Name: Angkon Dutta Joy

ID : 22101024

Section: 03

Answer to the ques. no - 1

(a)

For first loop $\rightarrow O(\log_6 n/2)$

For 2nd loop $\rightarrow O(\log_4 n)$

For 3rd loop $\rightarrow O(\log_3 n)$

$\therefore$ Time complexity $= O(\log_6(n/2) \cdot \log_4(n) \cdot \log_3(n))$

$= O(\log^3 n)$

(answer)

(b)

For 1st loop $\rightarrow O(n-1)$

For 2nd loop $\rightarrow O(n-1)$

$\therefore$ Time complexity $= O(n-1)(n-1)$

$= O(n^2 - 2n + 1)$

$= O(n^2)$

(answer)

(c)

The time complexity of A is $O(\log^3 n)$ which smaller than $O(n^2)$.

n^2 is bigger than the upper bound of A. So the statement of 'At least' $O(n^2)$ is meaningless.

(answer)

Answer to the question no - 2

(a)

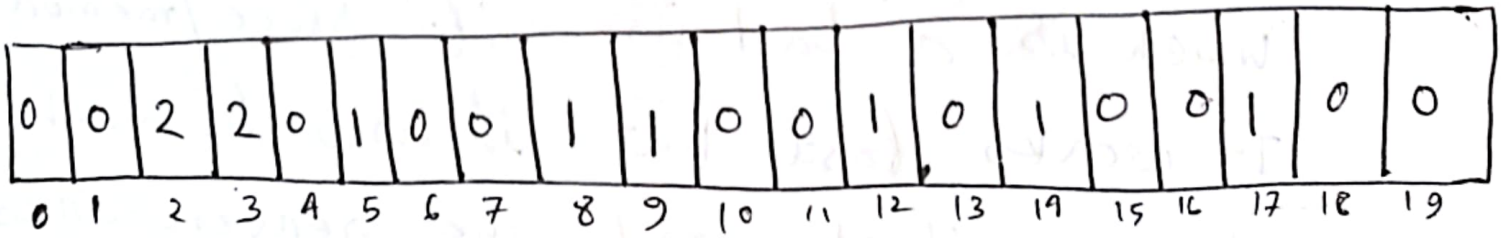
The algorithm is linear search. The pseudo code for linear search is -

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1
```

(b)

The algorithm is count sort. The steps are -

1) I will create a count array from the main array.



2) Now we have to create a position array.

0	0	2	4	4	5	5	5	6	7	7	7	8	8	9	9	9	10	10	10
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

3) Now we will create output array.

	0	1	2	3	4	5	6	7	8	9
	2	2	3	3	5	8	9	12	14	17
Id →	4	8	2	9	3	1	6	0	7	5

(answer)

(c)

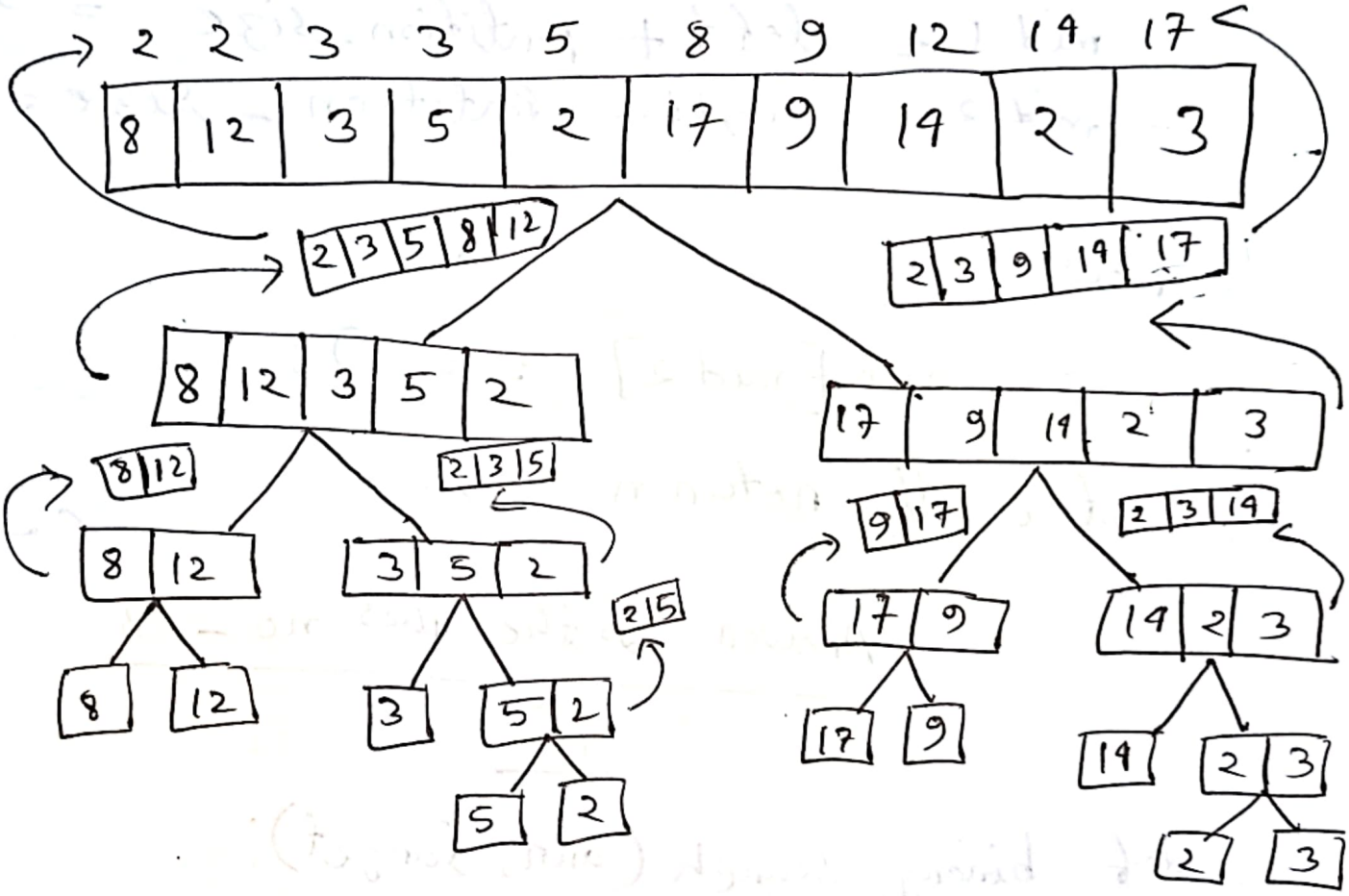
Yes, the crash was the result of count sort algorithm. Because suppose a new user comes and he uploads 1000 picture. for 11 People we need a array size of 1001. Which is a bad use of space/memory. It works fast but it uses so much extra space, which made the server crash.

Subject : .....

Date : .....

(d)

for a algorithm of $(n \log n)$ time complexity, I will use merge sort.

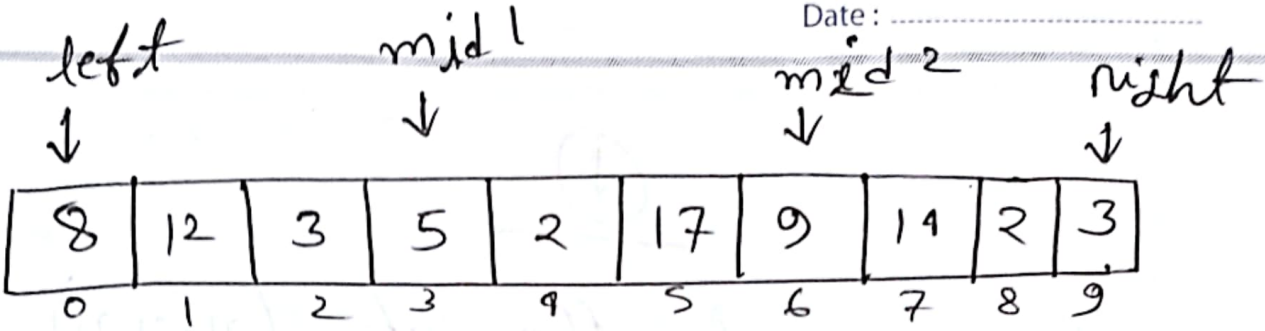


(e)

For efficiently searching 9, I will use ternary search algorithm.

Subject : .....

Date : .....



Partition_size = (right - left) // 3 = 3

mid 1 = left + Partition_size = 3

mid 2 = right - Partition_size = 6

now,

as. arr[mid 2] == 9 :

it will return 9.

Answer of the ques. no - 3

(a)

```
def binary_search (arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
```

Subject : .....

Date : .....

```
elif arr[mid] < target:
        low = mid + 1
else:
        high = mid - 1

return -1
```

(c)

(b)

To find the first occurrence we just have to modify arr[mid] == target block.

```
def binary_search_first_occurance(arr, target):
    l = 0
    r = len(arr) - 1
    result = -1
    while l <= r:
        mid = l + (r - l) // 2
        if arr[mid] == target:
            result = mid
            r = mid - 1
```

Subject : ----------

Date : ----------

```
elif arr[mid] > target:
        r = mid - 1
else:
        l = mid + 1

return result.
```

(—)

Subject : .....

Date : .....

(c)

```
def BFS(arr, l, r, x):
    while l <= r:
        mid = l + (r - l) // 2
        if arr[mid] == x:
            return x
        elif arr[mid] < x:
            l = mid + 1
        else:
            r = mid - 1
    return -1

def count(arr, n, x):
    idx = BFS(arr, 0, n-1, x)
    if idx == -1:
        return -1
    first_oc = idx
    while first_oc > 0 and arr[first_oc - 1] == x
        first_oc -= 1
    count, left, right = 1, idx - 1, idx + 1
```

Subject : .....

Date : .....

```
while left >= 0 and arr[left] == x:
        count += 1
        left -= 1
while right < n and arr[right] == x:
        count += 1
        right += 1
return first_oc, count
```

(o—)

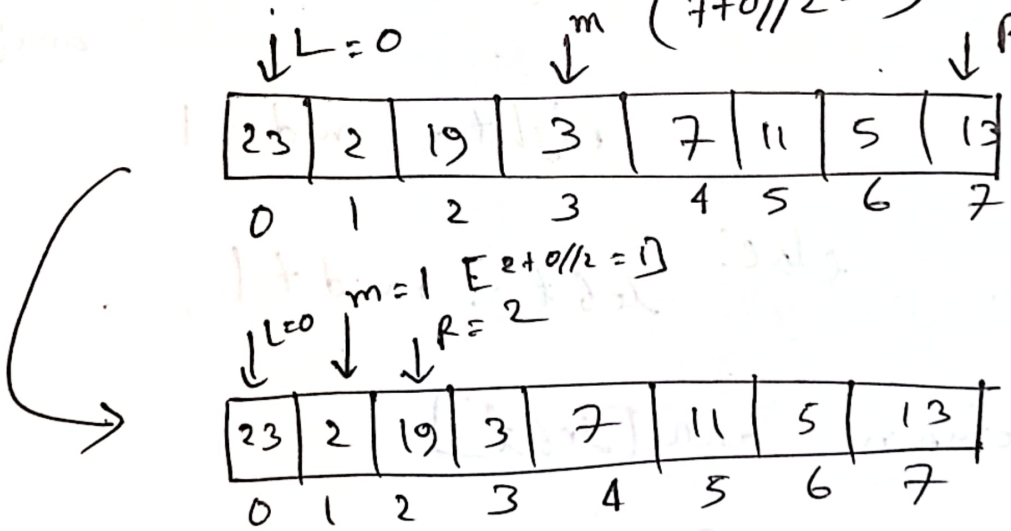
(d)

```
def minimum_wave(arr):
    l = 0
    r = len(arr) - 1
    while left < right:
        mid = l + (r-l)//2
        if arr[mid] < arr[mid-1] and arr[mid] <
                                    arr[mid+1]:
            return arr[mid]
        elif arr_mid >= arr[l]:
            l = mid + 1
        else:
            r = mid - 1
return arr[left]
```

(7)

Answer to the ques. no - 4

The code will find T = 2, but it is an exception. Because binary search works on sorted list. so, it will be an exception. But yes, the code will find T = 2.



as arr[m] == 2 ; it is found.

(answer)

Subject : .....

Date : .....

Answer to the ques no. - 5

(i)

```
def max_wave (arr):
    left = 0
    right = len(arr) - 1
    while left < right:
        mid = left + (right - left)//2
        if mid > 0 and arr[mid] > arr[mid-1]
                and arr[mid] > arr[mid+1]:
            return arr[mid]
        elif arr[mid] > 0 and arr[mid] < arr[mid-1]:
            right = mid - 1
        else:
            left = mid + 1
    return arr[left]
```

(ii)

Time complexity of this code is $O(\log n)$

Subject : .....

Date : .....

Answer to the Ques. no - 6

(a)

If we are performing search operation only one time, then linear search is better than sorting and performing binary search. But if we have to perform search operation multiple time, sorting the list and using binary search will be away more efficient than multiple linear search.

(b)

To handle negative number, we have to change the normal count sort algo. we just have to ~~add~~ substract every element with the smallest number in the list. Then all negative number will become positive and we can perform count sort. But we

Subject : .....

Date : .....

have sto add sthe number at last to find sthe final output.

```
def count_sort_modified (arr):

    min_val = min(arr)
    max_val = max(arr)
    range = int(max_val - min_val) + 1

    count = [0] * range

    for i in arr:
        count[int(i - min_val)] += 1

    sorted_arr = [ ]
    for i in range(range):
        while count[i] > 0:
            sorted_arr.append(i + min_val)
            count[i] -= 1
    return sorted_arr.
```

c →)

Subject : .....

Date : .....

(d)

The time complexity of merge sort = $O(n \log n)$

" " " " quick sort = $O(n+k)$

For very big dataset merge sort will be a better option as it will be more efficient. But as the constraint is space, quick sort will be better in this case. Because space complexity of quick sort is better than merge sort.

(e)

[100, 90, 80, 70, 60, 50, 40, 30, 20, 10]

In this array quick sort will fail to work in $O(n \log n)$ and it will take $O(n^2)$ to perform.