Ans. to the Q. No-1

$$C(n, r) = C(n-1, r-1) + C(n-1, r)$$

$$\Rightarrow {}^{n}C_{r} = {}^{n-1}C_{r-1} + {}^{n-1}C_{r}$$

we know that,

$${}^{n}C_{r} = \frac{n!}{r!(n-r)!}$$

$$n! = n(n-1)!$$

$$\therefore {}^{n-1}C_{r-1} = \frac{(n-1)!}{(r-1)!\,[(n-1)-(r-1)]!}$$

$$= \frac{(n-1)!}{(r-1)!\,(n-r)!}$$

$${}^{n-1}C_{r} = \frac{(n-1)!}{r!\,\{(n-1)-r\}!}$$

$$\therefore R.H.S = {}^{n-1}C_{r-1} + {}^{n-1}C_{r}$$

$$= \frac{(n-1)!}{(r-1)!(n-r)!} + \frac{(n-1)!}{r!(n-r-1)}$$

$$= \frac{r(n-1)!}{r(r-1)!(n-r)!} + \frac{(n-1)!\,(n-r)}{r!\,(n-r)(n-r-1)!}$$

$$= \frac{r(n-1)!}{r!\,(n-r)!} + \frac{(n-1)!\,(n-r)}{r!\,(n-r)!}$$

$$\left[\begin{array}{l} r(r-1)! = r! \\ (n-r)(n-r-1) = (n-r)! \end{array}\right]$$

$$= \frac{r(n-1)! + (n-1)!\,(n-r)}{r!\,(n-r)!}$$

$$= \frac{(n-1)!\,\{r+n-r\}}{r!\,(n-r)!}$$

$$= \frac{n(n-1)!}{r!\,(n-r)!}$$

$$= \frac{n!}{r!\,(n-r)!}$$

$$= {}^{n}C_{r}$$

$= L.H.S$

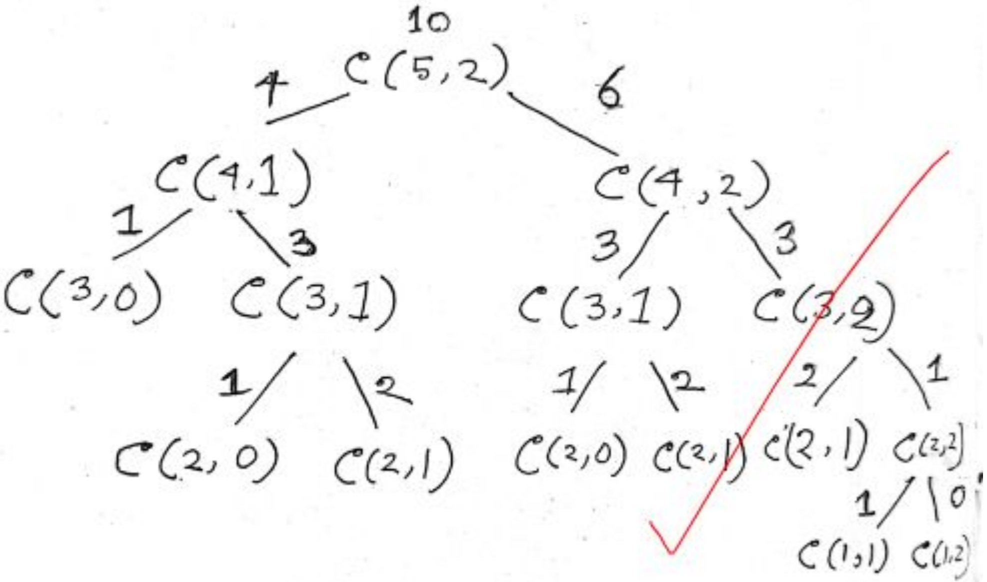
[Proved]

Ans.to.the.Q. No-2

```
def combination(n,r):
        if n < r:
            return 0
        elif r==0 or n==1:
            return 1
        elif r==1:
            return n
        else:
          return combination(n-1, r-1) +
                                  combination(n-1, r)
```

Ans. to. the. Q. No - 3

c = combination



Ans.to.the.Q.No-4

```
def ternary_search(arr, l, r, key):
    while l <= r:
        mid1 = l + (r-l) //3
        mid2 = r - (r - l)//3

        if arr[mid1]==key:
            return mid1
        elif arr[mid2]== key:
            return mid2
        elif key < arr[mid1]:
            r = mid1-1
        elif key > rr[mid2]:
            l = mid2+1
        else:
            l = mid1 +1
            r = mid2 - 1
```

Ans.to.the.Q.No-5

Step	Element to Search
0	$n = \frac{n}{3^0}$
1	$\frac{n}{3} = \frac{n}{3^1}$
2	$\frac{n}{9} = \frac{n}{3^2}$
3	$\frac{n}{27} = \frac{n}{3^3}$
⋮	⋮
K	$\frac{n}{3^k}$

$$\frac{n}{3^k} = 1$$

$$\Rightarrow n = 3^k$$

$$\Rightarrow \log_3 n = \log_3 3^k$$

$$\Rightarrow \log_3 n = k \log_3 3$$

$$\Rightarrow k = \log_3 n$$

$$\therefore O(\log_3 n)$$

✓

Ans. to. the. Q. No-6

For first loop,

step	i		
0	1	$=$	$\frac{1}{7^0}$
1	$\frac{1}{7}$	$=$	$\frac{1}{7^1}$
2	$\frac{1}{49}$	$=$	$\frac{1}{7^2}$
3	$\frac{1}{343}$	$=$	$\frac{1}{7^3}$
$\vdots$			$\vdots$
k			$\frac{1}{7^k}$
			$= 7^{-k}$

$$7^{-k} = n$$

$$\Rightarrow \log_7 7^{-k} = \log_7 n$$

$$\Rightarrow -k = \log_7 n$$

$$\Rightarrow k = -\log_7 n$$

$$\Rightarrow O(-\log_7 n) \approx O(\log_7 n)$$ ✓

For second loop,

$$k = \frac{n}{3} \Rightarrow O(\tfrac{n}{3}) \approx O(n)$$

For third loop,

$$k = \frac{40}{1} \Rightarrow O(40) \approx O(1)$$

For fourth loop,

$$k = -\frac{n}{5} \Rightarrow O(-\tfrac{n}{5}) \approx O(n)$$

$\therefore$ Total Time complexity $= O(\log_7 n) \times O(n) \times \left\{ \begin{array}{l} O(1) \\ + O(n) \end{array} \right.$

$$= O(\log_7 n) \times O(n) \times O(n)$$

$$= O(n^2 \log_7 n)$$

Saba
2210245

Ans. to the Q. No. 7

Step	i	j
0	1	a
1	2	1
2	3	1
3	4	1
4	5	2
5	6	2
6	7	2
7	8	2
8	9	2
9	10	3
⋮	⋮	⋮
13	16	3
⋮	⋮	⋮
k	k+1	$\sqrt{k}$

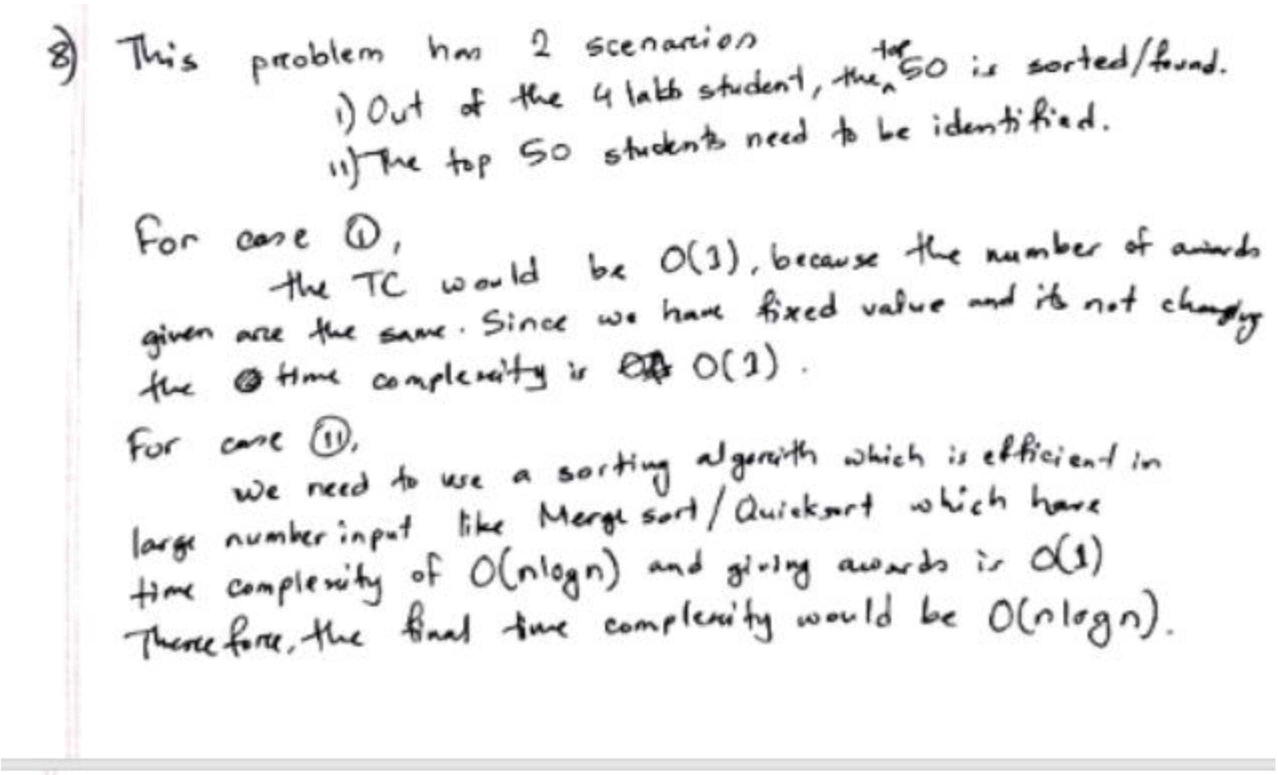
For inner loop, it requires $\sqrt{i}$ times iteration

∴ Time complexity for inner while loop = $O(\sqrt{n})$

For outter loop → ~~O(?)~~ $O(n)$

∴ Total time complexity → ~~(e~~ $O(\sqrt{n}) * O(n)$

$= O(n\sqrt{n})$



8) This problem has 2 scenarios

i) Out of the 4 lakh student, the top 50 is sorted/found.

ii) The top 50 students need to be identified.

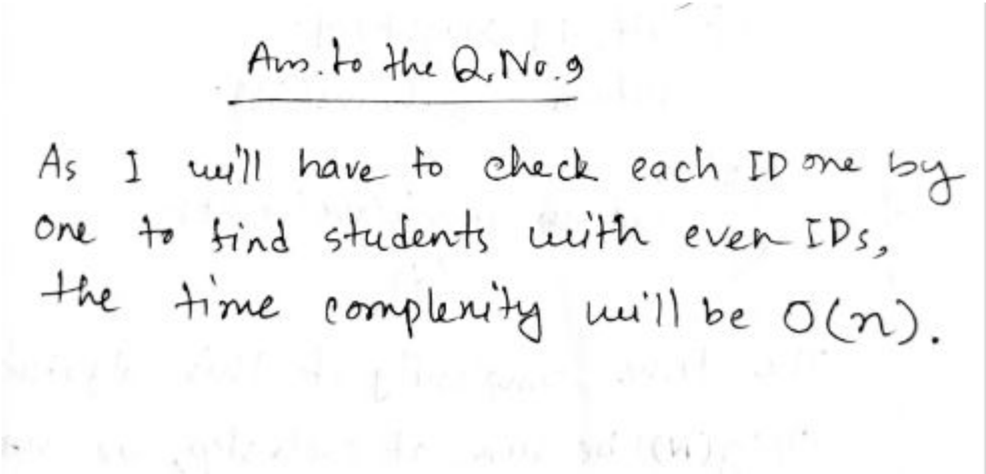
For case (i),

the TC would be O(1), because the number of awards given are the same. Since we have fixed value and its not changing the time complexity is O(1).

For case (ii),

we need to use a sorting algorith which is efficient in large number input like Merge sort/Quicksort which have time complexity of O(nlogn) and giving awards is O(1)

Therefore, the final time complexity would be O(nlogn).



Ans. to the Q. No. 9

As I will have to check each ID one by one to find students with even IDs, the time complexity will be $O(n)$.

10. (a) Pseudocode:

```
def findmax(arr, left, right):
        m = (left +right)//2

        if arr[m] < arr[m+1]:
            return findmax(arr, m+1, right)
        elif arr[m] < arr[m-1]:
            return findmax(arr, left, m-1)
        else:
            return arr[m]
```

Here, we find the mid value of the array. We see if the mid value is more or less than its neighbors. If it is ~~more~~ les than the right value, then it recursively searches the right half. Otherwise, it recursively searches the left half. If it is larger than its neighbours, it returns the element as the max number.

CamScanner

(b) Time complexity will be same as binary search, which is = $O(\log n)$