Notebook
Pg. No.
Date / /

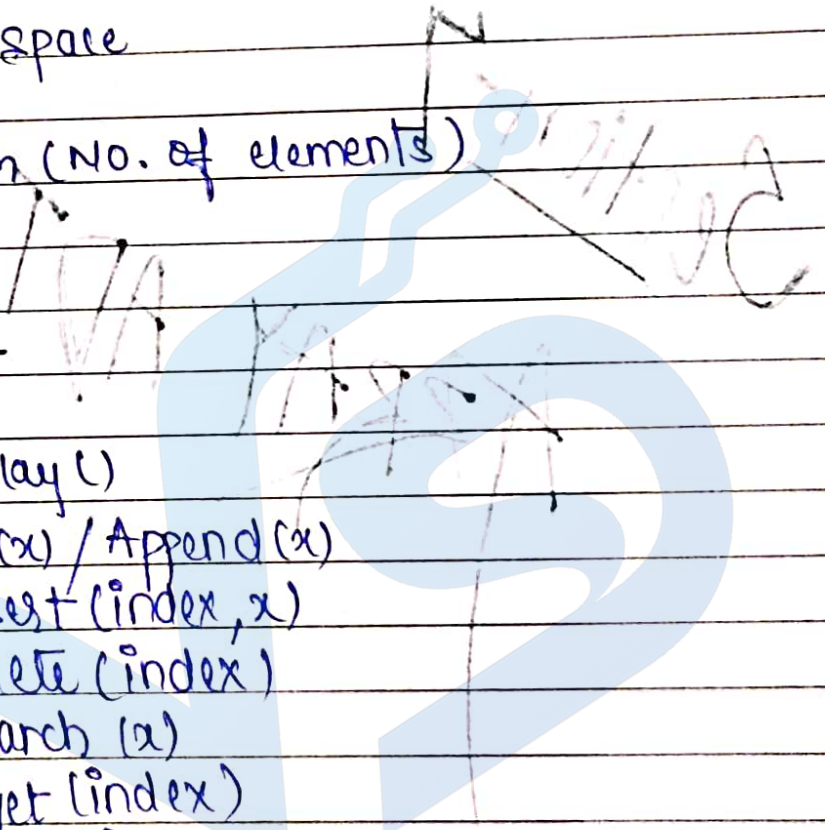
Section-7

ARRAY ADT

Array ADT → Array Abstract Data Type

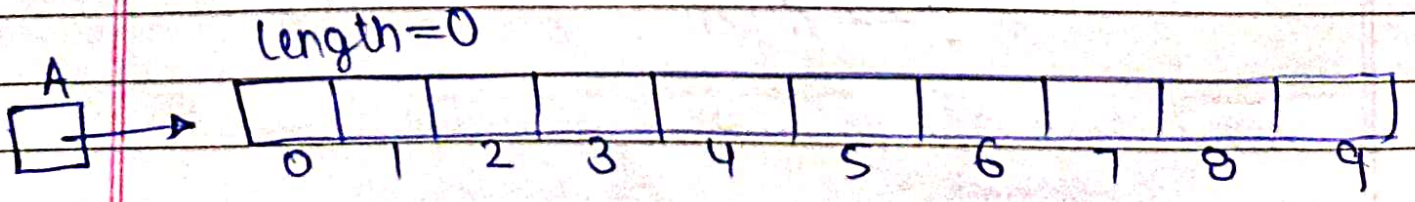
Data →

- Array space
- Size
  - Length (No. of elements)



Operations :-

- Display()
- Add(x) / Append(x)
- Insert(index, x)
- Delete(index)
- Search(x)
- Get(index)
- Set(index, x)
- Max() / Min()
- Reverse()
- Shift() / Rotate()



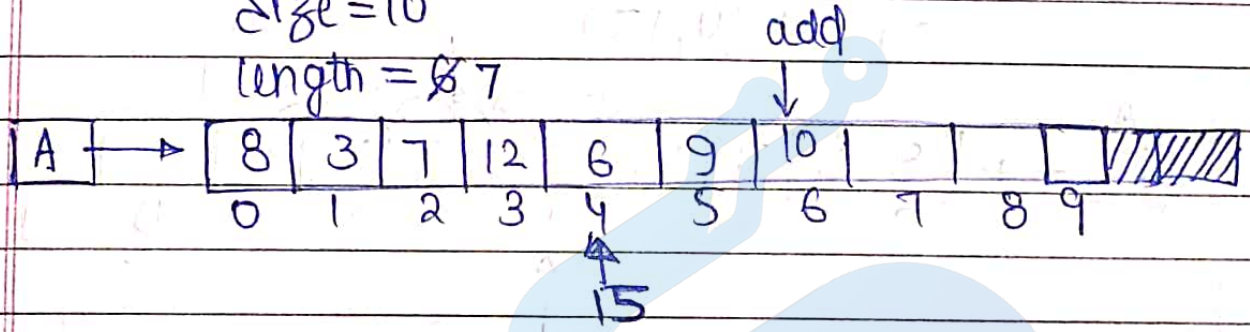
① int A[10];

Size = 10

(2) int *A;
A = new int [size];

Inserting in an Array :-

Size = 10
length = ~~8~~ 7



- Display :-

```
cout << A[0];

for(i=0; i<length; i++)
{
    cout<< A[i];

}
```

- Add (x) / Append (x)

```
A[length] = x;      ——— 1
length++;           ——— 1
                    ———
                    O(1)
```

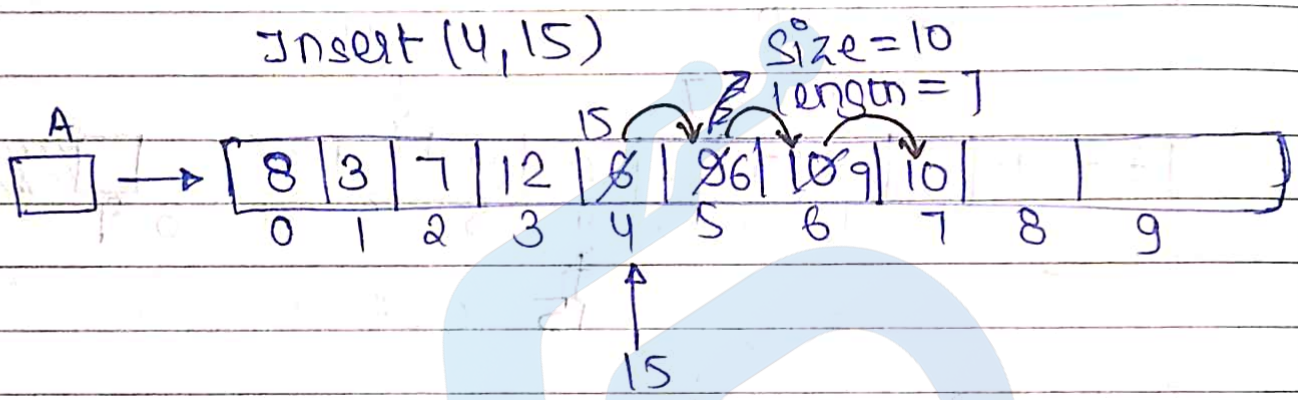
f(n) = 2

$f(n) = an^0$

$O(n^0) = O(1)$

- Insert(index, x)

Insert(4, 15)



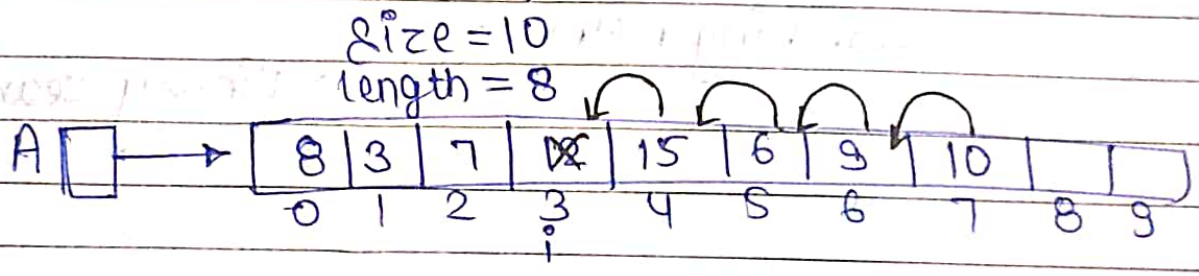
~~int~~ ~~i=length;~~

```
for(i=length; i>index; i--)
{
    A[i] = A[i-1]; ——— O(n)
}
A[index] = x; ——1
length++;
```

O(1) → Min

O(n) → Max

- Delete (index)



index

Delete (3)

index should not be out of range.

```
x = A[index];  ——— 1

for(i=index; i<length-1; i++)
{
    A[i] = A[i+1];  ——— 0-n
}
length--;  ——— 1
```

O(2) Min

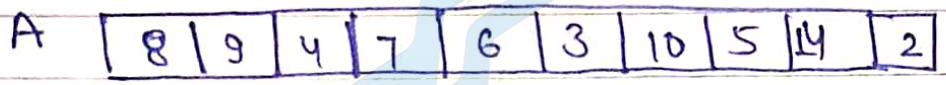
O(n+2) Max

Best : O(1)

Worst : O(n)

Searching Method → Linear Search
Searching Method → Binary search

Linear Search :-



for searching → Element should be unique

Key = 5 (successful)
↳ found at index = 7

Key = 12 (unsuccessfull)
↳ not found

```
for (i=0; i<length; i++)
{
    if (key == A[i])
    {
        return i;
    }
    else
    {
        return -1;
    }
}
```

Analysis of linear search:-

Successful Search
- Best case → O(1)
- Worst case → O(n)

unsuccessful search → O(n)

Average case →

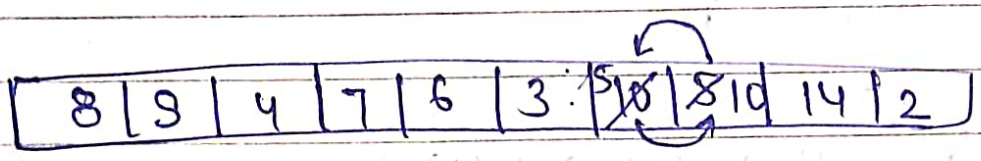
$$= \frac{1+2+3+\ldots\ldots+n}{n}$$

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

$$= n+1$$

$$O(n)$$

Improving linear search :- (Transposition)



Key → 5
↳ index = 7

1. Transposition

```
for(i=0; i<length; i++)
{
    if(key==A[i])
    {
        swap(A[i], A[i-1]);
        return i-1;
    }
}
```

2. Move to front/Head

```
for(i=0; i<length; i++)
{   if(key==A[i])
    {
        swap(A[i], A[0]);
        return 0;
    }
}
```

Notebook
Pg. No.
Date / /

Binary Search :-

A	4	8	10	15	18	21	24	27	29	33	34	37	39	41	43
	↑ l			↑ mid		↑ mid (h)		↑ mid							↑ h

(Element must be sorted)

Key = 18

↳ index = 4

l → lower

h → higher

$mid = \left\lfloor \frac{(l+h)}{2} \right\rfloor$

l	h	mid	
0	14	$\frac{0+14}{2} = 7$	18 < 27

Now

0	7-1=6	$\frac{0+6}{2} = 3$	18 > 15

Now,

3+1=4	6	$\frac{6+4}{2} = 5$	18 < 21

4	mid-1 = 4	$\frac{4+4}{2} = 4$	18 = 18

key = 34,

l	h	mid	
0	14	7	34 > 27
7+1=8	14	$\frac{14+8}{2}=11$	34 < 37
8	11-1=10	$\frac{10+8}{2}=9$	34 > 39
9+1=10	10	$\frac{10+10}{2}=10$	34 = 34 //

key = 25

l	h	mid	
0	14	7	25 < 27
0	7-1=6	$\frac{0+6}{2}=3$	25 > 10
3+1=4	6	$\frac{4+6}{2}=5$	25 > 21
6	6	6	25 > 24
6+1=7	6	$\frac{7+6}{2}=$	terminated because (l > h) Condition

Algorithm of binary search :-

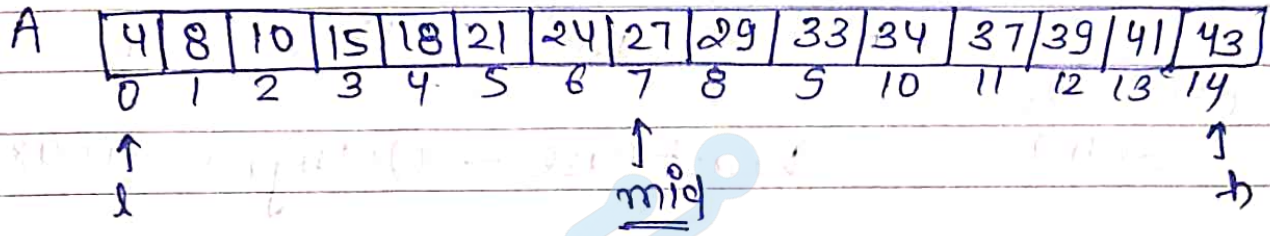
```
Algorithm BinarySearch (l, h, key)
{   while (l ≤ h) {
        mid = ⌊(l+h)/2⌋ ;
        if (key == A[mid])
        {
            return mid;
        }
        elseif ( key < A[mid])
        {
            h = mid - 1;
        }
        elseif ( key > A[mid])
        {
            l = mid + 1;
        }
    }
    return -1;
}
```

Recursive Version :-

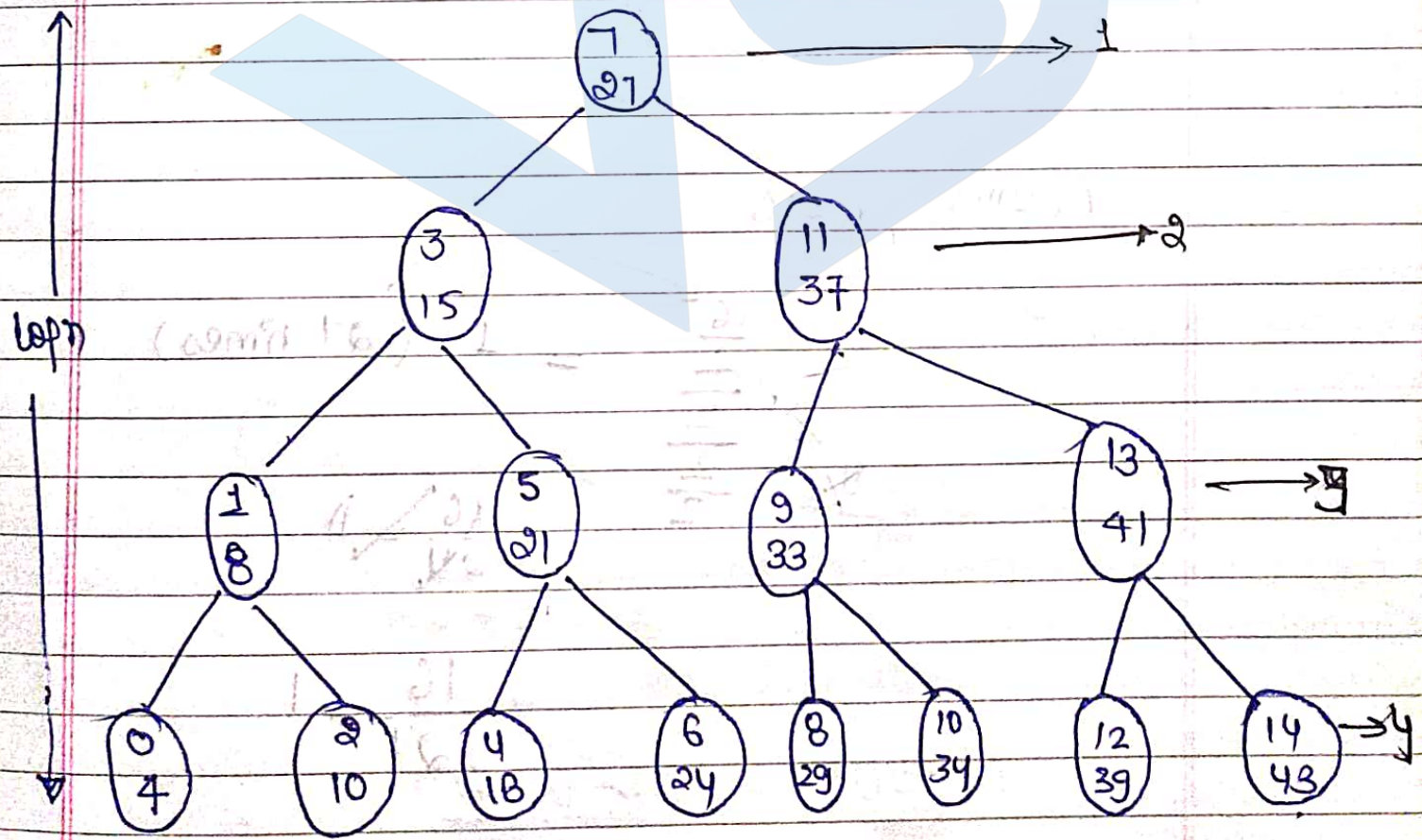
```
Algorithm RecursiveBinarysearch(l, h, key)
{

    if(l<=h)
    {
        mid = ⌈(l+h)/2⌉
        if(key==A[mid])
        {
            return mid;
        }
        else if(key < A[mid])
        {
            return RecursiveBinarysearch
                   (l, mid-1, key);
        }
        else if(key > A[mid])
        {
            return RecursiveBinarysearch
                   (mid+1, h, key);
        }
    }
    return -1;
}
```

Tail Recursion

Analysis of Binary Search :-



key = 27,
↳ index = 7



No. of comparison = Height of Tree

Successful search → Best case — O(1) (Min)

Worst case — O(logn) (max)

Unsuccessful search → O(logn)

Assume, n = 16

$\frac{16}{2} = 1$ (2^4 times)

$\frac{2}{2}$

$\frac{2}{2}$

$\frac{16}{2^4} = 1$

$\frac{16}{2^4} = 1$

$2^4 = 16$

$4 = \log_2 16$

$O(\log_2 n)$

$O(\log_2 n)$

Exact formula → $O(\log_2(n+1))$

Average Case Analysis of Binary Search :-

$= 1 + 1*2 + 2*4 + 3*8 + \cdots\cdots$

$= 1 + 1\times 2^1 + 2\times 2^2 + 3\times 2^3 + \cdots\cdots$

$= \frac{\sum_{i=1}^{\log n} i \times 2^i}{n} \Rightarrow \frac{\log n \times 2^{\log n}}{n}$

$\Rightarrow \frac{\log n \times n^{\log_2 2}}{n}$

$= \frac{n \log n}{n}$

$= \log n$

Average case → $O(\log n)$

For Binary search :-

Best case – O(1)

Average case – O(log n)

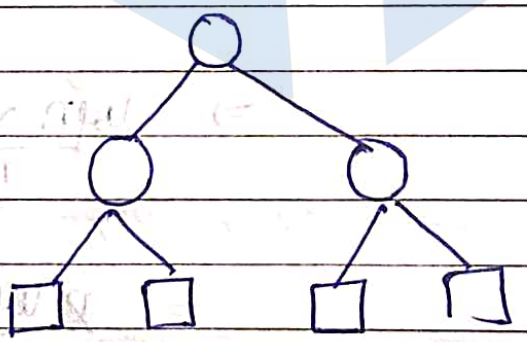
worst case – O(log n)

I → sum of path of internal node

E → ~~Total~~ sum of path of external node

$$E = I + 2n$$

↳ Always True



$n = 3$

$I = 2$

$E = 8$

$E = I + 2n$

$= 2 + 2 \times 3$

$E = 8$

$$e = i + 1$$

Average path :→

$$A_{successful}(n) = \frac{I}{n} = \frac{\text{(total path of all internal nodes)}}{\text{Number of node}}$$

Average successful time

$$A_s(n) = 1 + \frac{I}{n}$$

$E = I + 2n$

$I = E - 2n$

$$A_s(n) = 1 + \frac{I}{n}$$

$$= 1 + \frac{E - 2n}{n}$$

$$= 1 + \frac{E}{n} - 2$$

$$= 1 + \frac{n \log n}{n} - 2$$

$$= 1 + \log n - 2$$

$$= \log n - 1$$

$$A_s(n) = \log n$$

Average time for unsuccessful

$$A_u(n) = \frac{E}{n+1}$$

$$E = n \log n$$

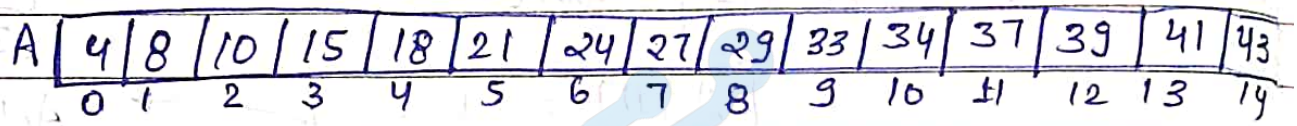
$$A_u(n) = \frac{n \log n}{n+1}$$

$$A_u(n) = \log n$$

Notebook
Pg. No.
Date / /

functions on Array

→ Get(), set(), Avg(), Max()



1. Get(index)
2. set(index, x)
3. Max()
4. Min()
5. sum()
6. Avg()

- Get(index)

```
if( index >=0 && index < length)
{
    return A[index];
}
```

- Set(index, x)

```
if(index >= 0 && index < length)
{
    A[index] = x;
}
```

- max()

```
{
```

A

8	3	9	15	6	10	7	2	12	4

O(n) → time

```
max = A[0];  —— 1
```

max: ~~8~~ 15

```
for(i=1; i<length; i++)  —— n
{
    if(A[i] > max)  —— n-1
    {
        max = A[i];
    }
}
return max;  —— 1
```

$f(n) = 2n + 1$

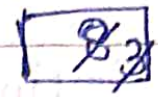
~~O(n)~~

O(n)

- Min()

```
min = A[0]; —1

for(i=1; i<length; i++) —n
{
    if( A[i] <min) —n-1
    {
        min = A[i];
    }
}
return min; —1
```

min



2

$f(n) = 2n+1$

$O(n)$

- sum()

Total = 0+8+3+9+ ....

```
int Total =0 ; —1
for (i=0; i<length; i++) —n+1
{
    Total = Total + A[i]; —n
}
return Total; —1
```

$f(n) = 2n+3$

$O(n)$

- ~~Avg()~~

Recursive function for sum of Array

$$Sum(A,n) = \begin{cases} 0 & n<0 \\ Sum(A,n-1) + A[n] & n \geq 0 \end{cases}$$

```
int sum(A,n)
{
    if(n<0)
    { return 0;
    }
    else
    {
        return sum(A,n-1) + A[n];
```

O(n)

call sum(A, length-1)

- Avg():-

```
Total = 0;
for(i=0; i<length; i++)
{
    Total = Total + A[i];
}
return total/n;
```

Reverse and shifted an Array :-

1. Reverse
2. left shift
3. left Rotate
4. Right shift
5. Right Rotate

A	8	3	9	15	6	10	7	2	12	4
	0	1	2	3	4	5	6	7	8	9

- Reverse :-

figure A → B

B	4	12	2	7	10	6	15	9	3	8
	0	1	2	3	4	5	6	7	8	9

$A \longrightarrow B \quad n$

$B \longrightarrow A \quad n$

$O(2n)$

$O(n)$

first Method ;

```
n+1 ——  for (int i=length-1 ; j=0 ; i>=0 ; i--, j++)
        {
n   ————————  B[j] = A[i];
        }
```

→ see figure A)

→ for reversing element and store in B Array

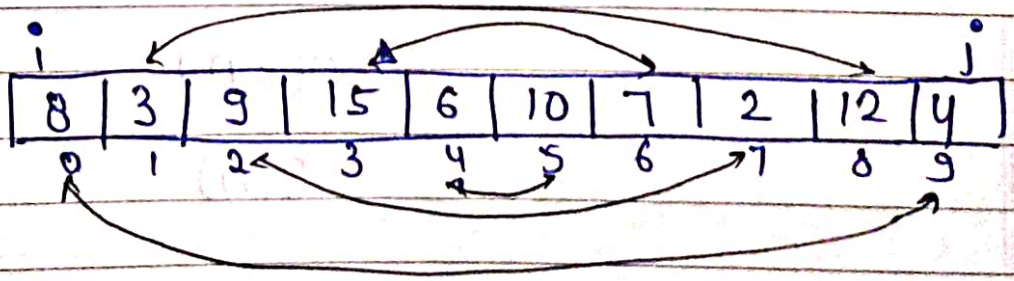
```
n+1 ——  for (i= 0; i< length ; i++)
        {
n   ——————  A[i] = B[i];
        }
```

overwriting the value of array A.

$f(n) = 4n+2$

$O(n)$

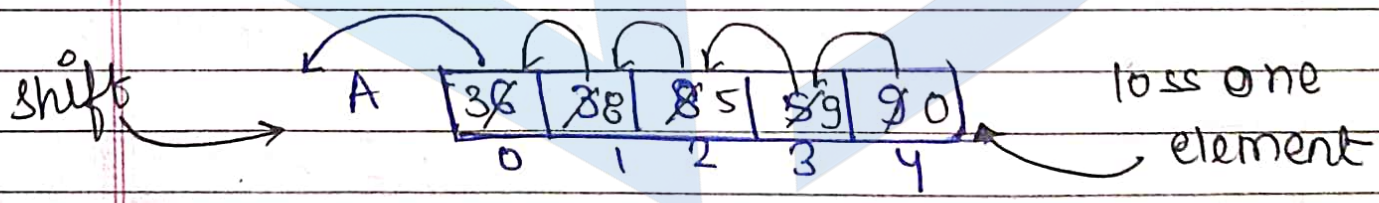
Second Method :-

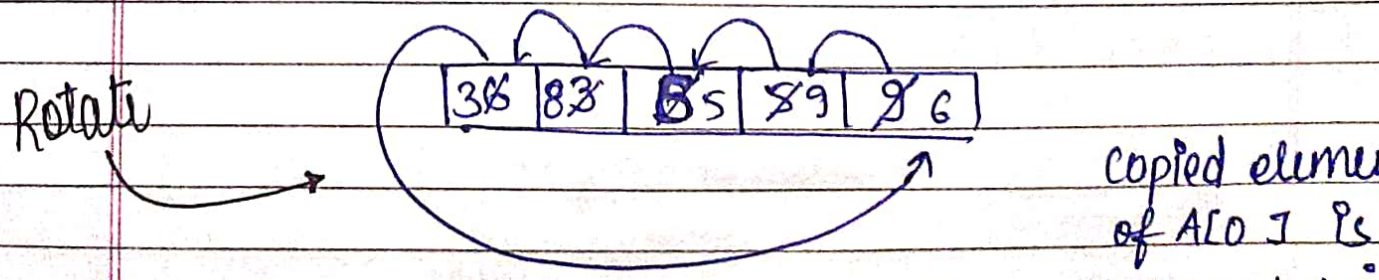


~~int i=0; j=length-1;~~

```
int i,j;

for(i=0 ; j=length-1 ; i<j ; i++ ,j--)
{
    ~~temp=A~~
    temp = A[i];
    A[i] = A[j];
    A[j] = temp;

}
```

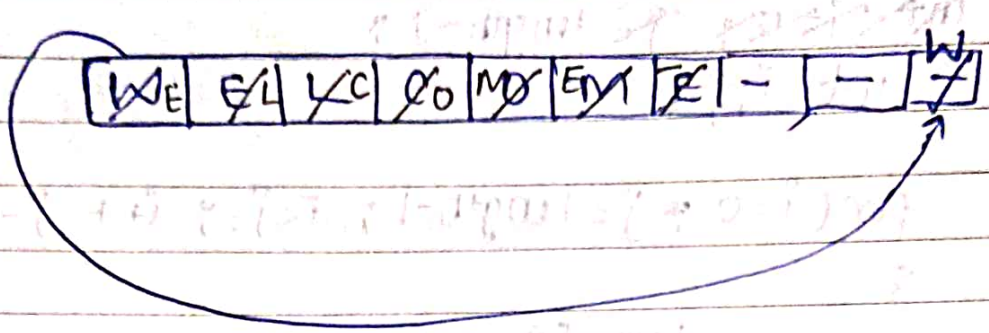
Left Shift / Rotate :-





Copied element of A[0] is replaced with loss element

Use:-



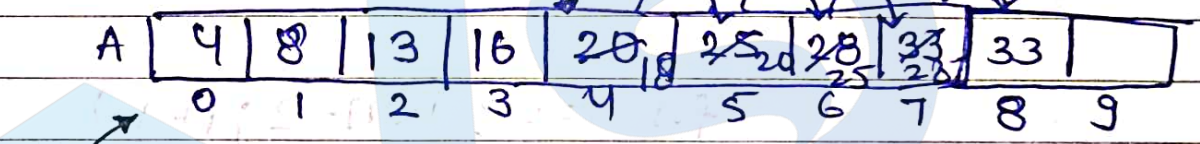
for scrolling → shift

for Rolling → Rotation

Check if Array is Sorted :-

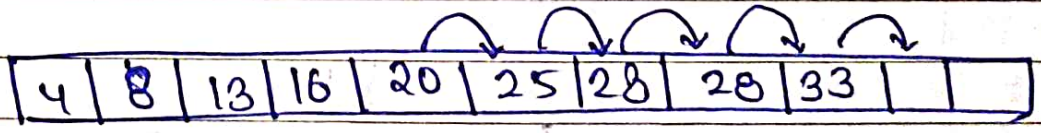
1. Inserting in a sorted Array
2. Checking if an Array is sorted
3. Arranging -ve on left side

- Insert → 18



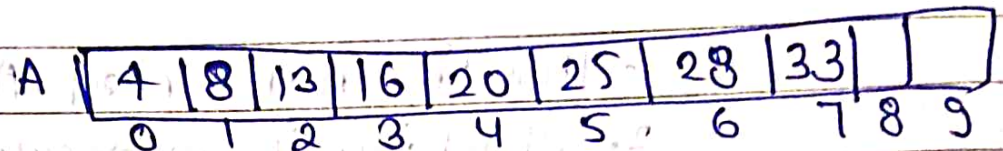
Processor:-

```
x = 18;
```




```
i = length - 1;
while (A[i] > x)
{   A[i+1] = A[i];
    i--;
}
A[i+1] = x;
```

Sorted :-

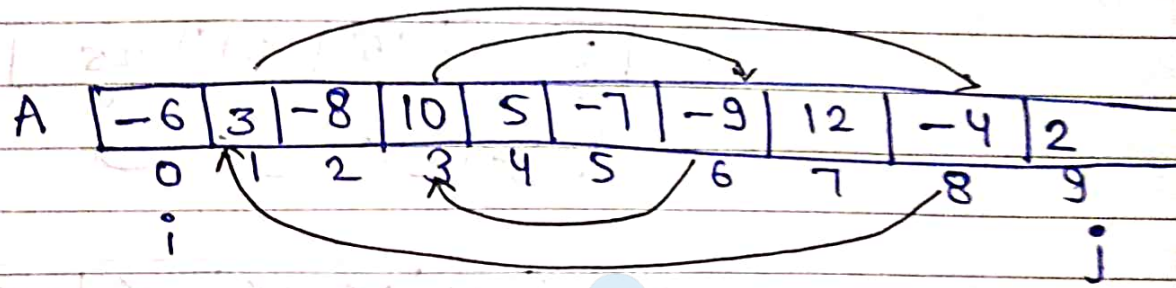


Check current number with next number

```
Algorithm isSorted (A, n)
{
    for(i=0; i<n-2; i++)
    {
        if(A[i]> A[i++])
        {
            return false;
        }
    }
    return true;
}
```

O(n)

- -ve number on left side :-

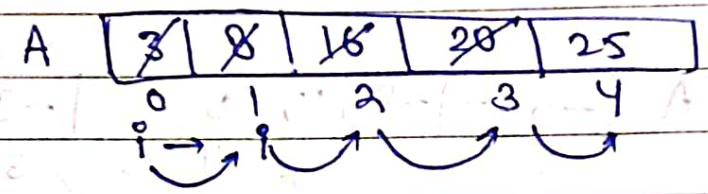


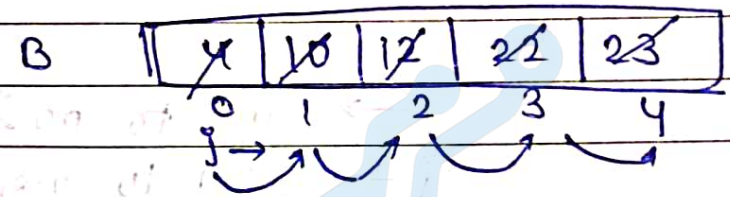
i → find to positive number
j → find to negative number

```
i=0;  ——1
j=length-1;  ——1
while(i<j){
  while(A[i]<0) → stop when the number is positive
  {
    i++;
  }
  while(A[j]>=0) ⟶ stop when it less than 0 or when the number is negative
  {
    j--;
  }
  if(i<j){
  swap(A[i], A[j]);
  }
}
```

we compare n+2 element

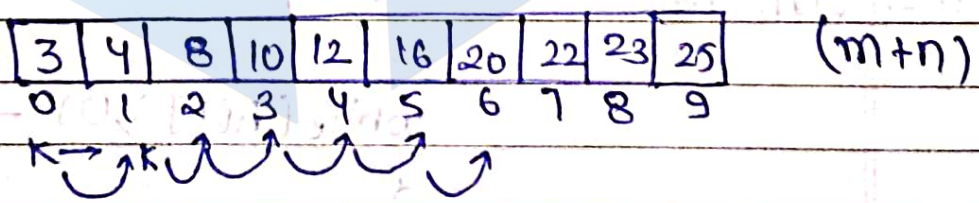
Merging Array :-





Binary operation on Array

1. Append
2. Concat
3. Compare
4. Copy



Merging can be done only in sorted Array.

Analysis :-

$\Theta(m+n)$

```
int i=0, j=0, k=0;
while(i<m && i<n){
    if(A[i]< B[j])
    {
            C[k++]= A[i++];
    }
    else
    }
            C[k++] = B[j++];

    }
}
while
for( ; i<m; i++)  // for remaining left in a Array A.
{
        C[k++]=A[i];
}
for( ; j<n; j++) // for remaining left in a Array B
{
        C[k++] = B[j];
}
```

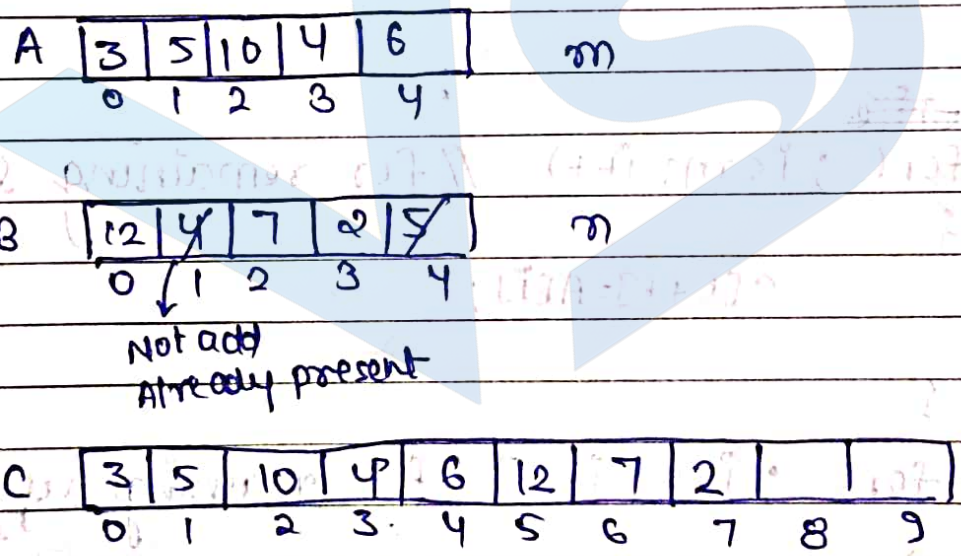
Set operation on Array - Union, Intersection and Diffference :-

Set Operations :-

1. Union
2. Intersection
3. Difference
4. set Membership

for Unsorted Element :-

Union



m → searching

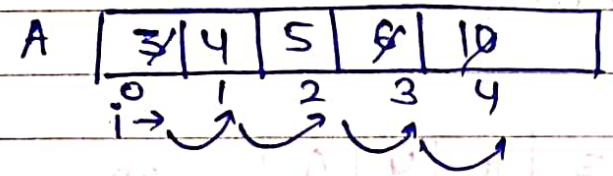
copy Array(A) ← m + m * n → compare

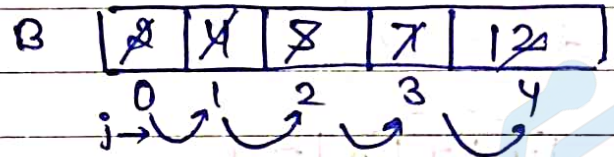
$m + n * n$

$O(n^2)$

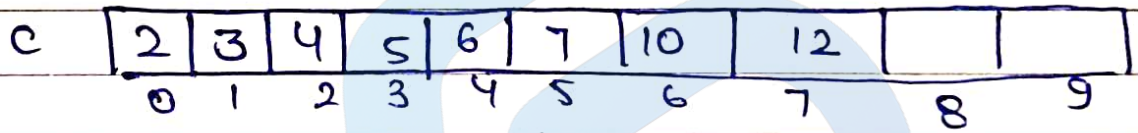
Union :- (A U B)

for sorted Element





Used Merged proccesor



$O(m+n)$

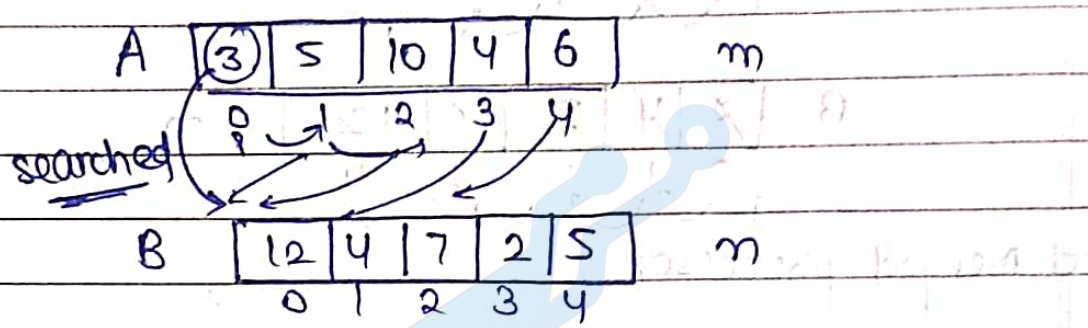
$O(n+n)$

$O(2n)$

$O(n)$

Intersection :- (A ∩ B)

for unsorted Array :-



$n * m$

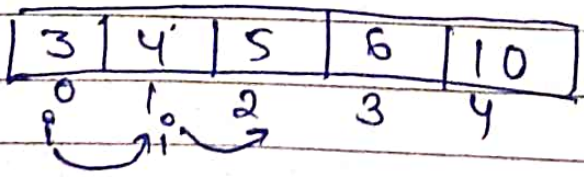
$n \times n$

$O(n^2)$

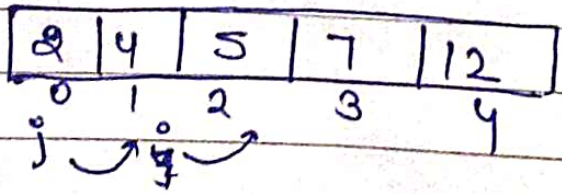
for sorted Array :-

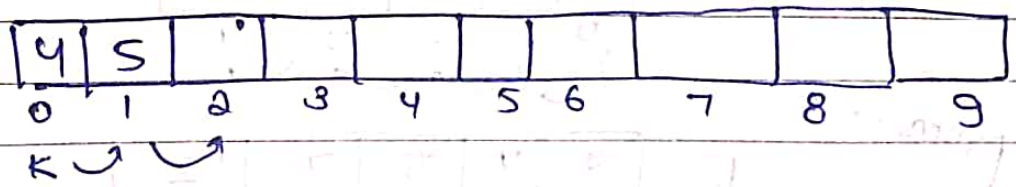
Merge processor

Move which is smaller element index.



when both are equal then copy them in ←

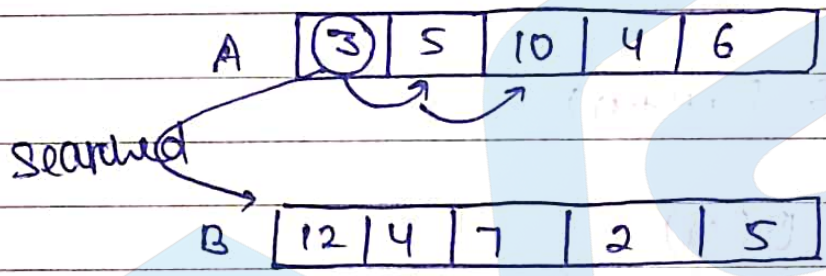


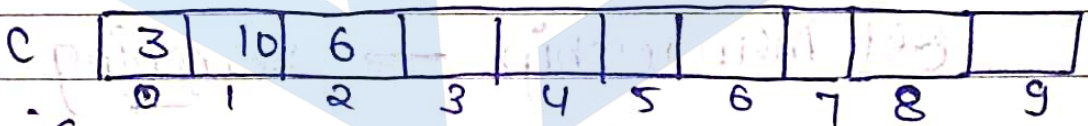


Difference ÷ (A - B)

for unsorted Array

(only in A) but





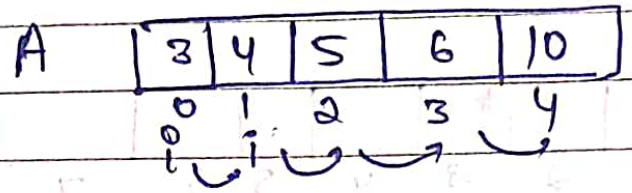
copy only element of A which not present in B.

$m * n$

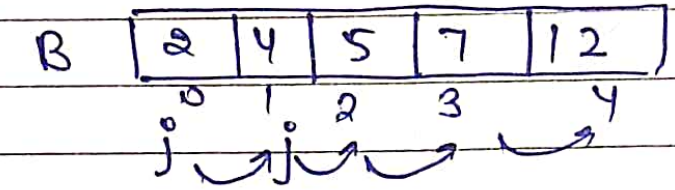
$n * n$

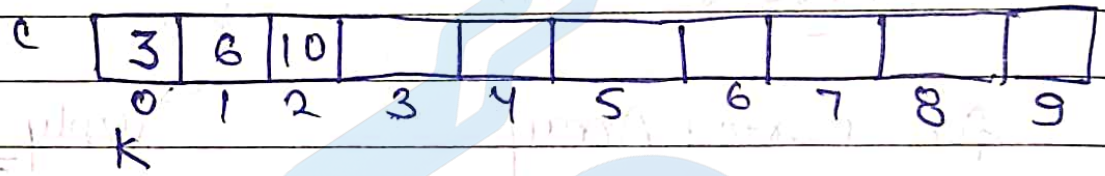
$O(n^2)$

for sorted Array :-



Merge processor





$\Theta(m+n)$

$\Theta(n)$

Set Membership → Searching

↳ A ∈ 10

Notebook
Pg. No.
Date / /

6

Student Challenge

(sorted Array)

- Finding Single Missing Element in an Array-

A	1	2	3	4	5	6	8	9	10	11	12
	0	1	2	3	4	5	6	7	8	9	10

finding single Missing element in Sorted Array

$\frac{n(n+1)}{2}$ → first 10 natural number

$\frac{12(13)}{2} = 78$

Approach 1 :-

```
sum = 0; n = arr[length-1]
for(int i=0; i<n; i++)
{
    sum = sum + arr[i];
}
S = n*(n-1)/2;
Missing_number = S-sum;
```

A	6	7	8	9	10	11	13	14	15	16	17
	0	1	2	3	4	5	6	7	8	9	10

l = 6 ;
h = 17 ;
n = 11 ;

6 - 0 = 6
7 - 1 = 6
8 - 2 = 6
⋮
11 - 5 = 6

```
diff = l - 0;
for (i=0; i<n; i++)
{
    if (~~diff~~ A[i] - i == diff)
    {
        cout << "missing element
             is " << i + diff;
        ~~break~~;
    }
    break;
}
```

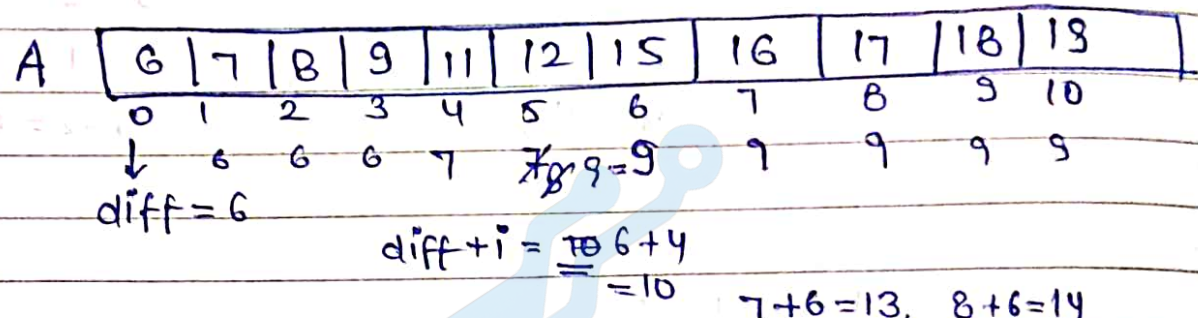
13 - 6 = 7

6 + 6 = 12 → missing

Time-complexity → O(n)

(Sorted Array)

- finding Multiple Missing Element in an Array

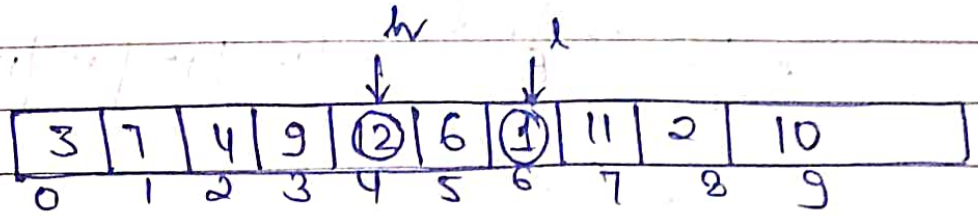



```
for(i=0; i<n; i++)
{
  diff = 6 - 0;
  for(i=0; i<n; i++)
  {
    if(A[i] - i != diff)
    {
      while(diff < (A[i] - i))
      {
        cout << i + diff;
        diff++;
      }
    }
  }
```

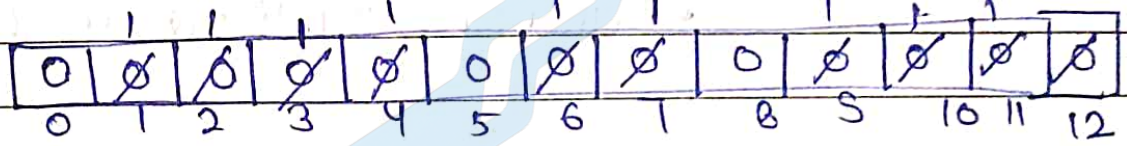
→ This take negligible time

Time complexity → O(n)

- finding Multiple Missing Element in an Unsorted Array:-



Hash/Bitset table →



Time complexity → O(n)

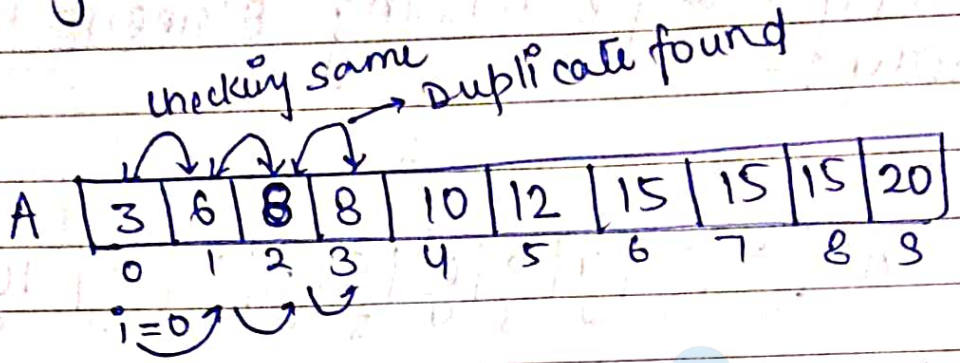
l = 1
h = 12
n = 10

for Hash table, we need ~~equal~~ space in Hash table equal to largest element in an array.

```
for(i=0; i<n; i++) ——n
{
    H[A[i]]++;
}
for(i=l; i<=h; i++) ——n
{
    if (H[i]==0)
    {
        cout<<i<<endl;
    }
}
```

O(n)

Finding Duplicate Element in an Sorted Array



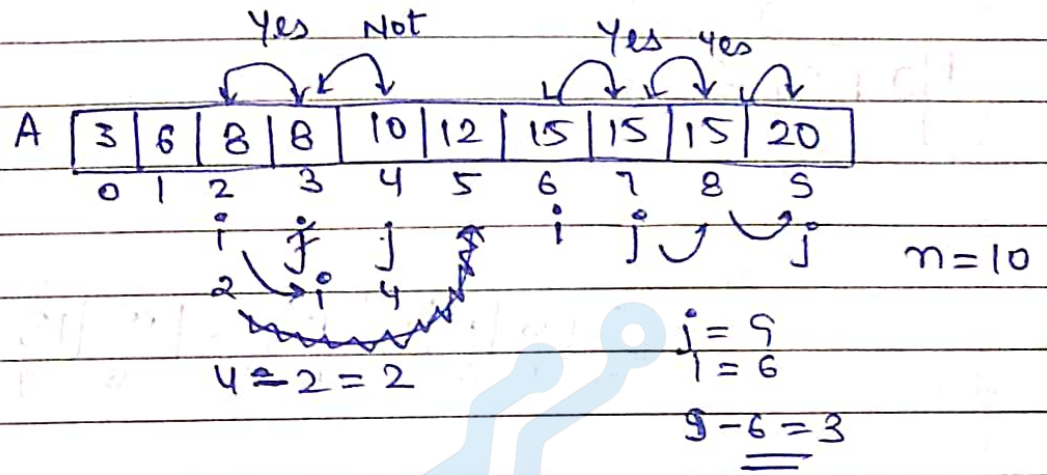
- finding Duplicates in Sorted Array
- Counting Duplicates in Sorted Array

n = 10

lastDuplicate = ~~0~~; 15

```
int lastDuplicate = 0;
for(int i=0; i<n; i++)
{
    if( A[i]==A[i+1] && A[i]!=LastDuplicate)
    {
        cout << A[i];
        lastDuplicate = A[i];
    }
}
```

• counting →

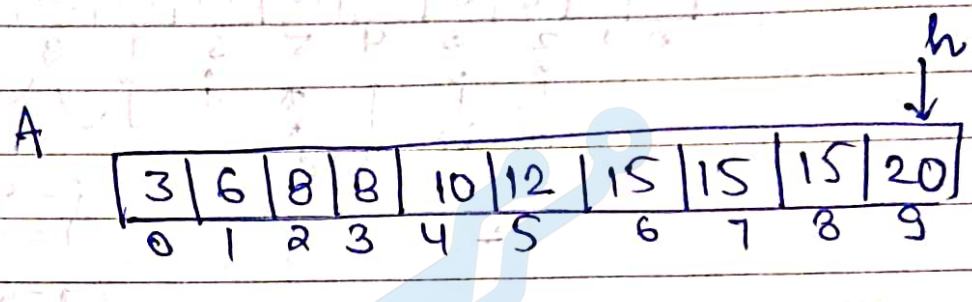


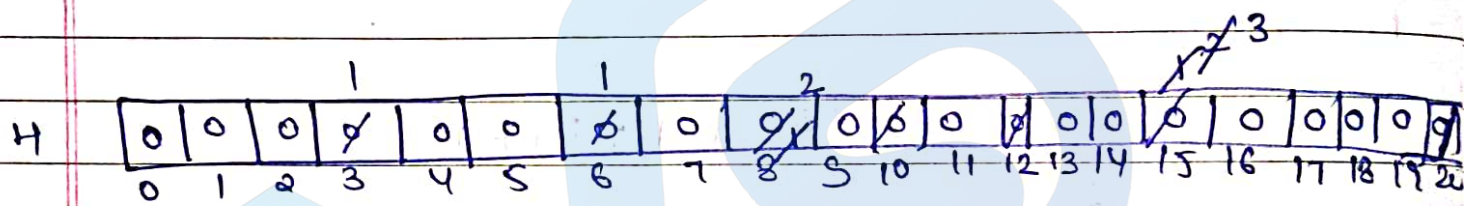

```
for(i=0; i<n-1; i++)
{
    if(A[i]==A[i++])
    {
        j=i+1;
        while(A[j]==A[i])
        {
            j++;
        }
    }
    cout<<A[i]<<"is appearing"<<j-i<<
    times;
    i=j-1;
}
}
```

Negligible

O(n)

Finding Duplicates in Sorted Array using Hashing :-





size of array = 20 because largest element in an array is 20.

$$n + n = 2n$$ (Also linear)

$O(n)$

```
for (int i=0; i<n; i++)
{
        A[A[i]]++;
}
```

— n

```
for(i=0 ; i<=n ; i++)
{
    if( H[i]>1)
    {
        cout<<i<<"is duplicated "<<H[i]-1<<"times"<<endl;
    }
}
```

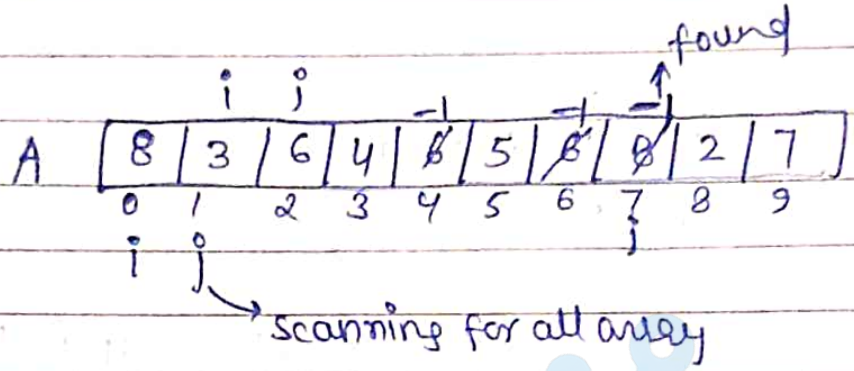
n

$f(n) = n + n$

$= 2n$

$O(n)$

Finding Duplicates in a Unsorted Array :-



for 8 :- count = 1; 2

for 3 :- count = 1;

for 6 :- count = 1; 2 3

for 4 :- count = 1;

for 5 :- count = 1;

for

-1 because if we not convert that then again check

Ex :-
for - 6 → 2
6 → 4
6 → 6

Analysis

~~n-1 (compare)~~

$= n-1 + n-2 + n-3 + \dots + 3 + 2 + 1$

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

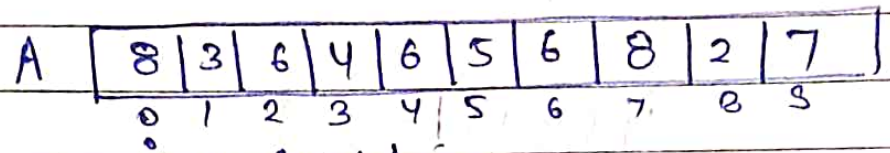
$= O(n^2)$

```
for(i=0; i<n-1; i++)
{
    count=1;

    if(A[i]!=-1){
        for(j=i+1; j<n; j++)
        {
            if(A[i]=A[j])
            {
                count++;
                A[j]=-1;
            }
        }
        if(count>1)
        {
            cout<<A[i]<<" is duplicated "<<count
            <<"times"<<endl;
        }
    }
}
```

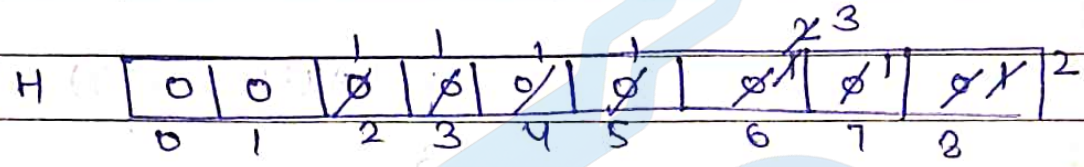
n → n

$O(n^2)$

finding duplicate in an unsorted Array using Hashing:-



h = max_element l = min_element



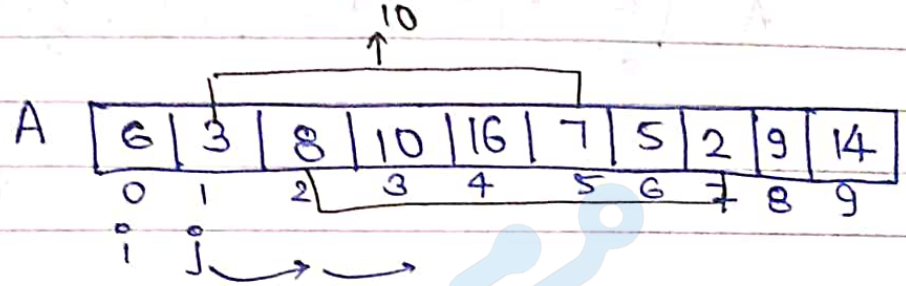

```
int n = 8
int H[h+1] = {0};

for(int i=0; i<n; i++)
{
        H[A[i]]++;

}
for(int i=1; i<=h; i++)
{
    if(H[i]>1)
    {
        cout<<i<< is repeated <<H[i]<<"times"
        <<endl;
    }
}
```

n + n = 2n (Linear)

Unsorted Array

Finding a Pair of Elements with sum K

$(a+b=K)$



$a+b=10$

check A[i]+A[j] = k

```
for(int i = 0; i<n-1; i++)
{
    for(j=i+1; j<n; j++)
    {
        if(A[i]+A[j]==K)
        {
            cout<<A[i]<<"+"<<A[j]<<"="
            K<<endl;
        }
    }
}
```

n n

$O(n^2)$

Analysis :-

$= n-1+n-2+ \cdots \cdots +3+2+1$

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

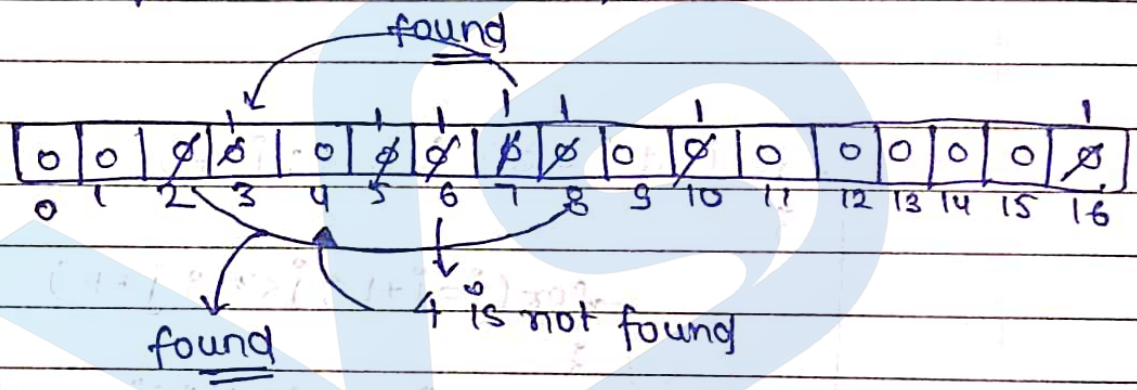
$= O(n^2)$

Faster method by using Hashing

A	6	3	8	10	6	7	5	2	9	14
	0	1	2	3	4	5	6	7	8	9

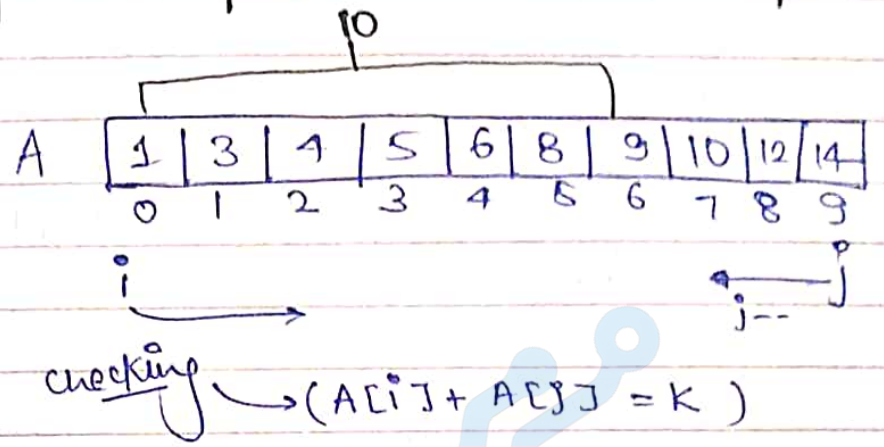
a+b=10

Size of Hash table = largest_element_in_an_array




```
for(int i=0; i<n; i++)
{
        if(H[K-A[i]]!=0)
        {
            cout<<A[i]<<"+"<<K-A[i]<<"="<<K;
        }
        H[A[i]]++;
}
```

Finding a Pair of Elements with sum K in Sorted Array



if total is greater than K,
then j--;

if total is less than K
then i++;

O(n)

```
i=0, j=n-1;
while(i<j)
{
    if (A[i] + A[j] <= K)
    {
        cout<< A[i] <<'+" << A[j] <<"= " <<K;
        i++;
        j--;
    }
    if (A[i] + A[j] < K)
    {
        i++;
    }
    else
    {
        j--;
    }
```

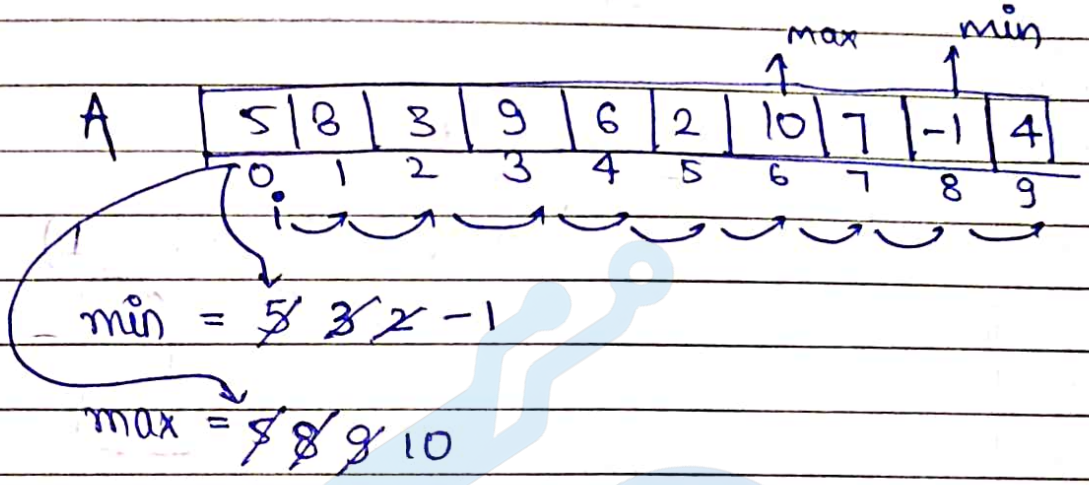
Notebook
Pg. No.
Date / /

Another method

```
for (i=0, j=n-1; i<j ; )
{
        if (A[i] + A[j] == K)
        {
            cout<<A[i] <<" + " << A[j] <<" = " <<K;
            i++;
            j--;
        }
        else if ( A[i] + A[j] < K)
        {
                i++;
        }
        else
        {
                j--;
        }

}
```

Notebook
Pg. No.
Date / /

Finding Max and Min in a Single Scan ---




```
min = A[0];
max = A[0];                                    O(n)
for (i=1; i<n; i++)
{
    if (A[i] < min)
    {
        min=A[i];

    }
    else if (A[i] > max
    {
        max = A[i];
    }
```

Pg. No.
Date / /

10, 9, 8, 7, 2, 1

↳ No. of comparatio = n - 1 → best

↳ minimum

1, 2, 3, 5, 8, 9, 10

↳ comparati ⇒ 2(n - 1) → Maximum

↳ worst

Quiz

Q1. Suppose you are given an array s[1.....n] and a procedure reverse(s, i, j) which reverse the order of elements in between position i and j (both inclusive)

What does the following the sequence do, where $1 \leq k \leq n$:

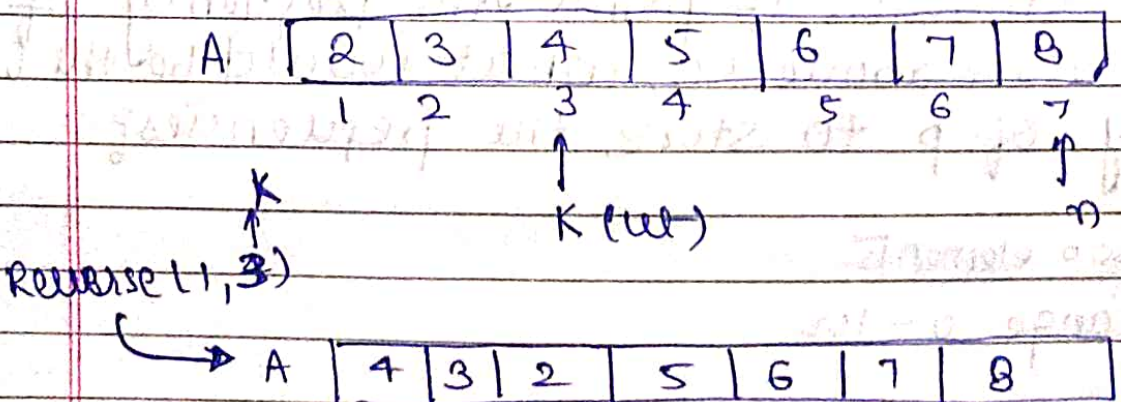
```
reverse(s, 1, k);
reverse(s, k+1, n);
reverse(s, 1, n);
```

→ s[1.....n]

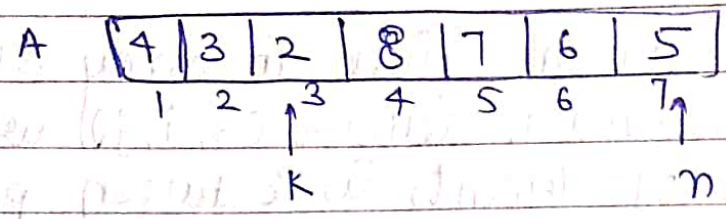
reverse(1, k)

reverse(k+1, n)

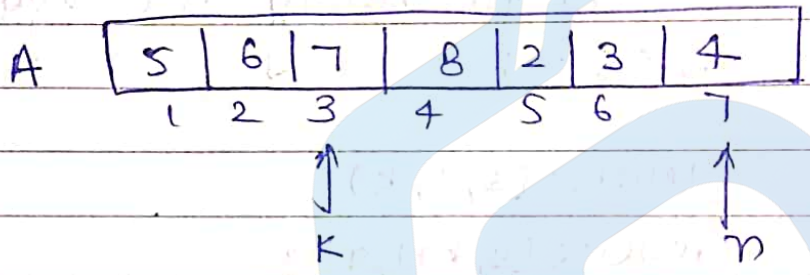
reverse(1, n)



reverse(k+1, n) or reverse(4, 7)



reverse(1, n) or reverse(1, 7)



(a) rotates S left by K position

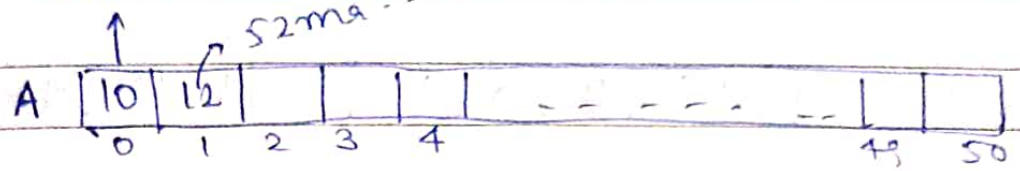
Q2. A program P reads in 500 integers in the range (0,100) representing the scores of 500 students. if prints the frequency of each score above 50, what would be the best way of p to store the frequencies?

→ 500 elements

range 0 - 100

frequency of element > 50

student who get 51 marks (at)



(a) an array of 50 numbers

Q3. Let a be array containing n integers in increasing order. The following algorithm determines whether there are two distinct numbers in the array whose difference is a specified number, s>0.

```
i=0; j=1;
while(j<n)
{
    if(E)          // E → find
    {
        i++;
    }
    else if(a[j]-a[i]==s)
    {
        break;
    }
    else
    {
        i++;
    }
}
if(j<n)
{
    cout<<"Yes";
}
```

→ A

2	4	8	9	13	16	18	23	25
0	1	2	3	4	5	6	7	8

i j

s = 10

a[4] - a[2]

↳ difference is less than 10

so, j++;

a[3] - a[2]

↳ difference is greater than 10

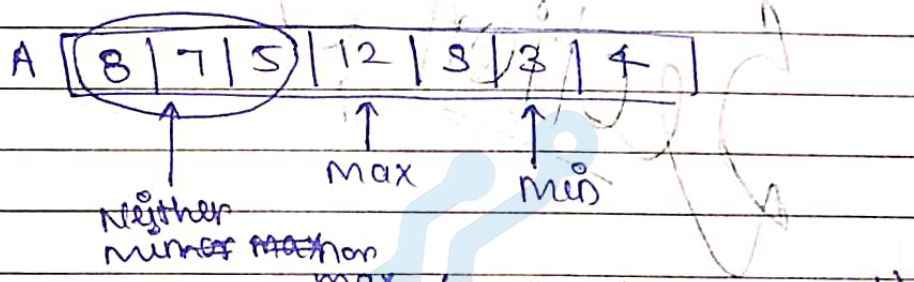
so, i++;

```
if(a[j] - a[i] < s)
{
    j++;
}
else if (a[j] - a[i] > s)
{
    i++;
}
```

(b) a[j] - a[i] < s ✓

Q4. What is the time taken to find any element from an Array of elements, which is neither maximum nor minimum.

→



(In first three element)

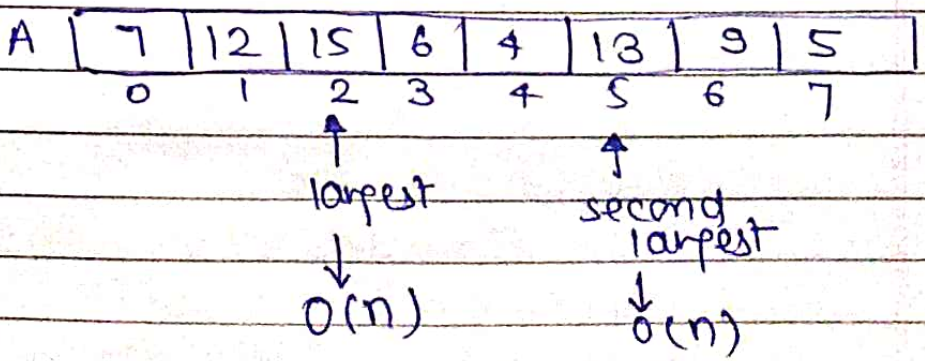
(a) O(1)

Take first three element

O(1) → Take constant time.

Q5) What is the time taken to find the second largest element in a Array.

→



$= n + n = 2n$

(a) O(n)

O(n)