Name: Rollno:

ESO207: Data Structures and Algorithms (Midsem Exam)

15th September 2025

Total Number of Pages: 8 Time: 2 hr Total Points 80

Instructions

1. All questions are compulsory.
2. Answer all the questions in the space provided in the question paper booklet. They should not be answered anywhere else.
3. Use the space provided in the paper for rough work. No extra rough sheets will be provided
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	Points	Score
1	10	
2	10	
3	11	
4	13	
5	10	
6	8	
7	10	
8	8	
Total:	80	

Name: Rollno:

Question 1. State whether the following statements are true or false and give suitable justification for your answer.

(a) (2 points) If $f(n) = O(g(n))$ and $f(n) = O(h(n))$ then $g(n) = O(h(n))$.

> **Solution: False**, giving a counter example suffices. One such counter example : $f(n) = 1,\ g(n) = n^2,\ h(n) = n$.

(b) (2 points) If $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$ then $f(n) = O(g(n))$.

> **Solution: False**, giving a counter example suffices. One such counter example : $f(n) = n^2,\ g(n) = n$, therefore, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty > 0$ but $f(n) \neq O(g(n))$.
> *OR*
> **True**, if we assume that limit must be finite. Let the limit be c, then, there exists some δ such that $\forall\ n > \delta,\ c\,g(n) - \epsilon < f(n) < c\,g(n) + \epsilon$, where ϵ can be arbitrarily small. Therefore $f(n) = O(g(n))$.

(c) (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 $n^2/4$.

> **Solution: False**, Maximum number of iterations of the while loop can be n^2. Consider the case when the grid has no obstacles, start position is top-left and destination is bottom-right. Since every node (which is not an obstacle) is enqueued exactly once, and in each iteration exactly one node is dequeued, there will be n^2 iterations.

(d) (2 points) The maximum number of red nodes in a RedBlackTree consisting of n nodes is $n/2$.

> **Solution: False**, RBT is a full binary tree $\implies$ number of internal nodes (say k) = number of leaf nodes - 1. Therefore, $n = k + (k+1) = 2k + 1$. All the $(k+1)$ leaf nodes are null nodes which are black. Also, the root must be black. So, remaing nodes = $k - 1 < n/2$. Hence, maximum number of red nodes cannot be $n/2$.

(e) (2 points) Deleting an element in a binary search tree consisting of n nodes can be done in $O(\log n)$ time.

> **Solution: False**, in the worst case deleting an element in a BST can take $O(n)$ time due to skewing. Consider the case when BST is linear, $1 \to 2 \to \ \ldots\ \to n$, and we want to delete n.

Question 2. Fill in the blank type questions.

(a) (2 points) What is the complexity of **SpecialUnion** operation in a Red-Black tree storing n values? ___$O(\log n)$___

(b) (2 points) The number of bit operations required to multiply two n bit numbers is ___$O(n^{\log_2 3})$___. (Give the bound of the best algorithm discussed in class)

(c) (2 points) Inserting a node into a doubly linked list can be done in ___$O(1)$___ time.

(d) (2 points) The **Partition** function used in QuickSort can be implemented in ___$O(n)$___ time and ___$O(1)$___ extra space.

(e) (2 points) An arbitrary Binary Search Tree consisting of n nodes and height h can be converted to a perfectly balanced Binary Search Tree in ___$O(n)$___ time.

Question 3. For each of the following functions, what is the smallest possible function class in which they belong? Give your answer in big O notation?

(a) (4 points) $g(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (i+j)$

Solution: Given.

$$g(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (i+j) = \underbrace{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} i}_{(1)} + \underbrace{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} j}_{(2)}.$$

Compute (1). For each fixed i, there are $n-i$ terms:

$$(1) = \sum_{i=1}^{n-1} i(n-i) = n \sum_{i=1}^{n-1} i - \sum_{i=1}^{n-1} i^2 = n \cdot \frac{(n-1)n}{2} - \frac{(n-1)n(2n-1)}{6} = \frac{n(n-1)(n+1)}{6}.$$

Compute (2). For each fixed j, it appears $j-1$ times (with $i = 1, \ldots, j-1$):

$$(2) = \sum_{j=2}^{n} j(j-1) = \sum_{j=1}^{n} (j^2 - j) = \frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2} = \frac{n(n+1)(n-1)}{3}.$$

Combine.

$$g(n) = (1) + (2) = \frac{n(n-1)(n+1)}{6} + \frac{n(n+1)(n-1)}{3} = \frac{n(n-1)(n+1)}{2} = \frac{1}{2}(n^3 - n).$$

$g(n) = \Theta(n^3)$, hence the smallest Big-O class is $\boxed{O(n^3)}$.

(b) (3 points) $T(n) = 7T(n/8) + 3n^2$ and $T(1) = 1$

Solution: We have the recurrence

$$T(n) = 7\,T\Big(\frac{n}{8}\Big) + 3n^2, \qquad T(1) = 1.$$

Compare with the Master Theorem form $T(n) = a\,T(n/b) + f(n)$:

$$a = 7, \quad b = 8, \quad f(n) = 3n^2.$$

Compute

$$n^{\log_b a} = n^{\log_8 7}.$$

Since $\log_8 7 < 1$, we have $n^{\log_8 7} = n^{0.935\ldots}$. Thus

$$f(n) = 3n^2 = \Theta\big(n^{\log_8 7 + \varepsilon}\big) \quad \text{with} \quad \varepsilon = 2 - \log_8 7 > 0,$$

so $f(n)$ is polynomially larger than $n^{\log_b a}$ and we are in **Master Theorem Case 3**.
Regularity condition:

$$a\,f\Big(\frac{n}{b}\Big) = 7 \cdot 3\left(\frac{n}{8}\right)^2 = \frac{21}{64}n^2 = \frac{7}{64}\,f(n) \le c\,f(n) \quad \text{with } c = \frac{7}{64} < 1,$$

so the condition holds.
Therefore,

$$\boxed{T(n) = \Theta\big(n^2\big)}.$$

(c) (4 points) $S(n) = S(n/4) + S(3n/4) + n$ and $S(1) = 1$

Solution: We have the recurrence:

$$S(n) = S\left(\frac{n}{4}\right) + S\left(\frac{3n}{4}\right) + n, \qquad S(1) = 1.$$

Approach (Recursion tree method): At each level of recursion, the total work done (excluding recursive calls) is proportional to the sum of the sizes of the subproblems.

Level 0: one problem of size n, work $= n$.

Level 1: two subproblems of sizes $\frac{n}{4}$ and $\frac{3n}{4}$, total work $= \frac{n}{4} + \frac{3n}{4} = n$.

Level 2: Each subproblem again splits into two smaller ones:

$$S\left(\frac{n}{4}\right) \to S\left(\frac{n}{16}\right) + S\left(\frac{3n}{16}\right), \quad S\left(\frac{3n}{4}\right) \to S\left(\frac{3n}{16}\right) + S\left(\frac{9n}{16}\right).$$

Total size at this level $= \frac{n}{16} + \frac{3n}{16} + \frac{3n}{16} + \frac{9n}{16} = n$.

Hence, at every level, the combined size of all subproblems remains n. Thus, each level contributes $\Theta(n)$ work.

Depth of recursion: The largest subproblem has size $\frac{3n}{4}$ at each level, so the depth L satisfies:

$$\left(\frac{3}{4}\right)^L n \leq 1 \quad \Rightarrow \quad L = O(\log n).$$

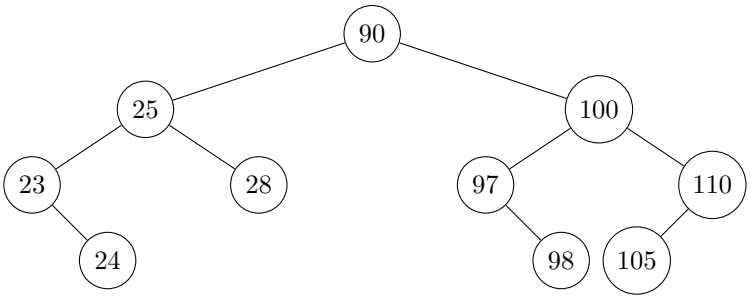
Total work:

$$S(n) = (\text{work per level}) \times (\text{number of levels}) = \Theta(n) \times O(\log n) = \boxed{\Theta(n \log n)}.$$

Name: Rollno:

Question 4. (a) (5 points) In the following BST, there are some keys that if inserted, will increase the height of the tree.

List all the integer keys (distinct from those already present in the BST) for which this is true. If no such keys exists, explain why not. (Each key should increase the height if inserted alone, not with other keys.)



Solution: The current height of the BST is 4. The deepest nodes are 24, 98, and 105. We check possible insertions (as left or right child) under these nodes.

Node 24: Left child: $23 < k < 24 \Rightarrow$ no integer possible.
Right child: $24 < k < 25 \Rightarrow$ no integer possible.

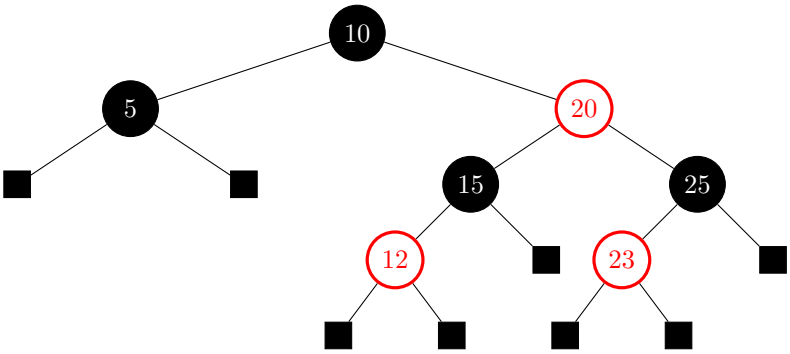
Node 98: Left child: $97 < k < 98 \Rightarrow$ no integer possible.
Right child: $98 < k < 100 \Rightarrow k = 99$.

Node 105: Left child: $100 < k < 105 \Rightarrow k \in \{101, 102, 103, 104\}$.
Right child: $105 < k < 110 \Rightarrow k \in \{106, 107, 108, 109\}$.

Keys that increase the height: $\{99, 101, 102, 103, 104, 106, 107, 108, 109\}$

Each of these insertions adds a new level (depth = 5), thereby increasing the height of the BST.

(b) (8 points) Insert a node with value 24 in the following RedBlack Tree. Show all computation steps and the operation that you perform in that step to obtain the resultant RedBlackTree.



Solution: Node 24 will be inserted with red color to the right of node 23. As node 23 and node 24 both have red color, there is a color imbalance.

To solve this, we observe that parent(24) is red and uncle(24) is black, which points us to case 3.2 of insertion in a red-black tree.

Refer to case 3.2 from lecture notes for further steps.

Question 5. (10 points) Given an array containing n integers with at most k distinct values, design an $O(n \log k)$ algorithm to sort the array. Give the time complexity analysis.

Solution:

We use a Red-Black Tree where each node is augmented with a `count` field. We iterate through the n elements of the input array one by one. For each element, we perform a search operation on the Red-Black Tree. If a node containing the element's value already exists, we increment its associated `count` field. If the element is not found, we insert a new node into the tree with the element's value and initialize its `count` to one. Since there are at most k distinct values, the tree will never contain more than k nodes, ensuring its height remains bounded at $O(\log k)$. As we perform n such search or insert operations, the total time complexity for building this tree is $O(n \log k)$.

To reconstruct the sorted array we perform a single in-order traversal of the tree. As we visit each node, we append its value to our final output array a number of times equal to the value of its stored `count` field.

Question 6. (8 points) Given a sorted array of distinct integers $A[0, ..., n-1]$, design an $O(\log n)$ time algorithm to find out whether there is an index i such that $A[i] = i$.

Solution: We can use a modified binary search since the array is sorted and all elements are distinct.

Algorithm:

1. Initialize two pointers: $low = 0$, $high = n - 1$.
2. While $low \leq high$:
   (a) Compute $mid = \lfloor (low + high)/2 \rfloor$.
   (b) If $A[mid] = mid$, then return **true**.
   (c) If $A[mid] > mid$, then for all $j > mid$, since A is strictly increasing, $A[j] \geq A[mid] + (j - mid) > j$, so no such index can exist in the right half. Hence, search the left half: set $high = mid - 1$.
   (d) If $A[mid] < mid$, then for all $j < mid$, $A[j] \leq A[mid] - (mid - j) < j$, so no such index can exist in the left half. Hence, search the right half: set $low = mid + 1$.
3. If the loop ends without finding any index, return **false**.

Time Complexity: Each iteration halves the search range, so the time complexity is $O(\log n)$.

Question 7. (10 points) Two words u and v (over lower case alphabet symbols) are said to be *anagrams* if one can be obtained from the other by permuting the characters. For example *silent* and *listen* are anagrams, whereas *server* and *verses* are not anagrams.

Given two words u and v (over lower case alphabet symbols) each of the same length n give an $O(n)$ time algorithm to decide whether u and v are anagrams. Give the time complexity analysis.

Solution:

We will use an array of size 26 to store the frequencies of characters, indexed using the difference from the ASCII value of `'a'`. For u and v to be anagrams, we must have the corresponding frequencies of each character equal in both.

function AnagramCheck(u, v)

```
// Although, we are given both to be of equal length, for completeness
If length(u) ≠ length(v) then                                          O(1)
   return False

// Frequency array for 26 letters
Initialize F[0...25] ← 0
For i ← 0 to length(u) − 1 do                                          O(n)
   // To index, we use the ascii difference from 'a'.
   F[u[i] − 'a'] ← F[u[i] − 'a'] + 1          // Increment count for char in u
   F[v[i] − 'a'] ← F[v[i] − 'a'] − 1          // Decrement count for char in v

For i ← 0 to 25 do                                                     O(1)
   If F[i] ≠ 0 then
     return False                                  // An imbalance was found
return True
end function
```

The overall time complexity is $T(n) = O(1) + O(n) + O(1) = O(n)$. The auxiliary space complexity is $O(1)$ because the frequency array F is a fixed size (26).

Question 8. (8 points) You are given a matrix of size $n \times n$, where each row and column is sorted in ascending order. Design an $O(n)$ time algorithm to find whether a target value x exists in the matrix. Give the time complexity analysis.

Solution: We exploit the monotone structure (rows and columns sorted ascending). At the *top-right* corner $A_{1,n}$, every element to its left is $\leq A_{1,n}$ and every element below is $\geq A_{1,n}$. Comparing $A_{i,j}$ with x lets us discard an entire row or column each step: if $A_{i,j} > x$ then all entries below in column j are too large, so move left; if $A_{i,j} < x$ then all entries to the left in row i are too small, so move down. This "staircase" walk either finds x or exits the matrix if x is absent.

Algorithm (staircase search).

1. Initialize $i \leftarrow 1$, $j \leftarrow n$ (top-right corner of A).
2. While $i \leq n$ and $j \geq 1$:
   (a) If $A_{i,j} = x$, **return true**.
   (b) If $A_{i,j} > x$, set $j \leftarrow j - 1$ *(move left; discard column j).*
   (c) If $A_{i,j} < x$, set $i \leftarrow i + 1$ *(move down; discard row i).*
3. If the loop ends, **return false**.

Correctness. Because rows and columns are sorted in ascending order:

- If $A_{i,j} > x$, then for all $r \geq i$ we have $A_{r,j} \geq A_{i,j} > x$, so x cannot be in column j; moving left is safe.

- If $A_{i,j} < x$, then for all $c \leq j$ we have $A_{i,c} \leq A_{i,j} < x$, so x cannot be in row i; moving down is safe.
- If $A_{i,j} = x$, we have found the target.

Therefore the algorithm returns true iff x appears in A.

Time complexity. Each iteration moves either one step left (decreasing j) or one step down (increasing i). Since j can decrease at most n times and i can increase at most n times, there are at most $2n$ iterations. Hence the running time is $\mathcal{O}(n)$ and the extra space is $\mathcal{O}(1)$.