28/12/21

* Section 7: Array ADT

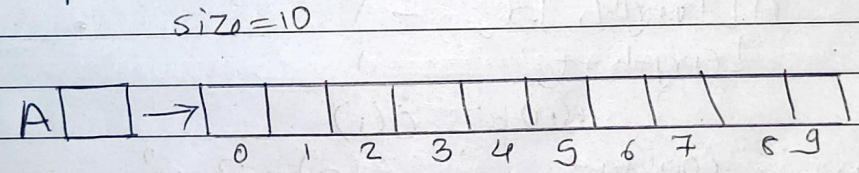
94. Array ADT

* Data

1. Array Space
2. Size
3. Length (No. of elements)

* Operations

1. Display() ✓
2. Add(x) / Append(x) ✓
3. Insert(index, x) ✓
4. Delete(index) ✓
5. Search(x) ✓
6. Get(index)
7. Set(index, x)
8. Max() / Min()
9. Reverse()
10. Shift() / Rotate()

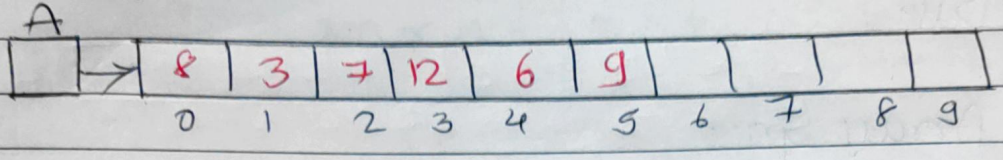


1) int A[10];
2) int *A;
   A = new int[size];

Anyone

36. Inserting in an Array

Size = 10
Length = 6

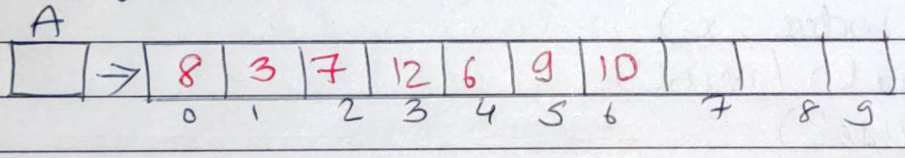


1) Display()

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

2) Add(x) / Append(x)

Size = 10
Length = 7




```
A[Length]=x;      -1
Length++;         -1
                  f(n) = 2    O(1)
```

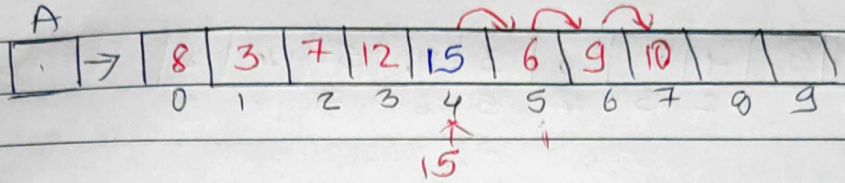
Time is constant $\therefore f(n) = 2n^0$

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

3) Insert (index, x) (4, 15)

Size = 10
Length = ~~4~~ 8

A




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

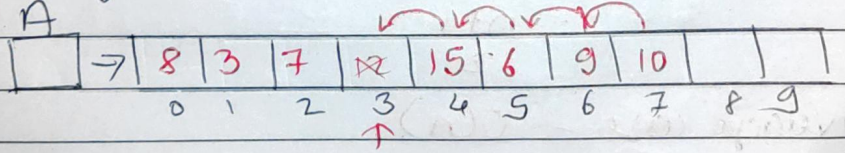
Time complexity: O(1) - Min, O(n) - Max

4) Deleting from Array

4) Delete (index) (3)

Size = 10
Length = ~~8~~ 7

A

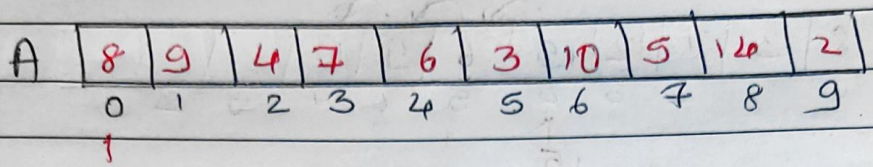



```
x = A[index];                          — 1
for(i = index; i < length-1; i++){
    A[i] = A[i+1];                     — 0-n
}
Length--;
```

Time: min = 2 = O(1); max = n+2 = O(n) — 1/2

100. Linear Search

Size = 10
Length = 10



key = 5 ← Successful
key = 12 ← Unsuccessful

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

Best case - O(1) } for successful
Worst case - O(n) }
Time - O(n) = for unsuccessful

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

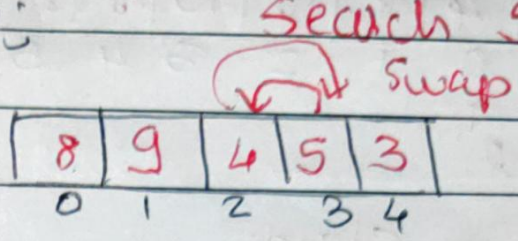
$$= \frac{n+1}{2}$$

Average case - O(n)

101. Improving Linear Search

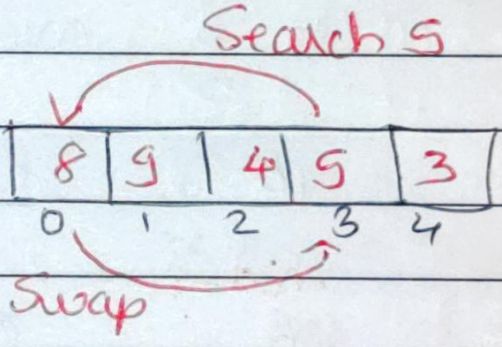
1. Transposition

```
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;
    }
}
```



103. Binary Search // Array should be sorted
↓ Splits list into two

Size = 15
Length = 15

A	4	8	10	15	18	21	24	27	29	33	34	37	39	41	43
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	l							mid							h

l → start
h → end
l < h

if (key > mid) then l = mid+1
if (key < mid) then h = mid-1
if (key = mid) then l = h

i) key = 18

l	h	mid = $\left[\frac{l+h}{2}\right]$
0	14	$\frac{0+14}{2} = 7$
	mid-1	
0	6	$\frac{0+6}{2} = 3$
mid+1		
4	6	$\frac{4+6}{2} = 5$
	mid-1	
4	4	$\frac{4+4}{2} = 4$ Found

2) key = 34

l	h	mid = $\left\lfloor \frac{l+h}{2} \right\rfloor$
0	14	7
mid+1 8	14	11
8	mid-1 10	9
mid+1 10	10	10 Found

3) key = 25

l	h	mid = $\left\lfloor \frac{l+h}{2} \right\rfloor$
0	14	7
0	6	3
4	6	5
6	6	6
7	6	x ← Terminate as l > h

104. Binary Search Algorithm

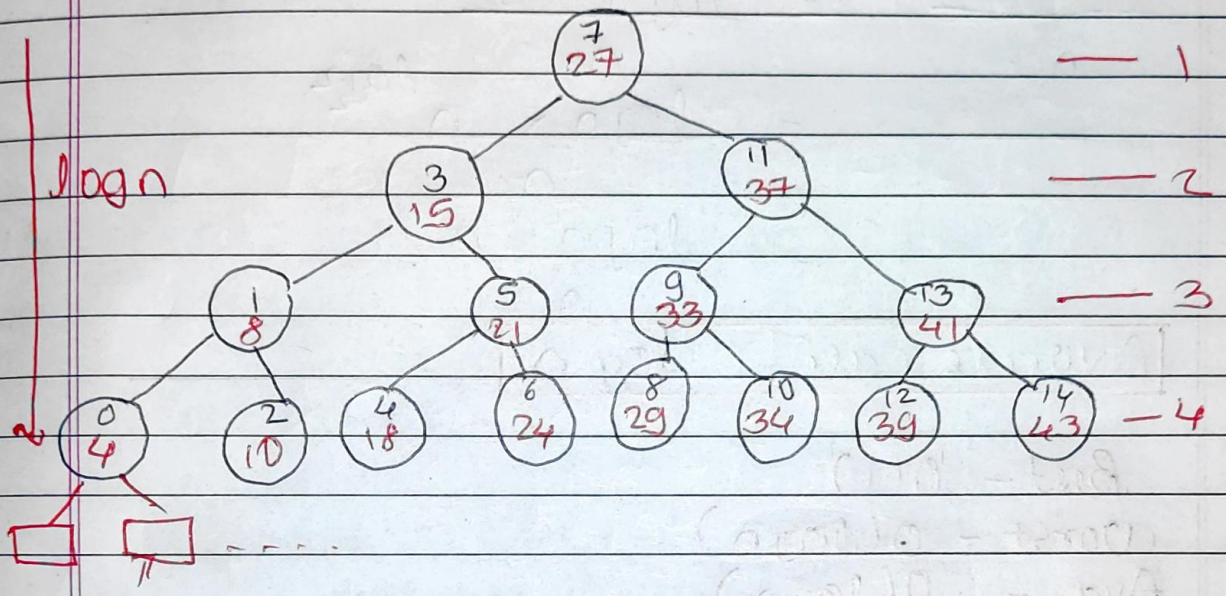
```
* BinSearch (l, h, key) {                    // Iterative
      while (l <= h) {                       // Better
          mid = [(l+h)/2];                   procedure
          if (key == A[mid])
              return mid;
          else if (key < A[mid])
              h = mid - 1;
          else
              l = mid + 1;
      }
      return -1;
}

* BinSearch (l, h, key) {                    // Recursive
      if (l <= h) {                          // Tail Recursion
          mid = [(l+h)/2];
          if (key == A[mid]);
              return mid;
          else if (key < A[mid])
              return BinSearch (l, mid-1, key);
          else
              return BinSearch (l, mid+1, h, key);
      }
      return -1;
}
```

106. Analysis of Binary Search

Size = 15
Length = 15

A	4	8	10	15	18	21	24	27	29	33	34	37	39	41	43
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14



Unsuccessful

* For Successful
  Best case min - O(1)
  Worst case max - O(log n)

* For unsuccessful
  O(log n)

107. Average case analysis of Binary Search.

Comparisons required

$= 1 + 1 \times 2 + 2 \times 4 + 3 \times 8 \ldots$

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

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

$= \log n \times 2^{\log n}$

$= \log n \times n^{\log_2 2}$

$= \log n \times n^1$

$= n \log n$

Average case = $\log n$

Best - O(1)
Worst - O(log n)
Avg - O(log n)

I (Internal node) - ○
E (External node) - □
$E = I + 2n$ $E = n \log n$
$e = i + 1$

* Average successful time
$A_s(n) = 1 + \frac{I}{n}$

* Average unsuccessful time
$A_u(n) = \frac{E}{n+1} = \frac{n \log n}{n+1} = \log n$

$As(n) = 1 + \frac{I}{n}$ $\quad E = I + 2n$ $\quad I = E - 2n$

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

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

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

$\approx \log n$

108. Get() Set() Avg() Max() functions on Array.

6. Get (index)

```
if(index >=0 && index < Length)
    return A[index];
```

Time - O(1)

7. Set (index, x)

```
if(index >= 0 && index < Length)
    A[index] = x;
```

8. max( )

```
int max = A[0];                    -1
for (i = 1; i < length; i++){      -n
    if (A[i] > max)                n-1
        max = A[i];
}
return max;                        -1
```

max: [89] (15)

$f(n) = 2n + 1$

Time - O(n)

9. min()

```
int min = A[0];
for (i = 1; i < length; i++) {
    if (A[i] < min)
        min = A[i];
}
return min;
```

min: [83] (2)

Time - O(n)

10. Sum()

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

$f(n) = 2n + 3$

Time - O(n)

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

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

Time $\in O(n)$;

11. Avg ()

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

Time - $O(n)$ $\quad f(n) = 2n+3$

110. Reverse and shift an Array

Size = 10
Length = 10

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

1> Reverse

* First Method

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

A → B n
B → A n
2n

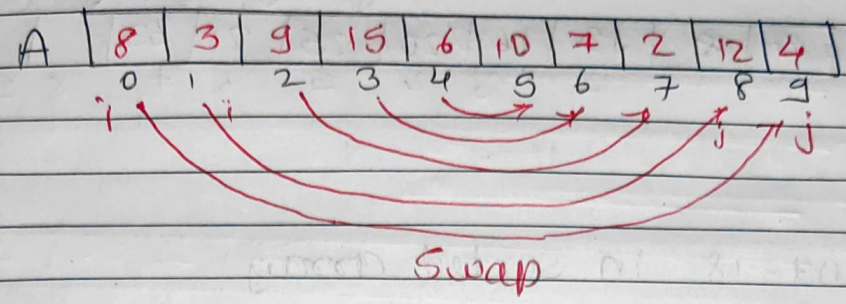
Time - O(n)

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

B → A
```
for(i=0; i<Length; i++){
    A[i] = B[i];                          -n
}                                         2n
```

Time - O(n)

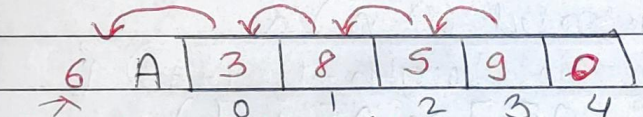
* Second Method

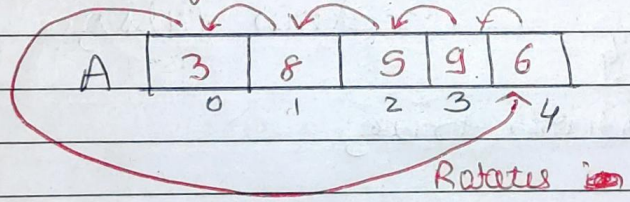



```
for(i=0, j=Length-1; i<j; i++, j--){
        temp = A[i];
        A[i] = A[j];
        A[j] = temp;
}
```

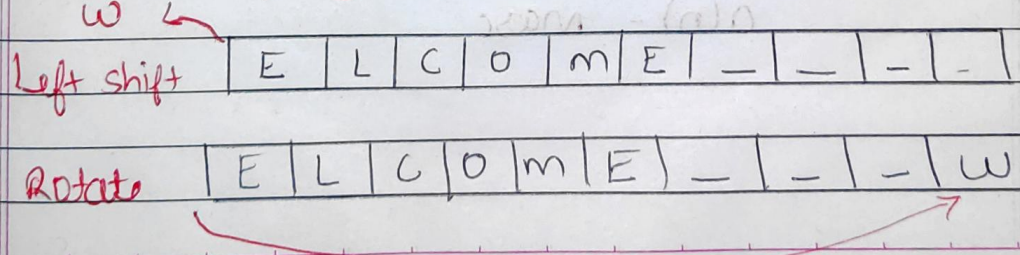
2) Left Shift / Rotate

Size=5
Length=5

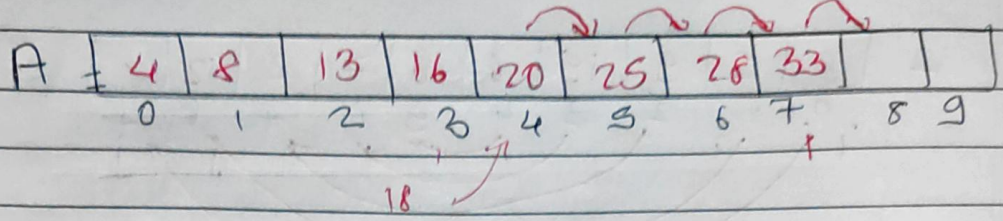




* LED Displays



112. Check if Array is Sorted



1) Insert - 18 in sorted array

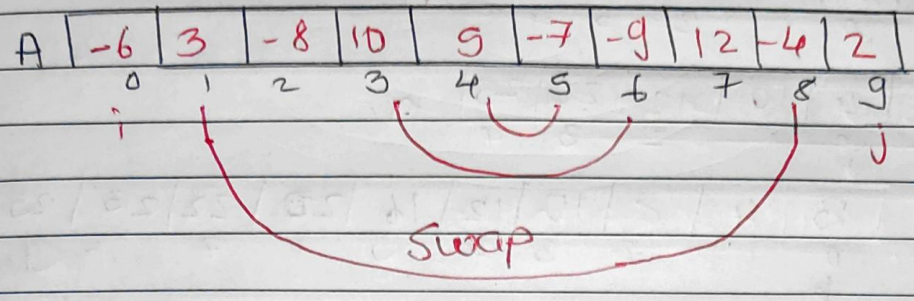
```
x = 18;
i = Length-1;
while (A[i] > x) {
    A[i+1] = A[i];
    i--;
}
A[i+1] = x;
```

2) Sorted or not

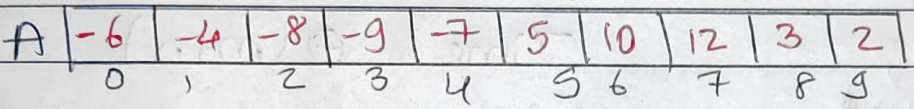
```
Algorithm isSorted (A, n) {
    for (i=0; i < Length-1; i++) {
        if (A[i] > A[i+1])
            return false;
    }
    return true;
}
```

Time - O(1) - Min
O(n) - Max

3) -ve on left side and +ve on right side



After swapping

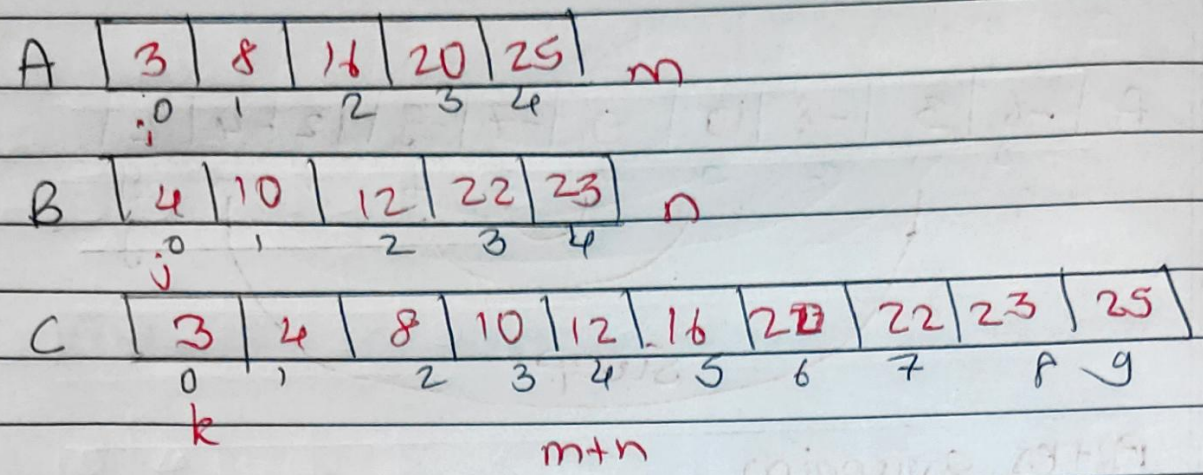



```
i = 0;
j = Length-1;
while (i < j) {
    while (A[i] < 0) {
        i++;
    }
    while (A[j] > 0) {
        j--;
    }
    if (i < j) {
        swap(A[i], A[j]);
    }
}
```

Time - O(n)

D D M M Y Y Y Y

14. Merging Arrays Arrays should be sorted



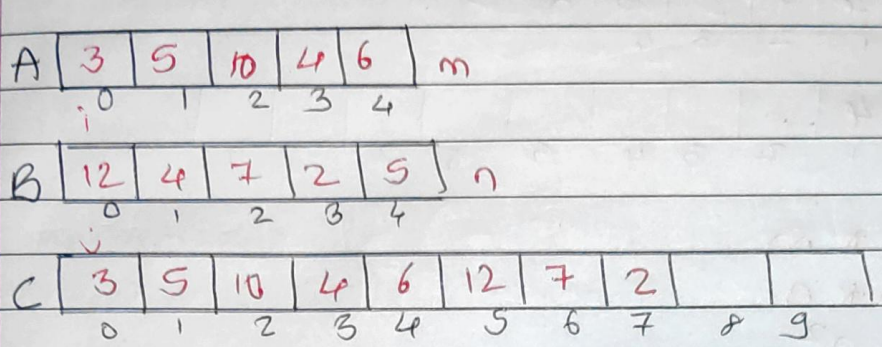
Time - $\Theta(m+n)$

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

116. Set operations on Array - Union, Intersection and Difference.

1> Union (A U B)

*UnSorted



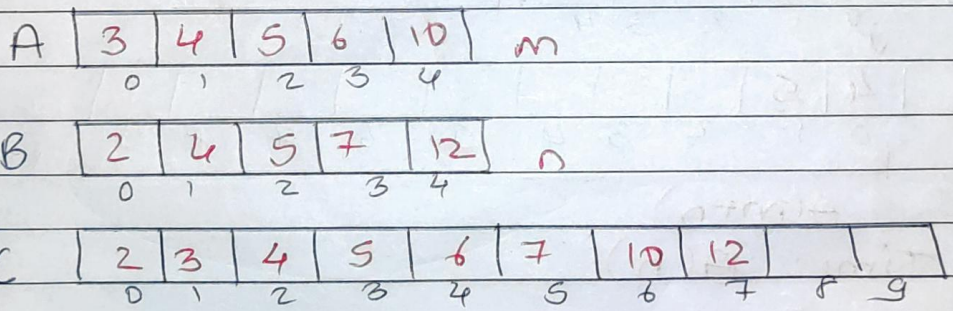
Time : $n + m * n$

$= n + n * n$

$= n + n^2$

Time - $O(n^2)$

* Sorted



$\theta(m+n)$

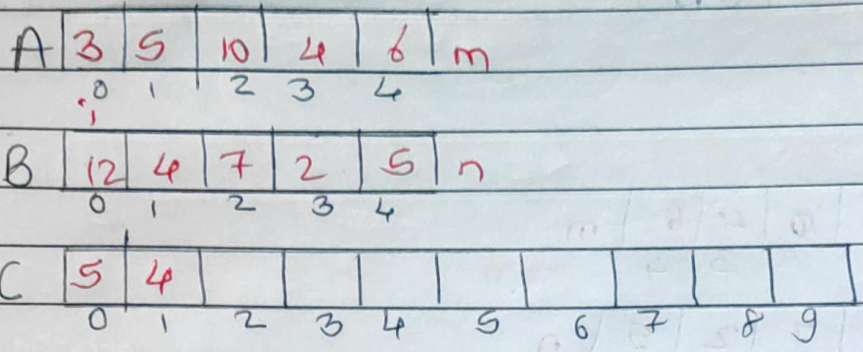
$\theta(n+n)$

$\theta(2n)$

Time : $\theta(n)$

2) Intersection (A ∩ B)

* Unsorted



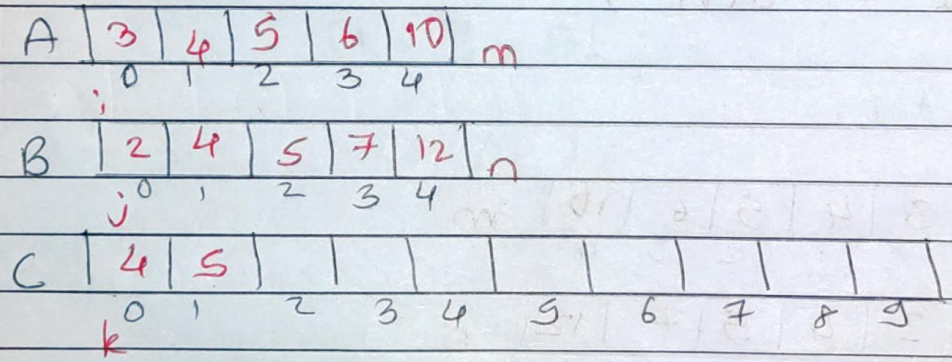
$n * m$

$= n * n$

$= n^2$

Time $O(n^2)$

* Sorted



$\Theta(m+n)$

Time: $\Theta(n)$