



University of Maryland College Park

Department of Computer Science

CMSC132 Spring 2025

Exam #3

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g., 123456789):

Instructions

- Please print your answers and use a pencil.
- This exam is a closed-book, closed-notes exam with a duration of 50 minutes and 100 total points.
- **Do not remove the exam's staple.** Removing it will interfere with the scanning process (even if you staple the exam again).
- Write your directory id (e.g., terps1, not UID) at the bottom of pages with **DirectoryId**.
- Provide answers in the rectangular areas.
- Do not remove any exam pages. Even if you don't use the extra pages for scratch work, return them with the rest of the exam.
- Your code must be efficient and as short as possible.
- If you continue a problem on the extra page(s) provided, make a note on the particular problem.
- You don't need to use meaningful variable names; however, we expect good indentation.
- **You must write your name and id at this point (we will not wait for you after time is up).**
- You must stop writing once time is up.

Grader Use Only

#1	Part #1 (Short Answer)	45	
#2	Part #2 (Code 1)	16	
#3	Part #3 (Code 2)	39	
Total	Total	100	

Part #1 (Short answer – 3 points each)

1. A free tree is a graph that is undirected, connected, and _____.
2. Which statement is **false**:
 - A. All rooted trees have a degree of 2.
 - B. In the worst case, merge sort will run in $O(n \log n)$.
 - C. The Java hashCode method can return a negative integer.
 - D. Insertion sort runs in $O(n)$ in the best case.
3. Which statement is **true**:
 - A. Linear probing is just another name for open addressing.
 - B. The load factor when using separate chaining will always be 1 or less.
 - C. The default version of the clone method does a shallow copy.
 - D. $\lg(1024)$ is 11.

For #4 to #6, provide one integer as the answer. No mathematical expressions.

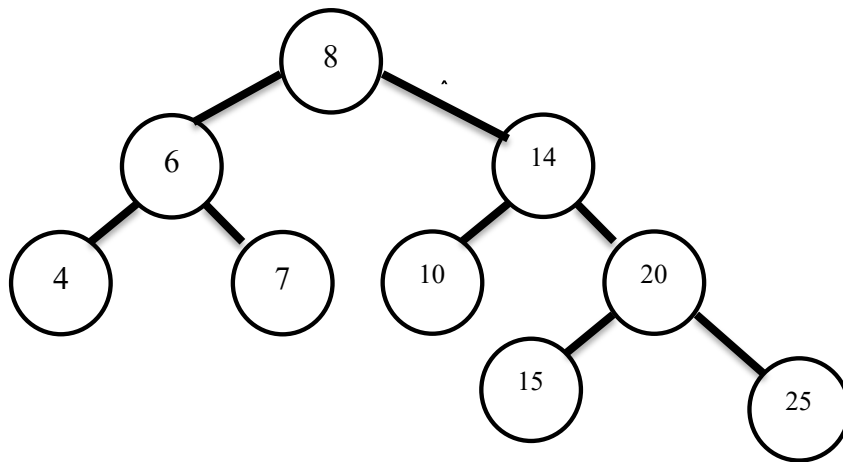
4. $\log_2(2048 \times 512)$ is _____
5. $\log_2(256^{10})$ is _____
6. The root node is at level 0. What is the maximum number of nodes that can appear at level 7 of a binary search tree? _____
7. _____ is a sorting algorithm described as having a hard split, but easy merge.
8. Assume you use a linked list to implement a Set ADT. Although insertion into the list is $O(1)$, explain why insertion into the set is $O(n)$ in the worst case. *(No more than 2 sentences.)*

9. Assume you are inserting the integers 1 to 5. Give a sequence of these numbers such that inserting them into an empty binary search tree would result in the worst-case degenerate tree:

10. Explain why a recursive solution to a coding problem is not always efficient in terms of memory usage. *(No more than 2 sentences.)*

11. Draw the **binary search tree** if the keys are inserted in the following order: 30, 20, 40, 60, 70, 10, 50.

For questions 12 to 15, assume the following **binary search tree**:



12. Give the keys in pre-order: _____

13. Give the keys in post-order: _____

14. What is the height of the tree? _____

15. Why is this tree an example of a full tree? _____

Directory ID:

Part #2 (Code 1)

16. Finish the code for the **difference** method below. See sample **main** method on the next page.

```
public class SetDifferenceDemo {  
    /**  
     * Returns a new set containing elements in set1 but not in set2.  
     *  
     * Assumptions:  
     * - Both input sets have one or more elements (No Nulls)  
     * - The only library methods ALLOWED are: Creating a new HashSet via default constructor, and the contains and add methods  
     *  
     * @param set1 the set to subtract from  
     * @param set2 the set whose elements should be removed  
     * @param <T> the element type  
     * @return a new set with the difference  
     */  
    public static <T> Set<T> difference(Set<T> set1, Set<T> set2) {
```

```

public static void main(String[] args) {
    Set<String> fruits = new HashSet<>();
    fruits.add("apple"); fruits.add("banana"); fruits.add("tangerine");

    Set<String> unwanted = new HashSet<>();
    unwanted.add("banana"); unwanted.add("date");

    Set<String> result1 = difference(fruits, unwanted);
    System.out.println("Sample 1 difference: " + result1); // Should print [apple, tangerine]

    Set<Integer> nums1 = new HashSet<>();
    nums1.add(1); nums1.add(2); nums1.add(3); nums1.add(4);

    Set<Integer> nums2 = new HashSet<>();
    nums2.add(3); nums2.add(4); nums2.add(5);

    Set<Integer> result2 = difference(nums1, nums2);
    System.out.println("Sample 2 difference: " + result2); // Should print [1, 2]
}
}

```

Part #3 (Code 2)

Write a method to find the **second smallest key** in a **Binary Search Tree (BST)**. First, answer these questions for 3 points each. They will help guide your thinking before you write the code.

17. What is another case (besides the empty tree) where there is no **second smallest key**?

18. The smallest key is located at the leftmost node. One possible location for the **second smallest key** is which *ancestor* of the smallest node?

19. Another possible location for the **second smallest key** is which *descendant* of the smallest node?

```

public class BinarySearchTree<K extends Comparable<K>> {
    private class Node {
        private K key;
        private Node left, right;
        private Node(K key) {
            this.key = key; } }

    private Node root;

    public K secondSmallest() { //write the code on the next page}

```

- Assume BST add method as seen in class. If smaller go to left, if larger go to the right. No duplicate keys allowed
- Assume import of `java.util.NoSuchElementException`;

```

    public static void main(String[] args) {
        BinarySearchTree<Integer> tree = new BinarySearchTree<>();
        tree.add(40); tree.add(20); tree.add(60);
        tree.add(10); tree.add(30); tree.add(50); tree.add(70);
        System.out.println("2nd Smallest: " + tree.secondSmallest()); // 2nd Smallest: 20
        tree.add(15); tree.add(12); tree.add(18);
        System.out.println("2nd Smallest: " + tree.secondSmallest()); // 2nd Smallest: 12
    }
}

```

20. Finish **secondSmallest()** and have it return the second smallest key in the BST. Throw a **NoSuchElementException** with any message, if the tree has no second smallest key. **Must be iterative.** No library methods other than creating the exception.

```
public K secondSmallest() {
```