QUEENS COLLEGE Department of Computer Science
CSCI 313 Final Exam Fall 2019 12.17.19

Instructor: Alex Ryba

08.30am – 10.30am, Tuesday, December 17, 2019

Problem 1 For the following functions give a *simplified* Θ estimate of the run time in terms of `N`

(a)

```
public static int f1(int N) {
   int x = 0;
   for (int i = 0; i < N * N; i++)
      for (int j = N; j < N * N; j++)
         x++;
   return x;
}
```

Answer: $\Theta(N^4)$

(b)

```
public static int f2(int N) {
   int x = 0;
   for (int i = 0; i < Math.sqrt(N); i++)
      for (int j = i; j > 1; j /= 2)
         x++;
   return x;
}
```

Answer: $\Theta(\sqrt{N} \log N)$

For the next three parts, consider the following function:

```
public static int f3(int N) {
   if (N <= 1) return 1;
   PriorityQueue<Double> q = new PriorityQueue<>();
   for (int i = 0; i < N; i++) q.add(Math.random());
   for (int i = 1; i <= 4; i++) q.add(f3(N/2));
   return q.removeMin();
}
```

(c) Give an estimate for the run time of `f3` assuming that the PriorityQueue has a heap implementation.

Answer: $\Theta(N^2)$

(d) Give an estimate for the run time of `f3` assuming that the PriorityQueue is implemented to use a sorted ArrayList instead of a heap.

Answer: $\Theta(N^2 \log N)$

(e) Give an estimate for the run time of `f3` assuming that the PriorityQueue is implemented to use an unsorted ArrayList instead of a heap.

Answer: $\Theta(N^2)$

Problem 2 Suppose that the `class Point` has two instance variables `int x` and `int y`. It has a hash function that returns the last digit of `x`. We write (a, b) for the Point whose instance variable `x` has value a and instance variable `y` has value b.

A hash table that uses open addressing and has capacity 8 stores Point data. The table currently stores $(0, 0)$ at index 1, $(22, 0)$ at index 2, $(12, 1)$ at index 3, $(25, 3)$ at index 5 and all other entries are either `null` or the `ghost`.

For each of the following parts give an answer and a brief reason to justify your answer.

(a) What is stored at index 0?

Answer: The ghost.

Since the table contains elements like $(0, 0)$ that hashes to 0, that entry must have already been occupied when it was inserted. The entry can never again be `null`, but since it does not contain Point data it must contain the ghost.

(b) If the table applies linear probing, at which index would $(19, 2)$ be added to the table?

Answer: Index 4.

The hash code is 9 which reduces to 1 modulo 8. Linear probing will attempt to place the entry at the sequence of indices $1, 2, 3, 4, 5, \ldots$. The first available space has index 4.

(c) If the table applies quadratic probing, at which index would $(11, 2)$ be added to the table? (Assume that the data in the table is as originally described.)

Answer: This point cannot be placed.

The hash code is 1 and quadratic probing would consider the sequence of indices $1, 2, 5, 2, 1, 2, 5, 2, \ldots$. All of these indices are already taken.

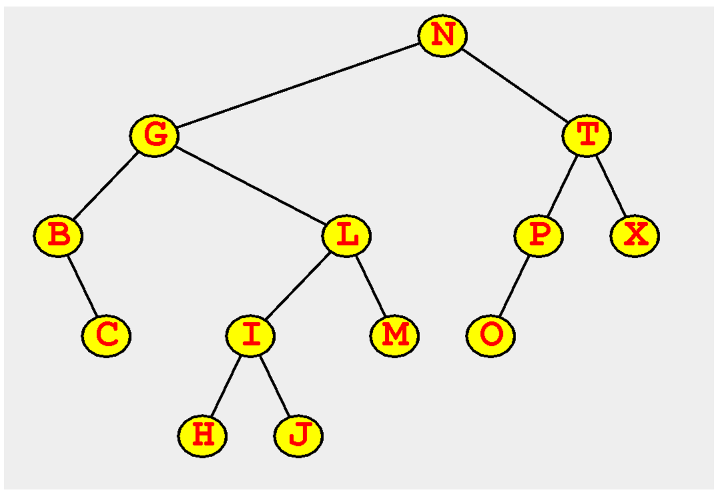
(d) Would it be better to use a hash function that adds the last digits of x and y?

Answer: This hash function is a little better because it makes use of more of the information stored in a Point.

(e) Would it be better to use a hash function that adds a randomly selected digit of x to a randomly selected digit of y?

Answer: This cannot be used as a hash function since it does not always give the same result when applied to a given Point.

Problem 3 Each part of this question asks about the following binary tree. Even if it is changed by operations in a part, all later parts refer to the original tree (as shown in the diagram).

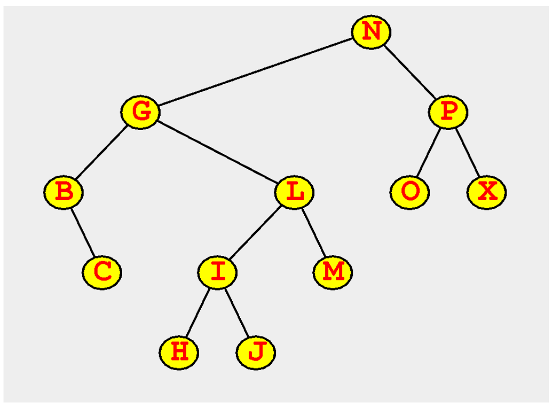


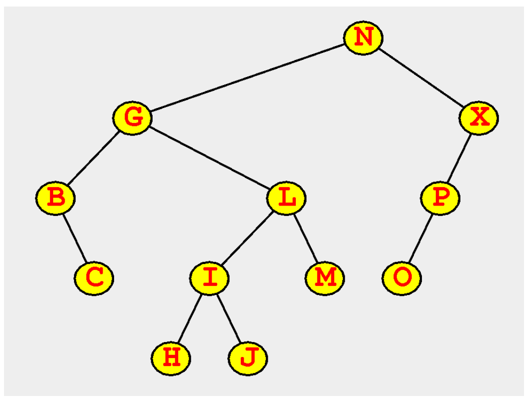
(a) What is the postorder traversal of the tree? **Answer:** `C B H J I M L G O P X T N`

(b) Which nodes have height 2? **Answer:** `L T`

(c) If the tree is considered as a Binary Search Tree, show the 2 possibilities that could result when T is removed.

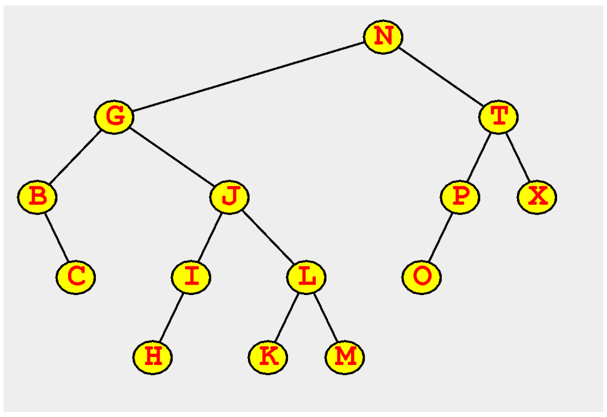
Answer:





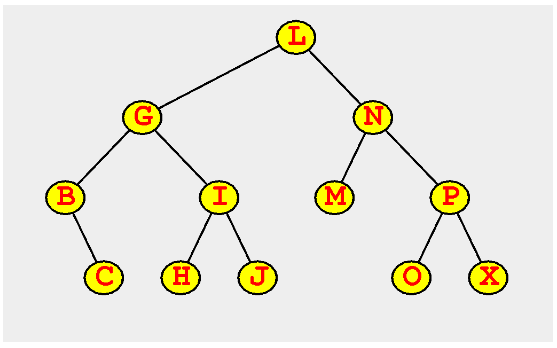
(d) If the tree is considered as an AVL Tree, show how the tree would be modified if K is added.

Answer:



(e) If the tree is considered as an AVL Tree, show how the tree would be modified if T is removed.

Answer:



Problem 4 Suppose that `class Graph` is implemented to store an (undirected) graph. Its only instance variable is `ArrayList<Edge> edges` (in other words, it uses an edge list implementation).

The classes `Egde` and `Vertex` are as follows.

```
public class Vertex {
   public Vertex() {
   }
}
```

```
public class Edge {
   public Vertex a, b;
   public Edge(Vertex a, Vertex b) {
      this.a = a; this.b = b;
   }
}
```

Write a complete implementation of a `Graph` method to find the degree of a given vertex in the graph. It should have title line:

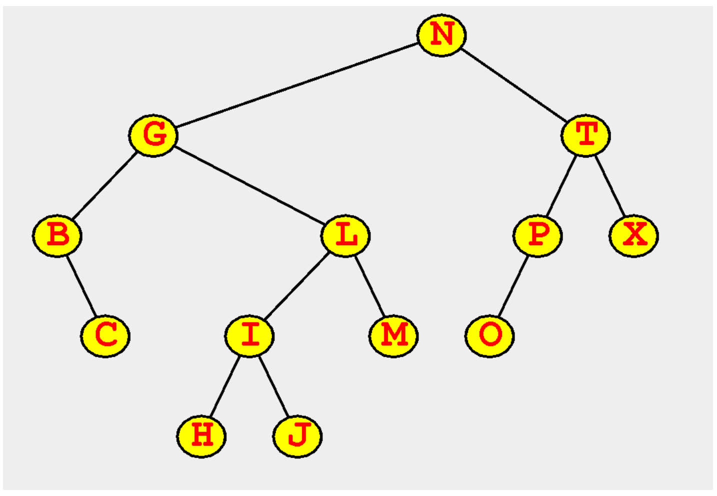
`int degree(Vertex v)`

Answer:

```
int degree(Vertex v) {
   int count = 0;
   for (Edge e:edges)
      if (e.a == v || e.b == v) count++;
   return count;
}
```

Problem 5 This question gives you the code for a recursive method called `rightNodes`. Five small pieces of the code have been omitted and replaced by PART (a), PART (b) and so on. You should answer each part by supplying the omitted code. (In each case it is at most one line of code.)

The method `rightNodes` returns the *"right nodes"* of a binary tree as an `ArrayList` that shows the rightmost entry in each level of the tree. For example, if the tree is as follows:



the method `rightNodes` returns `N,T,X,O,J`.

The method operates as follows. By recursion, it finds array lists that give the rightNodes of the right and left children of the root. These will store `T,X,O` and `G,L,M,J` . It uses the first of these lists as an initial part of the answer and adds extra elements.

First, the data at the root is added to the beginning of the answer (in our example this gives us `N, T,X,O`). Then, any extra entries at the end of the second array list are added to the end of the answer (in our example, we get `N, T,X,O, J`).

The method applies to a standard `BinaryTree` whose nodes contain `String` data. It returns an array list of `String`.

```
public ArrayList<String> rightNodes() {
   return rightNodesStartingAt(root);  // call a recursive helper method
}

private ArrayList<String> rightNodesStartingAt(BinaryNode n) {
   if (n == null)  PART (a)  ; // base case for empty tree
   ArrayList<String> rightSideOfRight = rightNodesStartingAt(  PART (b)  );
   ArrayList<String> rightSideOfLeft =  PART (c)  ;
   ArrayList<String> answer = rightSideOfRight;
   answer.add(0,  PART (d)  );
   while (rightSideOfLeft.size()  PART (e)  answer.size())
      answer.add(rightSideOfLeft.get(answer.size() - 1));
   return answer;
}
```

(a) Give a replacement for PART (a). **Answer:** `return new ArrayList<String>()`

(b) Give a replacement for PART (b). **Answer:** `n.getRight()`

(c) Give a replacement for PART (c). **Answer:** `rightNodesStartingAt(n.getLeft())`

(d) Give a replacement for PART (d). **Answer:** `n.getData()`

(e) Give a replacement for PART (e). **Answer:** `>=`