Quiz 5 — Data Struct. & More (T. III/21–22)

Name: Pearploy Chaicharoensin

ID: 6381278

Directions:

- This exam is "paper-based." Answer all the questions either directly onto this PDF or in a separate file, which you'll then turn into a PDF for submission.
- Have the webcam on for the duration that you're working on the quiz.
- No consultation with other people is permitted. But feel free to use your notes, books, and the Internet. You are also allowed to write code and run it.
- You can chat with the instructors via the built-in chat.
- This quiz is worth a total of 40 points, but we'll grade out of 30. Anything above 30 is extra credit. You have 110 minutes. Good luck!

Problem 1: Quick Running Time Analysis (8 points)

1. [2 points] Order the following functions from small to large, assuming n is a large number.

   $\cancel{42n^2}$ $\cancel{n}$ $\cancel{2^n}$ $\cancel{2n^3}$ $n\cancel{\log n}$ $\cancel{n^{15}}$ $\cancel{\log n}$

   Your answer should look like below:

   $\log n$ < n < $n \log n$ < $42n^2$ < 2^n < $2n^3$ < n^{15}

2. [2 points x 3] For each of the following, write its running time in Θ notation. Just provide an answer.

(i)
```
int unknown1(int[] a) {
    int n = a.length;      Θ(1)
    int c = 0;   Θ(1)
    for (int k=n-1;k>0;k=k/2) {     → Θ(log n)×[Θ(n)+Θ(log n)]
                                      Θ(log n)Θ(n) = Θ(n log n)
        for (int t=0;t<n;t+=1) {
            if (a[t]>0) c++;        } Θ(n)
        }
        for (int t=n-1;t>0;t=t/2) {
            if (a[t]>a[t/2]) c++;   } Θ(log n)
        }
    }
    return c; → Θ(1)
}
```

(i) $\Theta(n \log n)$ times

I	t
1	0
2	1
3	2
⋮	
k	k−1

$k-1 = n-1$

$k = n$

$\Theta(n)$

I	t
1	$n-1$
2	$\frac{n-1}{2}$
3	$\frac{n-1}{4}$
⋮	⋮
k	$\frac{n-1}{2^{k-1}}$

$\frac{n-1}{2^{k-1}} = 1$

$n-1 = 2^{k-1}$

$\log(n-1) = k-1$

$k = \log(n-1)+1$

$k = \log(n)$

$\Theta(\log n)$

I	k
1	$n-1$
2	$\frac{n-1}{2}$
3	$\frac{n-1}{4}$
⋮	⋮
m	$\frac{n-1}{2^{m-1}}$

$\frac{n-1}{2^{m-1}} = 1$

$n-1 = 2^{m-1}$

$\log(n-1) = m-1$

$m = \log(n-1)+1$

$\Theta(\log n)$

(ii)
```
int unknown2(int[] xs) {
    if (xs.length == 1) return xs[0];  Θ(1)
    else {
        int[] ys = Arrays.copyOfRange(xs, 1, xs.length);  Θ(n-1) = Θ(n)
        return xs[0]+unknown2(ys);  Θ(1) + T(n-1)
    }
}
```

ii) $\Theta(n)$ times

(iii)
```
void unknown3(double[] a) {
    int m = Math.min(25, a.length);
    for (int i=0;i<a.length;i++) {  Θ(n)×Θ(n) = Θ(n²)
        for (int j=0;j<m;j++)
            a[i] *= a[j];           } Θ(n)
    }
}
```

iii) $\Theta(n^2)$ times

I	j
1	0
2	1
3	2
⋮	⋮
k	k−1

$k-1 = n-1$

$k = n$

$\Theta(n)$

I	i
1	0
2	1
⋮	⋮
k	k−1

$k-1 = n-1$

$k = n$

$\Theta(n)$

Problem 2: Running Time With Data Structures (8 points)

For each of the following functions, analyze the running time for what happens in the worst-case. Answer in the tightest possible Big-O or Θ. Optionally, explain your reasoning briefly. **Pay close attention to the choice of data structures used.**

(i) $O(n)$ times

```
int numUnique(int[] a) {
    int n = a.length;  O(1)
    Set<Integer> seen = new HashSet<>(); O(1)
    for (int elt: a) { O(n)
        seen.add(elt);O(1)
    }
    return seen.size(); O(1)
}
```

(ii) $O(n \log n)$ times

```
Map<Character, Integer> histogram(String st) {
    Map<Character, Integer> hist = new TreeMap<>();   O(1)
    for (int i=0;i<st.length();i++) { O(n)
        char ch = st.charAt(i); O(1)
        if (!hist.containsKey(ch)) { hist.put(ch, 0); } O(log n)
        hist.put(ch, hist.get(ch)+1); O(log n)
    }
    return hist; O(1)
}
```

(iii) $O(n)$ times

```
// an implementation of binary search on a (doubly) linked list
// the get function is similar to what you implemented in the assignment
Integer bsHelper(LinkedList<Integer> xs, int l, int r, int key) {  → 2T(n/2)+O(1) = O(n)
    if (l>=r) return null;  O(1)
    int m = (l+r)/2;  O(1)
    if (xs.get(m) == key) return m;  O(1)
    else if (xs.get(m) > key) return bsHelper(xs, l, m, key);  T(n/2)
    else return bsHelper(xs, m+1, r, key);  T(n/2)
}
Integer binarySearch(LinkedList<Integer> xs, int key) {
    return bsHelper(xs, 0, xs.size(), key);  → O(n)
}
```

(iv) $O(n)$ times

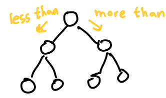
```
// reverse a LinkedList into an ArrayList
List<Integer> rev(LinkedList<Integer> xs) {
  List<Integer> reversed = new ArrayList<>(); O(1)
  for (int elt : xs) {  O(n)
     reversed.add(0, elt); // add elt at the beginning of reverse  O(1)
  }
  return reversed; O(1)
}
```

Problem 3: HashMap vs. TreeMap (6 points)

The `HashMap` class internally implements a bucketed hash table to store values associated with the keys. The `TreeMap` class internally keeps a (balanced) binary search tree ordered by keys. Reason about the following questions. They are somewhat open-ended, but your analysis should be grounded on facts that you know.

(i) (2 points) Why is accessing a specific key in `TreeMap` takes (much) longer than accessing a key in a `HashMap`? Explain briefly.

Since TreeMap needs to go through each element from the root of the tree to find the key that we want. Fortunately, it is sorted, so it takes only O(log n) times to drive in the tree by recursively go through left and right of the node.



(ii) (2 points) In what scenarios would the `TreeMap` be faster than—and hence preferred over— the `HashMap`?

When we need to find lowerKey, floorKey, ceilingKey, higherKey of a given node. TreeMap takes only O(log n) time, while HashMap takes O(n) from doing those.

Moreover, it is faster to get sorted items.

(iii) (2 points) The `TreeMap` is believed to use more memory than than `HashMap`. Argue in support of this or provide a counterargument.

TreeMap takes less memory than HashMap, since the elements in TreeMap are more well-order than HashMap.

Problem 4: Graph Representation Quickies (3 points = 0.5 points/blank)

We have seen a number of graph representations, including the following two options:

(1) an **adjacency map** represents a graph as a `HashMap` mapping each vertex to a `HashSet` of its neighbors

(2) an **edge list** represents a graph as an `ArrayList` of edges (an edge is a `Pair`)

Let $G = (V, E)$. You will complete the table below with the running time of **(i)** `G.deg(u)` for computing the degree of a vertex u in the graph G, **(ii)** `G.isAdj(u, v)` for determining whether there is an edge from u to v in G, and **(iii)** `G.nbrs(u)` for returning the set of the neighbors of u in G. Remember $n = |V|$ and $m = |E|$.

Operation	Adjacency Map	Edge List
`deg(u)`	$O(1)$	$O(m)$
`isAdj(u, v)`	$O(1)$	$O(m)$
`nbrs(u)`	$O(1)$	$O(m)$

Problem 5: Randomness (6 points)

A biased-pentahedron die has 5 faces numbered 1, 2, 3, 4 and 5. The die is rolled and the number on the face of the die, denoted by X, is recorded. The probability distribution of X is

i	1	2	3	4	5
$\mathbf{Pr}[X = i]$	0.3	0.2	0.1	a	b

(1) Because of the law of total probability, we know that $a + b = 0.4$. Given that $\mathbb{E}[X] = 2.9$, write down a second equation involving a and b.

$E[x] = \sum x P_r[x] = 2.9$

$= 1(0.3) + 2(0.2) + 3(0.1) + 4a + 5b = 2.9$

$0.3 + 0.4 + 0.3 + 4a + 5b = 2.9$

$4a + 5b = 1.9$

(2) Solve your two equations for a and b. (*Hint:* The final answer is simple.)

from (1) ← $4a + 5b = 1.9$ — ①

from the given ← $a + b = 0.4$ — ②

$a = 0.4 - b$ — ③

plug ③ in ①

$4(0.4 - b) + 5b = 1.9$

$1.6 - 4b + 5b = 1.9$

$b = 0.3$

plug $b = 0.3$ in ②

$a + 0.3 = 0.4$

$a = 0.1$

Thus $a = 0.1$ and $b = 0.3$

(3) Determine the value of $\mathbb{E}[X^2]$. Show your work.

$E[x^2] = \sum x^2 P_r[x]$

$= 1^2(0.3) + 2^2(0.2) + 3^2(0.1) + 4^2(0.1) + 5^2(0.3)$

$= 0.3 + 0.8 + 0.9 + 1.6 + 7.5$

$E[x^2] = 11.1$

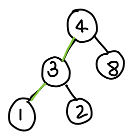
Problem 6: Binary Search Trees (4 points)

Remember that the height of a tree is the number of nodes on the longest path from the root to a leaf. A binary search tree contains 5 keys: $3, 1, 4, 2, 8$. 1 2 3 4 8

(i) What is the bare minimum height of such a BST? Draw a BST on these keys that achieves this height.

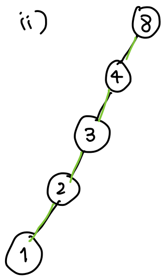
(ii) What is the maximum height of such a tree? Draw a BST on these keys that achieves this height.

i)



The minimum height is 2

ii)



The maximum height is 4

Problem 7: Binary Tree Equals (5 points)

In this problem, a binary tree is represented using the familiar `TreeNode`, as shown on the right, so every node stores a key in an attribute known as `key`. Two binary trees S and T are *equal* if (i) they have the same structure and (ii) each node in S and its corresponding node in T (i.e., the node at the same position) store the same string value (ignoring cases, i.e., `CaT` is the same as `CAt`).

```
class TreeNode {
    String key; // key

    // left and right children
    TreeNode left, right;
}
```

Your Task: Write a static method **`public static boolean`** `treeEqual(TreeNode S, TreeNode T)` that takes in the roots of `S` and `T`, respectively, and returns whether these two trees are equal. (*Hint:* Recursion. Keep in mind that `S` or `T`, or both, can be **null**.)

```
public static boolean treeEqual (TreeNode S, TreeNode T) {
    if (S.key == null && T.key != null) { return false; }
    else if (S.key != null && T.key == null) { return false; }
    else if ( S.key.equals(T.key)) {
        treeEqual (S.left , T.left);
        treeEqual (S.right, T.right);
        return true;
    }
    return false;
}
```