CS140 Final Exam - December 10, 2019

Please answer all questions.
Please put your answers on the answer sheets provided.
Please do not put answers on the exam.

Question 1: 27 Points

Please answer the following multiple choice / true-false questions. The potential answers are on the answer sheet. All running times should be worst-case.

Part A: T/F: When I insert a value into an AVL tree, I will perform a maximum of two rotations.
Part B: T/F: $a(n) = O(b(n))$ if there are positive constants c and n_0 such that for all $n > n_0$, $ca(n) \geq b(n)$.
Part C: Suppose I have a deque with n elements and an iterator to one of its elements. What is the running time of deleting the element.
Part D: Suppose I have an AVL tree with n elements. What is the running time of deleting an element from it.
Part E: Suppose I have a binary search tree with n elements. What is the running time to create a sorted vector from the binary search tree?
Part F: What is the big-O expression for $15n^2 + 400n + 5 \log(n)$?
Part G: Suppose I have a list with n elements and an iterator to one of its elements. What is the running time of deleting the element.
Part H: Suppose I have a binary search tree. What kind of traversal do we use to create a sorted vector from the binary search tree.
Part I: T/F: If **b** is a map, and I put the line `"a = b[i]"` in my method, I can declare the method as **const**.
Part J: T/F: $a(n) = O(b(n))$ if there are positive constants c and n_0 such that for all $n > n_0$, $cb(n) \geq a(n)$.
Part K: Suppose I have a binary search tree with n elements. What is the running time of deleting an element from it.
Part L: What is the big-O expression for $55n + 4n \log(n) + 20 \log(n)$?
Part M: What is the big-O running time of inserting a value between 0 and n into a multimap with m elements that are numbers between 0 and n.
Part N: T/F: If **b** is a map, and I put the line `"a = b.find(i)->second"` in my method, I can declare the method as **const**.
Part O: Suppose I have a binary search tree class. I implement **Clear()** without declaring any local variables. What kind of traversal do I have to use?
Part P: Suppose I have a tree that represents an arithmetic expression. What kind of traversal do we use to evaluate/calculate that expression.
Part Q: T/F: When I delete a value from an AVL tree, I will perform a maximum of two rotations.
Part R: Suppose I have a vector with n elements and an iterator to one of its elements. What is the running time of deleting the element.

Question 2: 20 Points

We have a class called **A**, which has a method **B()**. For each implementation of **B()** on the next page, please tell me its running time in big O notation. The big O should be in terms of the variables n and m. Please assume the following:

- **x** is a member variable that is a vector of ints, whose size is n, and whose values are beween 0 and $m-1$.
- **y** is a member variable that is a vector of ints, whose size is m, and whose values are between 0 and $n-1$.
- n is larger than m.
- **swap()** is a procedure in the algorithms library that is $O(1)$.
- The method **F(i)** has a running time that is $O(i)$.

Question 2, Continued

```
void A::B() // Implementation 1
{
  size_t i, j;

  for (i = 0; i < x.size(); i++) {
    for (j = 0; j < y.size(); j++) {
      swap(x[i], x[y[j]]);
    }
  }
}
```

```
void A::B() // Implementation 2
{
  size_t i;
  deque <int> d;

  for (i = 0; i < x.size(); i++) {
    d.push_front(x[i]);
  }
}
```

```
void A::B() // Implementation 3
{
  size_t i;
  set <int> s;

  for (i = 0; i < x.size(); i++) {
    s.insert(x[i]);
  }
}
```

```
void A::B() // Implementation 4
            // Don't burn time on this one.
            // If you can't get it quickly, make a guess.
{
  size_t i, j;

  for (i = 0; i < x.size(); i++) {
    for (j = 0; j < x[i]; j++) F(j);
  }
}
```

```
void A::B() // Implementation 5
{
  size_t i, j;

  for (i = 0; i < x.size(); i++) {
    for (j = 1; j < y.size(); j *= 2) {
      swap(x[i], x[y[i]]);
    }
  }
}
```

```
void A::B() // Implementation 6
{
  size_t i;
  multiset <int> s;

  for (i = 0; i < x.size(); i++) {
    s.insert(x[i]);
  }
}
```

```
void A::B() // Implementation 7
{
  size_t i;
  set <int> s;

  for (i = 0; i < y.size(); i++) {
    s.insert(y[i]);
  }
}
```

```
void A::B() // Implementation 8
{
  size_t i;

  for (i = 0; i < x.size(); i++) swap(y[0], y[x[i]]);
  for (i = 0; i < y.size(); i++) swap(x[0], x[y[i]]);
}
```

```
void A::B() // Implementation 9
{
  size_t i;
  vector <int> v;

  for (i = 0; i < x.size(); i++) {
    v.insert(v.begin(), x[i]);
  }
}
```

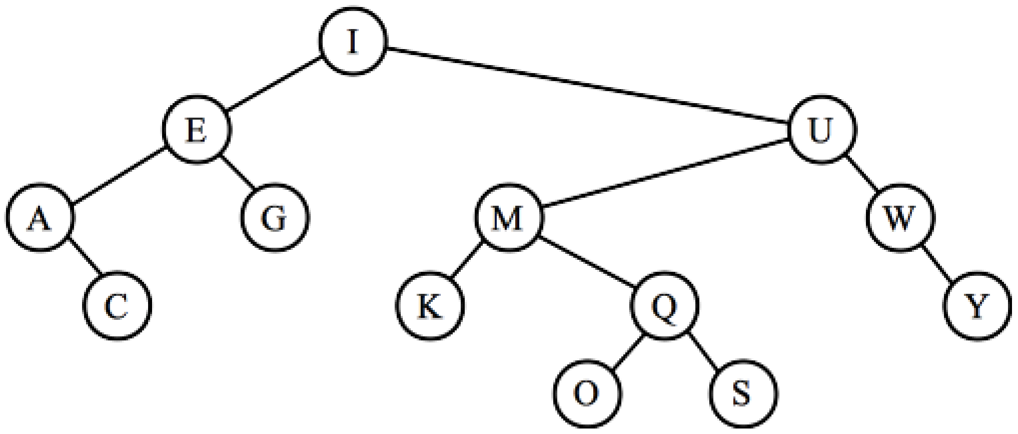
```
void A::B() // Implementation 10
            // Don't burn time on this one.
            // If you can't get it quickly, make a guess.
{
  size_t i;
  set <int> s;

  for (i = 0; i < x.size(); i++) {
    s.insert(y[x[i]]);
  }
}
```

There are no segmentation violations in this code. I'm not trying to trick you with any of this code.

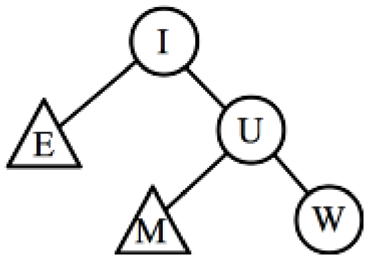
CS140 Final Exam - December 10, 2019

Question 3: 20 Points

All of these questions pertain to the tree below, which is both a binary search tree and an AVL tree.



When you answer the questions, if a subtree rooted at node *X* is unchanged, then simply represent the subtree with a triangle labeled with *X*. For example, suppose I ask, "Let's consider the tree to be a binary search tree, but not an AVL tree. Draw the tree that results when you delete **Y**". Your answer should be:



This is to make your life easier writing answers, and my life easier when I grade. *I will take off points if you draw out whole trees instead of using triangles as above*. Also, each of these questions is independent -- the questions do not depend on each other. Please answer the following:

Part A: If I insert **Z**, the tree is no longer an AVL tree. Which is the lowest imbalanced node?
Part B: If I insert **R**, the tree is no longer an AVL tree. Which is the lowest imbalanced node?
Part C: If I delete **G**, the tree is no longer an AVL tree. Which is the lowest imbalanced node?
Part D: If I delete **C**, the tree is no longer an AVL tree. Which is the lowest imbalanced node?
Part E: If I delete **I**, the tree is no longer an AVL tree. Which is the lowest imbalanced node?
Part F: Let's consider the tree to be a binary search tree, but not an AVL tree. Draw the tree after **L** is inserted.
Part G: Let's consider the tree to be a binary search tree, but not an AVL tree. Draw the tree after **U** is deleted.
Part H: Let's consider the tree to be an AVL tree. Draw the tree after **P** is inserted.
Part I: Let's consider the tree to be an AVL tree. Draw the tree after **Z** is inserted.
Part J: Let's consider the tree to be an AVL tree. Draw the tree after **K** is deleted.

CS140 Final Exam - December 10, 2019

Question 4: 17 Points

Here's the API for a queue, from **queue.hpp**. This is identical to what's in the lecture notes.

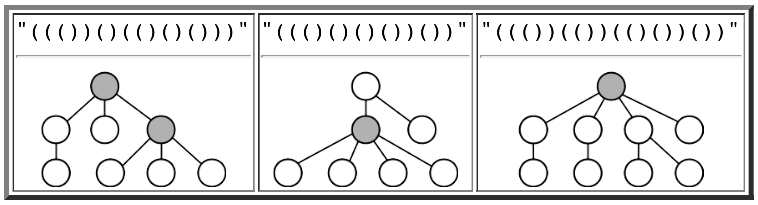
```
#pragma once
#include <string>

/* This defines the nodes of the queue.  It's only used internally to
   the queue, but we have to define it here.  */

class Qnode {
  public:
    std::string s;
    Qnode *ptr;
};

/* Here's the Queue class. */

class Queue {
  public:

    /* Constructors, Destructor, Assignment Overload */

    Queue();
    Queue(const Queue &q);
    Queue& operator= (const Queue &q);
    ~Queue();

    /* Same operators as stacks. */

    void Clear();
    bool Empty() const;
    size_t Size() const;

    /* Push puts the string on the end of the queue,
       and Pop removes the string from the beginning of the queue. */

    void Push(const std::string &s);
    std::string Pop();

  protected:
    Qnode *first;           // Pointer to the first node on the queue.
    Qnode *last;            // Pointer to the last node on the queue.
    int size;               // The queue's size.
};
```

Please implement the methods **Push()** and **Pop()**. Have **Pop()** throw the string `"Bad pop"` if it encounters an error.

CS140 Final Exam - December 10, 2019

Question 5: 16 Points

You can describe a tree with a string of parentheses. The string `"()"` represents a single node, and `"(`*`ABC`*`)"` represents a tree that has a root node with three children, which are the subtrees, *A*, *B* and *C*. For example, below are three trees and their representations:



Please write a procedure with the following prototype:

```
int max_children(const string &s);
```

This procedure should take a string composed of parentheses as above, and return the number of children that the node with the maximum number of children has. In the first example, it will return three, because the two shaded nodes each have three children. In the second example, it will return four, because the shaded node has four children, and in the third example, it will also return four.

You may assume that there are no errors in the string -- it will be a valid representation of a tree.

This is similar to a Topcoder problem that y'all found difficult earlier this semester (which was much harder than this problem). I suggest that you break this problem into two parts:

1. Create a vector **v**, where if **s[i]** is a left parenthesis, then **v[i]** contains the index of the right parenthesis that corresponds to **s[i]**. Otherwise **v[i]** is -1.

2. Define a recursive procedure that takes as arguments **v** and the index of a left parenthesis. It should return the answer for the subtree that is rooted at the substring starting with that index. I called my recursive procedure **rec_children()**.

Let's use the middle example above to help. The following shows what **v** is for that tree:

```
index:   0   1   2   3   4   5   6   7   8   9  10  11  12  13
s:       (   (   (   )   (   )   (   )   (   )   )   (   )   )
v:      13  10   3  -1   5  -1   7  -1   9  -1  -1  12  -1  -1
```

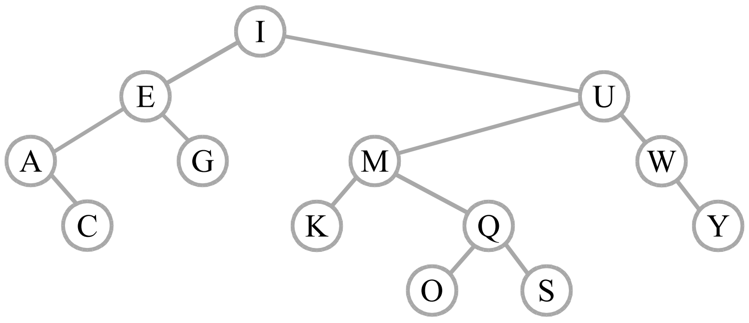
Here are the calls to **rec_children()** and what they return:

```
rec_children(v, 0)         Counts its children.  There are 2.  It makes two recursive calls, which I label below:
  1. rec_children(v, 1)    Counts its children.  There are 4.  It makes four recursive calls.
       rec_children(v, 2)  Returns 0 because it has no children.
       rec_children(v, 4)  Returns 0.
       rec_children(v, 6)  Returns 0.
       rec_children(v, 8)  Returns 0.
     rec_children(v, 1)    Returns 4:  It has four children, and the maximum recursive answer
                                       for those children is 0.  Max of 4 and 0 is 4, so return 4.
  2. rec_children(v, 11)   returns 0 because it has no children.

rec_children(v, 0)         Returns 4: It has two children, and the maximum recursive answer
                                       for those children is 4.  Max of 2 and 4 is 4, so return 4.
```

If you are panicking with this problem, consider the following:

- If simply write out what you think you should be doing, and it's reasonable, you'll get some points.
- Writing the code to create **v** correctly will get you half credit if you don't write **rec_children()** correctly.
- Writing **rec_children()** correctly will get you half credit if you don't create **v** correctly.

Scratch Paper
that you may find useful.


I
E
U
A
G
M
W
C
K
Q
Y
O
S