CpSc 2120: Algorithms and Data Structures

Instructor: Dr. Brian Dean — Fall 2017
Webpage: `http://www.cs.clemson.edu/~bcdean/` — TTh 12:30-1:45
Handout 8: Midterm Quiz (In-Class Part). 25 possible points. — McAdams 119

1. True/False (1 point each). Please circle T (true), F (false), or WTF (whither true or false?). Correct answers are worth 1 point, WTF answers 1/2 point, no answer or incorrect answers 0 points.

T F WTF The height of an AVL tree on n elements is always $\Omega(\log n)$.

T F WTF If an algorithm runs in $\Omega(n)$ time, then it runs in $O(n^2)$ time.

T F WTF If an algorithm runs in $\Theta(n^3)$ time, then it runs in $O(n^3)$ time and also $\Omega(n^3)$ time.

T F WTF A queue can be implemented so that the operations of inserting an element at the front and removing an element from the back both take $O(1)$ time.

T F WTF In a network with n webpages, each iteration of Pagerank will run in $O(n)$ time (here an "iteration" is what we performed 50 times on homework #1).

2. Running Time (3 points). For each of the code samples below, please write the running time as a function of n using $\Theta()$ notation.

```
// CODE EXCERPT 1:
for (int i=0; i<n; i++)
  cout << "Hello world\n";
```

```
// CODE EXCERPT 2:
for (int i=0; i<n; i++)
  for (int j=i+1; j<n; j+=2)
    cout << "Hello world\n";
```

```
// CODE EXCERPT 3:
for (int i=0; i<n; i++)
  for (int j=i; j>0; j/=2)
    cout << "Hello world\n";
```

3. Verify (4 points). Given two length-n integer arrays A and B, please describe, in English, a fast algorithm that can verify if B contains the sorted contents of A. That is, you want to verify if B is the result you would get from sorting A. Be sure to describe the running time of your algorithm.

4. Missing Element (4 points). Suppose you are given a balanced binary search tree T_1 containing n distinct elements, and another balanced binary search tree T_2 containing all but one of the elements in T_1. Please describe (preferably in English) a fast algorithm for determining the missing element – appearing in T_1 but not T_2. Please comment on the running time of your algorithm. Assume both T_1 and T_2 are balanced via randomization according to the code from lab 4.

5. Tree Copy (4 points). Please fill in the function below so it creates a duplicate copy of an entire tree.

```
struct Node {
  int key;
  int size;
  Node *left, *right;
  Node (int k, int s, Node *l, Node *r) {
     key = k; size = s; left = l; right = r;
  }
};

// Returns a pointer to the root of a duplicate copy of a tree
Node *copy_tree(Node *root)
{
```

6. Debugging (5 points). For each of the four code excerpts below, please either state that the code is correct, or clearly identify one or more compiler errors or "bugs" and show how to fix them. Here, a "bug" is either a specific part of the code or a problem with its general design that causes the program to exhibit undesired behavior, such as potentially crashing, incorrect output, much slower performance than clearly intended, or leaking memory. Line numbers are provided for reference to help you refer to parts of the code, if needed (the line numbers are not themselves part of the code).

```
1. // CODE EXCERPT 1:
2. // This function returns 1+2+3+4+...+n.  Assume n>=0.
4. int sum(int &n)
5. {
6.   if (n == 0) return 0;
7.   return n + sum(n-1);
8. }
```

```
1. // CODE EXCERPT 2:
2. struct Node {
3.   int key;
4.   Node *left, *right;
5. };
6. // Return maximum value in a standard binary search tree
7. // We guarantee that the tree isn't empty
8. int find_max(Node *root)
9. {
10.  // base case: are we at a leaf?
11.  if (root.left == NULL && root.right == NULL) return root.key;
12.  int max_left = find_max(root.left);
13.  int max_right = find_max(root.right);
14.  return max_left > max_right ? max_left : max_right;
15. }
```

```
// CODE EXCERPT 3:
// This uses the same Stringset class we wrote in lecture
#include "stringset.h"
void insert_hello_world(Stringset S)
{
  S.insert ("Hello world");
}
int main(void)
{
  Stringset S;
  insert_hello_world(S);
  return 0;
}
```

```
// CODE EXCERPT 4:
int ***A = new int **[10];  // Booyah!  3D Array!
for (int i=0; i<10; i++) {
  A[i] = new int *[20];
  for (int j=0; j<20; j++) {
    A[i][j] = new int [30];
    for (int k=0; k<30; k++)
      A[i][j][k] = new int;
  }
}
```

This page may be used for scratch work[1].

[1]What, you were expecting something silly here in the footnote?

This page may be used for scratch work[2].

[2] Ok, fine, just one data structure joke then: Q: Why are object-oriented programs so bad at baseball? A: Three structs and you're out!