

CSE 373 A 19Wi: Midterm February 15 2019

Name _____

Student ID# _____

Instructions

- This test is closed book, closed calculators, closed electronics.
- You are permitted a single sheet of 8.5" x 11" notes
- Do not start the exam until told to do so. You will receive a 10 point deduction if you work before the exam starts or keep working after the instructor calls for papers
- Please write your answers neatly in the provided space
- Be sure to **leave some room in the margins** as we will be scanning your answers and it's easy to miss markings too close to the edge of the paper
- Please note we will only be scanning the front of each page, so please **do not put any work you wish to be graded on the back of a page**
- If you have a question, please raise your hand to ask the course staff for clarification
- Some **mathematical definitions have been provided for you on the last page of this packet**, feel free to rip it off for easy reference

Advice

- **Write down assumptions, thoughts and intermediate steps so you can get partial credit.** Clearly circle your final answer
- The questions are not necessarily in order of difficulty, skip around

1. Trees

(A) Insert the following sequence of values into an empty **AVL tree** in the given order.

4, 20, 32, 17, 101, 74, 1, 12, 7

Draw your **final tree** in the following figure (*Figure 1*), you will not need to fill in all nodes. You may use the space below *Figure 1* to draw your intermediary trees.

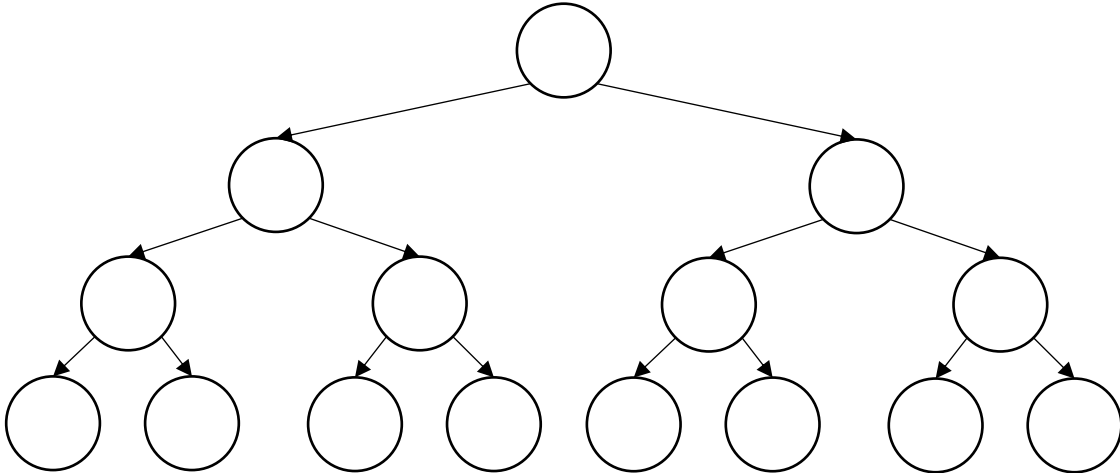


Figure 1: Fill in this tree with your **final** answer. Leave unused nodes empty.

(B) Given the Binary Search Tree in *Figure 2* imagine you are asked to delete the overallRoot, 74. Circle all the nodes in the tree that could take its place without breaking the AVL invariants or moving more than that single node.

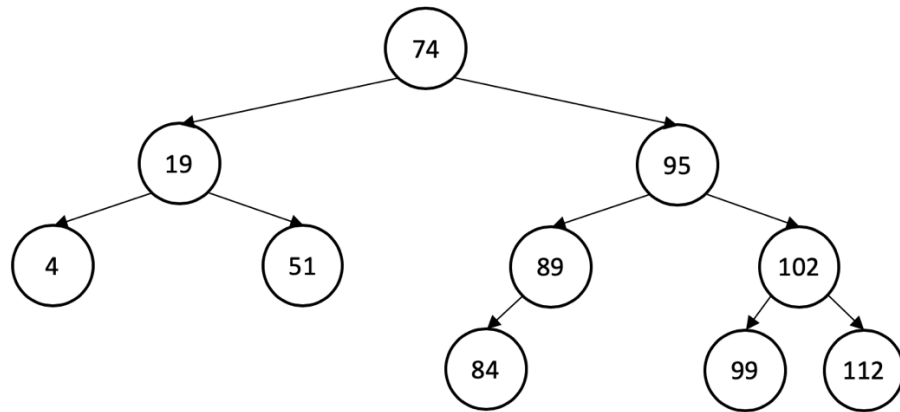


Figure 2

2. Code Modeling

(A) Give the code model $f(n)$ for the worst-case runtime of the `m1` method in terms of the number of Strings stored in the `ArrayList` `words`. You may simplify constants to stand in variables such as C_1 or C_2 (you do not need to count the exact number of operations).

- Assume all words have the same length, m .
- Assume `ArrayList` is Java's implementation of a `List` using an `Array` as underlying storage.
- Assume `ArrayDictionary` is implemented just as you did in Homework 2.

```
1 public static ArrayDictionary<String, int[]> m1 (ArrayList<String> words) {
2     ArrayDictionary<String, int[]> letCts = new ArrayDictionary<String, int[]>();
3     for (String w : words) {
4         int[] count = new int[26];
5         for (int i = 0; i < w.length(); i++) {
6             count[word.charAt(i) - 'a']++;
7         }
8         letCts.put(w, count);
9     }
10    return letCts;
11 }
```

- Worst case run-time of line #8 in terms of n
- Worst case run-time of loop between lines #5 and #7 in terms of n and m
- Overall worst case run-time of `m1` in terms of n (assume m is constant)
- Simplified tight big O of `m1`

(B) Give the code model $f(n)$ for the worst-case runtime of the `m2` method in terms of the number of Strings stored in the `ArrayList words`. You may simplify constants to stand in variables such as C_1 or C_2 (you do not need to count the exact number of operations).

- Assume all words have the same length, m .
- Assume `ArrayList` is Java's implementation of a List using an Array as underlying storage.
- Assume `ArrayDictionary` is implemented just as you did in Homework 2.
- Assume the `AVLMap` implements the `IDictionary` interface with an AVL tree as underlying storage.
- Assume `ChainedHashSet` is implemented just as you did in Homework 3 and will always resize at a given load factor λ .
- Assume `equals` methods compare each element in the given structures once.

```

1 public static AVLMap<String, ChainedHashSet<String>> m3 (
    ArrayList<String> words, ArrayDictionary<String, int[]> letterCounts) {
2     AVLMap<String, ChainedHashSet<String>> anagrams =
        new AVLMap<String, ChainedHashSet<String>>();
3     for (String word : words) {
4         ChainedHashSet<String> myAnagrams = new ChainedHashSet<String>();
5         int[] myLetterCounts = letterCounts.get(word);
6         for (String otherWord : words) {
7             if (!otherWord.equals(word)) {
8                 int[] otherLetterCounts = letterCounts.get(otherWord);
9                 if (Arrays.equals(myLetterCounts, otherLetterCounts)) {
10                    myAnagrams.add(otherWord);
11                }
12            }
13        }
14        anagrams.put(word, myAnagrams);
15    }
16    return anagrams;
17 }

```

- (i) Worst case run-time of line #5 in terms of n
- (ii) Worst case run-time of line #10 when $n < \lambda$ in terms of n and λ
- (iii) Worst case run-time of line #10 in terms of n
- (iv) Worst case run-time of line #14 in terms of n
- (v) Worst case run-time of loop between lines #6 and #13 in terms of n (assume m is constant)
- (vi) Overall worst case run-time of `m3` in terms of n
- (vii) Simplified tight big O of `m3`

(C) Consider the following method `m3` as well as your work in parts A and B. Give the code model $f(n)$ for the worst-case runtime of the `m3` method in terms of the number of Strings stored in the `ArrayList words`. You may simplify constants to stand in variables such as C_1 or C_2 (you do not need to count the exact number of operations).

- Assume `ArrayList` is Java's implementation of a `List` using an `Array` as underlying storage.
- Assume the `AVLMap` implements the `IDictionary` interface with an AVL tree as underlying storage.

```
1 public static ChainedHashSet<String> m3 (ArrayList<String> words){
2     ArrayDictionary<String, int[]> letterCounts = m1(words);
3     AVLMap<String, ChainedHashSet<String>> anagrams = m2(words, letterCounts);
4     String firstWord = words.get(0);
5     return anagrams.get(firstWord);
6 }
```

(i) Worst case run-time of line #5 in terms of n

(ii) Simplified tight big O of `m3`

(D) Give the recurrence $T(n)$ for the runtime of the following method `mystery`. You may simplify all constants to stand in variables such as C_1 or C_2 (you do not need to attempt to count the exact number of operations). **YOU DO NOT NEED TO SOLVE** this recurrence, just give the base case and a recurrence case.

```
public int mystery(int n) {
    if (n < 10000) {
        int result = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < i; j++) {
                result++;
            }
        }
        return result;
    } else {
        return 1 + mystery(n-1) + mystery(n-2);
    }
}
```


(2E alternative approach) Tree Method

$$T(n) = \begin{cases} 8 & \text{when } n \leq 1 \\ 4T\left(\frac{n}{2}\right) + n^2 & \text{otherwise} \end{cases}$$

- I. How many nodes are there on level i (assume the root is at level 0)?

- II. What is the size of input, n , on level i ?

- III. What is the total work done on the recursive levels in terms of n and i ?

- IV. How much work is done in the base case in terms of n ?

- V. Final closed form of $T(n)$

3. Hashing

For questions 3A and 3B imagine you are working with the following implementation of a hash table that stores integers greater than 0.

```
public class KaseyHash {
    private int[] data;
    private int size;

    public KaseyHash() {
        this.data = int[10];
        this.size = 0;
    }
    ...
}
```

(A) Below is one possible implementation of the `put(value)` method for the `KaseyHash` class. Based on the given code, fill out the table in *Figure 3* with the final state of `data` after calling `put` on each of the ints listed below in the given order. (note that the data will initially store all 0s)

```
public static void put(int value) {
    int naturalHash = value % data.length;
    int hashIndex = naturalHash;
    int i = 0;
    while (data[hashIndex] != 0) {
        i++;
        hashIndex = (naturalHash + i) % data.length;
    }
    data[hashIndex] = value;
}
```

202, 36, 12, 68, 126, 76, 88

0	1	2	3	4	5	6	7	8	9

Figure 3

(B) Below is one possible implementation of the `findPos(value)` method for the `KaseyHash` class. Based on the given code, and the state of data showing in *Figure 4*, how many probes would it take to locate each of the following values? A probe is counted as each time you investigate an index of `data`. Indicate your answers in the table in *Figure 5*.

```
public static int findPos(int value) {
    int naturalHash = value % data.length;
    int hashIndex = naturalHash;
    int i = 0;
    while (data[hashIndex] != 0 && data[hashIndex] != value) {
        i++;
        hashIndex = (naturalHash + i * i) % data.length;
    }
    return hashIndex;
}
```

0	1	2	3	4	5	6	7	8	9
56	103	72	203	33		156	83		

Figure 4

Value	203	33	156	72	102	83	56
# of probes	1						

Figure 5

4. Heaps

(A) Insert the following sequence of values in the given order, one at a time, into an empty min heap. Fill in *Figure 6* with your final answer, you will not need to fill in all nodes. You may use the space below *Figure 6* to draw your intermediary trees.

13, 42, 11, 35, 3, 22, 8, 9

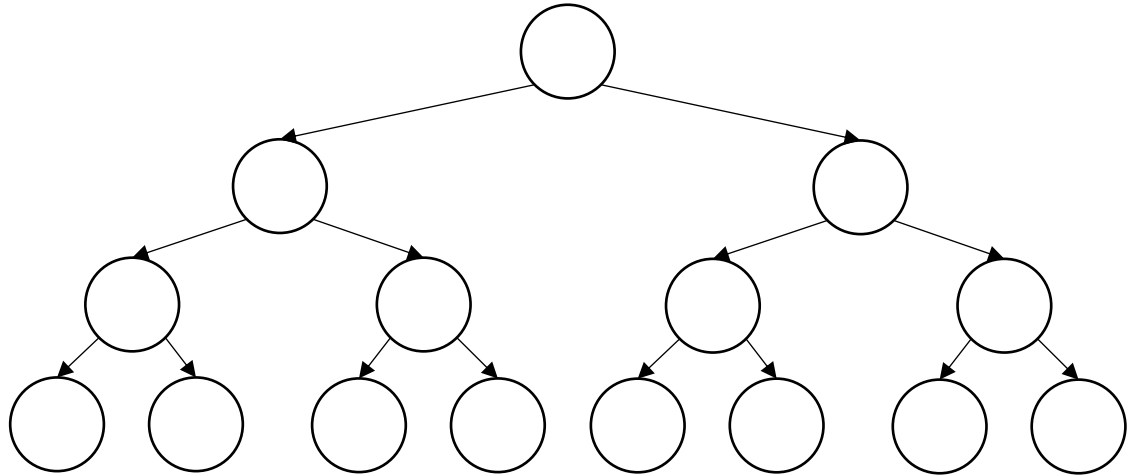


Figure 6: Fill in this tree with your **final** answer. Leave unused nodes empty.

(B) Given the array in *Figure 7* representing the current state of a min heap, fill in the empty array in *Figure 8* with the state of the heap after performing a single `removeMin()`. You may use the space below *Figure 8* to show your work.

0	1	2	3	4	5	6	7	8	9
8	15	13	32	24	41	29	54	82	

Figure 7

0	1	2	3	4	5	6	7	8	9

Figure 8

5. Asymptotic Analysis

(A) For each of the following functions fill in Figure 9 with the simplified tight O bound and whether the statement is True or False

Function	Tight O	Relationship Statement	Circle
$a(n) = 4n + 10$		$a(n)$ is in $O(n^2)$	True or False
$b(n) = 2n + n^2$		$b(n)$ is in $\Omega(n)$	True or False
$c(n) = \frac{1}{2}n + 2n^2 + 2^n$		$c(n)$ is in $O(n^2)$	True or False
$d(n) = \log_2(5n)$		$d(n)$ is in $\Omega(n)$	True or False
$e(n) = \log_2(n^3)$		$e(n)$ is in $\theta(\log n)$	True or False
$f(n) = \sum_{i=0}^{n-1} i$		$f(n)$ is in $O(n)$	True or False

(B) Demonstrate that $f(n)$ is dominated by $g(n)$ by finding a c and n_0 . You must show your work to receive any credit.

$$f(n) = 4n^2 - 2n + 9 \quad g(n) = n^2$$

6. Extra Credit (1 point)

Draw an artistic rendering of your current feelings about data structures and algorithms

(extra work space if necessary, will be scanned)