CSE 332 Autumn 2025 Midterm
Version A

Name: _____Sample Solution_____

Email address (UWNetID): _____

Instructions:

- The allotted time is 60 minutes.
- Please do not turn the page until the staff says to do so.
- This is a closed-book and closed-notes exam. You are NOT permitted to access electronic devices including calculators.
- Read the directions carefully, especially for problems that require you to show work or provide an explanation.
- When provided, write your answers in the box or on the line provided.
- Unless otherwise noted, every time we ask for an O, Ω, or Θ bound, it must be simplified and tight.
- Unless otherwise noted, assume all data structures are implemented as described in lecture.
- For answers that involve bubbling in a ◯ or ☐, make sure to fill in the shape completely.
- If you run out of room on a page, indicate where the answer continues. Try to avoid writing on the very edges of the pages: we scan your exams and edges often get cropped off.
- A formula sheet has been included at the end of the exam.

Advice

- If you feel like you're stuck on a problem, you may want to skip it and come back at the end if you have time.
- Look at the question titles on the cover page to see if you want to start somewhere other than problem 1.
- **Relax and take a few deep breaths. You've got this! :-).**

Total: 74 points

Q1: Short-answer questions (20 pts)

- Unless otherwise noted, assume all data structures are implemented as described in lecture.
- For questions asking you about runtime, give a simplified, tight Big-O bound. This means that, for example, $O(5n^2 + 7n + 3)$ (not simplified) or $O(2^{n!})$ (not tight enough) are unlikely to get points. Unless otherwise specified, all logs are base 2.
- For questions with a mathematical answer, you may leave you answer as an unsimplified formula (e.g. $7 \cdot 10^3$).

We will only grade what is in the provided answer box.

a. Floyd's Build Heap method moves from:

(Select one) ○ low indices to high indices or ● high indices to low indices of the array

and calls:

(Select one) ○ percolateUp or ● percolateDown.

b. Give a **simplified**, tight big-O bound for

$$f(N) = N^{0.0001} + \log_{10}(N^{10}) + 3$$

$O($ $N^{0.0001}$ $)$

c. Give the **worst case** runtime for inserting 2^N elements into an initially empty AVL tree. (Give your answer in terms of N.):

$O($ $N2^N$ $)$

d. Assuming only integers ≥ 1 may be inserted, what is the smallest possible integer in the 3rd level (the root node is level 0) of a binary min heap?

Heaps are PriorityQueues, which means they can contain duplicates!

So all the integers can be 1.

1

e. Give the **best case** runtime for inserting a (key, value) pair into an AVL tree used as a dictionary, containing N (key, value) pairs. (Give your answer in terms of N.)

$O($ 1 $)$

Q1: (continued)

(Same instructions as on previous page)

f. Give a **simplified**, tight big-O bound for

$$f(N) = (3^N + N^{20})^3$$

$O(\boxed{27^N})$

g. Give the **worst case** runtime of removing the $\log N$ largest elements from a binary max heap containing N elements. (Give your answer in terms of N.)

$O(\boxed{(\log N)^2})$

h. Give the **worst-case** runtime of removing the second smallest element from a binary search tree containing N elements. (Give your answer in terms of N.)

$O(\boxed{N})$

i. Give a **simplified**, tight big-O bound for

$$f(N) = \log^2(N) + \log(\log(N^2))$$

$O(\boxed{\log^2(N)})$

j. Give the **minimum number of leaf nodes** in a binary min heap of height 4. (Remember: A single node is a tree of height 0.) Give an exact number, not a formula.

$\boxed{8}$

Q2: Code Analysis (16 pts)

Describe the worst-case running time for the following pseudocode functions in Big-O notation in terms of the variable n. Your answer **MUST** be tight and simplified. **You do not have to show work or justify your answers for this problem.**

a)

```
public static void flipACoin(int n) {
      int coinSide = 0;  // even is heads, odd is tails
      for (int i = 1; i < n; i *= 2) {
            int j = i;
            while (j < n) {
                  coinSide += 1;
                  j += i;
            }
      }
      if (coinSide % 2 == 0) {
            System.out.println("Heads!");
      } else {
            System.out.println("Tails!");
      }
}
```

O(n)

Explanation:

We analyze how i changes throughout each iteration:

Iteration number	0	1	2	3	k
Value of i	1	2	4	8	2^k

This means we stop after $\log_2 n$ iterations.

Now we consider the inner loop. This inner loop iterates from j=0 to j=n, but j increments by i each iteration of the while loop. Therefore, the while loop iterates n/i times or $n/2^k$ times on the k'th iteration. This means we get the summation:

$$\sum_{k=0}^{log_2 n-1} \frac{n}{2^k} = n \bullet \sum_{k=0}^{log_2 n-1} \frac{1}{2^k} = n \bullet c = O(n)$$

b)

```
public static int mystery(int n, int k) {
      if (n < 100) {
          for (int i = 0; i < n * n * n; i++) {
            k++;
          }
          return k;
      } else if (n < 5000) {
          return n * n;
      }
      k +=  mystery(n / 2, k);
      return k * mystery(n / 2, k);
    }
```

O(n)

Recurrence becomes T(n) = 2T(n/2) + c. drawing the tree and solving the summation, this becomes 2^i * c work per level for log(n) levels. So the answer is O(n). This is similar to the example we saw in lecture for recursive list summation.

Q2: (continued)

c)
```
public static int calculateNetWorth(int n) {
      if (n < 10000) {
            return n + getStockBonus(n);
      }
      int wealth = 0;
      for (int i = 1; i < n; i *= 3) {
            if (i % 27 == 0) {
                  wealth += sellStocks(n);
            }
      }
      return wealth;
}
public static int getStockBonus(int shares) {
      int profit = 0;
      for (int stockNum = 0; stockNum <= shares; stockNum ++) {
            profit = profit + stockNum * 332;
      }
      return profit;
}
public static int sellStocks(int shares) {
      int total = 0;
      for (int stockNum = 1; stockNum <= shares; stockNum *= 3) {
            total = total + stockNum * 332;
      }
      return total;
}
```

For `calculateNetWorth`:

$O(\boxed{\log^2 n})$

d)
```
public static BinarySearchTree treeMystery(int n) {
      BinarySearchTree bst = new BinarySearchTree();
      bst.insert(0);
      for(int i = 1; i < n; i++){
            bst.insert(i);
            bst.insert(-i);
      }
      return bst;
}
```

$O(\boxed{n^2})$

Q3: O, Ω, and Θ (9 pts)

For each of the following statements, indicate whether it is always true, sometimes true, or never true. You do not need to include an explanation. Assume that the domain and codomain of all functions in this problem are natural numbers(1, 2, 3 ...).

a) A function that is O(n log n) is _____ O(log n)

○ Always ○ Never ● Sometimes

b) A function that is Ω(1) is _____ O(n).

○ Always ○ Never ● Sometimes

c) A function that is $\Theta(n^2)$ is _____ $\Omega(n^3)$.

○ Always ● Never ○ Sometimes

d) A function that is $\Omega(n^2)$ is _____ $O(n^2)$

○ Always ○ Never ● Sometimes

e) A function that is O(n) is _____ $\Theta(n^2)$.

○ Always ● Never ○ Sometimes

f) A function that is $\Theta(2^n)$ is _____ $\Omega(n^{1024})$

● Always ○ Never ○ Sometimes

g) A function that is $O(\log(n^n))$ is _____ $O(n^2)$.

● Always ○ Never ○ Sometimes

h) If f(n) is O(g(n)) and g(n) is Ω(h(n)), then f(n) is _____ O(h(n)).

○ Always ○ Never ● Sometimes

i) If f(n) is both Θ(g(n)) and Ω(h(n)), then g(n) is _____ Ω(h(n)).

● Always ○ Never ○ Sometimes

Q4: Write a Recurrence (5 pts)

Give a base case and a recurrence for the worst-case runtime of the following function. **Use variables appropriately for values that are constants (e.g. c_1, c_2, etc.) in your recurrence**; you do not need to attempt to count the exact number of operations, but you should include lower-order terms. **YOU DO NOT NEED TO SOLVE** this recurrence.

```
public static int fun(int n) {
    if (n > 9) {
        int sum = fun(n - 4);
        for (int i = 0; i < n * n; i++) {
            sum += i;
        }
        return fun(n/5);
    } else {
        for (int j = 0; j < n; j++) {
            System.out.println(j);
        }
        return n;
    }
}
```

$T(n) =$ _____ c_0 _____ For $n \leq 9$

$T(n) =$ _____ $T(n-4) + T(n/5) + c_1n^2 + c_2$ _____ For $n > 9$

Yipee!!!! YOU DO NOT NEED TO SOLVE *this* recurrence…

Q5: Tree Method (8 pts)

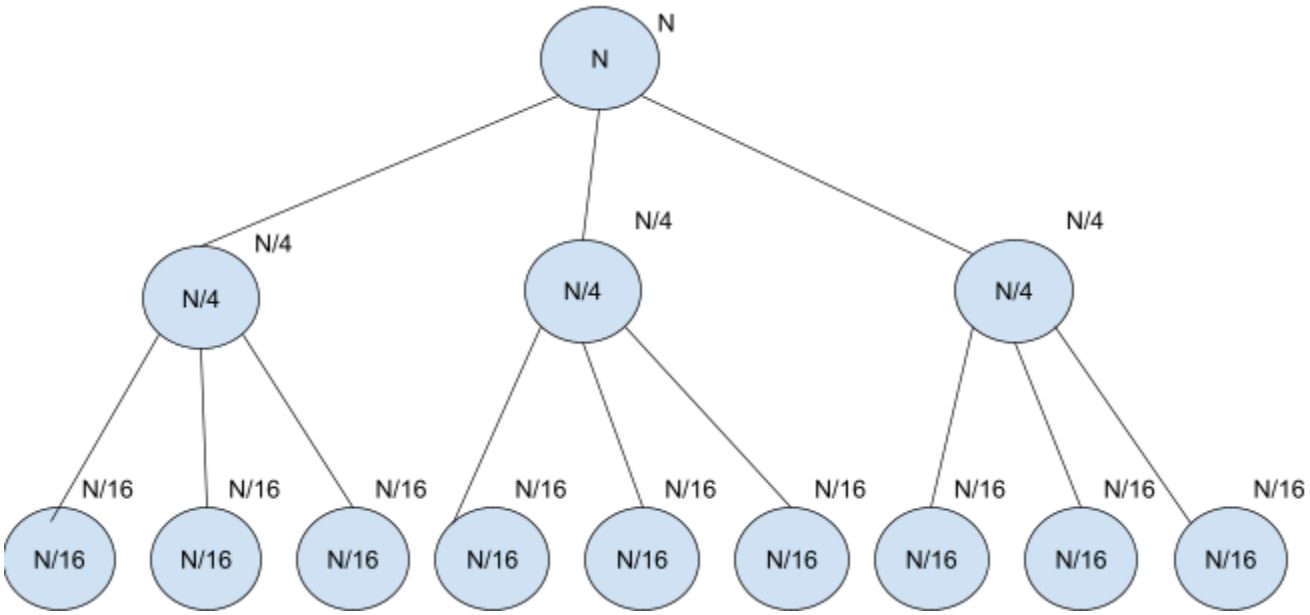
Suppose that the running time of an algorithm is expressed by the recurrence relation:

$$T(1) = 1$$

$$T(N) = 3T(\tfrac{N}{4}) + N \qquad \text{for integers N > 1}$$

For the following questions, use the tree method to solve the recurrence relation. We have broken up the process into subquestions to guide you through your answer. You may assume that N is always a power of 4. **Be sure to include bases in logs in your answers if any.**

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



b) 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.

$(3/4)^i n$

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

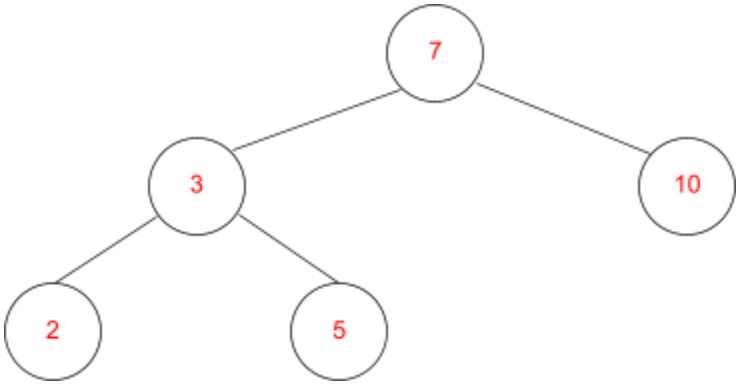
$\log_4 n$

d) Give a simplified Θ bound on the solution. When simplified, n should not appear in any exponents.

$\Theta(\ n\ )$

Q6: AVL (9 pts)

You are given the following AVL Tree used as a dictionary, containing (key, value) pairs (only the keys are shown). **Answer the questions on the next page**.



Q6 (continued)

a) (2 pts) Give a single integer key that if inserted would cause a **double rotation**. Write N/A if not possible.

Note: 6 would work as well

4

b) (2 pts) Give a single integer key that if inserted would cause a **single rotation**. Write N/A if not possible.

Note: any integer <= 1 works as well

1

c) (2 pts) Give a single new integer key (not already present) that if inserted would cause **no rotations**. Write N/A if not possible.

Note: 8, 9 and any integer >= 11 works as well

11

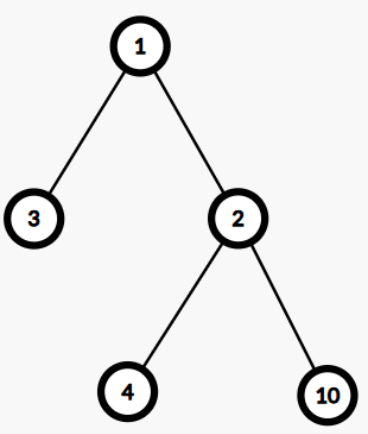
d) (3 pts) What is the minimum number of integer keys we need to insert to increase the height of the tree to be 2 more than its current height?

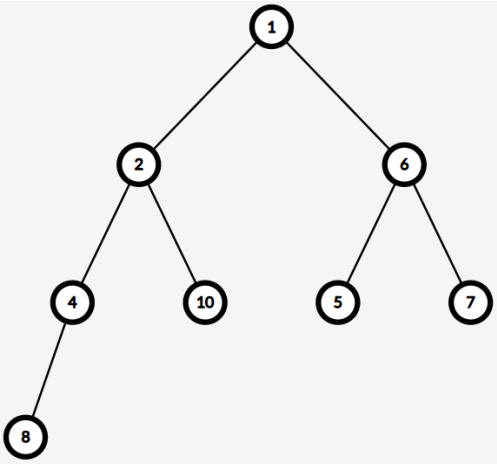
Currently Tree contains 5 nodes, 12 nodes total are needed to reach height 4.

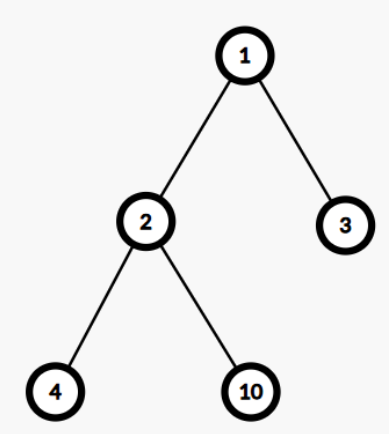
7

Q7: Heaps (7 pts)

a) (3 pts) For each binary **min** heap, select if it is valid or invalid.







i) ○ Valid / ● Invalid ii) ○ Valid / ● Invalid iii) ● Valid / ○ Invalid

b) This is an array representation of a 0-indexed binary **max** heap:

23	16	20	5	4	18	1			

i) (2 pts) Insert 19 into the **original** heap. Fill in the array representation of the resulting heap below. You are welcome to draw the heap structure, **but we will only grade what is written in the array**.

23	19	20	16	4	18	1	5		

ii) (2 pts) Perform a deleteMax on the **original** heap. Fill in the array representation of the resulting heap below. You may draw the heap structure, **but we will only grade what is written in the array**.

20	16	18	5	4	1				

Summations

1. $\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$ for $|x| < 1$
2. $\sum_{i=1}^{n} cf(i) = c\sum_{i=1}^{n} f(i)$
3. $\sum_{i=0}^{n-1} 1 = \sum_{i=1}^{n} 1 = n$
4. $\sum_{i=0}^{n} i = 0 + \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
5. $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$
6. $\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$
7. $\sum_{i=0}^{n-1} x^i = \frac{1-x^n}{1-x}$
8. $\sum_{i=0}^{n-1} \frac{1}{2^i} = 2 - \frac{1}{2^{n-1}}$

Logs:

1. $a^{\log_b(c)} = c^{\log_b(a)}$
2. $\log_b(a) = \dfrac{\log_d(a)}{\log_d(b)}$
3. $\log_b(b) = 1$
4. $\log_b(1) = 0$
5. $b^{\log_b(n)} = n$
6. $\log_b(n \cdot m) = \log_b(n) + \log_b(m)$
7. $\log_b(\frac{n}{m}) = \log_b(n) - \log_b(m)$
8. $\log_b(n^k) = k \cdot \log_b(n)$

This is a blank page! Enjoy!