Name: Rollno:

ESO207: Data Structures and Algorithms (Midsem Exam)

20th September 2023

Total Number of Pages: 11 Time: 2 hr Total Points 80

Question	Points	Score
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
7	10	
8	10	
Total:	80	

Instructions

1. All questions are compulsory.
2. Answer all the questions in the space provided in the question paper booklet.
3. Use the space provided in the paper for rough work.
4. The symbols or notations mean as usual unless stated.
5. You may cite and use algorithms and their complexity as done in the class.
6. Cheating or resorting to any unfair means will be severely penalized.
7. Superfluous and irrelevant writing will result in negative marking.
8. Using pens (blue/black ink) and not pencils. Do not use red pens. for answering.
9. Please bring your ID cards.

Helpful hints

1. It is advisable to solve a problem first before writing down the solution.
2. The questions are *not* arranged according to the increasing order of difficulty.

Question 1. Fill in the blank type questions.

(a) (3 points) Consider a red-black tree T having black height h. Then the minimum height of T is $\underline{\ h\ \textbf{(or}\ h-1)\ }$ and the maximum height of T is $\underline{2h-1\ \textbf{(or}\ 2h-2)}$. Also the number of nodes in T is at least $\underline{\quad 2^h - 1 \quad}$.

(b) (1 point) The size of a full binary tree having n internal nodes, is $\underline{\quad 2n+1 \quad}$.

(c) (2 points) The amount of extra space used in MergeSort is $O(\underline{\quad n \quad})$, and in QuickSort is $O(\underline{\quad \log n \quad})$.

(d) (2 points) Consider the recurrence relation $T(n) = T(\sqrt{n}) + 1$ for $n > 1$ and $T(1) = 1$. Then $T(n) = O(\underline{\quad \log\log n \quad})$.

(e) (2 points) Recall the problem of computing shortest distance in a $n \times n$ grid. The maximum number of times the **while** loop iterates is $\underline{\quad n^2 \quad}$ (give an exact upper bound, do not use O notation).

Question 2. Multiple choice type questions (**single** correct choice).

(a) (2 points) What is the complexity of **SpecialUnion** operation in a red-black tree storing n values?

A. $O(1)$
B. $O(\log n)$
C. $O(n)$
D. $O(n \log n)$

(b) (2 points) Consider the following two statements

I. If a node in a BST does not have a left child, then it does not have a predecessor in the BST.
II. Inorder traversal of a BST gives a sorted array.

A. Both statements I and II are correct
B. I is correct and II is incorrect
C. I is incorrect and II is correct
D. Both statements I and II are incorrect

(c) (2 points) Let $T(n) = 8T(n/3) + n^2$ for $n > 1$ and $T(1) = 1$. Then the best bound on $T(n)$ is

A. $O(n \log n)$
B. $O(n^2)$
C. $O(n \log^2 n)$
D. $O(n^2 \log n)$

(d) (2 points) How many bit operations are required to multiply two n bit numbers? Give the best possible upper bound.

A. $O(n \log_{3/2} n)$
B. $O(n^{3/2})$
C. $O(n^{\log_3 2})$
D. $O(n^{\log_2 3})$

(e) (2 points) Consider the stack based algorithm for evaluating an arithmetic expression involving parenthesis.

A. The InsideStackPriority of '(' is minimum and the OutsideStackPriority of '(' is maximum.
B. The InsideStackPriority of '(' is maximum and the OutsideStackPriority of '(' is minimum.
C. The InsideStackPriority and OutsideStackPriority of '(' are minimum.
D. The InsideStackPriority and OutsideStackPriority of '(' are maximum.

Question 3. Multiple choice type questions (**one or more** correct choices).

(a) (2 points) Which of the following properties does a red-black tree always have?

A. It is a full binary tree

B. It is always nearly balanced

C. Height of the tree is $O(\log n)$

D. Deleting a node may require $O(n)$ time.

Solution: Option D. can also be viewed as correct. Since $\log n$ is technically $O(n)$.

(b) (2 points) Which of the following statements are true?

A. $3^n = O(2^n)$

B. $2^n = O(3^n)$

C. $\log(n!) = O(n \log n)$

D. $n^{\log n} = O(2^{\log^2 n})$

(c) (2 points) In which of the following arrays, will Quick Sort perform the exact same number of comparisons as the array [3, 1, 5, 8, 4].

A. [10, 15, 20, 25, 30]

B. [15, 10, 25, 30, 20]

C. [15, 11, 6, 14, 9]

D. [10, 2, 20, 50, 15]

(d) (2 points) Suppose you have a singly linked list and in addition to the head pointer, you are given a pointer to a node p. Then which of the following statements are correct.

A. If p is the last node in the list, then the optimal algorithm for deleting p has time complexity $O(n)$ but not $O(1)$.

B. If p is the last node in the list, then the optimal algorithm for deleting p has time complexity $O(1)$.

C. If p is not the last node in the list, then the optimal algorithm for deleting p has time complexity $O(n)$ but not $O(1)$.

D. If p is not the last node in the list, then the optimal algorithm for deleting p has time complexity $O(1)$.

(e) (2 points) In which of the following data structures can we delete a node in $O(1)$ time.

A. Doubly linked list

B. Stack (the Pop operation)

C. Red-black trees

D. Array

Question 4. Let A and B be two arrays of size n each, sorted in ascending order. Consider the algorithm

`WhatsMyOutput` given below.

Algorithm 1: `WhatsMyOutput`$(A[1,\ldots,n], B[1,\ldots,n])$

```
if $n == 1$ then
    return $(A[1] + B[1])/2$;
else if $n == 2$ then
    return $(\max(A[1], B[1]) + \min(A[2], B[2]))/2$;
else
    if $n$ is even then
        $p_A \leftarrow (A[n/2] + A[n/2+1])/2$;
        $p_B \leftarrow (B[n/2] + B[n/2+1])/2$;
    else
        $p_A \leftarrow A[(n+1)/2]$;
        $p_B \leftarrow B[(n+1)/2]$;
    end
    if $p_A < p_B$ then
        return WhatsMyOutput($A[n/2+1,\ldots,n], B[1,\ldots,(n+1)/2]$);
    else if $p_A > p_B$ then
        return WhatsMyOutput($A[1,\ldots,(n+1)/2], B[n/2+1,\ldots,n]$);
    else
        return $p_A$;
    end
end
```

Note that the arrays A and B are indexed from 1 to n.

(a) (2 points) Let $T(n)$ be the time taken by the algorithm `WhatsMyOutput`$(A[1,\ldots,n], B[1,\ldots,n])$. Give an expression for $T(n)$ and solve it. (Do not forget the base case!)

Solution:

$$\begin{aligned} T(n) &= T(n/2) + c \qquad \text{for } n \geq 3 \\ T(2) &= d_1 \\ T(1) &= d_2 \end{aligned}$$

Therefore,

$$T(n) = O(\log n).$$

(b) (2 points) Let $S(n)$ be the extra space used by the algorithm `WhatsMyOutput`$(A[1,\ldots,n], B[1,\ldots,n])$. Give an expression for $S(n)$ and solve it.

Solution:

$$\begin{aligned} S(n) &= S(n/2) + c \qquad \text{for } n \geq 3 \\ S(2) &= d_1 \\ S(1) &= d_2 \end{aligned}$$

Therefore,

$$S(n) = O(\log n).$$

(c) (3 points) What does the algorithm `WhatsMyOutput`$(A[1,\dots,n], B[1,\dots,n])$ output? (Give a one line answer)

Solution: Median of all the elements in $A \cup B$.

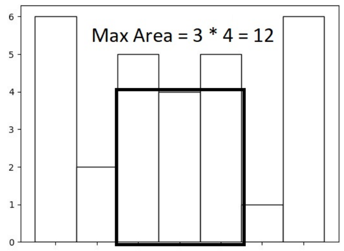
(d) (3 points) Let $S = \{i \mid 1 \le i \le 100\}$. Let A be a subset of S of size 50 and $B = S - A$, such that A and B are sorted in ascending order. Then what is the output of `WhatsMyOutput`$(A[1,\dots,50], B[1,\dots,50])$.

Solution: 50.5 (or 50 if you considered '/' as integer division)

Question 5. A *histogram* is a bar graph, in which the bars are placed adjacent to each other with no gap between them, and the width of each bar equal to 1.

We are given an array A of size n, where $A[i]$ $(1 \le i \le n)$ stores the length of i-th bar in the positive y direction. We need to find the area of the rectangle with the largest area contained in the histogram whose edges are parallel to the x and y axes.

Here is an example of the histogram corresponding to the array: $A = [6, 2, 5, 4, 5, 1, 6]$



(a) (7 points) Complete the below skeletal code for the above task. The function **LargestAreaRectangle** receives the histogram as an array A and size of array n and outputs the area of the largest rectangle in the histogram.

The function **UpdateStackAndArea** is an helper function which takes 4 inputs array A, stack S, and 2 integers max_area and i. Here consider the array A and stack S is passed in the function in constant time and the changes made in the function are also reflected in the original stack.

Algorithm 2: UpdateStackAndArea(A, S, max_area, i)

```
//Write a code to evaluate area of rectangle with height $A_{tp}$ and using histograms till $i$
$tp \leftarrow$ Top($S$);
__________ Pop($S$) __________;
if $IsEmpty(S) == true$ then
    $current\_area \leftarrow A_{tp} * (i - 1)$;
else
    $current\_area \leftarrow$ __________ $A_{tp} * (i - \textbf{Top}(S) - 1)$ __________;
end
if $max\_area < current\_area$ then
    __________ $max\_area = current\_area$ __________;
end
return __________ $max\_area$ __________;
```

Algorithm 3: LargestAreaRectangle(A, n)

```
CreateEmptyStack $S$;
$max\_area \leftarrow 0$;
$i \leftarrow 1$;
while $i \leq n$ do
    if $IsEmpty(S) == true$ or $A_{Top(S)} \leq A_i$ then
        __________ Push($S, i$) __________;
        $i \leftarrow i + 1$;
    else
        $max\_area \leftarrow$ UpdateStackAndArea($A, S, max\_area, i$);
    end
end
while __________ IsEmpty($S$) == false __________ do
    $max\_area \leftarrow$ UpdateStackAndArea($A, S, max\_area, n + 1$);
end
return __________ $max\_area$ __________;
```

(b) (3 points) Provide the time complexity of the given algorithm considering no overhead for passing array and Stack into the function **UpdateStackAndArea** and give a brief justification for that.

Solution: Time Complexity: $O(n)$, where n is the size of the array.

Justification: Each element in the array is pushed exactly once and popped exactly once from the stack. Also in every iteration of the **while** loop you either push or pop an element. Therefore the complexity would be $O(n)$.

Question 6. You are given a binary tree (not necessarily balanced) with n nodes in the form of an array $P[1, \ldots, n]$, where $P[i]$ stores the parent of node i. It is given that the root of the tree is node 1, and we assume the parent of the root node is taken to be root itself, therefore $P[1] = 1$.

Your task is to design a data structure of size $O(n \log n)$, that would allow you to answer the following type of query in time $O(\log n)$:

Ancestor(i, k) := Returns the k-th ancestor of node i.

Note that the 0-th ancestor of a node in the node itself, the first ancestor of a node is it parent, and so on.

(a) (2 points) Describe the data structure that you are planning to use.

Solution: The data structure is a two dimensional array, A, where $A[i][j]$ stores 2^j-th ancestor of node i, where $1 \leq i \leq n$ and $0 \leq j \leq \log n$.

(b) (4 points) Give an algorithm to populate the data structure in time $O(n \log n)$.

Solution: Since 2^0-th or 1-st ancestor of any node is it's parent in the tree, we populate $A[i][0]$ as follows:

```
for $i \leftarrow 1$ to $n$ do
    $A[i][0] \leftarrow P[i]$;
end
```

Now we compute $A[i][j]$ for $j \geq 1$ as follows. Here we use the fact that $2^k = 2^{k-1} + 2^{k-1}$.

```
for $j \leftarrow 1$ to $\log n$ do
    for $i \leftarrow 1$ to $n$ do
        $temp \leftarrow A[i][j-1]$;
        $A[i][j] \leftarrow A[temp][j-1]$;
    end
end
```

(c) (4 points) Using the data structure computed in the previous part, give an $O(\log n)$ time algorithm to compute **Ancestor**(i, k).

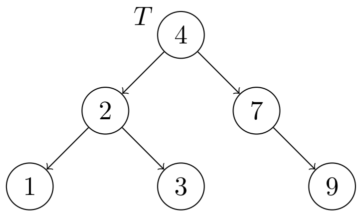
Solution: We calculate the k-th ancestor of the node u using the data structure A as follows:

Algorithm 4: Ancestor(i, k)

```
$count \leftarrow 0$;
$temp \leftarrow i$;
while $k > 0$ do
    if $k$ is odd then $temp \leftarrow A[temp][count]$ ;
    $k \leftarrow k/2$;
    $count \leftarrow count + 1$;
end
return $temp$;
```

Question 7. The preorder traversal of a binary search tree T is an array A that is defined recursively by the algorithm **PreOrderTraversal** given below. Let A and i be global variables, such that A is an array and $i = 1$ initially.

```
Algorithm 5: PreOrderTraversal($T$)
    if $T = NULL$ then return;
    $A[i] \leftarrow$ value($T$);
    $i \leftarrow i + 1$;
    PreOrderTraversal(left($T$)) ;
    PreOrderTraversal(right($T$)) ;
```



Example of a BST T

For example, in the binary search tree given above, the array A will be $[4, 2, 1, 3, 7, 9]$ after calling the function **PreOrderTraversal**(T).

(a) (8 points) Design an $O(n)$ time algorithm, **PreorderParent**($A[1, \ldots, n], k$), that given an array $A[1, \ldots, n]$ corresponding to the preorder traversal some BST T with n nodes and an index k (where $1 \leq k \leq n$), returns the index of k's parent in T. Assume all values in T are distinct.

Some examples are provided below for clarity. The same BST as given above is used in the examples:

- $A = [4, 2, 1, 3, 7, 9]$ and $k = 6$ then return 5. $A[6] = 9$, and the parent of 9 is 7 in T. The index of 7 in A is 5.
- $A = [4, 2, 1, 3, 7, 9]$ and $k = 4$ then return 2. $A[4] = 3$, and the parent of 3 is 2 in T. The index of 2 in A is 2.
- $A = [4, 2, 1, 3, 7, 9]$ and $k = 1$ then return 1. The **parent** of **the root** is taken to be itself.

Provide both a description of your idea, as well as a pseudocode. If you provide an $O(n \log n)$ time algorithm, you will get 5 marks and if you provide an $O(n^2)$ time algorithm, you will get 2 marks.

Description:

Solution: We divide the problem into three cases.

1. If $k = 1$ the its the root of the BST and hence the output is 1 as well.
2. If $A[k-1] > A[k]$ then k is a left child of its parent. This follows from the definition of preorder traversal of a BST. In this case we output $k - 1$ as the answer.
3. If $A[k-1] < A[k]$ then k is a right child of its parent. This is the interesting case.

Alternate solution: When k is the right child of its parent, the parent will always be the predecessor of the k-th node in the tree formed by removing the subtree rooted at k from the tree. This node is essentially the largest value less than $A[k]$, in the subarray $A[1...k]$.

Pseudocode:

Solution:

Algorithm 6: PreorderParent$(A[1,\ldots,n],k)$

```
if $k = 1$ then /* Case when k is root node */
    return 1;
else if $A[k-1] > A[k]$ then/* Case when k is left child of its parent */
    return $k - 1$;
else /* Case when k is right child of its parent */
    $p \leftarrow k - 1$;
    /* in the following while loop we move to an ancestor whose value is
       just bigger than A[k], or if such an ancestor does not exist, then
       the root node.  note that k lies on the left subtree of this
       ancestor. */
    while $A[p] < A[k]$ and $p > 1$ do
        $p \leftarrow p - 1$;
    end
    /* if p is not the root node, then we move to the left child of this
       ancestor */
    if $A[p] > A[k]$ then
        $p \leftarrow p + 1$;
    end
    /* we keep traversing this subtree and update p to be the next node
       on the rightmost path */
    $temp \leftarrow p + 1$;
    while $temp < k$ do
        if $A[temp] > A[p]$ then
            $p \leftarrow temp$;
        end
        $temp \leftarrow temp + 1$;
    end
    /* finally the parent of k will be the last but one node along this
       righmost path */
    return $p$;
end
```

(b) (2 points) Give the time complexity analysis of the algorithm you have designed.

Solution: There are two disjoint `while` loops in the algorithm and both iterate at most n times. Therefore complexity is $O(n)$.

Question 8. You are given a binary tree T having n nodes and height h. Each node has the following fields:

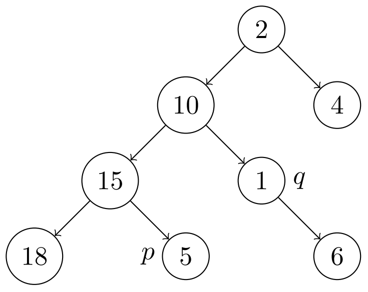
1. *left* : Pointer to the left child (NULL if left child does not exist)
2. *right* : Pointer to the right child (NULL if right child does not exist)
3. *parent* : Pointer to the parent of the node (for root node it is NULL)
4. *val* : Value stored at this node

For two nodes p and q in T,

- **Path**(p, q) = the ordered set of nodes on the unique path from p to q in T.

- **Dist**$(p, q) = \sum_{v \in \textbf{Path}(p,q)} val(v)$.

Here is an example of a binary tree storing values. The path from p to q has values $(5, 15, 10, 1)$. Therefore **Dist**$(p, q) = 31$,



(a) (8 points) Given a binary tree T and two nodes p and q in T, design an $O(h)$ time algorithm to compute the value of **Dist**(p, q). Give a brief description of the idea behind your algorithm and the pseudocode of your algorithm.

Description:

Solution: We compute the path from both nodes p and q upto the root and store them in to separate stacks (S_p and S_q respectively). We then remove the common elements from the stack, storing the last common element between the two paths (this is the least common ancestor of p and q). Next we sum the remaining elements in both stacks together with the least common ancestor.

Pseudocode:

Solution:

Algorithm 7: ComputeDist(T, p, q)

```
CreateEmptyStack($S_p$);
CreateEmptyStack($S_q$);
while $p \neq NULL$ do
    Push($p, S_p$);
    $p \leftarrow parent(p)$;
end
while $q \neq NULL$ do
    Push($q, S_q$);
    $q \leftarrow parent(q)$;
end
while $Top(S_p) == Top(S_q)$ and $IsEmpty(S_p) \neq TRUE$ and $IsEmpty(S_q) \neq TRUE$ do
    $temp \leftarrow$ Top($S_p$);
    Pop($S_p$);
    Pop($S_q$);
end
$sum \leftarrow value(temp)$;
while $IsEmpty(S_p) \neq TRUE$ do
    $sum \leftarrow sum + value(\text{Top}(S_p))$;
    Pop($S_p$);
end
while $IsEmpty(S_q) \neq TRUE$ do
    $sum \leftarrow sum + value(\text{Top}(S_q))$;
    Pop($S_q$);
end
return sum;
```

(b) (2 points) Show that the time complexity of your algorithm is $O(h)$.

Solution: Each `while` loop iterates at most h many times. Complexity is $O(h)$.