ESO207A: Data Structures and Algorithms

Theoretical Assignment 1

Due Date: 2nd September, 2023

Total Number of Pages: 7 Total Points 160

Instructions-

1. For submission typeset the solution to each problem and compile them in a single pdf file. Hand-written solutions will not be accepted. You can use LaTeX or Word for typesetting.
2. Start each problem from a new page. Write down your Name, Roll number and problem number clearly for each problem.
3. For each question, give the pseudo-code of the algorithm with a clear description of the algorithm. Unclear description will receive less marks. Less optimal solutions will receive only partial marks.
4. Assume that sorting would have $O(nlog(n))$ complexity.

Question 1. Ideal profits (40 points)

In the X world, companies have a hierarchical structure to form a large binary tree network (can be assumed to be a perfect binary tree). Thus every company has two sub companies as their children with the root as company X. The total number of companies in the structure is N. The wealth of each company follow the same general trend and doubles after every month. Also after every year, half of the wealth is distributed to the two child companies (i.e. one fourth to each) if they exist (i.e. the leaf node companies do not distribute their wealth). Given the initial wealth of each of the N companies, you want to determine the final wealth of each company after m *months*.
(A perfect binary tree is a special tree such that all leaf nodes are at the maximum depth of the tree, and the tree is completely filled with no gaps. Detailed explanation *here*)

(a) (20 points) Design an algorithm in $O(n^3log(m))$ complexity to find the final wealth of each company after m *months*.

(b) (10 points) Analyze the time complexity of your algorithm and briefly argue about the correctness of your solution.

(c) (10 points) Consider the case of a single company (i.e. only root) in the tree. Give a constant time solution to find the final wealth after m *months*.

Solution-
Wealth of a company doubles each month (at the start of the month) and becomes one half at the end of the year. At the end of a year thus if a company has wealth = w Then at the start of the next year wealth = $w/2$.
Assuming the parent of ith company as p_i (with the parent of root as -1) i.e. in the following binary search tree, the parent array is given as [-1,1,1,2,2,3,3]. It can be seen that parent of ith company is given by [i/2]
Assuming companies have initial wealth $w_1, w_2, ...w_N$

After one month, $w'_1, w'_2, ...w'_N = 2w_1, w_2, ...w_N$ and can be represented in matrix form as-

$$\begin{bmatrix} 2 & 0 & 0.... & 0 \\ 0 & 2 & 0.... & 0 \\ . & . & ..... & . \\ 0 & 0 & 0.... & 2 \end{bmatrix} \begin{bmatrix} w_{1,0} \\ w_{2,0} \\ .. \\ w_{N,0} \end{bmatrix} = \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ .. \\ w_{N,1} \end{bmatrix}$$

After 12 months,

$$\begin{bmatrix} 2 & 0 & 0.... & 0 \\ 0 & 2 & 0.... & 0 \\ . & . & ..... & . \\ 0 & 0 & 0.... & 2 \end{bmatrix}^{12} \begin{bmatrix} w_{1,0} \\ w_{2,0} \\ .. \\ w_{N,0} \end{bmatrix} = \begin{bmatrix} w_{1,12} \\ w_{2,12} \\ .. \\ w_{N,12} \end{bmatrix}$$

At the start of first year
$w'_1 = w_1/2$
$w'_2 = w_2/2 + w_1/4$
$w'_i = w_i/2 + w_{i/2}/4$
$w'_{N-l} = w_{N-l} + w_{N-l/2}/4$
$w'_N = w_N + w_{N/2}/4$
where l = number of leaf nodes = $(N+1)/2$
Thus at the start one year,

$$\begin{bmatrix} 2 & 0 & 0.... & 0 & 0 \\ 0 & 2 & 0.... & 0 & 0 \\ 0 & 0 & 2.... & 0 & 0 \\ . & . & ..... & . & . \\ 0 & 0 & 0.... & 0 & 2 \end{bmatrix}^{12} \begin{bmatrix} 1/2 & 0 & 0.... & 0 & 0 \\ 1/4 & 1/2 & 0.... & 0 & 0 \\ 1/4 & 0 & 1/2.... & 0 & 0 \\ . & . & ...... & . & . \\ 0 & 0 & .... & 1/4 & 1 \end{bmatrix} \begin{bmatrix} w_{1,0} \\ w_{2,0} \\ w_{3,0} \\ .. \\ w_{N,0} \end{bmatrix} = \begin{bmatrix} w_{1,12} \\ w_{2,12} \\ w_{3,12} \\ .. \\ w_{N,12} \end{bmatrix}$$

The multiplication of the matrices is commutative due to identity matrix. The final wealth can be given after m months i.e. $\lfloor m/12 \rfloor year$

$$\begin{bmatrix} 2 & 0 & 0.... & 0 & 0 \\ 0 & 2 & 0.... & 0 & 0 \\ 0 & 0 & 2.... & 0 & 0 \\ . & . & ..... & . & . \\ 0 & 0 & 0.... & 0 & 2 \end{bmatrix}^{m} \begin{bmatrix} 1/2 & 0 & 0.... & 0 & 0 \\ 1/4 & 1/2 & 0.... & 0 & 0 \\ 1/4 & 0 & 1/2.... & 0 & 0 \\ . & . & ...... & . & . \\ 0 & 0 & .... & 1/4 & 1 \end{bmatrix}^{\lfloor m/12 \rfloor} \begin{bmatrix} w_{1,0} \\ w_{2,0} \\ w_{3,0} \\ .. \\ w_{N,0} \end{bmatrix} = \begin{bmatrix} w_{1,m} \\ w_{2,m} \\ w_{3,m} \\ .. \\ w_{N,m} \end{bmatrix}$$

The matrix exponentiation can be carried in $log(m)$ steps and each matrix multiplication in $O(n^3)$ computation. Overall time complexity = $O(n^3 log(m))$

(c) For a single company simply: Initialwealth * (2^m)

Question 2. Moody Friends (40 points)

P friends arrive at a hotel after a long journey and want rooms for a night. This hotel has n rooms linearly arranged in form of an array from left to right where array values depict the capacities of the rooms. As these are very close friends they will only consider consecutive rooms for staying. As you are the manager of the hotel you are required to find cheapest room allocation possible for them (sum of the capacities of selected rooms should be greater than or equal to P). Cost of booking every room is same and is equal to C.

(a) (15 points) Design an algorithm in $O(n)$ time complexity for determining the minimum cost room allocation. The allocated rooms should be consecutive in the array and their capacities should sum to atleast P.

(b) (15 points) Now suppose they don't care about the cost and total capacity anymore. But they came up with a beauty criteria for an allocation. According to them, an allocation is beautiful if GCD (Greatest Common Divisor) of capacities of all rooms in the allocation is at least equal to or greater than a constant K. And they want to take maximum number of contiguous rooms possible. Your task is to design an algorithm in $O(nlog(n))$ time complexity for determining the maximum number of contiguous rooms they can get which satisfy the beauty constraints. You can assume access to a blackbox GCD algorithm which can give you GCD of two numbers in constant $O(1)$ time.

(c) (10 points) Give proof of correctness and time complexity analysis of your approach for parts (a).

Solution-

The problem can be solved in $O(n)$ complexity using sliding window approach. Let the window be denoted by the "start" and the "end" index (including both), i.e. we would consider the sum of consecutive elements a[start], a[start + 1],....., a[end]

Initialize start = 1 and iterate over the array incrementing "sum" until $sum >= P$

Psuedo-code:

```
start = end = 1
sum = capacity[1]
while sum  < P
    end++
    sum += capacity[end]
co = end - start + 1
for (i=end; i <= n; i++)
    sum += capacity[1]
    while (sum - capacity[start] >= 1)
        start ++
    co = min(co, end - start + 1)
```

(b) To find the maximum number of contiguous rooms possible

There are two points that are very important observations to solve this problem

1. GCD(a,b,c) = GCD(a, GCD(b,c)) = GCD(GCD(a, b),c)
2. for any k positive integers a_1, a_2 .... a_k GCD(a_1, a_2 .... a_k) $\leq$ GCD(a_1, a_2 .... a_{k-1})

We can use the first property to make a range query datastructure like Sparse table, segment tree or block decomposition. Segment tree is an overkill because we dont need to update any values in this case.

And due to second property we can apply binary search on the length of the subarrays.

We have given a bried psuedocode for Binary search using sparse table, but you can use binary search with any other RQ datastructure as well.

```
main()
for start=1; start <=n; start++
    low = start; high = n
    while(true)
        mid = start + (high - low)/2
        if Range-GCD(start, mid) >=k
            co = max(co, mid-start+1)
            low = mid +1
        else
            high = mid -1

Range-GCD(i,j)
    L = j-i
    t = Power-of-2(L)
    k = log(L)
    if(t == L) return B[i][k]
    else return gcd(B[i][k], B[j-k][k])

B = n x log(n) matrix where B[i,k] stores gcd of (A[i], A[i+1].... A[i+ 2^k])
```

Question 3. **BST universe** (30 points)

You live in a BST world where people are crazy about collecting BSTs and trading them for high values. You also love Binary Search Trees and possess a BST. The number of nodes in your BST is n.

(a) (10 points) The Rival group broke into your lab to steal your BST but you were able to stop them. But still they managed to swap exactly two of the vertices in your BST. Design an $O(n)$ algorithm to find which nodes are swapped and the list of their common ancestors.

(b) (20 points) Seeing you were able to easily revert the damage to your tree, they attacked again and this time managed to rearrange exactly k of your nodes in such a way that none of the k nodes remain at the same position after the rearrangement. Also all the values inside this BST are positive integers and upper bounded by a *constant* G. Your task is to determine the value of k and which nodes were rearranged. Design an algorithm of complexity $O(min(\ G + n,\ nlog(n)\ ))$ for the same. (Hint : Consider two cases for $G < nlog(n)$ and $G > nlog(n)$)

Solution:
For a BST, the inorder traversal is in sorted order
(a) Compute the inorder traversal in O(n) time and store it in an array $A[1], A[2]....A[n]$

Psuedo-Code:

```
A = new int[n]
co = 1
Inorder-Traversal(node)
    if(node->left)
        Inorder-Traversal(node->left)
    A[co] = node->value
    co++
    if(node->right)
        Inorder-Traversal(node->right)
```

For two nodes swapped in the array, let (i,j) i¡j $A[i] < A[j]$ the resultant array would be like
$A[1], A[2], ...A[i-1], A[j], A[i+1], ..., A[j-1], A[i], A[j+1]$
Thus $A[j] > A[i-1]$ and $A[j] < A[i+1]$. Thus there is a peak in the array. Similarly $A[i] < A[j-1]$ and $A[i] < A[j-1]$i.e. a valley in the array. The two indices can be found by a simple linear search.

Psuedo-Code:

```
for i=1;i<=n;i++
    if( (i<=1 || A[i]>A[i-1]) && (i >= N || A[i]>A[i+1])
        node1 = i
    if( (i<=1 || A[i]<A[i-1]) && (i >= N || A[i]<A[i+1])
        node2 = i
```

(b) After k nodes were swapped, inorder traversal is not sorted
If $nlogn < G$

Sort the array in $nlog(n)$ time complexity.
Compare the original array with the sorted to get the elements which are not at correct postions

If $nlogn > G$
Pseudo-Code:

```
B = new int[G]
for i=1;i<=N;i++
    B[A[i]] = 1
for i=1;i<=G;i++
    prefixcount[i] = prefixcount[i-1] + B[i]
for i=1;i<=N;i++
    if(prefixcount[A[i]] !=i)
        co++
        print(A[i])
print("Value of k=" + co)
```

Please note the following feedback in the graded answer sheets:

1. p11 - swaps are not correct, either consecutive element problem is there on output of inorder traversal or some other
2. p12 - Ancestor part not correct
3. p13 - whole list of ancestor not given
4. p14 - G<nlogn case not correct

Question 4. Helping Joker (20 points)

Joker was challenged by his master to solve a puzzle. His master showed him a deck of n cards. Each card has value written on it. Master announced that all the cards are indexed from 1 to n from top to bottom such that $(a_1 < a_2 < ... < a_{n-1} < a_n)$ .Then his master performed an operation on this deck invisible to Joker (Joker was not able to see what he did), he picked a random number k between 0 and n and shifted the top k cards to the bottom of the deck. So after the operation arrangement of cards from top to bottom looks like $(a_{k+1}, a_{k+2} \ldots a_n, a_1, a_2 \ldots a_k)$ where $(k+1, k+2 \ldots n, 1, 2 \ldots k)$ are original indices in the sorted deck. Joker's task is to determine the value of k. Joker can make a query to his master. In a query, joker can ask to look at the value of any card in the deck. Joker asked you for help because he knew you were taking an algorithms course this semester.

(a) (15 points) Design an algorithm of complexity $O(log(n))$ for Joker to find the value of k.

(b) (5 points) Provide time complexity analysis for your strategy.

Solution:
Final arrangement of cards after the operation is given as $(a_{k+1}, a_{k+2} \ . \ . \ . \ a_n, a_1, a_2 \ . \ . \ . \ a_k)$-
In this question we can use first or the last card of the shuffled deck as a comparison to find the pivot point. We can note that all the cards on the left of the pivot point are greater that the card at the end. And all the cards on the right side are less than or equal to the card at the end. Hence we use the card at the end to find the pivot point.
Pseudo-Code:

```
left=0; right = n
while left < right
    mid = left + (right - left)/2
    if( nums[mid] > nums[n-1])
        left = mid + 1
    else
        right = mid -1
k = n-left + 1
```

Question 5. One Piece Treasure (30 points)

Strawhat Luffy and his crew got lost while searching for One piece (worlds largest known treasure). It turns out he is trapped by his rival Blackbeard. In order to get out he just needs to solve a simple problem. You being the smartest on his crew are summoned to help. On the gate out, you get to know about a hidden string of lowercase english alphabets of length n. Also, an oracle is provided which accepts an input query of format (i, j), and returns true if the $substring(i, j)$ of the hidden string is a palindrome and false otherwise in $O(1)$ time. But there is a catch, the place will collapse killing all the crew members, if you ask any more than $n * log^2(n)$ queries to the oracle. The string is hidden and you can't access it. (Assume the string is very big i.e. n is a large number)

(a) (30 points) You need to design a strategy to find number of palindromic substrings in the hidden string so your crew can safely escape from this region. Please state your algorithm clearly with pseudocode.
(A contiguous portion of the string is called a substring)

Solution:
Iterate over the array and consider each index i(for odd length) or [i,i+1] (for even length strings) as the center of a substring i.e. for an index i consider the substrings of odd lengths a[i], a[i-1, i+1], a[i-2, i+2]... a[i-k,ai+k] and substrings of even length as a[i,i+1], a[i-1, i+2]...a[i-k, i+k+1]. IF a substring a[i-k,i+k] is a palindrome then all substrings centered at i and for all 1<= k' <= k are palindrome. Similar can be said for even length substrings. We can binary search for the value k for each index position i.

Psuedo-Code:

```
// For odd length substrings
for(center = 1; center <= n; center++)
    lim = min(center-1, n-center)
    low = 0; high=lim; k=low
    while low<=high
        mid = low + (high-low)/2
        if(palin(center-mid, center + mid))
            k = mid
```

```
                low = mid + 1
            else
                high = mid - 1
        co += k

    // For even length substrings
    for(center = 1; center < n; center++)
        lim = min(center-1, n-center-1)
        low = 0; high=lim; k=low
        while low<=high
            mid = low + (high-low)/2
            if(palin(center-mid, center + mid + 1))
                k = mid
                low = mid + 1
            else
                high = mid - 1
        co += k
```

Total queries $= 2nlog(n)$