Name - Varun Gupta

Roll No - 2023101108

Problem Set - 3

Ans1)

Using Binary Search to Solve this Problem.

Pseudocode:

```
n = arr.size()
low = 0, high = n-1
minimum = -∞
while low ≤ high:
    mid = (low + high) / 2
    if arr[mid] ≥ arr[low] then:
        if arr[low] ≤ arr[high] then:
            return min(minimum, arr[low])
        else:
            minimum = min(minimum, arr[mid])
            low = mid + 1
    else:
        if arr[low] ≤ arr[high] then:
            return min(arr[low], arr[mid])
        else:
            minimum = min(minimum, arr[mid])
```

```
        high = mid - 1
    return arr[high]
```

Time Complexity :- The algorithm divides the search space in half during each iteration.

∴ TC = O(log n)

Ans 2)

x [0, 1, - - -, m-1] and y [0, 1, - - - n-1]

Also n > m.

Pseudocode :-

```
Mult (x, m, y, n):
    Product = 0
    for i from 0 in range (0, [n/m]):
        partial = y [i*m, ----, min((i+1)*m-1, n-1)]
        Partial_product = Karatsuba (x, partial)
        product t = partial_product * 10^(i*m)
    return product
```

Algorithm & Logic :-

We will divide y(n) into sections of length almost equal to m. Then Multiply this section & xc using Karatsuba Algorithm. Then to get the required result we can just shift there position (powers of 10) and then add.

$$\text{Result} = \sum_{i=0}^{[n/m]-1} \text{Product}_i * 10^{i*m}$$

Time Complexity :-

For Calculating Partial Product, $TC = O(m^{\log_2 3})$

This is calculated $[n/m]$ times.

$$\therefore T(m, n] = [n/m] \cdot O(m^{\log_2 3})$$

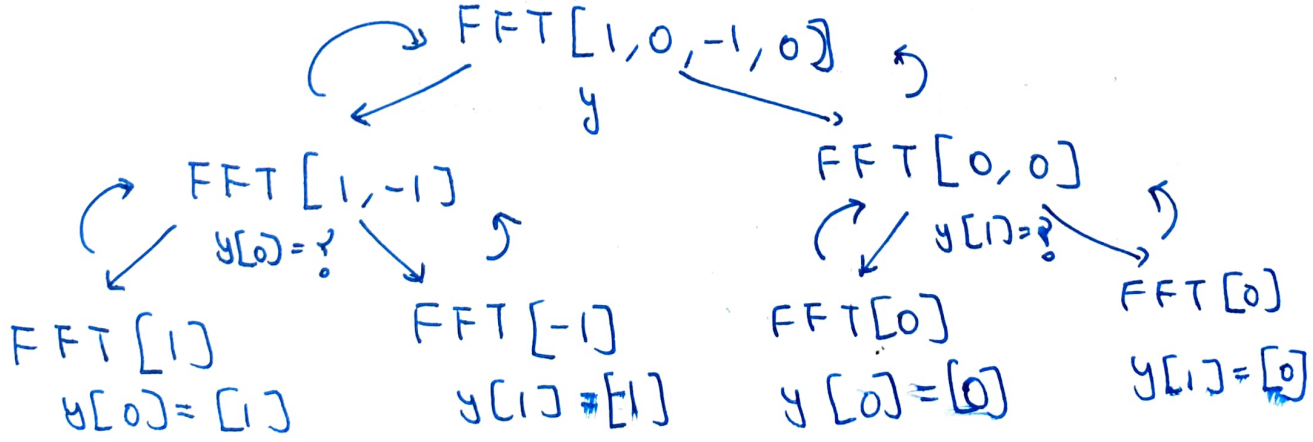
$$= O(m^{\log_2 3 - 1} \cdot n)$$

Ans 3)

(a) FFT of $[1, 0, -1, 0]$

Recursive FFT(a): [Code Ref: CLRS pg 911]

```
n = a.size()
if n = 1:
    return a
ω_n = e^(2πi/n)
ω = 1
a^[0] = (a_0, a_2, ---, a_{n-2})
a^[1] = (a_1, a_3, ---, a_{n-1})
y^[0] = FFT(a^[0])
y^[1] = FFT(a^[1])
for k in range(0, n/2):
    y_k = y_k^[0] + ω y_k^[1]
    y_{k+n/2} = y_k^[0] - ω y_k^[1]
    ω = ω · ω_n
return y
```



FFT [1, -1]:

$n = 2$

$\omega = 1$

$\omega_2 = e^{\frac{2\pi i}{2}} = -1$

for k = 0 to 0 (1 time)

$y_0 = y_0^{[0]} + \omega\, y_0^{[1]}$

$y_0 = [1] + 1\,[-1] = [0]$

$y_{0+2/2} = y_0^{[0]} - \omega\, y_{00}^{[1]}$

$= [1] - 1\,[-1] = [2]$

$y[0] = [0\ 2]$

FFT [0, 0]:

$n = 2$, $\omega = 1$

$\omega_2 = e^{2\pi i/2} = -1$

for k = 0 to 0 ~~etc~~:

$y_0 = y_0^{[0]} + \omega\, y_0^{[1]}$

$= [0] + 1\,[0] = [0]$

$y_1 = y_0^{[0]} - \omega\, y_0^{[1]} = [0]$

$y^{[1]} = [0\ 0]$

FFT [1, 0, -1, 0]:

$n = 4$

$\omega_4 = i$

for k = 0 to 1:

K=0 : →

$$y_0 = y_0^{[0]} + \omega\, y_0^{[1]}$$
$$= [0] + 1\,[0] = [0]$$
$$y_2 = y_0^{[0]} - \omega\, y_0^{[1]} = [0]$$

K=1 : →

$$y_1 = y_1^{[0]} + \omega\, i\, y_1^{[1]}$$
$$= [2] + i\,[0] = [2]$$
$$y_3 = y_1^{[0]} - i\, y_1^{[1]}$$
$$= [2] - i\,[0] = [2]$$

$$y = [0, 2, 0, 2]$$

FFT = (0, 2, 0, 2)

(b) Of which sequence is [1, 0, -1, 0] the FFT? Using Same Recursive FFT function as above used. To find the sequence from the FFT we will do the inverse of Recursion did in Part A.

$a = [a_0, a_1, a_2, a_3]$

$FFT = [1, 0, -1, 0]$

FFT(a): ⇝ $FFT = [1, 0, -1, 0]$

$n = 4, \omega = 1, \omega_4 = i$

for k = 0 to 1:

k = 0:

$$\left.\begin{aligned} y_0 &= y_0^{[0]} + y_0^{[1]} = 1 \\ y_2 &= y_0^{[0]} - y_0^{[1]} = -1 \end{aligned}\right] \Rightarrow \begin{aligned} y_0^{[0]} &= 0 \\ y_0^{[1]} &= 1 \end{aligned}$$

k = 2

$$\left.\begin{aligned} y_1 &= y_1^{[0]} + i\, y_1^{[1]} = 0 \\ y_3 &= y_1^{[0]} - i\, y_1^{[1]} = 0 \end{aligned}\right] \begin{aligned} y_1^{[0]} &= 0 \\ y_1^{[1]} &= 0 \end{aligned}$$

FFT(a_0, a_2) ⇝ $[0, 0]$

$n = 2, \omega = 1, \omega_2 = -1$

for k = 0 to 0

$$\left.\begin{aligned} y_0 &= y_0^{[0]} + 1\, y_0^{[1]} = 0 \\ y_1 &= y_0^{[0]} - 1\, y_0^{[1]} = 0 \end{aligned}\right] \Rightarrow \begin{aligned} y_0^{[0]} &= 0 \\ y_0^{[1]} &= 0 \end{aligned}$$

$a_0 = 0, a_2 = 0$

FFT(a_1, a_3) ⇝ $FFT = [1, 0]$

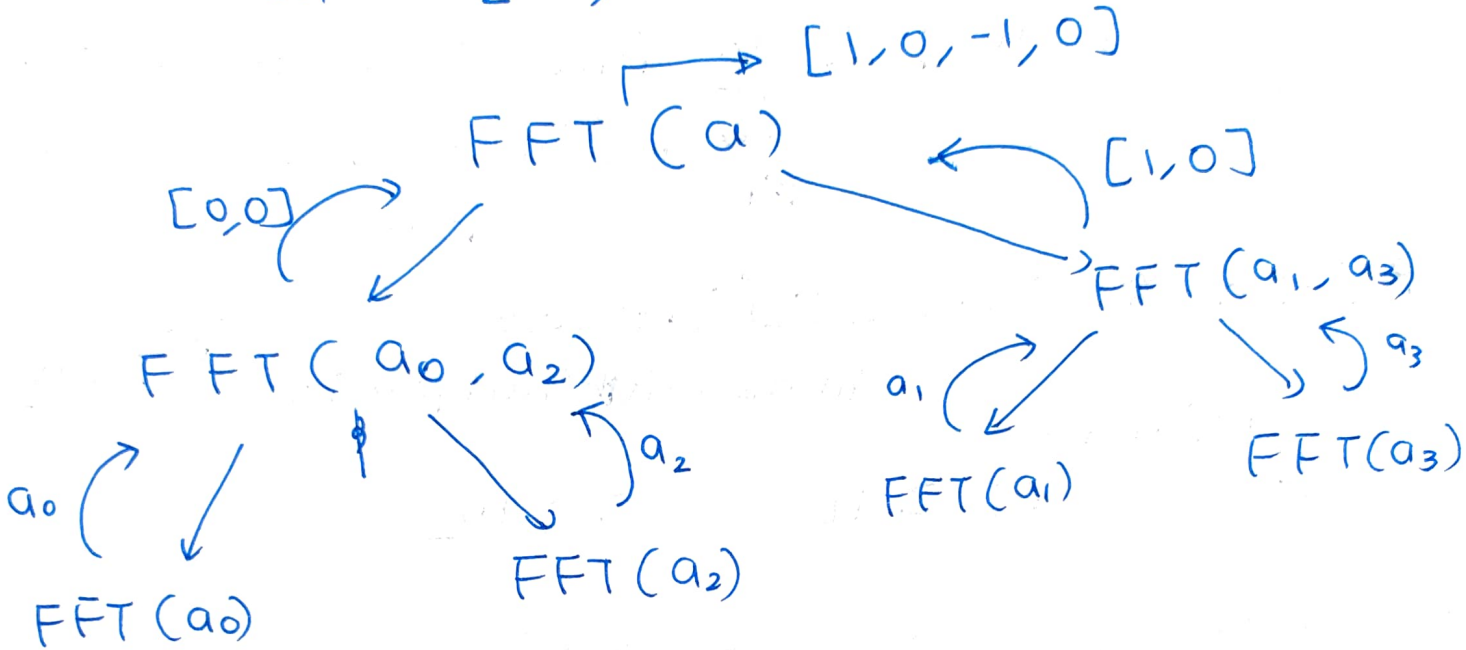
$n = 2, \omega = 1, \omega_2 = -1$

for $k = 0$ to 0

$$y_0 = y_0^{[0]} + 1\, y_0^{[1]} = 1$$
$$y_1 = y_0^{[0]} - 1\, y_0^{[1]} = 0$$

$$y_0^{[0]} = 1/2, \quad y_0^{[1]} = 1/2$$

$a_1 = 1/2$, $a_3 = 1/2$



Sequence = $a = [a_0, a_1, a_2, a_3]$

$= [0, 1/2, 0, 1/2]$

Ans 4)

We have N line segments between Parallel lines. The start point of each line is on $Y = 0$ & other end point is on $Y = 5$. The line can be represented by (x_1, x_2). Two segments (x_1, x_2) & (y_1, y_2) then they intersect if & only if :

$x_1 < y_1$ and $x_2 > y_2$ (or) $x_1 > y_1$ & $x_2 < y_2$

Algorithm :-

We will sort by starting Point. Then using the idea of merge sort we divide the problem into two halves. We will find no of intersecting pairs in each half. After solving for each half, we need to count the intersecting pairs where one line segment comes from first half & another comes from the second half.

Keeping the segments sorted by their ending points in $r = 5$, when merging two halves, count how many segments from the second half end before a segment from the first half. These are the intersecting pairs.

Pseudocode :-

```
for Count_Intersections (segments):
    Sort (segments by x[1])
    return count_and_sort (segments)

Count_and_sort (segments):
    if length (segments) ≤ 1:
        return 0

    mid = length (segment) // 2
    left_seg = segment (start to mid)
    right_seg = segment (mid to end)

    Count_left = count_and_sort (left_seg)
    count_right = count_and_sort (right_seg)

    count_bet = count_cross (left_seg, right_seg)
    return count_left + count_right + count_bet

Count_cross (left_seg, right_seg):
    j = 0, count = 0
```

```
right_ends = B[segment.x2 for segment in right_seg]
sort(right_ends)
for each left_seg in left_segments:
    while j < len(right_ends) and right_end[j] < left_seg.x2:
        j += 1
    count += j
return count
```

Time Complexity: The time complexity is O(N log N) where N is number of line segments.

⊖ For each recursive division takes O(N) for merging & counting & the recursion depth is O(log N).

Ans 5)

```
majority(files):
    Majority_elem, cnt = majority_count(files)
    if count > len(files)/2:
        return "YES"
    else
        return "NO"
```

```
majority_count (files):
    if len (files) = 1:
        return (files[0], 1)    # Base case

    mid = len(files) // 2
    left = files (start 0 to mid)
    right = files (mid to end)

    left_majority, left_cnt = majority_count (left)
    right_majority, right_cnt = majority_count (right)
    if left_majority = right majority:
        return(left_majority, left_cnt + right_cnt)

    total_l = 0, total_r = 0
    for each file in files:
        if comparator (file, left_majority) = true:
            total_l ++ = 1
        if comparator (file, right_majority) = true:
            total_r + = 1
    if total_l > total_r:
        return (left_majority, total_l)
    else
        return (right_majority, total_r)
```

Time Complexity :-

Per Recursion find total_l & total_r

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

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

O(n) for counting occurences of majority candidate and O(logn) would be the depth of Recursion.

Ans 6)

Proof of Correctness :-

Consider there are more than $n/2$ files of same type, so while dividing it in 2 halved it would have been a majority in atleast one half. Hence we will get our output correctly.

Ans 6) Intuition:

We will solve this problem using a divide and conquer approach similar to idea of binary search. We will find two partitions - one from each archive such that the artifacts just before the partion in first archive is less than or equal to artifact just after the partition in the second archive and viceversa.

Let these two archives be A and B and A[i], B[i] be the ith smallest element of the archive A & B.

Now, Comparing medians of Archive A & B,

Suppose, $A(n/2) < B(n/2)$ [Without loss of gene we can assume this]

$\Rightarrow B(n/2) \geq$ First $n/2$ elements of A and First $n/2 - 1$ elements of B

Position of $B(T) \geq 2(n/2)^{th}$ (atleast) in the combined portion $T > n/2$

(Or) Position of $B(T) \geq n$, $T > n/2$

$\therefore B(T)$ can be eliminate from further calc. Also first $n/2$ elements of A can also be eliminated as they are less than last $n-(n/2)+1$ elements of B and also less than last $(n/2)$ element of A so if we have to check in combined array their position will be $[n-(n/2)+1]+(n/2) = n+1^{th}$ from last atleast.

$\Rightarrow$ First $(n/2)$ elements of A can be ignored.

Similar is the case $A(n/2) > B(n/2)$ then last part of A & first part of B is ignored.

Now keep doing it recursively until only one element in each set & minimum of that is our required median.

Pseudocode:-

```
Median(n, A, B, a, b):
    if n = 1:
        return min(A[a+1], B[b+1])
    k = [n/2]
    if (A(a+k) < B(b+k) then:
        return median(k, A, B, a+k, b)
    else:
        return median(k, A, B, a, b+k)
```

Time Complexity:

To find median of whole set of elements we first will call median(n, A, B, 0, 0)

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

$$\theta(n) = 2\log n \Rightarrow Q(n) = O(\log n)$$