Assignment – 01

CSE-221 (Theory)

Name: Nafisa Khan Youkee

Student Id: 2130149

Section: 02

Ans: to the Q: NO- 01

if n = 5

Outer loop	Inner loop runs
k = 5 →	4 times
k = 4 →	3 times
k = 2 →	1 time
k = 1 →	0 time

Outer loop → O(n)

inner Loop → O(n-1)

∴ time complexity = O(n) * O(n-1)

= O(n²-n)

= O(n²) (Ans)

Ans: to the Q: No-02

Slowest ⟶ fastest

$$n! > 7^n > n^r > e^{\ln(n)} > \sqrt[3]{n}\log n > \sqrt{n}$$

Ans: to the Q: No-03

$2n^3 + 5n^2 + 6n + 18 = \Theta(n^3)$

(i) n^3 is the tight bound of $2n^3 + 5n^2 + 6n + 18$

Upper bound

From definition,

$f(n) = O(g(n))$ there exists a positive constant c_1, c_2 and n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

(ii) True

(iii)

$2n^3 + 5n^2 + 6n + 18 \leq c_1 n^3$

$n_0 = 1$

$c_1 = 31$

lower bound

From definition,

$f(n) = \Omega(g(n))$; there exists a positive constant c_2 and n_2 such that $f(n) \geq c_2 \cdot g(n)$. for all $n \geq n_0$.

(ii) True

(iv)

$$2n^3 + 5n^2 + 6n + 18 \geq c_2 n^3$$

$$n_0 = 1$$

$$c_2 = 1$$

$\therefore$ $2n^3 + 5n^2 + 6n + 18 = \Theta(n^3)$ (Shown)

Searching and Sorting

Ans: to the Q: NO-04

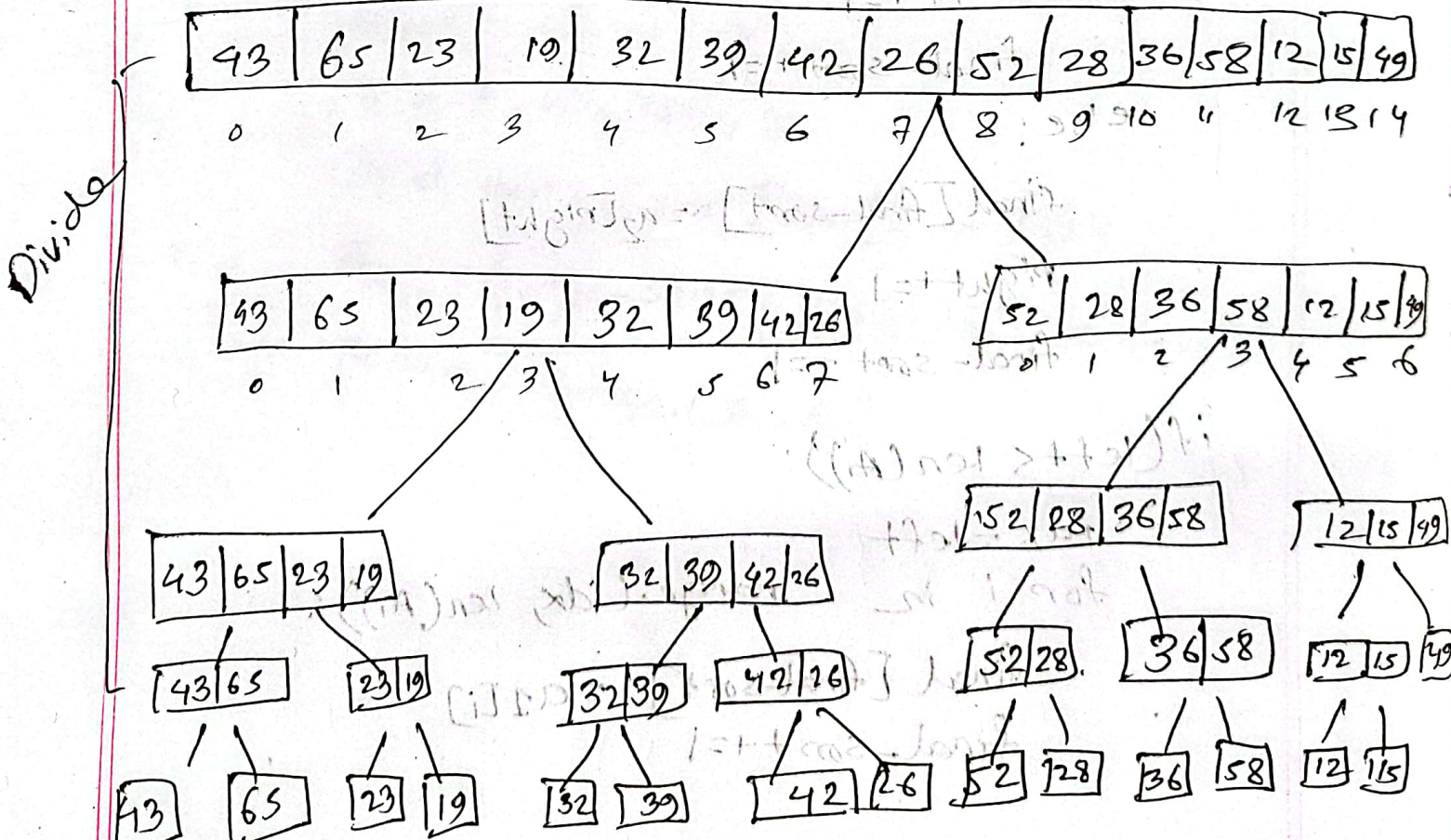
Merge sort algorithm:

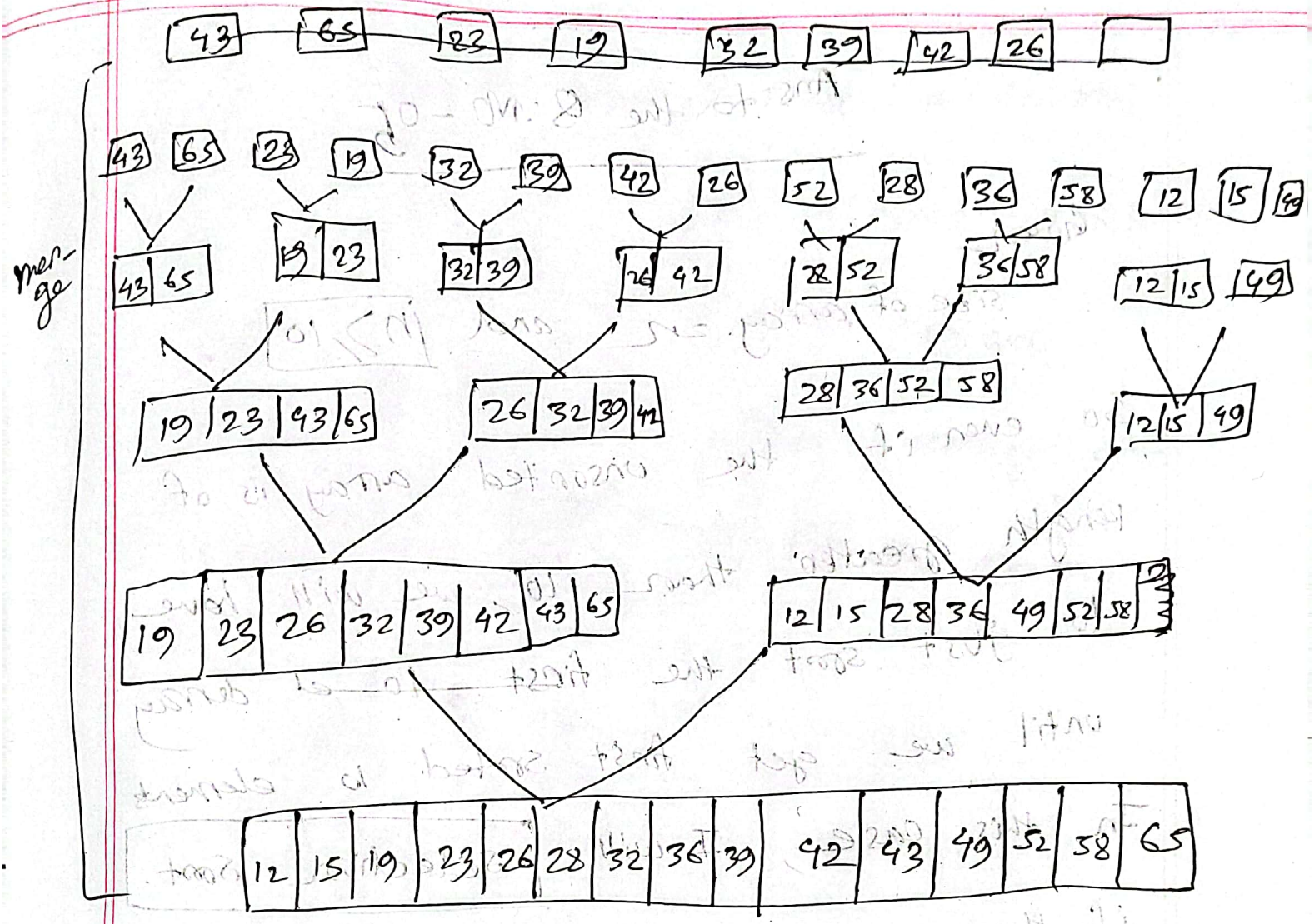
```
Mergesort(arr):
    if(len(arr) == 1):
        return arr
    else:
        mid = len(arr)//2
        A1 = merge-sort([arr[0:mid]])
        A2 = merge-sort(arr[mid: len(arr)])
        return merge(A1, A2)
```

```
merge(A1, A2):
    left=0
    right=0
    final_sort = 0
    final = [0]*(len(A1) + len(A2))
    while(left < len(a1) and right < len(a2)):
        if(a1[left] < a2[right]):
            final[final_sort] = a1[left]
            left+=1
            final_sort+=1
        else:
            final[final_sort] = a2[right]
            right+=1
            final_sort+=1
    if(left< len(A1)):
        idx=left
        for i in range(idx, len(A1)):
            final[final_sort] = a1[i]
            final_sort+=1
```

```
if (right < len(A2)):
    idx = right
    for right in range(idx, len(a2)):
        final[final_sort] = a2[right]
        final_sort += 1
return final
```

Sorting the given array using merge sort




43 65 23 19 32 39 42 26
43 65 23 19 32 39 42 26 52 28 36 58 12 15 49
merge
43 65 19 23 32 39 26 42 28 52 36 58 12 15 49
19 23 43 65 26 32 39 42 28 36 52 58 12 15 49
19 23 26 32 39 42 43 65 12 15 28 36 49 52 58
12 15 19 23 26 28 32 36 39 42 43 49 52 58 65

Ans: to the Q: NO - 05

Given size of array = n and $n \geq 10$

So even if the unsorted array is of length greater than 10 we will have to just sort the first 10 el derray until we get first sorted 10 elements.

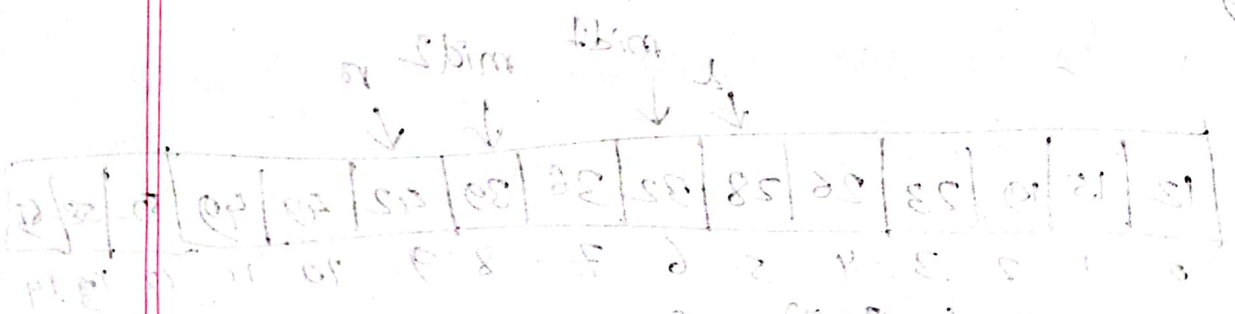
In this case, I will selection Soort.

if the outer loop is like: for(int i=0; i<10; i++)

there the outer loop will run 10 times.

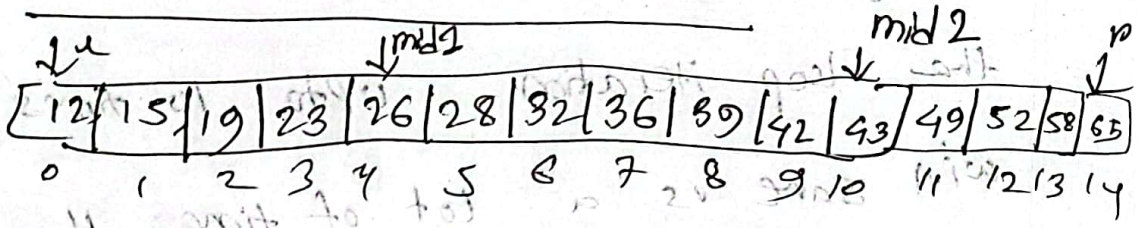
The Best case time complexity is O(n) and worst case is O(n²). After each iteration of outer loop, we need to ck each big element gets located in the right position.

Using selection sort, we can control the loop iteration upto 10 times which will save us a lot of time. Moreover, selection sort algorithm offers sorting with the least amount of swaps. For this reason I will use selection sort.



Ans: to the Q.No - 05

Step-1 array =



$l = 0$

$r = 14$

$mid1 = 0 + \frac{14-0}{3} = 4$

$mid2 = 14 - \frac{14}{3} = 10$

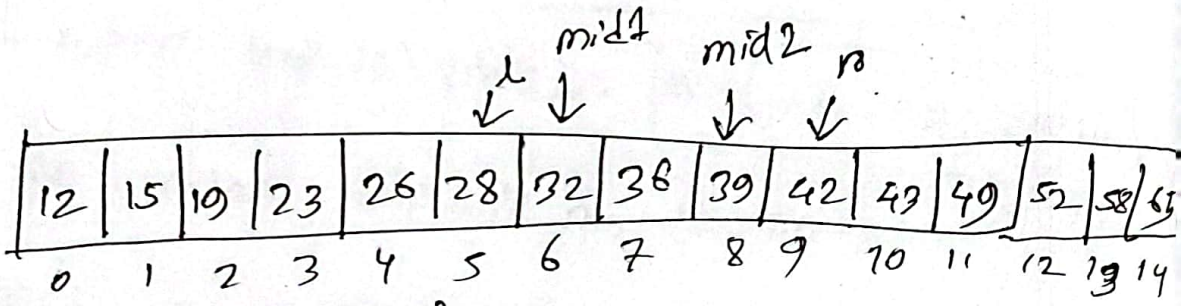
$arr[mid1] = 26$; element

$arr[mid2] = 43$

$l = 4+1 = 5$

$r = 9$

Step:2



$mid1 = 5 + (\frac{9-5}{3}) = 6$ $\quad l = 5$

$mid2 = 9 - 1 = 8$ $\quad r = 9$

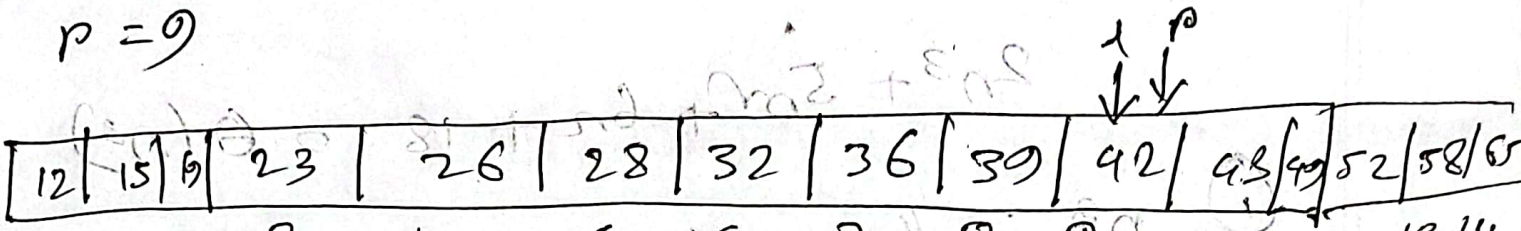
$arr[mid1] = 32$

$arr[mid2] = 39$; $l = 9$

Step : 3

l = 9

r = 9



$mid2 = 9 + \frac{0}{3}$

$= 9$

$mid2 = 9 - 0$

$= 9$

$arr[mid2] = 42$

$arr[mid2] = 42$

Step : 4 output: 42 found at index 9.

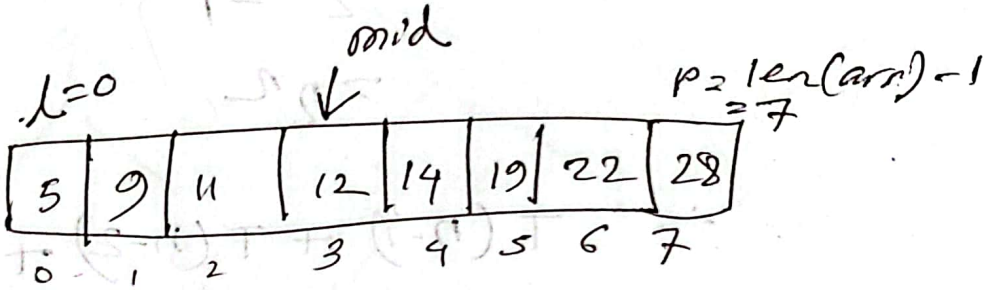
Ans: to the Q: No-07

Binary Search

Given array: [5, 9, 11, 12, 14, 19, 22, 28]

element to be found: 13 (doesn't exist in the array)

Step-1



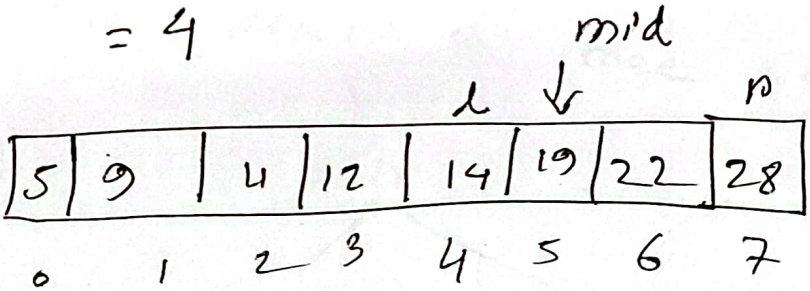
arr[mid] = 12

arr[mid] < element

l = mid+1

= 4

Step: 2



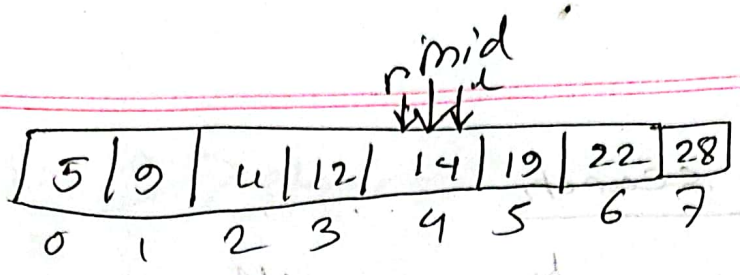
mid = 4+7//2 = 5

arr[mid] = 19

arr[mid] > element

r = 5-1 = 4

Step: 3



$mid = \frac{4+4}{2} = 4$

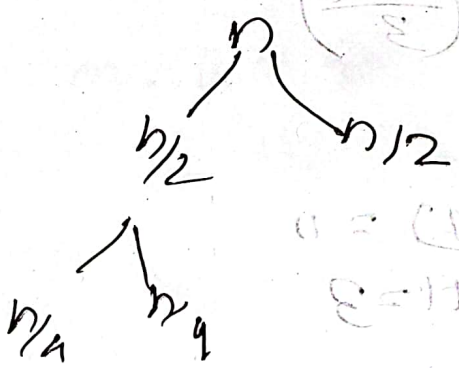
arr[mid] > element

r = 4 - 1 = 3

Now, (r < l) so the loop breaks

output: "element not found"

Time complexity:



$T(n) = O(\log_2 n)$

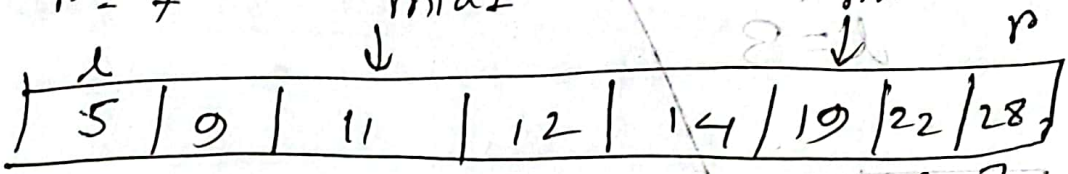
Ternary search

Here element = 13

l = 0

r = 7

Step 1



$mid1 = 0 + \frac{7-0}{3}$

$= 2$

$mid2 = 7 - \left(\frac{7-0}{3}\right)$

$= 5$

arr[mid1] = 11

arr[mid2] = 19 > element

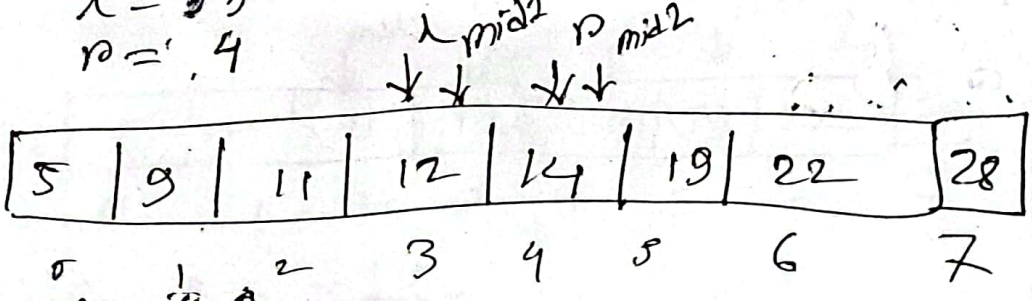
l = 3

r = 4

Step 2

l = 3

r = 4



$mid1 = 3 + \frac{4-3}{3} = 3$

$mid2 = 4 - 0 = 4$

arr[mid1] = 12 ; else

arr[mid2] = 14

l = 4
r = 3

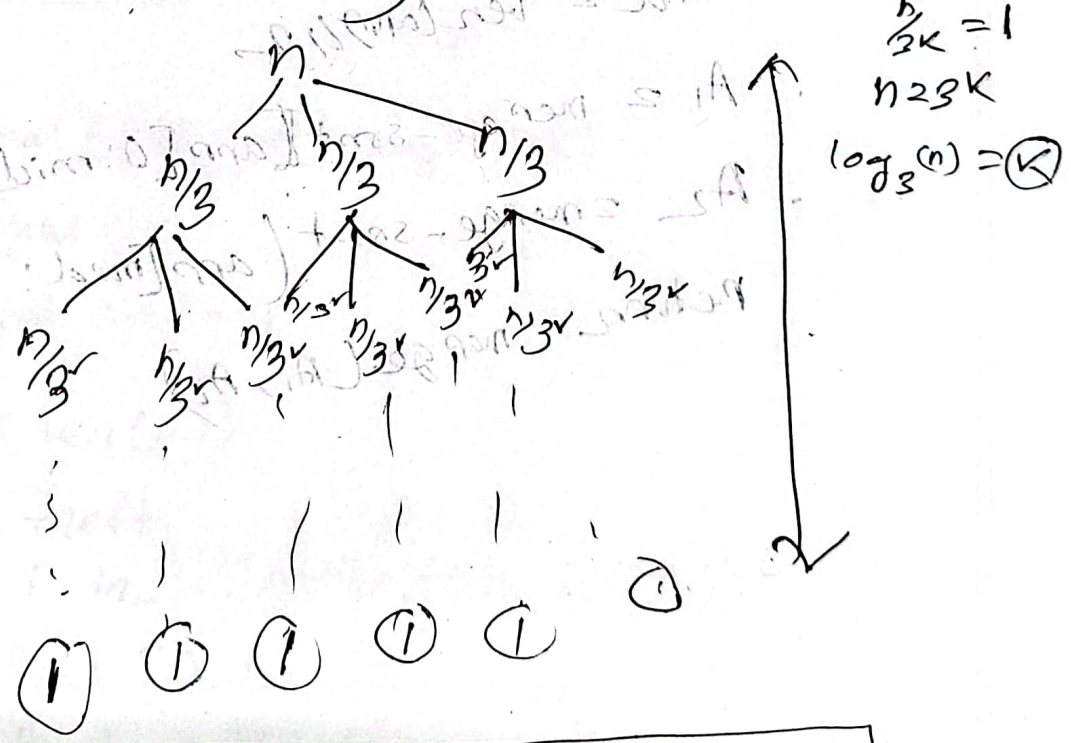
Step-3

l = 4
r = 3

Here l > r ; so loop ends

Step-4 output : "element not found"

Time complexity of ternary search



∴ time complexity = $\log_3(n)$

Ans: to the Q: No-08

8.a Quick sort algorithm using the last element as piviot:

main function

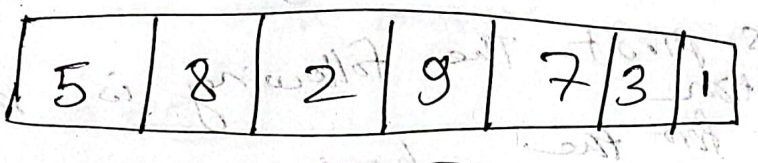
```
quick sort (arr, st, end):
    if ( st < end) {
        mid = Partition (arr, st, end)
        quicksort (arr, st, mid-1)
        quicksort (arr, st, mid+1)
```

Partition function

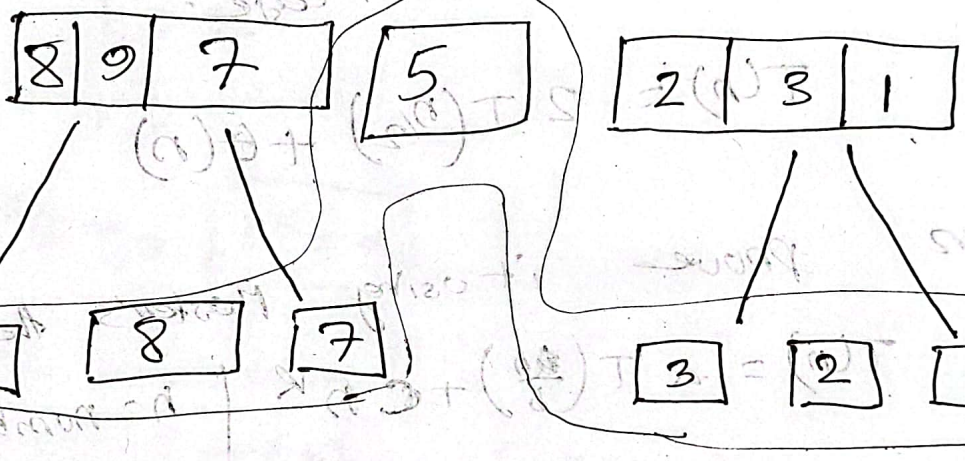
```
pivot = arr [end]
i = st-1
j = end-1
while (i<j):
    while (i<end and arr[i]<pivot):
        i+=1
    while (j>st and arr[j])=pivot):
        j-=1
    if (i<j):
        arr[i], arr[j] = arr[j], arr[i]
return i
```

8. b. Quick sort algorithm simulation (descending order)

Unsorted array, A = [5, 8, 2, 9, 7, 3, 1]



Pivot: 5



Pivot: 8

Pivot: 3

So now after running quick sort algorithm the array becomes A = [9, 8, 7, 5, 3, 2, 1]

8c Best case running time of the quick sort algorithm is $O(n \log n)$. It occurs when the partition process always picks the middle element as pivot. The following is recurrence equation relation for the best case:

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

We can prove it using Master's theorem,

$$T(n) = aT\left(\frac{n}{b}\right) + cn^k$$

n = number of elements in the array
a, b = constants

Now,

$$T(n) = \begin{cases} O(n^{\log_b a}) ; b^k < a \\ O(n^k \log n) ; b^k = a \\ O(n^k) ; b^k > a \end{cases}$$

Now,

$$T(n) = 2\,T(n/2) + O(n).$$

$$= 2\,T(n/2) + cn \quad \text{---} (1)$$

Comparing (1) with masters theorem,

$$a = 2$$

$$b = 2$$

$$k = 1$$

$\therefore$ Here, $\boxed{b^k = a}$

we can write,

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

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

(proved)

9.a

Recursive Time complexity:

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

Using Masters theorem

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

Comparing the constants with master's theorem

$a = 2$

$b = 2$

$c = 1$

$k = 3$

Here, $b^k = 2^3 = 8$. So, $b^k > a$

Therefore, $T(n) = O(n^k)$

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

Ans to the 8: No 9.b

$$T(n) = T(n/4) + T\left(\frac{5n}{8}\right) + n$$

For $T(n/4) + n$, —①

Comparing ① with Masters theorem,

$$T(n) = a\,T(n/b) + c \cdot n^k$$

Here $a = 1$

$b = 4$

$c = 1$

$k = 1$

$b^k = 4 > a = 1$

$\therefore \boxed{b^k > a}$

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

$$= O(n)$$

Now, For $T\left(\frac{5n}{8}\right) + n$

$a = 1$

$b = 8/5$

$c = 1$

$k = 1$

$b^k = 8/5 = 1.6$

$\therefore b^k > a$

$\therefore T(n) = O(n^k)$

$= O(n)$

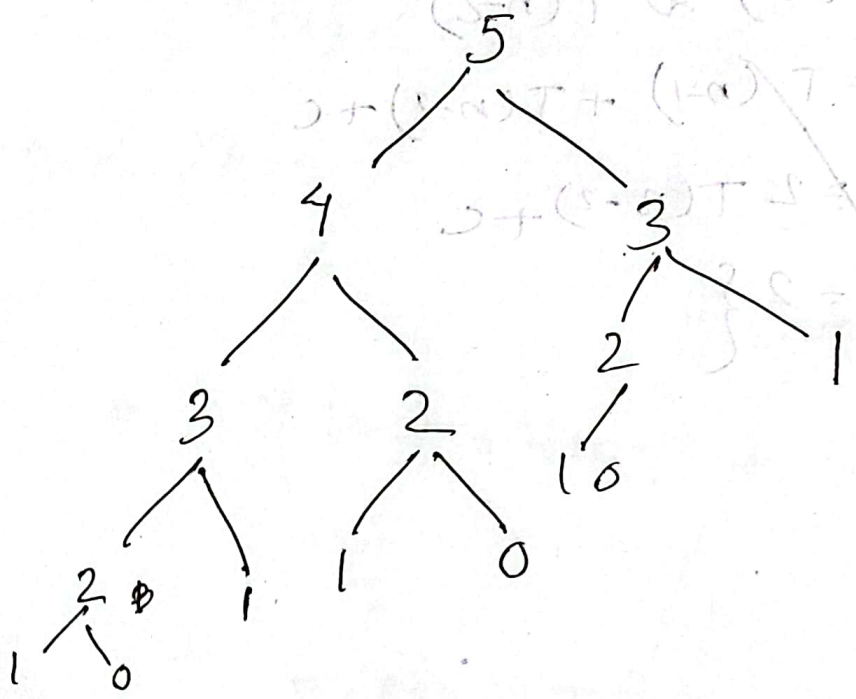
Now, $T(n) = O(n) + O(n)$

$= 2O(n)$

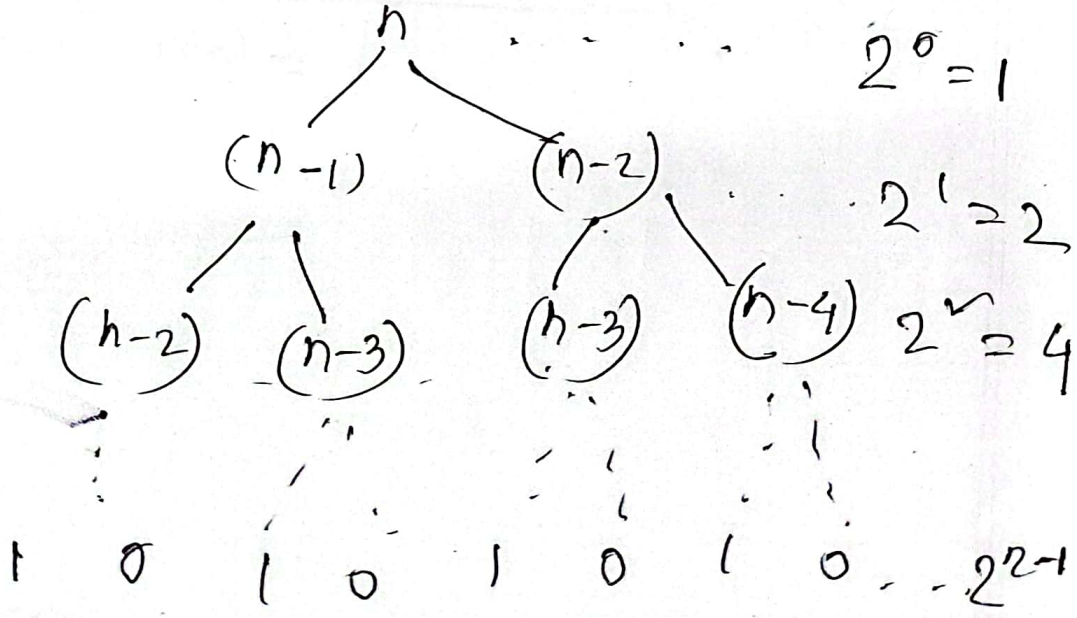
$= O(n)$

Q.C $T(n) = T(n-1) + T(n-2) + C$

Let's take, $n = 5$.



Now if we draw a tree for (n)



∴ total number of function call = $2^0 + 2^1 + 2^2 + \ldots + 2^{n-1}$

there, sum = $\frac{2^0(2^n - 1)}{2-1}$ [geometrical progression]

$= 2^n - 1$

$= 2^n$

$\therefore T(n-1) + T(n-2) + c = O(2^n)$

(Ans:)

Q.2

(d) $T(n) = 8\,T(n/2) + bn^2$

Applying masters theorem:

$$T(n) = 8\,T(n/2) + cn^k$$

Comparing the constants:

$$a = 8$$
$$b = 2$$
$$c = 1$$
$$k = 2$$

Here, $b^k = 8^2 = 64$ $\quad 2^2 = 4$

$\therefore b^k < a$

$\therefore T(n) = O(n^{\log_b a})$

$= O(n^{\log_2 8})$

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

$\log_2 8$

$\log_2 2^3$

$= 3$