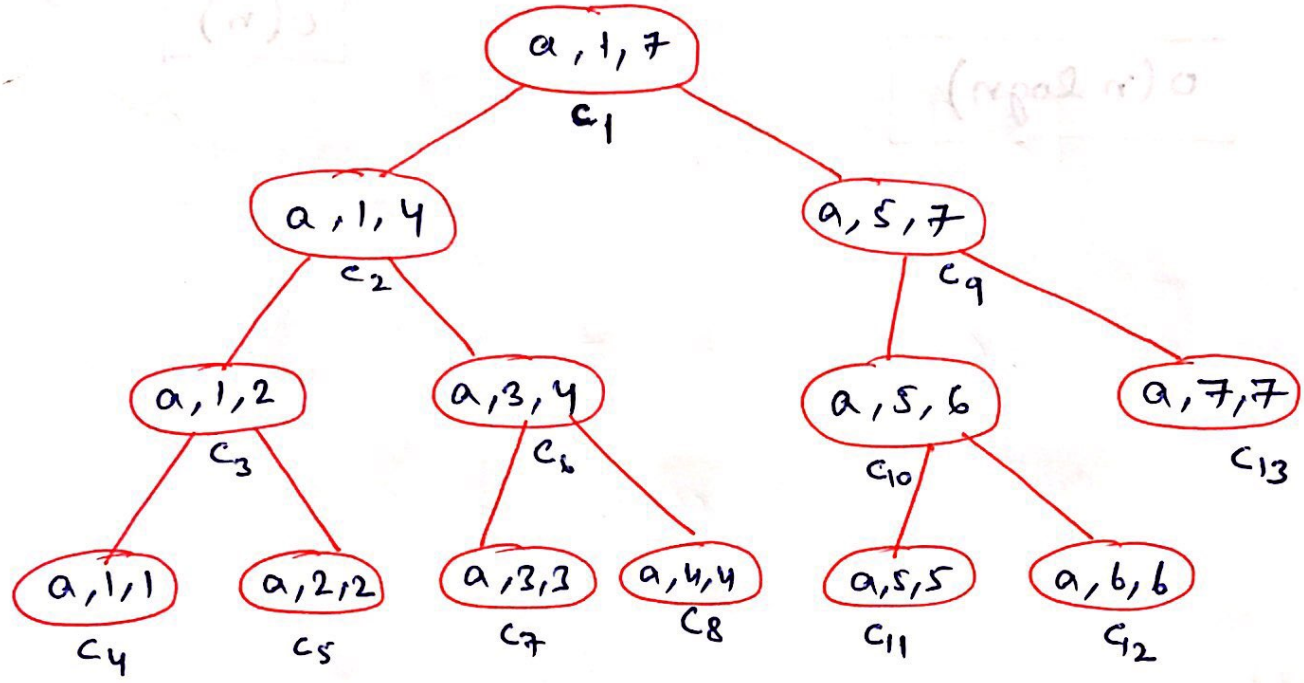
Merge sort :-

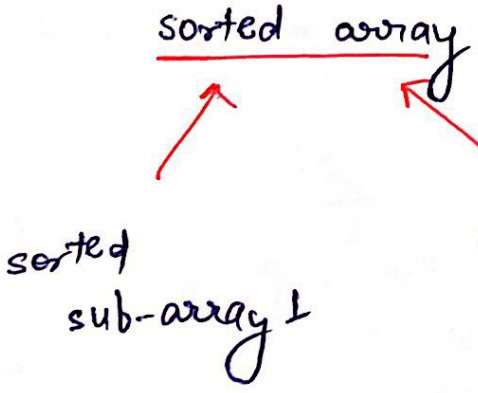
- Merging two sorted sub-arrays.

eg:-

$$A = \begin{bmatrix} 55 & 25 & 75 & 15 & 95 & 17 & 11 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{bmatrix}$$

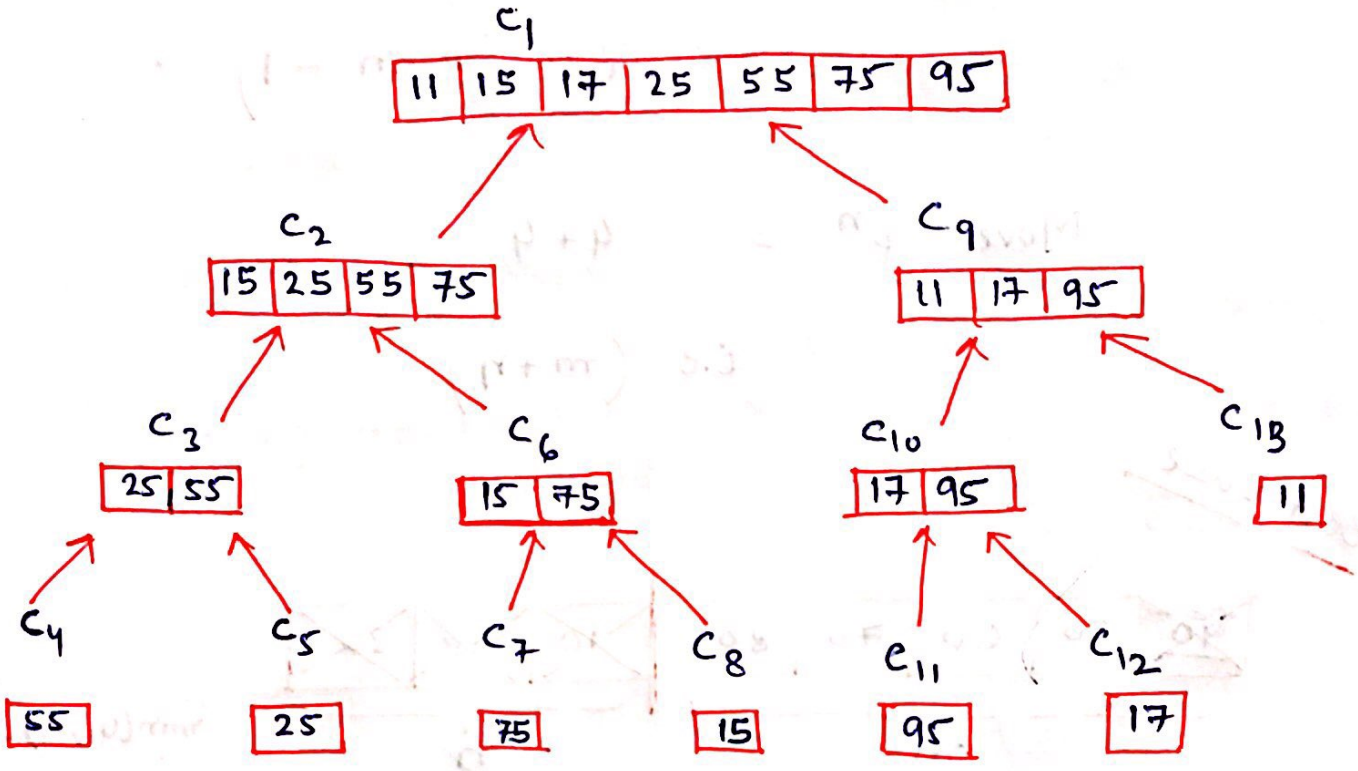


- Merge algorithm or combine algorithm in Merge sort -

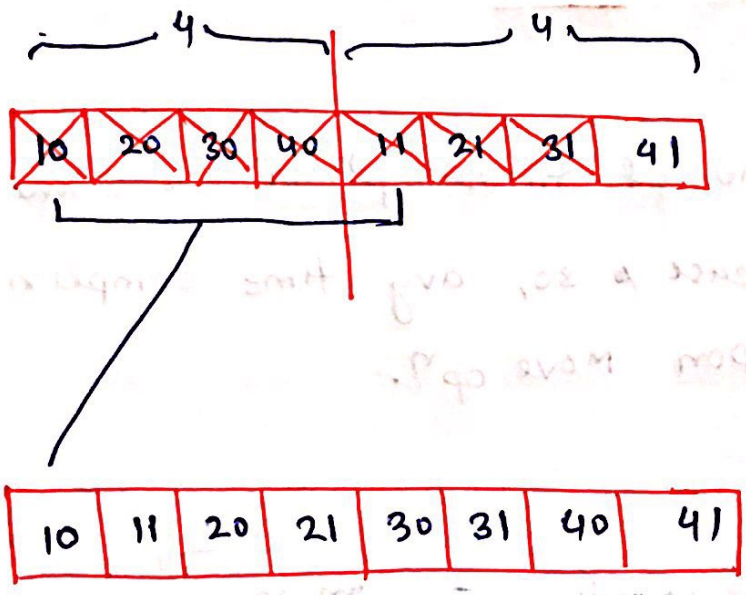


I/p :- 2 sorted sub-array

O/p :- 1 sorted array



↳ Merge algorithm -



compare :- 7

Moves :- 8

compare

min (10, 11) = 10

min (20, 11) = 11

min (20, 21) = 20

min (30, 21) = 21

min (30, 31) = 30

min (40, 31) = 31

min (40, 41) = 40

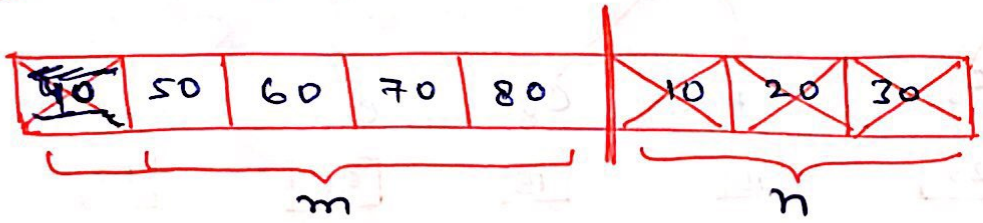
so, compare op^n = 4 + 4 − 1

i.e $(m+n-1)$

Move op^n = 4 + 4

i.e $(m+n)$

Best case



min(40,10) = 10

min(40,20) = 20

min(40,30) = 30

compare op^n = 3 ie min(m,n).

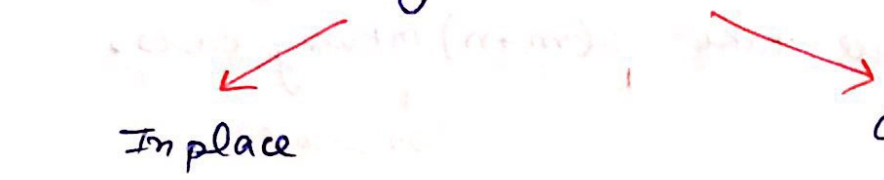
move op^n = $m+n$

NOTE:- So, no. of move op^n is a same in best, and worst case so, avg time complexity depends upon Move op^n.

Time complexity = $m+n$

= $O(n)$

Space complexity -



In place

(merge into same array)

O(n) no. extra space required

Out place

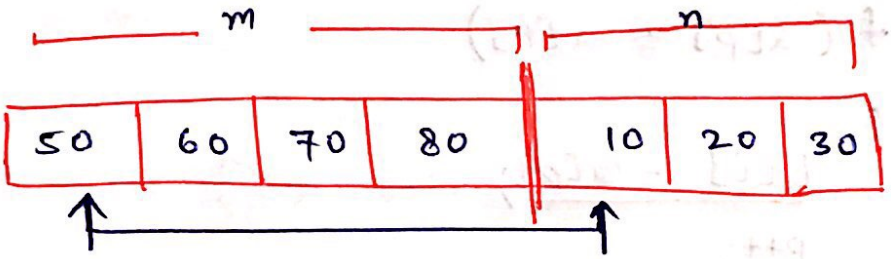
(merge into another array)

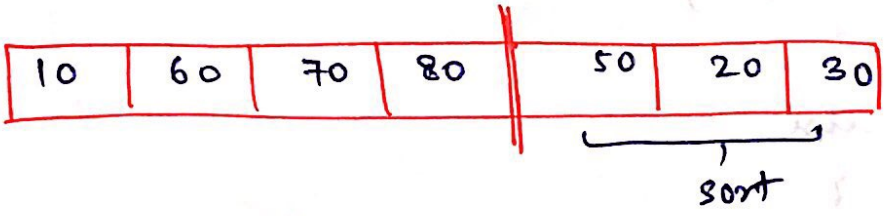
~~no auxillary space required~~ O(n)

$T(n) = n^2$

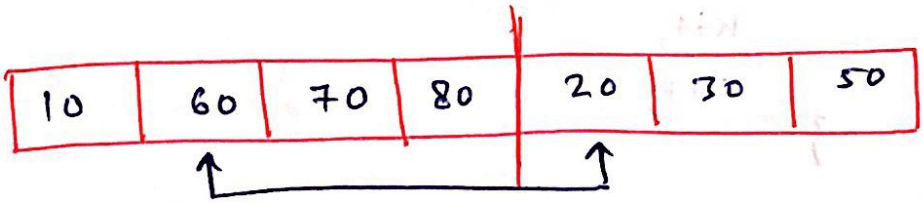
$T(n) = n^2$

In ~~Out~~ place :-





O(n)



$T(n) = m \times n$

NOTE:- Merging two sorted sub-array each of size m and n, will take O(m+n) in Every cases.

outplace

For inplace merging T(n) = O(m×n)

Merge algo code :-

```
k = mid + 1;        (mid = p+q/2)
i = 1;
while ( p ≤ mid  &&  k ≤ q)
{
    if ( a[p] ≤ a[k])
    {
        b[i] = a[p];
        p++;
        i++;
    }
    else
    {
        b[i] = a[k];
        k++;
        i++;
    }
}
```

Move left or remaing element of the sub-array

NOTE:- Merge sort is stable as repeating elements are remain in order after sorting.

Merge - sort algo :-

```
mergeSort(a, p, q)        ---- T(n)
{
    if (p == q)
        return (a[p])     ---- O(1)
    else
    {
        mid = ⌊(p+q)/2⌋ ;  ---- O(1)
        mergeSort(a, p, mid);      ---- T(n/2)
        mergeSort(a, mid+1, q);    ---- T(n/2)
        merge(a, p, mid, mid+1, q); ---- O(n)
    }
    return a;
}
```

(array, last index, start index)

Recurrence relation for Time - complexity :-

$$T(n) = \begin{cases} c & \text{if } n=1 \\ 2T(n/2) + n & n>1 \end{cases}$$

Now, apply substitution method :-

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

$$T(n/2) = 2T(n/4) + n/2$$

$$T(n/4) = 2T(n/8) + n/4$$

So,

$$T(n) = 2\left(2T(n/4) + n/2\right) + n$$

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

$$T(n) = 2^3 T(n/2^3) + 3n$$

↓ k times

$k = \log n$ (space - complexity)

$$T(n) = 2^k T(1) + kn$$

$$= c + n\log n$$

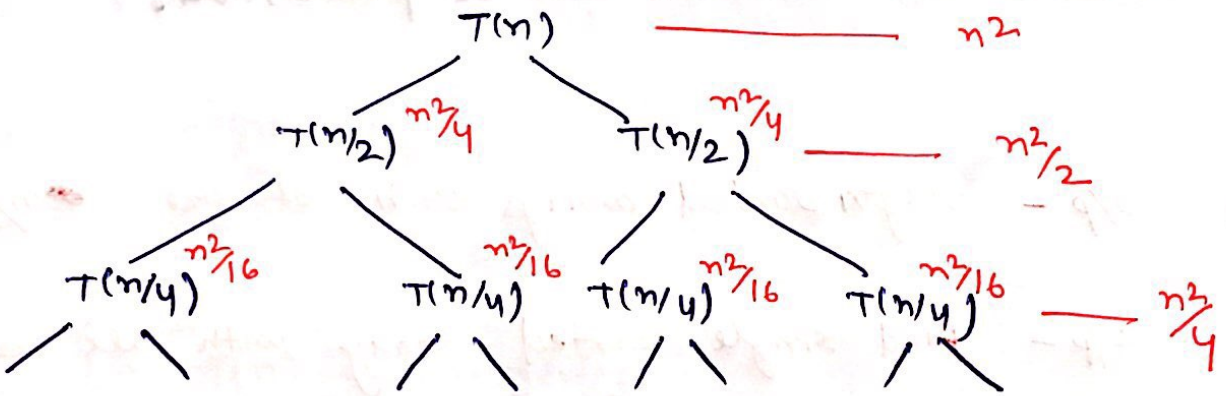
$$T(n) = O(n \log n)$$

NOTE:- - Merging can be inplace or outplace, but in merge sorting merging is outplace.

- $T(n) = O(n \log n)$ in all case best, worst or avg.

Inplace merge-sort :-

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



$$T(n) = (n^2 + n^2/2 + n^2/4 + \cdots\cdots) \quad \text{K times}$$

$$= n^2 \left(1 + 1/2 + 1/2^2 + \frac{1}{2^3} + \cdots\cdots \right)$$

$$T(n) = O(n^2)$$

Merge sort :-

- stack space ($\log n$)
- Total space: stack space + Extra space ($\log n + n$)

out place

- Merge sort is prefferable for larger size array, for smaller size insertion sort is prefferable.

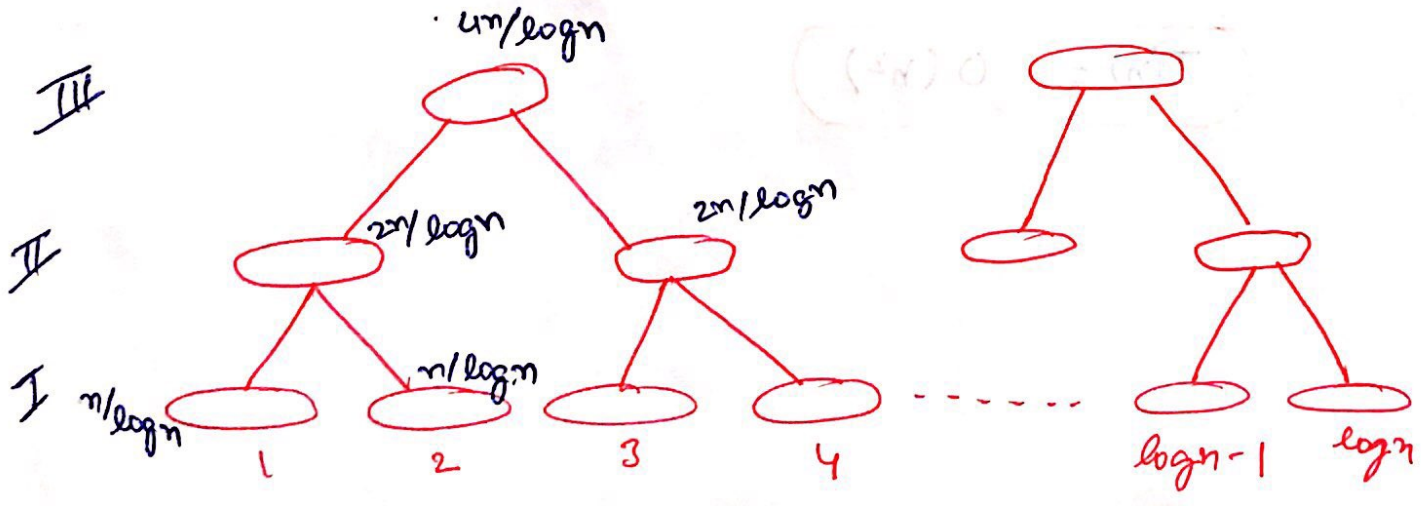
-

Q. I/p - $\log n$ sorted arrays each of size $n/\log n$

O/p - Find single sorted array with all elements.

Find the complexity ?

Total elements = n



III $\quad 4n/\log n \times \frac{\log n}{4} = n$

II $\quad 2n/\log n \times \log n/2 = n$

I $\quad n/\log n \times \log n = n$

For k-level – $T(n) = kn$

After K level – $\frac{2^k \cdot n}{\log n} \times \left(\frac{\log n}{2^k}\right) = n$

$\left(\frac{\log n}{2^k}\right)$ — no. of groups at kth level

so, $\frac{\log n}{2^k} = 1$ so, $k = \log(\log n)$

so, $\boxed{T(n) = n \log(\log n)}$

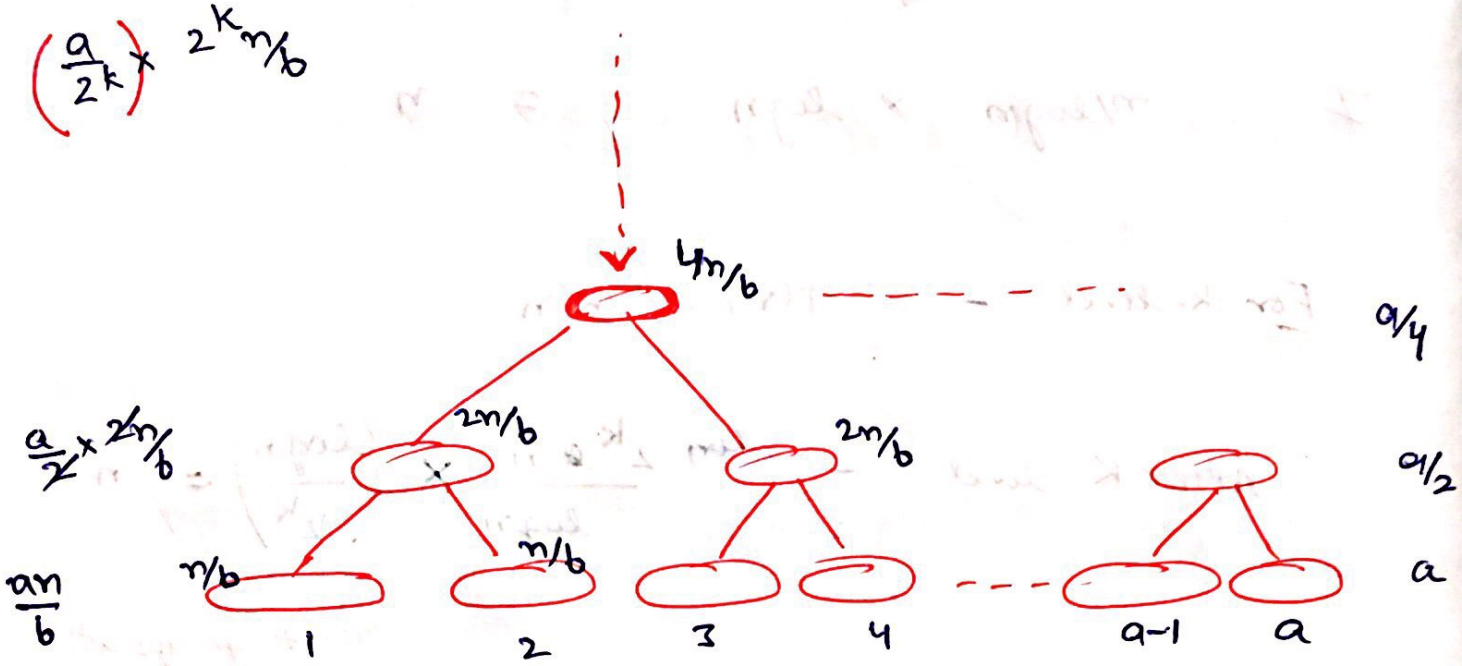
Q: I/p – a-sorted sub arrays each of size n/b

O/p – Find single sorted array with all elements

Find Time complexity ?

$a/2^k = 1$ $\quad k = \log a$

$\left(\frac{a}{2^k}\right) \times 2^k n/b$



so, $T(n) = K \cdot \frac{an}{b}$

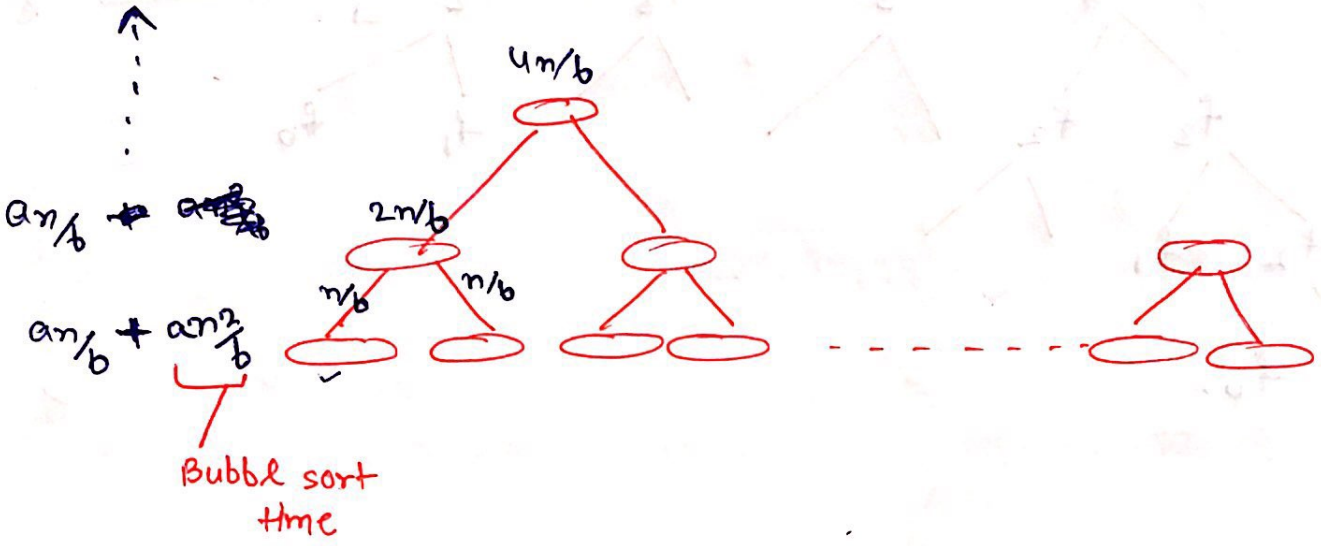
$= \frac{a}{b} \log n \left[\frac{an}{b} \log a\right]$

$$T(n) = O\left(\frac{an \log a}{b}\right)$$

Q. I/p – a sub-array each of size n/b where every sub-arrays are sorted using bubble sort.

O/p – Find single sorted array with all elements.

Find Time complexity ?



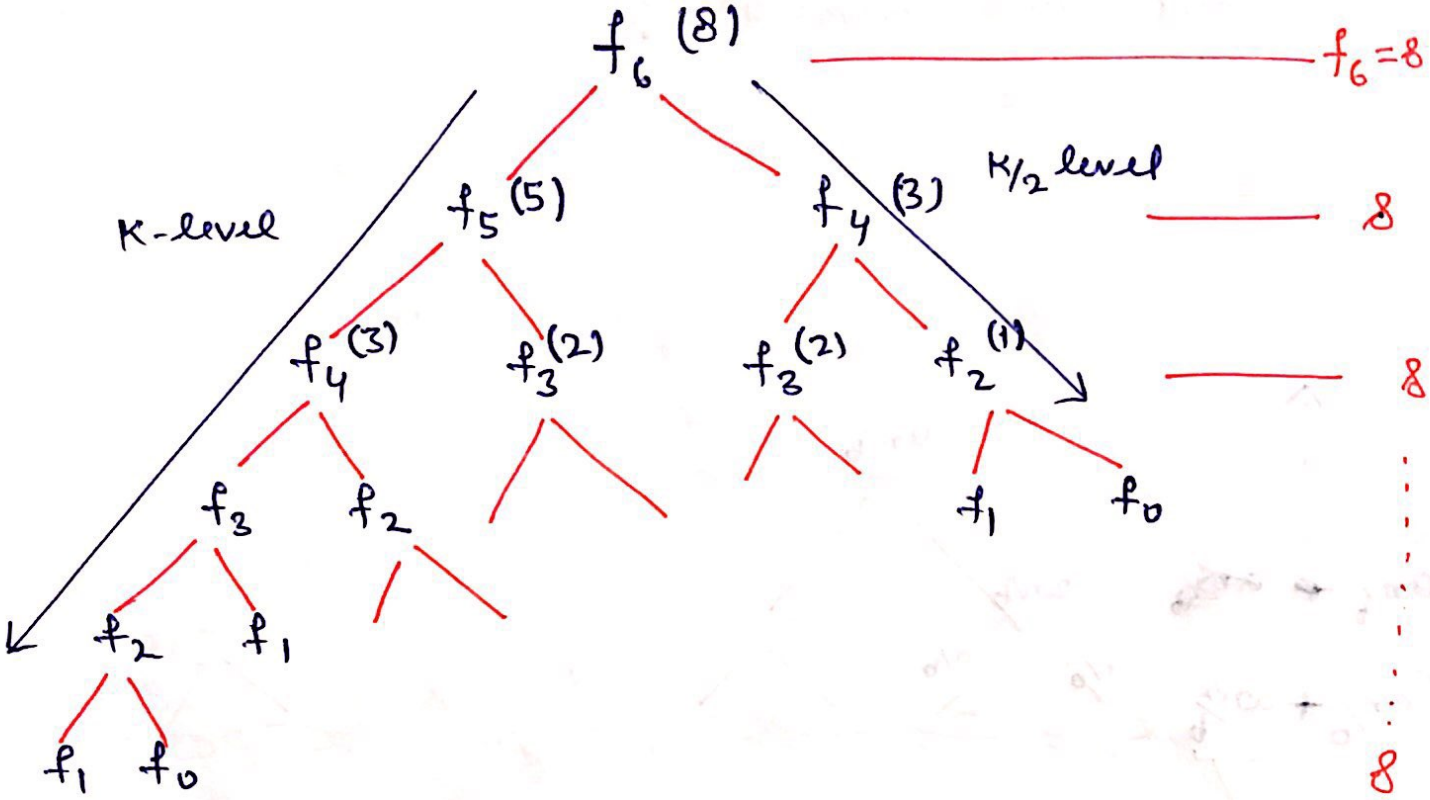
so, $T(n) = K\,an/b + an^2/b$

$= \left(\frac{an \log a}{b} + an^2/b \right)$

eg:-

n	0	1	2	3	4	5	6	7	8	9
f(n)	0	1	1	2	3	5	8	13	21	34

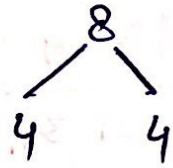
fibinocci merge sort → Divide op^n follow fibnocci series format



Normal merge sort

mid = $\frac{p+q}{2}$

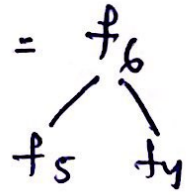
divide = half

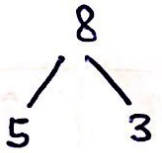


$T(n) = O(n \log n)$

fibnocci merge sort

divide = f_6





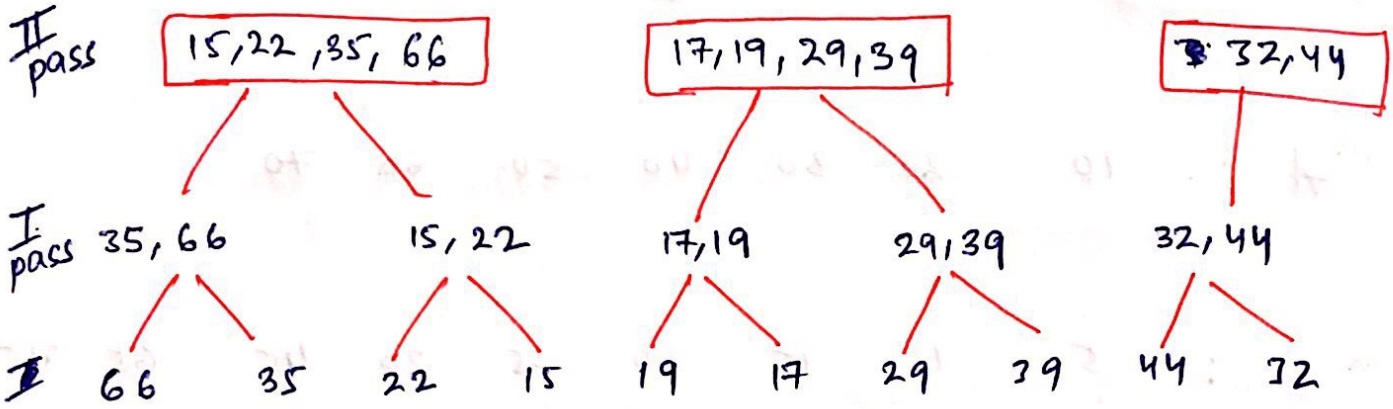
UB $T(n) = O(K \cdot f(K))$

LB $= \Omega(K/2 \, f(K))$

Q. Consider the following array :-

66 35 22 15 19 17 29 39 44 32

what will be the o/p after second pass of the straight 2-way merge sort algorithm?



Total no. of passes $= \lfloor \log_2 n \rfloor$ where n is no. of elements.

Normal merge-sort	straight 2-way merge sort
- First divide then merge	- Direct merge 2 elements
- Top-down approach	- Bottom-up approach
- Practically possible	- Thorotical possible

Q. I/p → 2 sorted array A and B where:

A having m distinct element

B having n distinct element

O/p → Find $A \cap B$, $A \cup B$
$O(n)$ $O(n)$

Find Time complexity?

m A : 10 20 30 40 50 60 70

n B : 5 10 15 20 25 30 45 65 75

For $A \cap B$

(i) Perform Linear search for each m elements:

$$T(n) = m \times n$$

$$= O(mn) = \boxed{O(n^2)}$$

(ii) Perform Binary search for each m elements:

$$T(n) = m \times \log n$$

$$= \boxed{O(n \log n)}$$

(iii) Apple merge algorithm :- (modified)

$$T(n) = O(m+n) = \boxed{O(n)}$$

```
while ( i ≤ m or ~~and~~ j ≤ n)
{
    if ( a[i] == a[j] )
    {
        b[K] = b[K+1] = a[i];
        i++;
        j++;
        K++;
    }
    else if ( a[i] < a[j] )
    {
        i++;
    }
    else
    {
        j++;
    }
}
```

For AUB:-

$T(n) = O(m+n) = O(n)$ merge algorithm

⇒ Quick Sort :-

- Divide and conquer
- In-place
- Not stable
- Practised sorting algo

Partition algo:-

```
Partition ( a, p, q)
{
    x = a[p] ;  # pivot
     i = p;
    for (j = p+1; j ≤ q ; j++)
    {
        if ( a[j] ≤ x)
        { i=i+1;
          swap( a[i], a[j]);
        }
    }
     swap ( a[i], a[p]);
     return (i) ;
}
```

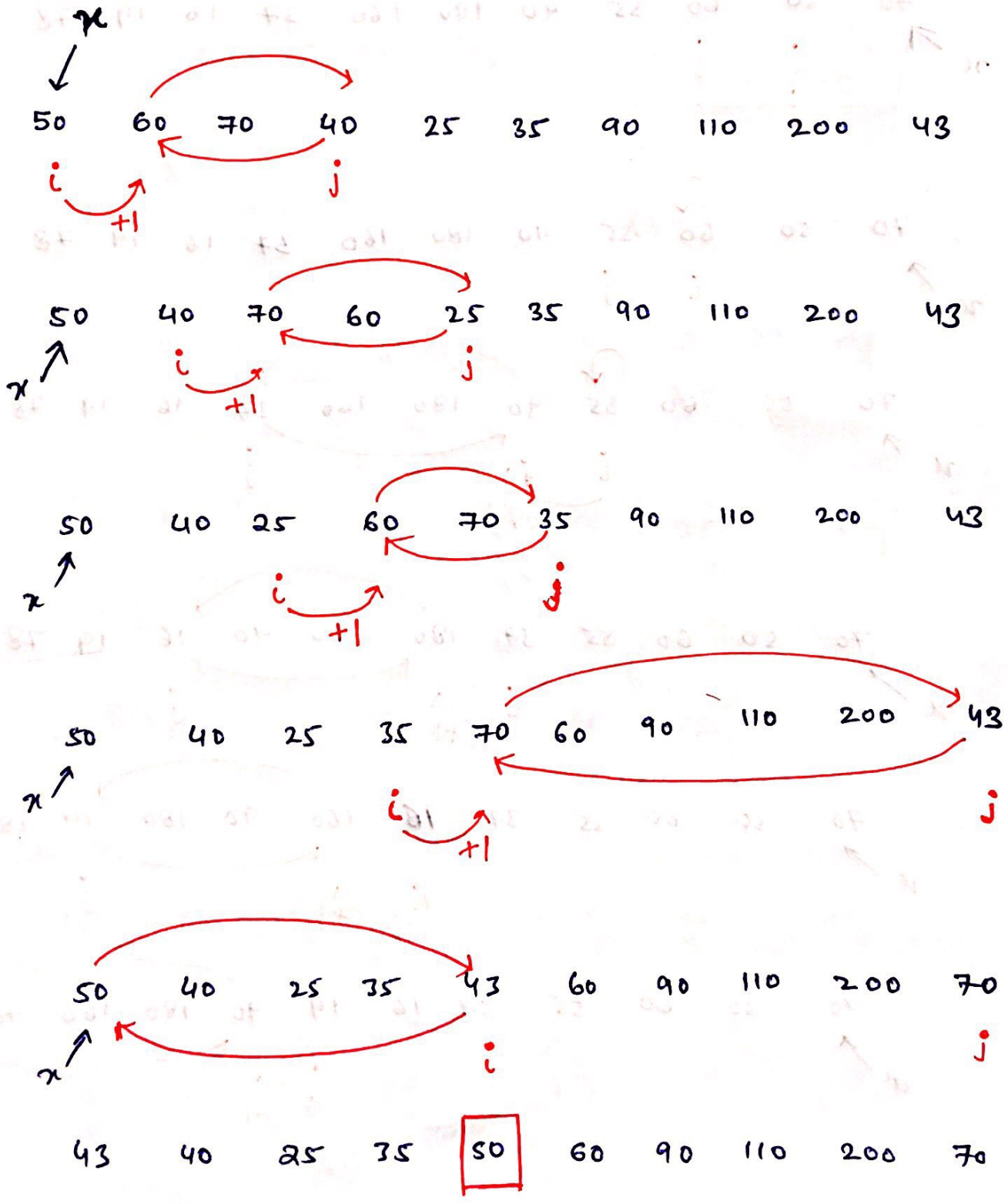
$T(n) = O(n)$

$A = \begin{bmatrix} 50 & 60 & 70 & 40 & 25 & 35 & 90 & 110 & 200 & 43 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{bmatrix}$

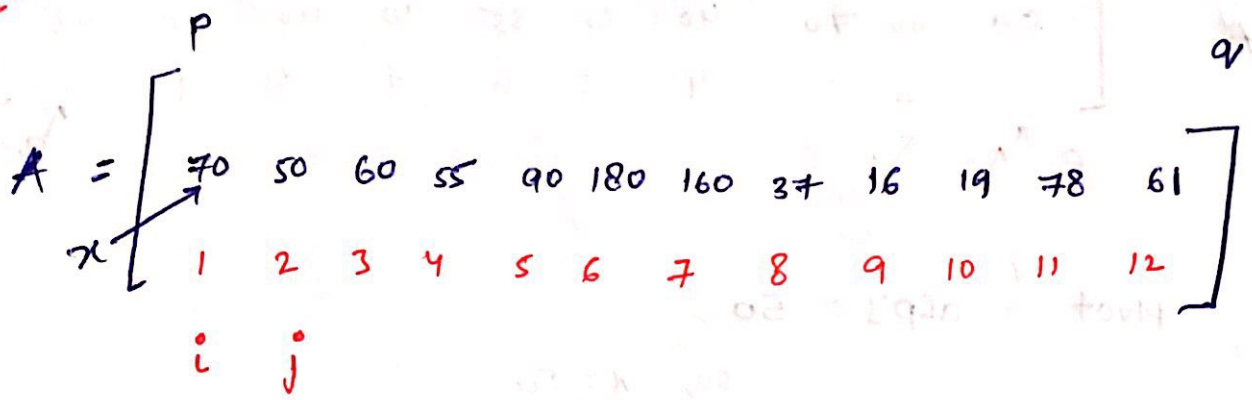
p ↑ i ↑ j ↑ a

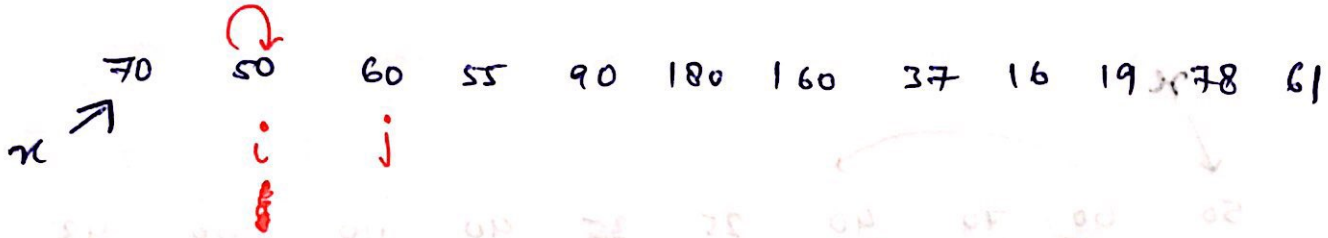
pivot = a[p] = 50

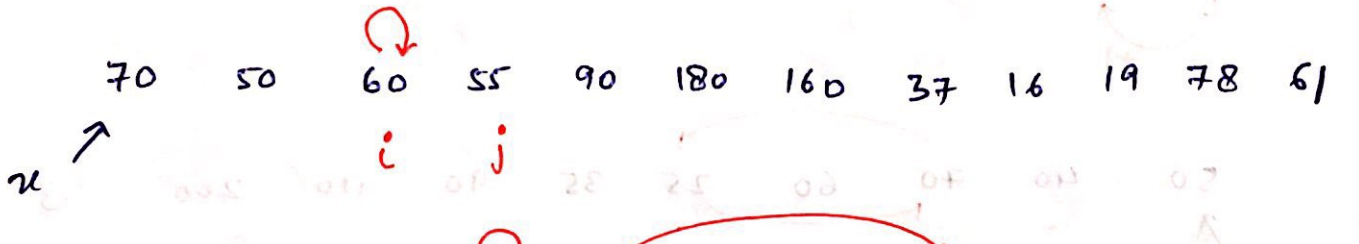
so, $x = 50$

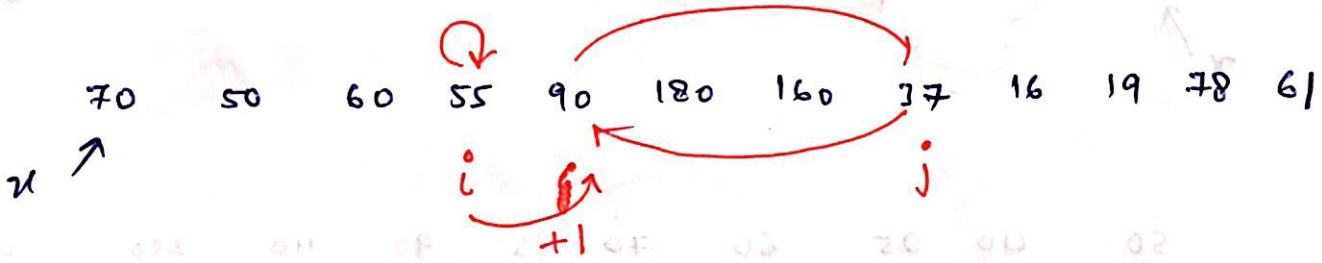


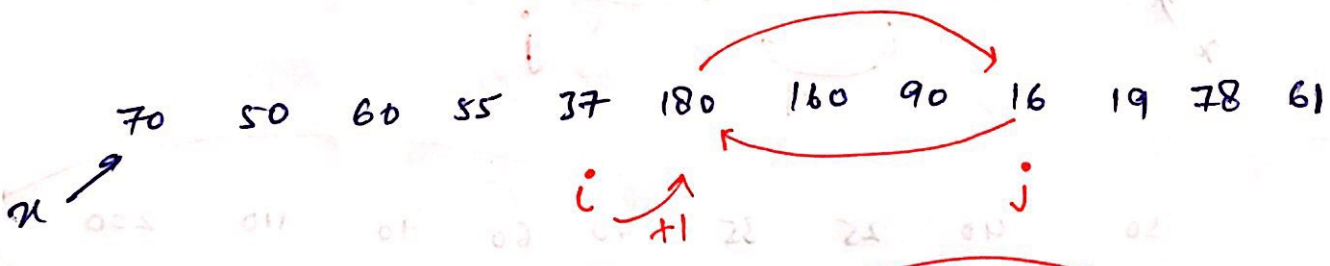
eg:-

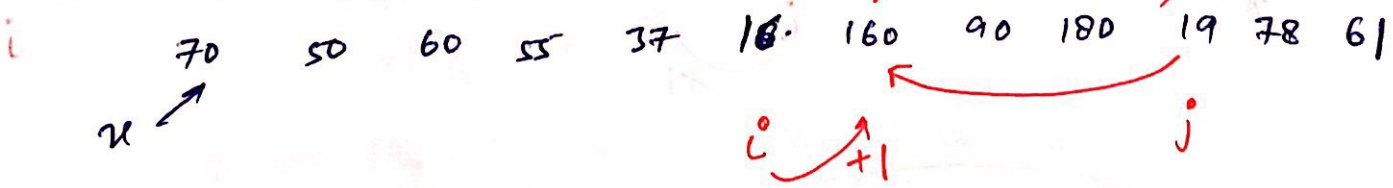

p
q
A = 70 50 60 55 90 180 160 37 16 19 78 61
x
1 2 3 4 5 6 7 8 9 10 11 12
i j

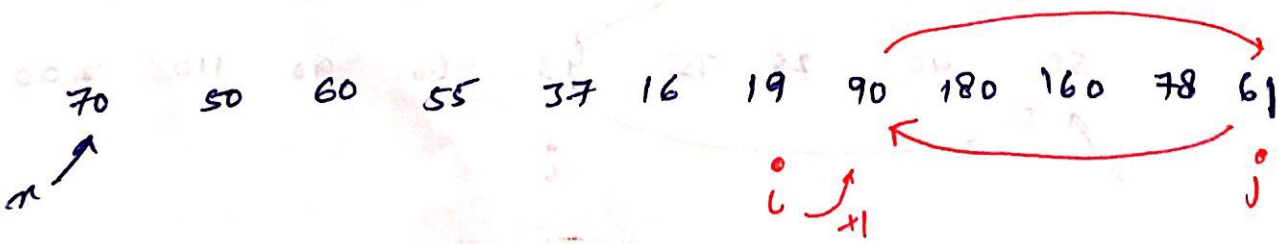

70 50 60 55 90 180 160 37 16 19 78 61
x
i j

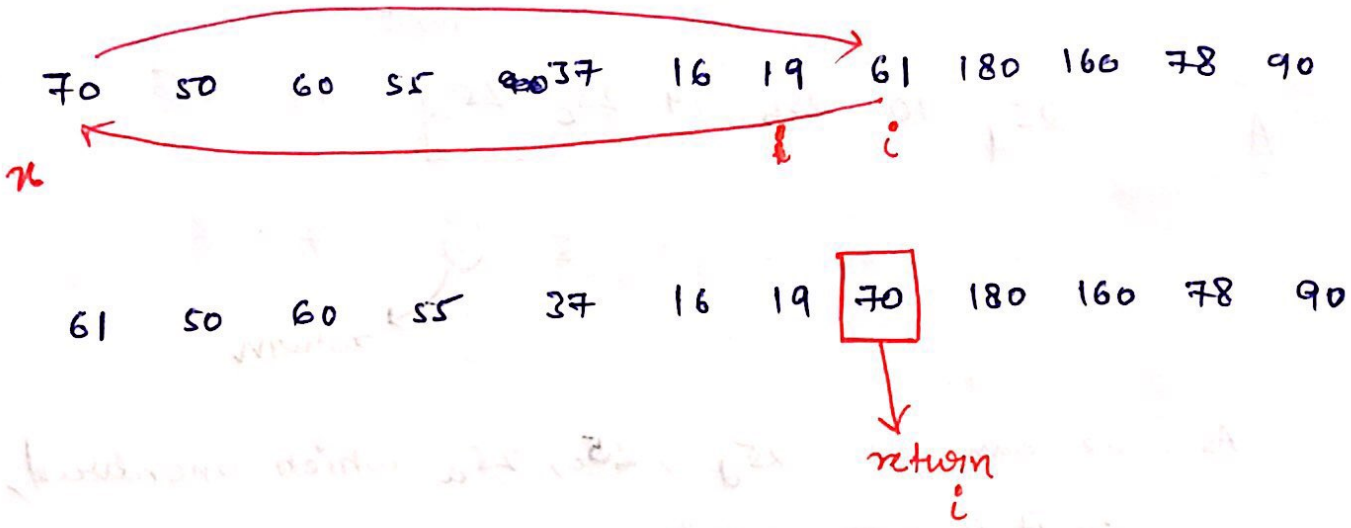

70 50 60 55 90 180 160 37 16 19 78 61
x
i j


70 50 60 55 90 180 160 37 16 19 78 61
x
i
j
+1

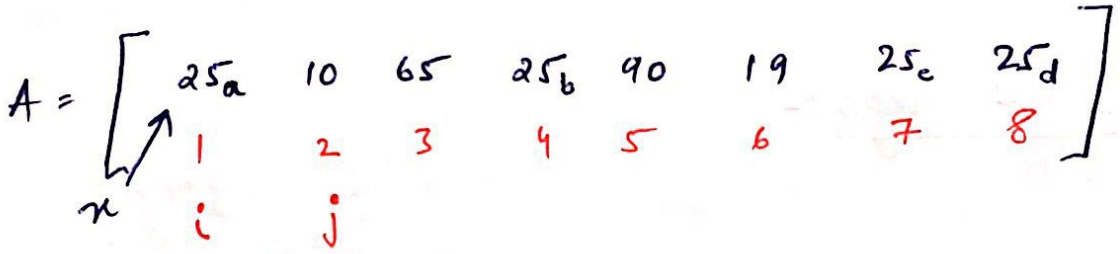

70 50 60 55 37 180 160 90 16 19 78 61
x
i
j
+1

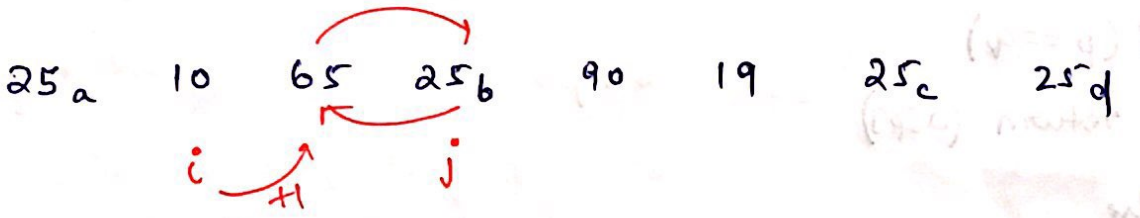

70 50 60 55 37 16 160 90 180 19 78 61
x
i
j
+1

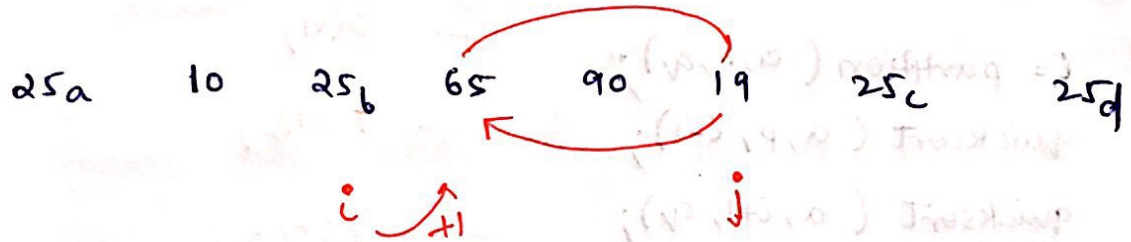

70 50 60 55 37 16 19 90 180 160 78 61
x
i
j
+1

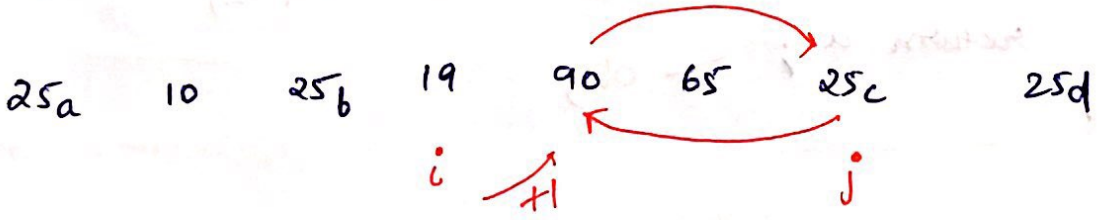

70 50 60 55 37 16 19 61 180 160 78 90
x
i
i
61 50 60 55 37 16 19 70 180 160 78 90
return
i

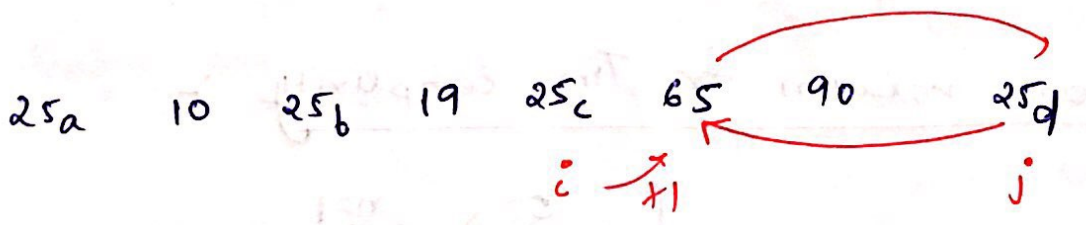
eg:-

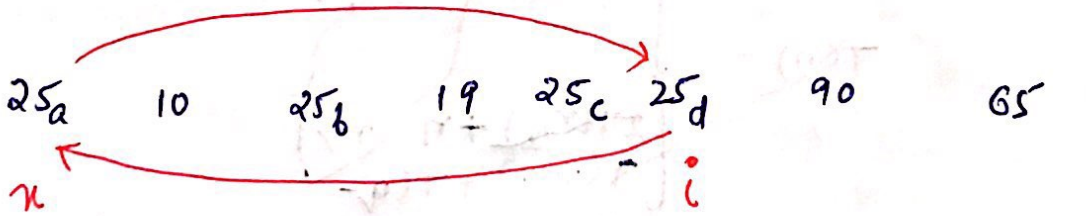

A = [25a 10 65 25b 90 19 25c 25d]
1 2 3 4 5 6 7 8
x
i
j


25a 10 65 25b 90 19 25c 25d
i
+1
j


25a 10 25b 65 90 19 25c 25d
i
+1
j


25a 10 25b 19 90 65 25c 25d
i
+1
j


25a 10 25b 19 25c 65 90 25d
i
+1
j


25a 10 25b 19 25c 25d 90 65
x
i

pivot

$A =$ 25_d 10 25_b 19 25_c 25_a 90 65

1 2 3 4 5 6 7 8

return

As, we can see 25_d, 25_c, 25_a which unordered, so it is not stable.

Quicksort Algo :-

```
quicksort ( a, P, q)                    ——— T(n)
{
    if (p == q)
      return (a[p])                     —— O(1)
    else
    {
      i = partition ( a, P, q);         —— O(n)
      quicksort ( a, P, i-1);           —— T(n/2)
      quicksort ( a, i+1, q);           —— T(n/2)
      return a ;                        —— O(1)
    }
}
```

Recurrence relation for Time complexity :-

$$T(n) = \begin{cases} c & n=1 \\ T(n/2) + n & n>1 \\ T(n-p) + T(q- \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ T(i-p) + T(q-i) + n & n > 1 \end{cases}$$

Best case ↙ Worst case ↘

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

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

$= O(n \log n)$

partition return i such that $i = n/2$

[divide array into 2 equal parts]

$T(n) = n + T(0) + T(n-1)$

$= n + T(n-1)$

$\vdots$

$= O(n^2)$

partition return i such that $i = 1$ or $i = N$ (end positions)

* Average case also belongs to recurrence relation of Best case algorithm.

Space complexity :-

Best case

stack space $= O(\log n)$

$n/2^k = 1$ $\quad k = \log_2 n$

Worst case

stack space $= O(n)$

$n - k = 1$

$k = n - 1$

NOTE :- Quicksort algorithm in worst case behaves as Tail Recursion, so we can write corresponding equivalent non-recursive program of stack space of $O(1)$.

So, quicksort minimum stackspace is 1 and maximum is $\log n$

$$1 \leq \text{stackspace} \leq \log n$$

1 : worst case

$\log n$: best case

So, by using non-recursive program for worst case of quicksort we have modify the space complexity from $O(n)$ to $O(1)$.

Quicksort

$$\text{Spau} = \text{I/p} + \text{Extra}$$
$$= \text{I/p} + \text{stack spau}$$
$$= n + \log n$$
$$= O(n)$$

MergeSort

$$\text{Spau} = \text{I/p} + \text{Extra}$$
$$= \text{I/p} + \text{stack spau} + n$$

n : outplace

$$= n + \log n + n$$
$$= O(n)$$

NOTE:- (Stack space $> \log n$) outplace

(Stack space $\leq \log n$) inplace

Due to this merge sort is outplace.

Q. Consider the following 3 I/p's :

I/p 1 — 1, 2, 3, 4, ... , n-1, n

I/p 2 — n, n-1, n-2, ... 2, 1

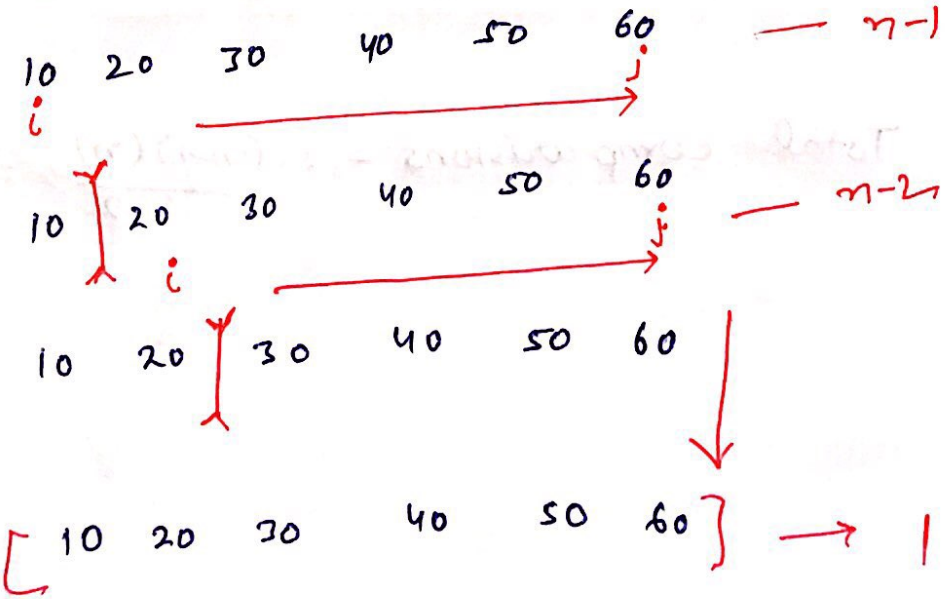
I/p 3 — n, n, n, n - - - - - - n

Let c_1, c_2, c_3 be the number of comparisions made for the I/p 1, I/p 2, & I/p 3, to arrange the array in acending order. Relation b/w c_1, c_2, c_3 ?

for I/p 1 :-

Let n = 6

I/p :



Total no. of levels = n

Total no. of comparision = $n-1 + n-2 + \cdots\cdots 2+1$

($n-1$: 1st level; $n-2$: 2nd)

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

for I/p 2 :-

I/p : 60 50 40 30 20 10

1 partion : 10 50 40 30 20 60 $n-1$

2 " : 10 50 40 30 20 60 $n-2$

3 " : 10 20 40 30 50 60 $n-3$

⋮

Total no. of levels = n

Total comparisions = $\frac{(n-1)(n)}{2} = O(n^2)$

for I/p 3 :-

I/p : 10_a 10_b 10_c 10_d 10_e 10_f

partition 1 : 10_f 10_b 10_c 10_d 10_e $\boxed{10_a}$

10_e 10_b 10_c 10_d $\boxed{10_f}$ $\boxed{10a}$

Total no. of levels = n

So, total comparision = $n-1 + n-2 + n-3 + \cdots 2+1$

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

So, $\boxed{c_1 = c_2 = c_3}$

NOTE:- If array is already sorted or almost sorted than quicksort will give worst case time complexity i.e $O(n^2)$.

But Insertion sort will give best case i.e $O(n)$.

Generalized form for Best & Average case :-

$T(n) = n + T(n/2) + T(n/2) \Rightarrow O(n \log_2 n)$

$T(n) = n + T(n/5) + T(4n/5) \Rightarrow O(n \log_{5/4} n)$

$\Omega(n \log_5 n)$

$T(n) = n + T(n/10) + T(9n/10) \Rightarrow O(n \log_{10/9} n)$

$\Omega(n \log_{10} n)$

$\Theta(n \log n)$

$$T(n) = n + T(\alpha n) + T((1-\alpha)n)$$

$(0 < \alpha < 1)$

Generalized form for Worst case :-

$T(n) = n + T(0) + T(n-1) \Rightarrow n/1 \times n = O(n^2)$

$T(n) = n + T(1) + T(n-2) \Rightarrow n/2 \times n = O(n^2)$

$T(n) = n + T(9) + T(n-10) \Rightarrow \frac{n}{10} \times n = O(n^2)$

$$T(n) = n + T(c-1) + T(n-c) \quad (c>1)$$

$n/c \times n = O(n^2)$

Q. In Quicksort, the sorting of n numbers, the $n/5^{th}$ smallest Element is selected as pivot using $O(n^2)$ time complexity. Then what will be the best case time complexity of Quicksort.

(a) $O(n^2)$ (b) $O(n^3)$

(c) $O(n \log n)$ (d) $O(\log n)$

* $O(n \log n)$ and $O(n)$ can't be possible as pivot selection time complexity is already $O(n^2)$.

In conventional partition algorithm :-

$T(n)$ = Pivot selection + Loop for swapping

$= O(1) + O(n)$

$= O(n)$

But here Pivot selection $= O(n^2)$

so, $T(n)$ of partition algo $= O(n^2) + O(n)$

$= \underline{O(n^2)}$

$n/5^{th}$ smallest element will go to $n/5^{th}$ position after partion

eg:-

1	2	3	4	5	6	7
40	20	10	50	70	60	30

( )'40 ( )

Since 40 is 4th smallest element so it is shifted to 4th position after partition.

So, Now Time complexity for Quick sort -

$$T(n) = O(1) + O(n) + T(n/2) + T(n/2)$$

pivot selection — swapping loop — partition algo

Now,

$$T(n) = O(n^2) + O(n) + T(n/5 - 1) + T(n - n/5)$$

$$=$$

$$(\ n/5 - 1\ )\ n/5\ (\ n - n/5\ )$$

$$T(n) = O(n^2) + O(n) + T(n/5) + T(4n/5)$$

After solving, we get

$$T(n) = O(n^2)$$

Q: In quicksort the sorting of n numbers, $n/10^{th}$ element is selected as pivot using $O(\log n)$ time complexity algorithm. Then what will be the worst case time complexity of quick sort?

$$T(n) = \underbrace{O(\log n)}_{\text{pivot selection}} + \underbrace{O(n)}_{\text{Loop}} + \underbrace{T(0) + T(n-1)}_{\text{worst case}}$$

$\underbrace{\qquad\qquad}_{\text{partition algo}}$

- since here $n/10^{th}$ element can go anywhere after partition so we have to choose the extreme case for worst case as given in question.

so, $T(n) = T(n-1) + n$

$$T(n) = O(n^2)$$

Q: In quicksort, the sorting of n numbers, 25th largest element is selected as pivot using $O(n^2)$ time complexity, then what will be the best case time complexity of algorithm?

$$T(n) = O(n^2) + O(n) + T(24) + T(n-25)$$

$$\left(\frac{n-24-1}{}\right) E \left(\frac{24}{}\right)$$

E : $n-24$

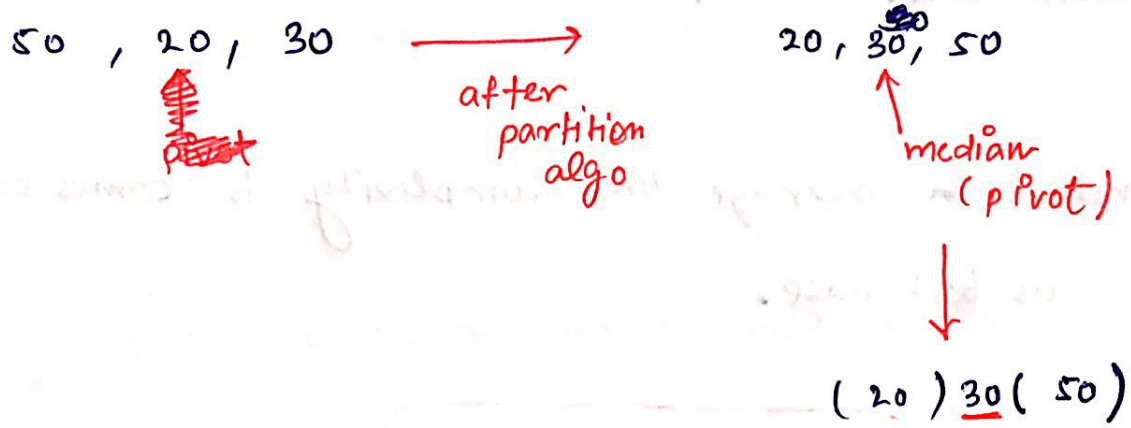
$$T(n) = n^2 + \underbrace{T(24)}_{\text{Const}} + T(n-25)$$

$$= n^2 \times n/25 = O(n^3)$$

$$\boxed{T(n) = O(n^3)}$$

- The 25th largest element will be at $(25-1)^{th}$ position from last.

NOTE :- If median element is selected as pivot then Quick-sort follows best case with $T(n) = O(n \log n)$.



= Randomized Quicksort :-

Select pivot element randomly in partition then sorting is known as randomized Quicksort.

	1	2	3	4	5	6	7
A [	10	20	30	40	50	60	70]

$r \leftarrow$ random (A, 1, 7)

$r \leftarrow 4$ (Let)

pivot $\leftarrow$ A[r] i.e 40

then this becomes best case

But sometime r can be 1 or n i.e

r

pivot = A[1] or A[n] then this leads to

worst case. $O(n^2)$

Hence, in average time complexity is comes same as best case.

$$T(n) = O(n \log n)$$

But if all element in an array are same then randomized quicksort will also gives worst case time complexity $O(n^2)$.

NOTE:- Randomized quicksort can solve problem of sorted array but not for identical element array.

But on an average Randomized quicksort is efficient then ordinary quicksort.