Midterm Reattempt

Winter 2026

Name Answer Key

Net ID _____ (@uw.edu)

Academic Integrity: You may only complete this reattempt if you registered to do so using the google form distributed to the class. You may not use any resources on this exam except for your one page (front and back) reference sheet, writing instruments, your own brain, and the exam packet itself. This exam is otherwise closed notes, closed neighbor, closed electronic devices, etc.. Your answer for each question should fit in the answer box provided.

Instructions: Before you begin, **Put your name and UW Net ID at the top of this page.** Make sure that your name and ID are LEGIBLE. Please ensure that all of your answers appear within the boxed area provided.

Section	Max Points
ADTs and Data Structures	6
Asymptotic Analysis	13
Heaps	12
AVL Trees	14
Algorithms	9

Section 1: ADTs

(6 pts) **Question 1: ADT Applications**
For each scenario below, identify which of the ADT options provided is the most appropriate choice. Options: Stack, Queue, Priority Queue, Ordered Dictionary

1. A robotic vacuum is cleaning a house. The robot begins at its docking station. It follows a path to clean its current room. If at any point it discovers a new room, it will pause the cleaning path of the current room and then begin a new path for the new room. After finishing with the new room it will return to the original room, picking up where it left off. Which ADT should the vacuum use to store the rooms?

   Stack

2. When a user is editing a google doc our computers do not send every mouse or keyboard interaction to the server independently. Instead they gather several keyboard/mouse interactions in a data structure, then send that data structure to the server for processing. What ADT should this data structure implement?

   Queue

3. When an ATM is used, it photographs the user every 30 seconds. These photos, along with their time stamps, are stored in a data structure which will be used to look up pictures by range of time stamps (e.g. get all photos taken between 10:08 and 10:23). What ADT should this data structure implement?

   Ordered Dictionary

4. We have a tool which analyzes course web pages for quality control. Any instructor at any time can request for their web page to be analyzed. The analysis takes a very long time, so there is often a backlog of requests. When selecting the next request, the system will always select the request for the course with the highest enrollment. What ADT should be used for storing the requests?

   Priority Queue

5. Several processors work together to scan for security vulnerabilities in a computer cluster. Each scanner removes a computer out of a data structure, scans that computer, then returns the computer to the data structure. Each time a processor removes a computer we need it to be the computer that has had the longest time elapsed since its last scan. What ADT should we use for storing the computers?

   Queue

6. I have an AI tool that movie producers use for generating music. Since computing time is limited, producers submit a prompt and a bid for how much they will pay for the result. We always process the prompt with the highest current bid. What ADT should be used for storing the bids?

   Priority Queue

Section 2: Asymptotic Analysis

(4 pts) **Question 2: Code Analysis**
Give the best and worst case running times for each algorithm below. Both algorithms add all of the contents from an AVL tree of size n into a binary search tree of size m, but do so in a different order.

Your running time may depend on *both* the values of n and m.

1. Suppose we populate the BST using an **in-order** traversal of the AVL tree as follows:

```
public void populate1(TreeNode AVLRoot, BST tree){
    if(AVLRoot != null){
        populate1(AVLRoot.left, tree);
        tree.insert(AVLRoot);
        populate1(AVLRoot.right, tree);
    }
}
```

Best Case Running Time: $\theta(n^2)$

Worst Case Running Time: $\theta(m \cdot n^2)$

2. Suppose we populate the BST using an **pre-order** traversal of the AVL tree as follows:

```
public void populate1(TreeNode AVLRoot, BST tree){
    if(AVLRoot != null){
        tree.insert(AVLRoot);
        populate1(AVLRoot.left, tree);
        populate1(AVLRoot.right, tree);
    }
}
```

Best Case Running Time: $\theta(n \log n)$

Worst Case Running Time: $\theta(m \cdot n \log n)$

(5 pts) **Question 3: Tree Method**

Suppose that the running time of an algorithm is expressed by the recurrence relation:

$$T(n) = 4 \cdot T\left(\frac{3n}{4}\right) + n$$

$$T(1) = 1$$

For the following questions, use the tree method to solve the recurrence relation. We have broken up the process into parts to guide you through your answer. You may assume that $2n/3$ is always an integer.

1) Fill in the values of a and b used to express the recurrence as $T(n) = a \cdot T(\frac{n}{b}) + f(n)$

$a =$ 4 $b =$ $\frac{4}{3}$

2) Sketch the tree in space below. Include at least the first 3 levels of the tree (i.e. the root, its children, and its grandchildren), make clear the input size for each recursive call as well as the work per call.

...

3) Indicate exactly the total amount of work done at level i of the tree (define the root to be level 0). Include all constants and non-dominant terms.

$n \cdot 3^i$

4) Indicate the level of the tree in which the base cases occur.

$\log_{4/3}(n) = \log_2(n)/\log_2(4/3)$.

5) Give a simplified Θ bound on the solution. When simplified, n should not appear in any exponents. (Hint: the log rule $a^{\log_b(c)} = c^{log_b(a)}$ may be helpful!)

$\Theta\left(n \cdot 3^{\log_{4/3}(n)} = n \cdot n^{\log_{4/3}(3)} = n^{1+\log_{4/3}(3)} = n^{\log_{4/3}(4)} \right)$

(4 pts) **Question 4: Change c or n_0**

For each part below, we give $f(n)$ and $g(n)$. Our goal is to show that $f(n) = O(g(n))$. We have provided a choice of c and n_0 that **do not** work to show that $f(n) = O(g(n))$. First, show *why* that choice of c and n_0 does not work, then correct the issue by suggesting either a different c or a different n_0 (you must change one and the other must stay the same). You do not need to justify that your c and n_0 work.

1. $f(n) = n^3 + 3n - 3$, $g(n) = n^3$, $c = 1$, $n_0 = 1$

   Justification:

   consider $n = 3$. In this case $f(3) = 2 \cdot 3^2 + 10 \cdot 3 - 10 = 18 + 30 - 10 = 38$
   $c \cdot g(n) = 2 \cdot 3^2 = 18$

   Variable you're changing (c or n_0): c

   New value: any $\geq 1.44444...$

2. $f(n) = \sqrt{\log_8 n}$, $g(n) = \log_8 n$, $c = 1$, $n_0 = 1$.

   Hint: consider when taking the square root of a value makes it larger vs. smaller.

   Justification:

   Consider $n = 2$. In this case $\log_8 2 = \frac{1}{3}$ and so $\sqrt{\log_8 2} = \sqrt{\frac{1}{3}} = \frac{1}{\sqrt{3}} > \frac{1}{3} = \log_8 2$.

   Variable you're changing (c or n_0): n_0

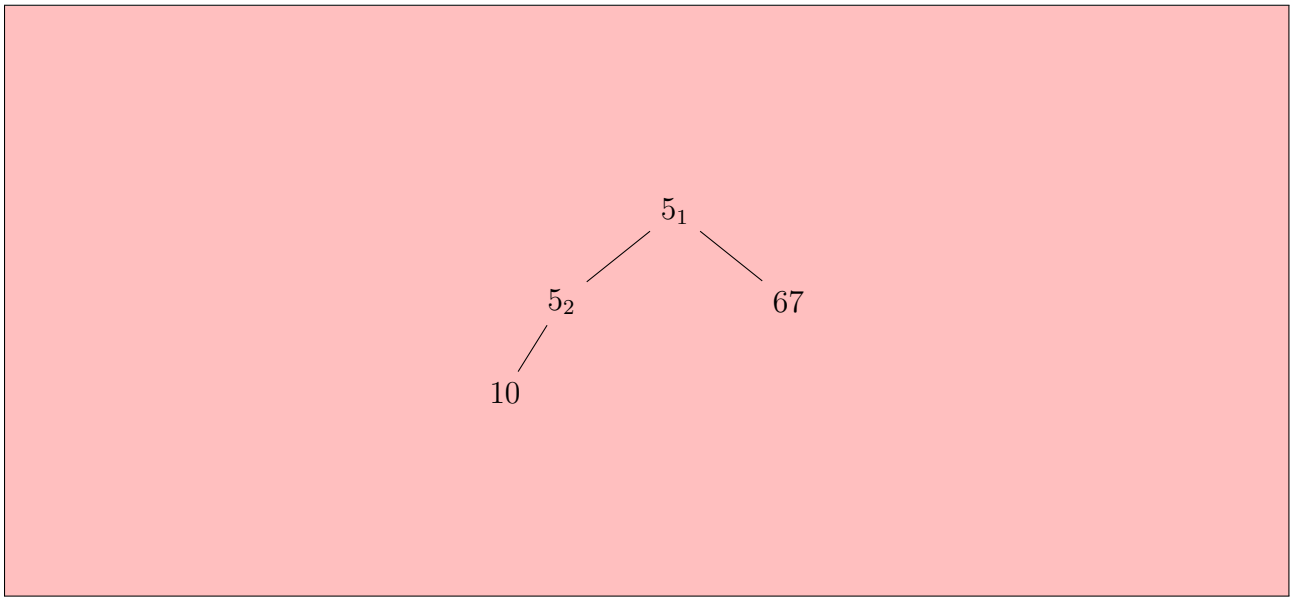
   New value: any ≥ 8

Section 3: Heaps

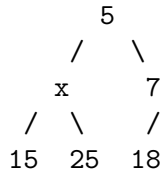
(5 pts) **Question 5: Heap Inserts**
Draw the Binary Min-Heap that results from performing the following operations in order: insert(5_1), insert(2), insert(67), insert(5_2), insert(10), extract()

The items 5_1 and 5_2 refer to two distinct items both with priority 5, make sure we can tell which is which in your drawing.

Assume that elements with equal keys are not swapped during insertion or deletion.



(4 pts) **Question 6: Heap Property**
Consider the following Binary Min-Heap:



1. What is the smallest integer value for x that maintains the heap property?

   5

2. Give an integer key which, when inserted, will cause exactly **1 swap** during `percolateUp`.

   any ≥ 5 and < 7

(3 pts) **Question 7: Heap Arithmetic**
Consider a Binary Max-Heap with n nodes.

1. How many leaf nodes are in a heap with $n = 17$ nodes.

   $$\text{leaves} = \lceil \tfrac{17}{2} \rceil = \lceil 8.5 \rceil = 9$$

2. How many nodes are on the last level of a heap with 21 nodes.

   $$6$$

3. What is a height of a heap with 127 nodes? (*height* = number of edges from root to the deepest node).

   $$\text{height} = \lfloor \log_2 127 \rfloor = \lfloor 6.99 \rfloor = 6$$

Section 4: AVL Trees

(6 pts) **Question 8: Largest Two Items**

For each question below we give a data structure and an operation (that is not an ADT operation). For each, describe an algorithm for that operation (in either English or pseudocode), then give your algorithm's worst case asymptotic running time. Only algorithms with the minimum possible asymptotic running time will receive full credit. Correct but slower algorithms will receive partial credit.

1. Given a key that is present in the dictionary, find the **next largest** key in an AVL Tree of size n (i.e. the smallest key present that is larger than the one given).

   Algorithm:

   first navigate to the given key's node using a BST search (recurse left if smaller than the current, right if larger), then return the smallest of: the parent of that node, and smallest node in its right subtree.

   Worst Case Running time: $O(\log n)$

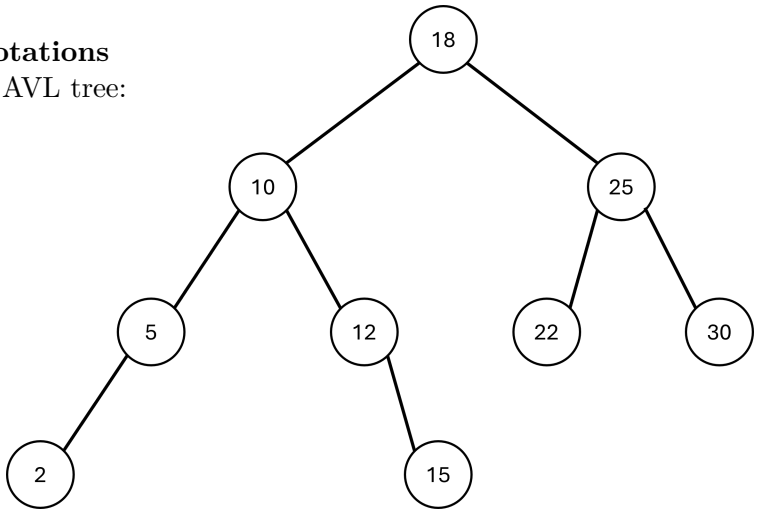
2. Find the **third smallest** element in an AVL tree of size n. You may assume that $n \geq 3$. (Hint: you may use the "next largest" operation described above as a subroutine without re-implementation.)

   Algorithm:

   navigate to the minimum node (keep going left as long as there is a left child), then call next largest twice.

   Worst Case Running time: $O(\log n)$

(8 pts) **Question 9: Rotations**
Consider the following AVL tree:



For each question below, indicate the *problem node* and *type of rotation* that would occur if the given insertions are done on the tree above. Recall that the problem node is the node whose subtrees we need to re-balance. Write one of the following to indicate your answer:

– **R** for a single right rotation

– **L** for a single left rotation

– **LR** for a left-then-right double rotation

– **RL** for a right-then-left double rotation

– **NR** for no rotation (write this for the "Problem node" box as well)

Each question should be considered completely independently (i.e. "reset" to the image between questions).

1. `insert(19)`

   Problem node: 12 Rotation type: L

2. `insert(31)`

   Problem node: NR Rotation type: NR

3. `insert(28)`
   `insert(29)`

   Problem node: 30 Rotation type: LR

4. `insert(7)`
   `insert(6)`

   Problem node: 18 Rotation type: R

Section 5: Algorithms - Returning A Deepest Leaf

For both questions in this section, suppose we have a collection of n integers stored in the tree-like data structure named in the question. Our goal is to return **any one of the integers stored in a leaf of maximal depth**. For example, if we had a tree of height 3, then we would want to return any integer stored in a leaf that is exactly 3 edges away from the root.

(5 pts) **Question 10: Binary Search Tree**
You may assume the following TreeNode class is used by BST.

```
public TreeNode(K key, V value) {
    this.key = key;
    this.value = value;
    height = 0;
    this.left = null;
    this.right = null;
}
```

1. Give an asymptotically optimal algorithm that achieves this for the Binary Seach Tree data structure.

   From the current node, if its height is 0 then return its value. Otherwise, return the result of a recursive call for any child whose height is exactly 1 less than the current.

2. What is the best case runtime for your algorithm? $O(\log n)$

3. What is the worst case runtime for your algorithm? $O(n)$

4. What is the worst case runtime for your algorithm if the BST is an AVL Tree? $O(\log n)$

5. If the worst running time is different, give an intuitive justification for why. If the worst running time is the same, give an intuitive justification for why. 1-2 sentences should be sufficient

It is the different because AVL trees are guaranteed to have log height due to the height balancing requirement.

(4 pts) **Question 11: Binary Min Heap**

Assume the binary min heap is represented in an array with the root stored at index 1.

1. Give an asymptotically optimal algorithm that achieves the goal of returning the value if any of the leaves of largest depth of the Binary Min Heap.

The first node at level d appears at index 2^{d-1}. The height of the heap is $\lfloor \log_2 n \rfloor$. return any values from between $2^{\lfloor \log_2 n \rfloor} - 1$ up to $n - 1$

2. Give the best case runtime of your algorithm.

$\Theta(1)$

3. Give the worst case runtime of your algorithm.

$\Theta(1)$

