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.

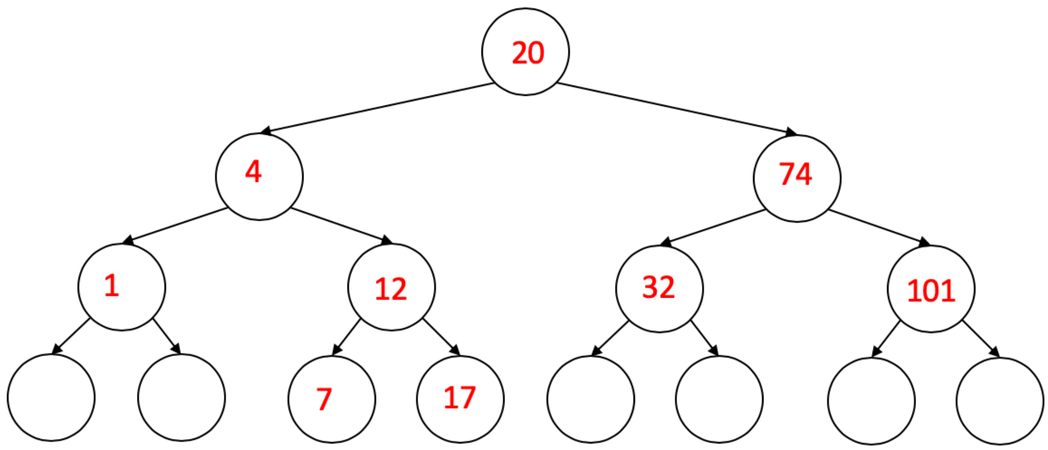

Figure 1: Fill in this tree with your **final** answer. Leave unused nodes empty.

(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.

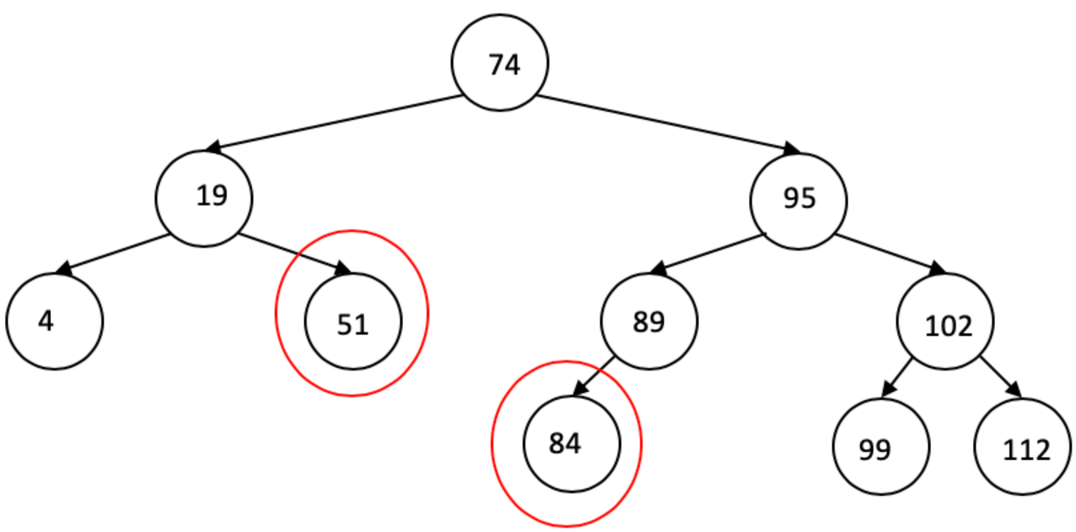

Figure 2

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 }
```

[5 points]

(i) Worst case run-time of line #8

n accept any $\Theta(n)$ expression

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

$c_1 m + c_2$ accept any $\Theta(m)$ expression

(iii) Overall worst case run-time of `m1` in terms of n (assume m is constant)

`C1 + n(C2 + n) accept any` $\Theta(n^2 m^\alpha)$ for all α

(iv) Simplified tight big O of `m1`

$O(n^2)$

(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 word : 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 n

(ii) Worst case run-time of line #10 when $n < \lambda$ in terms of n and λ $1 + \lambda$ TAs say Strike this ☹

(iii) Worst case run-time of line #10 in terms of n n

(iv) Worst case run-time of line #14 in terms of n $\log n$

(v) Worst case run-time of loop between lines #6 and #13 in terms of n (assume m is constant)
n(C1 + n + n) i.e. $\Theta(n^2)$

(vi) Overall worst case run-time of `m3`
`C1 + n(C2 + n + n(C3 + 2n) + n)  i.e.` $\Theta(n^3)$

(vii) Simplified tight big O of `m3`
O(n^3)

(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 logn [1 point]

(ii) Simplified tight big O of `m3` O(n^3) [1 point for writing dominant term of c(i), a(iv), and b(vii)]

(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). **<u>YOU DO NOT NEED TO SOLVE</u>** 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);
   }
}
```

$$T(n) = \begin{cases} C_1 + \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} C_2 & n < 10000 \\ C_3 + T(n-1) + T(n-2) & otherwise \end{cases}$$

(E) 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 method**. If using unrolling, answer I-V on this page, 7, if using tree method answer I-VI on the following page, 8.

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

Unrolling [20 points]

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

$$i{:}\,1 \quad T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

$$i{:}\,2 \quad = 4\left(4T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^2\right) + n^2$$

$$i{:}\,3 \quad = 4\left(4\left(4T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)^2\right) + \left(\frac{n}{2}\right)^2\right) + n^2$$

$$i{:}\,4 \quad = 4\left(4\left(4\left(4T\left(\frac{n}{16}\right) + \left(\frac{n}{8}\right)^2\right) + \left(\frac{n}{4}\right)^2\right) + \left(\frac{n}{2}\right)^2\right) + n^2$$

$$i{:}\,2 = 4^2T\left(\frac{n}{2^2}\right) + 4\left(\frac{n^2}{2^2}\right) + n^2$$

$$i{:}\,3\ 4^2\left(4T\left(\frac{n}{2^3}\right) + \left(\frac{n}{2^2}\right)^2\right) + \frac{4}{2^2}n^2 + n^2 = 4^3T\left(\frac{n}{2^3}\right) + \frac{4^2}{4^2}n^2 + n^2 + n^2$$

$$i{:}\,4\ \ 4^3\left(4T\left(\frac{n}{2^4}\right) + \left(\frac{n}{2^3}\right)^2\right) + n^2 + n^2 + n^2 = 4^4T\left(\frac{n}{2^4}\right) + \frac{4^3}{4^3}n^2 + n^2 + n^2 + n^2$$

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

$$T\left(\frac{n}{2^i}\right) = 4^iT\left(\frac{n}{2^i}\right) + \sum_{j=0}^{i-1} 4^j\left(\frac{n}{2^j}\right)^2 = 4^iT\left(\frac{n}{2^i}\right) + i \cdot n^2$$

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

$$= T\left(\frac{n}{2^i}\right) = T(1) \rightarrow \frac{n}{2^i} = 1 \rightarrow n = 2^i \rightarrow i = \log_2 n$$

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

$$T\left(\frac{n}{2^{\log_2 n}}\right) = 4^{\log_2 n}(8) + \sum_{i=1}^{\log_2 n} 4^{i-1}\left(\frac{n}{2^{i-1}}\right)^2$$

OR $4^{\log_2 n}(8) + n^2\log_2 n$

V. Final closed form of T(n)

$$T(n) = 8n^2 + \sum_{i=1}^{\log_2 n} 4^{i-1}\left(\frac{n^2}{(2^{i-1})\text{^}2}\right) = 8n^2 + \sum_{i=1}^{\log_2 n} n^2 = 8n^2 + n^2\log_2 n$$

(2E alternative approach) Tree Method

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

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

$$4^i$$

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

$$\frac{n}{2^i}$$

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

$$\sum_{i=0}^{\log_2 n-1} 4^i \left(\frac{n}{2^i}\right)^2$$

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

$$8\left(4^{\log_2 n}\right) = 8n^2$$

IV. Final closed form of T(n)

$$T(n) = 8n^2 + \sum_{i=0}^{\log_2 n-1} 4^i \left(\frac{1}{4}\right)^i n^2$$
$$= 8n^2 + \sum_{i=0}^{\log_2 n-1} n^2$$
$$= 8n^2 + n^2 \log_2 n$$

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*
88	**0**	**202**	**12**	**0**	**0**	**36**	**126**	**68**	**76**

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*.

```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;
}
```

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

Figure 4

Value	156	72	203	33	83	56	103
# of probes	1	**1**	**1**	**2**	**3**	**3**	**redacted**

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

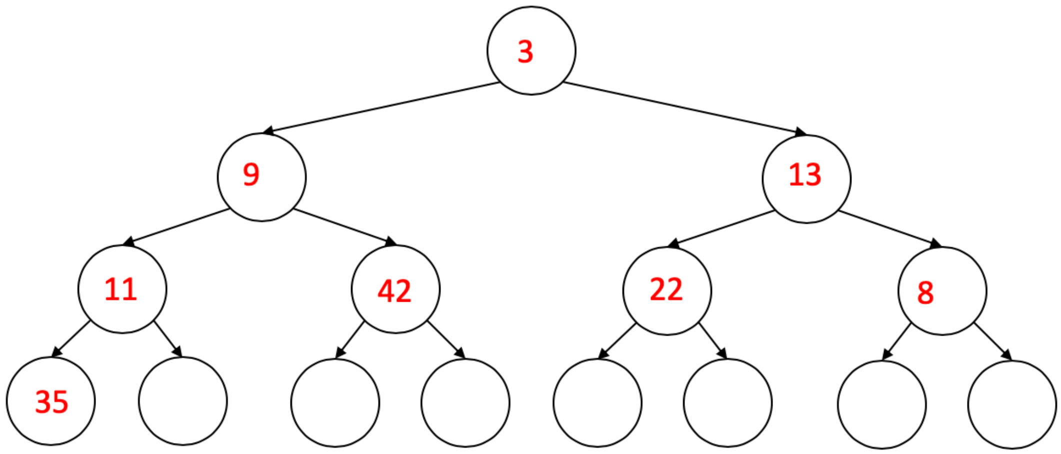

Figure 6: Fill in this tree with your **final** answer. Leave unused nodes empty.

(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*
13	**15**	**29**	**32**	**24**	**41**	**82**	**54**		

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

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

Two solutions are given here, but there are infinite possible solutions, if correct and complete work is shown.

$$4n^2 \le c \cdot n^2\ when\ c = 4\ for\ n \ge 1$$
$$-2n \le c \cdot n^2\ when\ c = 0\ for\ n \ge 1$$
$$9 \le c \cdot n^2\ when\ c = 9\ for\ n \ge 1$$
$$4n^2 - 2n + 9 \le 4n^2 + 0n^2 + 9n^2 = 13n^2 \text{ for n} \ge 1$$
$$4n^2 - 2n + 9 \le c \cdot n^2\ when\ c = 13\ and\ n_0 = 1$$

$$4n^2 \le c \cdot n^2\ when\ c = 4\ for\ n \ge 1$$
$$-2n \le c \cdot n^2\ when\ c = 1\ for\ n \ge 1$$
$$9 \le c \cdot n^2\ when\ c = 1\ for\ n \ge 3$$
$$4n^2 - 2n + 9 \le 4n^2 + n^2 + n^2 = 6n^2 \text{ for n} \ge 3$$
$$4n^2 - 2n + 9 \le c \cdot n^2\ when\ c = 6\ and\ n_0 = 3$$