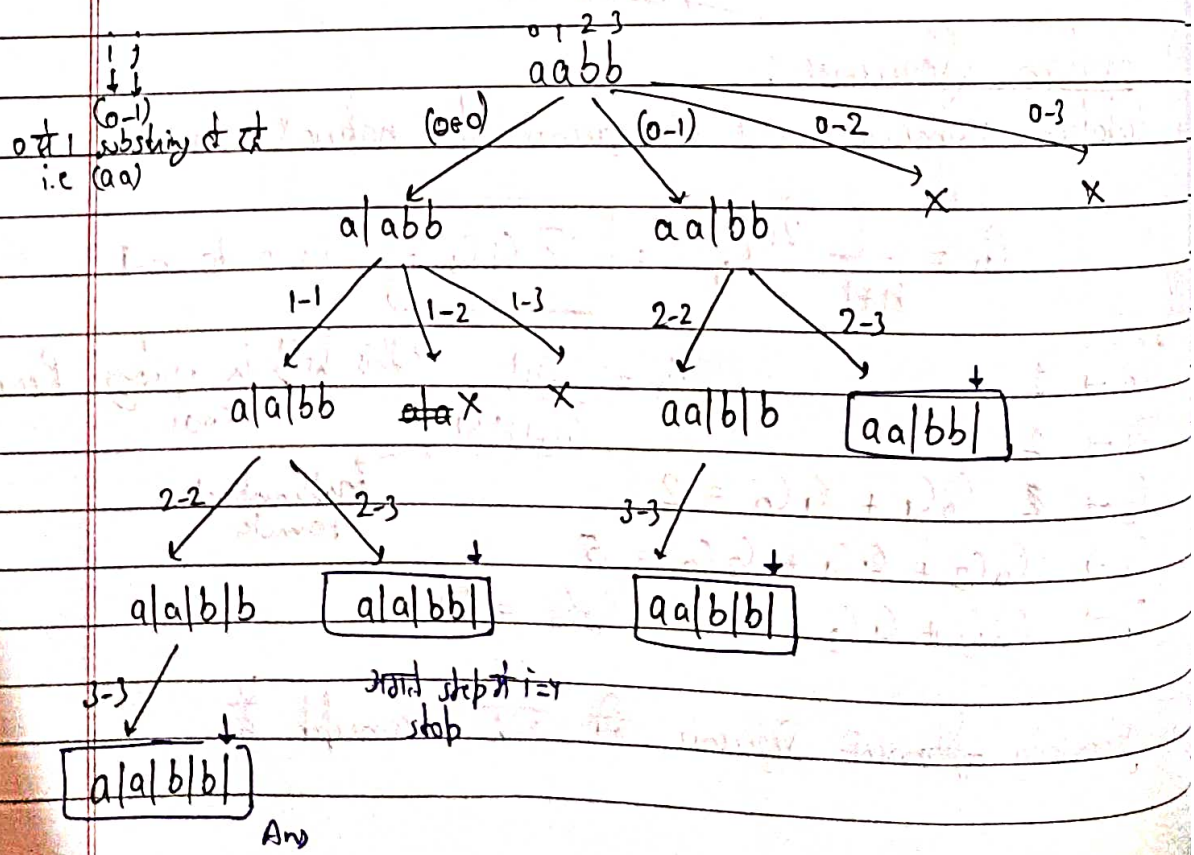
Code

```
int f(int n)
{
    if(n<=1)
        return 1;
    int ans=0;
    for(int i=0;i<n;i++)
    {
        ans += f(i) * f(n-i-1);
    }
    return ans;
}
```

(XVIII) Palindrome Partioning :

I/P → "aabb"

O/P → {{a,a,b,b}, {a,a,bb}, {aa,b,b}, {aa,bb}}



Code

```
class Solution{
    public:
    void f(vector<vector<string>> &ans, string s1, vector<string> &ds,
           int i)
    {
        if(i == s1.length)
        {
            ans.push_back(ds);
            return;
        }
        for(int j=i; j<s1.length(); j++)
        {
            string s = s1.substr(i, j-i+1);
            if(p(s))
            {
                ds.push_back(s);
                f(ans, s1, ds, j+1);
                ds.pop_back();
            }
        }
    }
    public:
    bool p(string s)
    {
        for(int i=0; i<s.length()/2; i++)
            if(s[i] != s[s.length()-i-1])
                return false;
        return true;
    }
    public:
    vector<vector<string>> partition(string s1){
        vector<vector<string>> ans;
        vector<string> ds;
        f(ans, s1, ds, 0);
        return ans;
    }
};
```

or s1.size()

palindrome check

s.find("str");
↑ string
return → index

eg: string s = "aabbc";
int ind = s.find('a');
cout << ind;
o/p → 0 → return first occurance index

s.find("aab") o/p → 0
s.find("bb") o/p → 2

s.substr(i, j) → j characters लेगा
↑ i index से start

eg: string s = "abcde";
cout << s.substr(1, 3);
o/p → bcd

logic
string s1 = "aabb"
हर step में partition कर दे देगा, partition के left वाला palindrome है, इसे check करना है तथा right में भी palindrome है।

a|a|b|b → palindrome है → now check for this
a|a|b|b → then check for this

(XIX) Sudoku Solver : (Only one solution we find)
(9×9)
O(9^(9×9)) → n×n has 9×9

```
class Solution{
public:
    void SolveSudoku(vector<vector<char>>&v){
        f(v);
    }
public f(vector<vector<char>>&v)
{
    for(int i=0;i<9;i++)
    {
        for(int j=0;j<9;j++)
        {
            if(v[i][j]=='.')   → character - don't need "." means
            {
```

count(a, a+n, 3);
counts no. 3 occurrence

→ valid for vector, array, string

index for accessing ith character of string

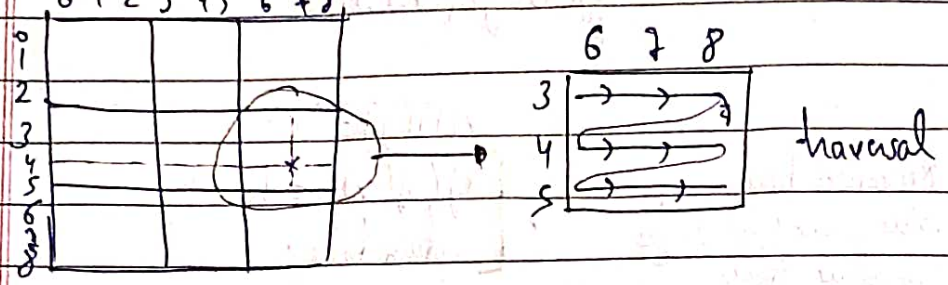
```
for(char c='1'; c<='9'; c++)
{
    if(issafe(v,c,i,j))
    {
        v[i][j]=c;
        if(f(v)==true)
        return true;
        else
            v[i][j]='.';
    }
}
return false;
}
}
}
return true;
}
bool issafe(vector<vector<char>> &v, char c, int i, int j)
{
    for(int k=0; k<=8; k++)
    {
        if(v[i][k]==c) return false;
        if(v[k][j]==c) return false;
        if(v[3*(i/3)+k/3][3*(j/3)+k%3]==c) return false;
    }
    return true;
}
```

Recursion here
सारे c value from '1' to '9' try
if false no value issafe
return false

logic of this ?

$v[3*(i/3)+k/3][3*(j/3)+k\%3]$

Say $i=4, j=7$ $v[4][7]$ we have to check $v[4][2]$ safe or not



ये ऐसा से देखो

for(k → 0 to 8) $i=4, j=7$

$v[i]$ i formula

in row=3 we want that i should be cond. & equal to
i.e 3+0

in row=4 we want that ——— 4
i.e 3+1

in row=5 ——— 5
i.e 3+2

$3 = 3+0$ } k
$4 = 3+1$ }
$5 = 3+2$ } → k/3

We get 0 when k/3 and k values from 0 to 2
We get 1 when k/3 & ——— 3 to 4
We get 2 when k/3 ——— 6 to 8

$[3*(i/3)+k/3]$

$\frac{0}{3} = \frac{1}{3} = \frac{2}{3} = 0$

for k → 0 to 2

Similarly Concept

for ~~v[i]~~ $[3*(j/3)+k\%3]$

$0\%3 = 0$
$1\%3 = 1$
$2\%3 = 2$

col

6 7 8

6+0 6+1 6+2

k%3 for k=0 to 2

6=3*(j/3)

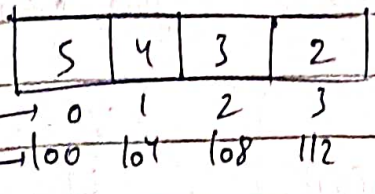
~~I/p vector<vector<char>> v;~~

Array → contiguous memory.

it is data structure

classmate Date Page

→ elements are stored in continuous Manner



Advantage
① Random Access (Indexing)
② Cache Friendliness
- it is memory closest to CPU
- fastest of all memory (RAM, Harddisk, secondary memory)

- Linked list, BST don't provide cache friendliness bz elements are stored at random location not in a continuous manner.

→ Types of Array:

① Fixed Sized Array:

```
int a[100];   } → SA
int a[n];
int *arr = new int[n]; → C++ | int *arr = malloc int[n]; } → HA
int a[] = {10,15,20} → SA
```

compiler itself initialize size = 3

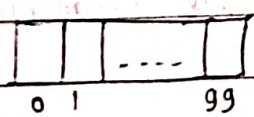
SA → stack allocated
HA → heap "

} basically internal representation of array is different

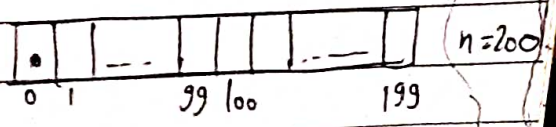
② Dynamic Sized Array:
Resize Automatically
C++ - Vector
Java - Arraylist
Python - list

O(1) → Vector insertion at end

Say vector



when we want to insert element at vector[100] location then vector size doubles



→ when again vector is full then again size double (0-399) n=400

Vectors in C++:

→ Dynamic Size
→ provides 8 inbuilt library functions. (find, erase, insert)
→ easy to know size

Array	Vector
int n = sizeof(arr)/sizeof(arr[0]);	int n = v.size();

→ array can't be returned but Vector can.
→ default value 0.
→ We can copy a vector (v1=v2;)

Operation on Array:

① Search
② Insertion

Insertion in Dynamic Sized Array:

Remember when Array(vector) is full then new array of double the size is created, and elements of previous array is copied in new array and old array is deleted.

③ Delete
first occurance is deleted

(I) Largest Element in an Array

I/P → a[10, 5, 20, 8]
O/P → 2 return index

Naive → $O(n^2)$
Efficient → $O(n)$

```
f(int a[], int n)
{
    int ms = a[0];
    for(int i=1; i<n; i++)
    {
        if(a[i]>ms)
            ms = a[i];
    }
    cout << ms;
}
```

**

```
ms = max(ms, a[i]);
```

↑
→ inbuild function in C++
→ #include<algorithm>

***

(II) Second largest Element

```
int SecondLargest(int a[], int n)
{
    int m = *max_element(a, a+n);
    int ms = INT_MIN;
    for(int i=0; i<n; i++)
        if(a[i]!=m)
            ms = max(ms, a[i]);
    if(ms==INT_MIN)
        return -1;
    return ms;
}
```

O(n)
two traversal

O(n) one traversal

```
int Secondlargest(int a[], int n)
{
    int l=a[0], sl=INT_MIN;
    for(int i=0;i<n;i++)
    {
        if(a[i]>l)
        {
            sl=l;
            l=a[i];
        }
        if(a[i]<l)
        {
            if(a[i]>sl)
                sl=a[i];
        }
    }
    if(sl==INT_MIN)
        return -1;
    return sl;
}
```

→ when 6 largest then l

→ when 6 smaller then l
→ now we check if 6 is greater sl if yes then sl=6;

when all element are same then sl not exist ∴ return -1

logic

1, 7, 5, 6

Say till here we have find out largest & second largest

→ When 6 come either it is > then largest or smaller then largest
→ equality ignore

(III) Check if an Array is Sorted :

Naive → $O(n^2)$
efficient → $O(n)$

```
bool f(int a[], int n)
{
    for(int i=1; i<n; i++)
        if(a[i] < a[i-1])
            return false;
    return true;
}
```

(IV) Reverse an Array : (O(n))

I/P → 5, 2, 1, 8, 4, 3 n=6
O/P → 3, 4, 8, 1, 2, 5

```
void reverse(int a[], int n)
{
    for(int i=0; i<n/2; i++)
    {
        swap(a[i], a[n-i-1]);
    }
}
```

swap → inbuilt function

→ reverse(a, a+n);
↓
reverse the array

→ in vector
reverse(v.begin(), v.end());

* When you want to use swap use while loop

```
while(low < high)
{
    swap(a[low], a[high]);
    low++;
    high--;
}
```

(V) Remove Duplicate elements from sorted Array :

Naive → O(n) , space = O(n)
efficient → O(n) " = O(1)

return new size of Array

```
int f(int a[], int n)
{
    int i=1;
    for(int j=1; j<n; j++)
    {
        if(a[j] != a[j-1])
        {
            a[i] = a[j];
            i++;
        }
    }
    return i;
}
```

eg: I/P → 2,2,3,4,5,5
O/P → 2,3,4,5

(VI) Move all zeros to end of array :

Naive → O(n²)
efficient → O(n)

c is count of positive numbers

```
void f(int a[], int n)
{
    int c=0;
    for(int i=0; i<n; i++)
    {
        if(a[i] != 0)
        {
            swap(a[i], a[c]);
            c++;
        }
    }
}
```

eg: I/P → {3,5,0,0,4}
O/P → 3,5,4,0,0

I/P → 3,5,7,2,0,0,4

यहा तक ठीक है, 4

4 != 0 ∴ swap(a[i], a[c])

↳ anticlockwise

(VII) Left Rotate an Array by One :

I/P → 1, 2, 3, 4, 5
O/P → 2, 3, 4, 5, 1

```
void f(int a[], int n)        O(n)
{
    int temp = a[0];
    for(int i=1; i<n; i++)
        a[i-1] = a[i];
    a[n-1] = temp;
}
```

(VIII) Left Rotate an Array by d :

(a) Naive : O(nd)

→ Left Rotate an Array by d (say 3) times means Left Rotate an Array by one d times.

eg: 1, 2, 3, 4, 5 ⇒ 2, 3, 4, 5, 1 ⇒ 3, 4, 5, 1, 2 ⇒ 4, 5, 1, 2, 3
d=3 ↑ o/p

(b) Using temp[] Array

~~simply copy~~

(c) Efficient O(n)

rotate by d means:

logic eg

1, 2, 3, 4, 5 ⇒ reverse a then reverse b then reverse whole array
d=2 (a | b)

→ reverse whole array

2, 1, 3, 4, 5 ⇒ 2, 1, 5, 4, 3 ⇒ 3, 4, 5, 1, 2
↑ reverse a ↑ reverse b ↑ final o/p

```
void Rotate(int a[], int d, int n)
{
    r(a,0,d-1);
    r(a,d,n-1);
    r(a,0,n-1);
}
void r(int a[], int i, int j)
{
    while(i<j)
    {
        swap(a[i], a[j]);
        i++;
        j--;
    }
}
```

(01)

```
reverse(a, a+d);
reverse(a+d, a+n);
reverse(a, a+n);
```

(IX) **Leaders in an Array :** Naive → $O(n^2)$, effective → $O(n)$

I/P → {16, 17, 4, 3, 5, 2}

O/P → 17, 5, 2

→ a no. is a leader if on the right every no. is smaller

→ last no. is a leader.

```
vector<int> leader(int a[], int n)
{
    vector<int> v;
    int m = a[n-1];
    v.push_back(m);
    for(int i=n-2; i>=0; i--)
    {
        if(a[i] > max)
        {   v.push_back(a[i])
            m = a[i];
        }
    }
```

```
    reverse( v.begin(), v.end());
    return v;
}
```

(X) Maximum Difference a[j]-a[i] such that j>i :

Naive → O(n²)
efficient → O(n)

I/P → {2, 3, 10, 6, 4, 8, 2}
O/P → 8

```
int f(int a[], int n)
{
    int m = a[0], ans = INT_MIN;   or  ans = a[1]-a[0];
    for(int i=1; i<n; i++)
    {
        ans = max(ans, a[i]-m);
        m = min(m, a[i]);
    }
    return ans;
}
```

(XI) Frequency in a Sorted Array :

I/P → {40, 50, 50, 50}	a[] = 5	1, 1, 2, 2, 2, 3, 4
O/P → 40 1	O/P 5 1	1 2
50 3		2 3
		3 1
		4 1

```
void f(int a[], int n)
{
    int c=1, i=1;
    while (i<n)
    {
```

```
while(a[i]==a[i-1] && i<n)
{
    c++;
    i++;
}
cout<<a[i-1]<<" "<<c<<endl;
i++;
c=1;
}
if(n==1 || a[n-1]!=a[n-2])
    cout<<a[n-1]<<" "<<1;
}
```

1, 1, 2, 2, 3 ✗

3!=2

(XII) Stock Buy & Sell :

I/P → {7, 1, 5, 3, 6, 4}

O/P → 7 (5-1=4 + 6-3=3) → 7

→ on each day you may decide to buy / sell the stock

```
int Profit(int a[], int n)
{
    int ans=0;
    for(int i=1; i<v.size(); i++)
    {
        if(v[i]>v[i-1])
            ans += v[i]-v[i-1];
    }
    return ans;
}
```

जो बाद stock बढ़ा add to ans.

Remember

1, 5, 6, 9

9-1 = 8

5-1+6-5+9-6 = 8

I/P → {7, 1, 5, 6, 9, 1, 4, 6, 3, 2}

b ↑ s ↑ b ↑ s ↑

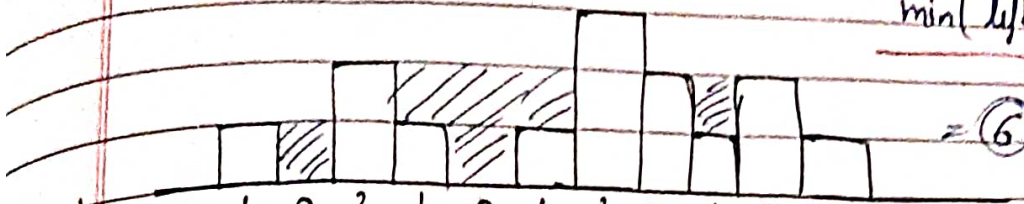
→ Remember We are calculating how much water building i store at top of it

→ index 0 & n-1 never contribute to ans.

(XIV) Trapping Rainwater problem:

Naive → $O(n^2)$

$\min(leftH(i), rightH(i)) - a[i]$



↑ logic

* i we find this

We maintain two array

prefix array $p[n] \rightarrow \{0,1,1,2,2,2,2,3,3,3,3,3\}$ $n=12$

suffix array $s[n] \rightarrow \{3,3,3,3,3,3,3,3,2,2,2,1\}$ $n=12$

$O(n)$ | space → $O(2n)$

```
int trap(vector<int> &a){
    int n=a.size();
    int p[n], s[n], ans=0;
    p[0]=a[0];
    for(int i=1; i<n; i++)
        p[i]=max(a[i], p[i-1]);
```

→ prefix array

```
    s[n-1]=a[n-1];
    for(int j=n-2; j>=0; j--)
        s[j]=max(a[j], s[j+1]);
```

→ suffix array

```
    for(int i=1; i<n-1; i++)
    {
        ~~ans += p[i] - s[i]~~
        ans += min(p[i], s[i]) - a[i];
    }
    return ans;
}
```