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](image-url)
2. Code Modeling

(A) Give the code model $f(n)$ for the worst-case runtime of the $m_1$ method in terms of the number of Strings stored in the ArrayList $\text{words}$. 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 $\text{ArrayList}$ is Java’s implementation of a List using an Array as underlying storage.
- Assume $\text{ArrayDictionary}$ is implemented just as you did in Homework 2.

```java
public static ArrayDictionary<String, int[]> m1(ArrayList<String> words) {
    ArrayDictionary<String, int[]> letCts = new ArrayDictionary<String, int[]>();
    for (String w : words) {
        int[] count = new int[26];
        for (int i = 0; i < w.length(); i++) {
            count[w.charAt(i) - 'a']++;
        }
        letCts.put(w, count);
    }
    return letCts;
}
```

(i) Worst case run-time of line #8 in terms of $n$

(ii) Worst case run-time of loop between lines #5 and #7 in terms of $n$ and $m$

(iii) Overall worst case run-time of $m_1$ in terms of $n$ (assume $m$ is constant)

(iv) Simplified tight big O of $m_1$
(B) Give the code model \( f(n) \) for the worst-case runtime of the \( m^2 \) method in terms of the number of Strings stored in the ArrayList \( \text{words} \). 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 \( \lambda \).
- 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 \( \lambda \).

(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 \( m^3 \) in terms of \( n \)

(vii) Simplified tight big \( O \) of \( m^3 \)
(C) Consider the following method $m_3$ as well as your work in parts A and B. Give the code model $f(n)$ for the worst-case runtime of the $m_3$ 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.

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

(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.

```java
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);
    }
}
```
Given the following recurrence $T(n)$ find the exact closed form. You need only to reduce it down so that it no longer includes $T(n)$ or a summation. You may use either unrolling or tree method, please select your method and answer only those questions corresponding to your chosen method. If using unrolling, answer I-V on this page, 7, if using tree method answer I-VI on the following page, 8.

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

Unrolling

I. What are the first three levels of the recurrence unrolled?

II. Give an expression for the $i$th level of unrolling in terms of $T()$, $n$ and $i$

III. What is the last level of unrolling in terms of $n$?

IV. Give an expression for $T(n)$ without any recursion (no remaining $T(n)$ terms)

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

```java
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)

```java
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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3
(B) Below is one possible implementation of the `findPos` 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.

```java
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;
}
```

![Table: Figure 4](image)

<table>
<thead>
<tr>
<th>Value</th>
<th>203</th>
<th>33</th>
<th>156</th>
<th>72</th>
<th>102</th>
<th>83</th>
<th>56</th>
</tr>
</thead>
<tbody>
<tr>
<td># of probes</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Table: Figure 5](image)
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()` operation. You may use the space below Figure 8 to show your work.

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>15</td>
<td>13</td>
<td>32</td>
<td>24</td>
<td>41</td>
<td>29</td>
<td>54</td>
<td>82</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 7**

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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.

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

**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)