CMPSC24 Final exam
Winter 2018
03/19/2018

This exam is closed book, closed notes. You may use a one-page A4 size paper containing your notes. Write your name on the your notes paper and all other sheets of the exam. No calculators or phones are allowed in the exam. **Write all your answers on the answer sheet. All exam sheets, notes paper and scratch paper must be turned in at the end of the exam.**

By signing your name below, you are asserting that all work on this exam is yours alone, and that you will not provide any information to anyone else taking the exam. In addition, you are agreeing that you will not discuss any part of this exam with anyone who is not currently taking the exam in this room until after the exam has been returned to you. This includes posting any information about this exam on Piazza or any other social media. Discussing any aspect of this exam with anyone outside of this room constitutes a violation of the academic integrity agreement for CMPSC 24.

Signature: _____

Name (please print clearly):_____

Umail address: _____@umail.ucsb.edu

Perm number: _____

You have 3 hours to complete this exam. Work to maximize points. A hint for allocating your time: if a question is worth 10 points, spend no more than 10 minutes on it if a question is worth 20 points, spend no more than 20 minutes on it etc. If you don't know the answer to a problem, move on and come back later. Most importantly, stay calm. You can do this.

WRITE ALL YOUR ANSWERS IN THE PROVIDED ANSWER SHEET IN PEN OR DARK PENCIL!

DO NOT OPEN THIS EXAM UNTIL YOU ARE INSTRUCTED TO DO SO.
GOOD LUCK!

Part 1 [50 points] Short answers: Provide answers to each of the following questions in your answer sheet

1. **[5 pts]** Assume that you are given N+1 elements that are in sorted (ascending or descending) order. The first N are inserted into each of the following data structures one at a time (in sorted order). What is the Big-O running time of inserting the last element into each of the following data structures? Provide the tightest bound.

i. Binary Search Tree that is NOT necessarily balanced.
ii. Balanced BST. Assume that the BST is balanced after each insert and re-balancing can be done in O(log i) where i is the number of nodes in the BST
iii. Unsorted linked-list. Assume that insertions are done at the head of the list
iv. Min-heap
v. Stack

2. **[5 pts]** For the following questions select the operation that is "faster" based on its Big-O running time? *Write A, B or C in your answer sheet in each case*

i. (1 pt) **Deleting a value**:
   A. **Deleting** a value at the head of a linked-list
   B. **Deleting** a value from a sorted array
   C. Both are equally fast

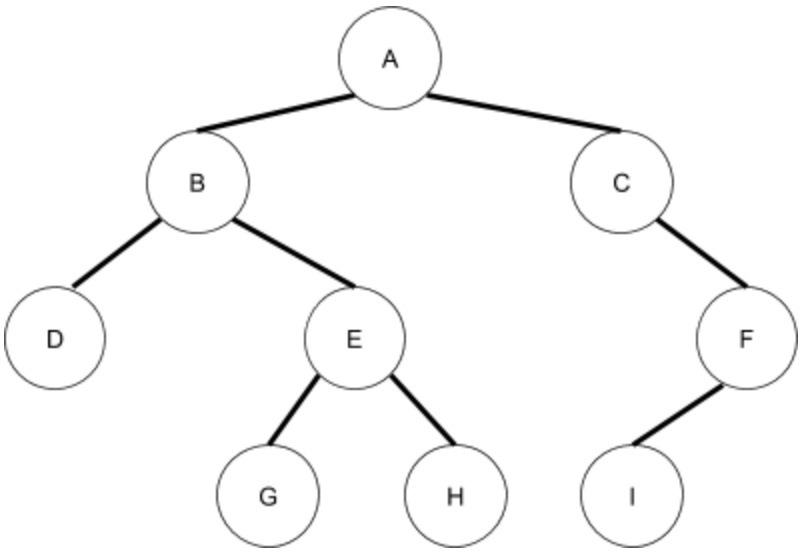
ii. (1 pt) **Searching** for an element:
   A. **Searching** for an element in a balanced BST
   B. **Searching** for an element in a sorted array using binary search
   C. Both are equally fast

iii. (1 pt) **Finding the minimum** element:
   A. **Finding the minimum** element in a min Heap
   B. **Finding the minimum** element in a sorted array where elements are stored in ascending order
   C. Both are equally fast

iv. (1 pt) **Finding the minimum** element:
   A. **Finding the minimum element in an unsorted array**
   B. **Finding the minimum element in a balanced BST**
   C. Both are equally fast

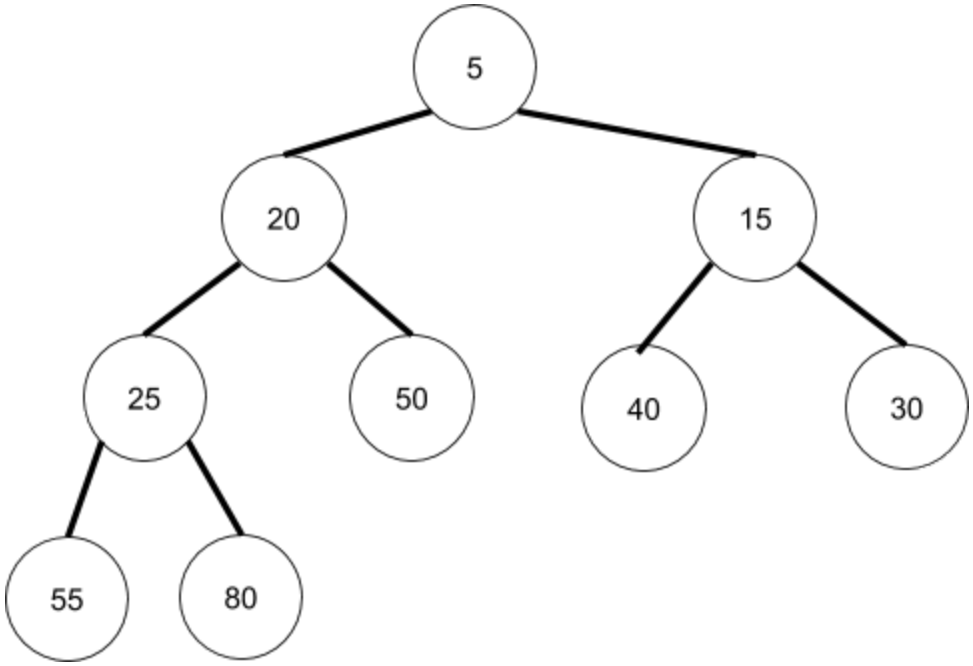
v. (1 pt) **Searching for an element**:
   A. **Searching for an element in a sorted linked list**
   B. **Searching for an element in a balanced BST**
   C. Both are equally fast

3. **[20 pts]** Consider the following BST with keys labeled A through I. You are not given the ordering scheme used to insert the keys into the BST. Do not assume alphabetical ordering. Instead use the properties of BSTs to answer the following questions:



i. (1 pt) Is B < A ? (Yes/No/Cannot be determined)
ii. (1 pt) Is H < A ? (Yes/No/Cannot be determined)
iii. (1 pt) Is G < I ? (Yes/No/Cannot be determined)
iv. (1 pt) Is I > F ? (Yes/No/Cannot be determined)
v. (1 pt) What is the key with the minimum value?
vi. (1 pt) What is the key with the maximum value?
vii. (1 pt) Which key is the successor of B?
viii. (1 pt) Which key is the successor of H?
ix. (8 pts) Write the output of printing the keys in the BST when the tree is traversed using an:
    a. Inorder traversal
    b. Preorder traversal
x. (4 pts) Draw the BST after deleting the root node (A)

4. **[10 pts] Below is a tree representation of a min heap that stores integer keys**



i. (4 pts) Suppose that an array was used to store the heap data structure shown above. Draw the contents of the array at indices 0 through 8

ii. (4 pts) Insert the elements 65, 55 and 2 into the heap in that order. Then answer the following questions about the resulting heap after all the insertions have been completed.
   a. What is the key value of the parent of 65?
   b. What is the key value of the parent of 55?
   c. What are the key values of the left and right children of 2?
   d. What are the key values of the left and right children of 5?

iii. (2 pts) Delete the ROOT node (5) from the heap shown in the original diagram. Then draw an array representation of the resulting heap.

5. **[10 pts]** The following code that uses the stack container class of the Standard Template Library to check if the parentheses in an infix expression are balanced. The code has a bug. To detect the bug, answer the questions that follow:

```
bool balanced( string input){
    stack<char> st;
      for(int i = 0 ; i < input.length(); i++){
         if( input[i] == '('){
            st.push(input[i]);
         } else if ( input[i]== ')' && !st.empty()){
            st.pop();
         }
     }
     if(!st.empty()){
            return false;
     }
     return true;

}
```

i. (3 pts) Suppose that the function **`balanced()`** is called with each of the following input strings. What is the return value of the function in each case?

a. "(27 + 58) - ((2 * 28) - (100 / 2))"
b. "(27 + 58) - ((2 * 28) - (100 / 2)))"
c. "(27 + 58) - (((2 * 28) - (100 / 2))"

ii. (3 pts) Rewrite the part of code that has a bug.

iii. (2 pts) What is the Big O running time of the above implementation for an input string of length N? Give the tightest bound.

iv. (2 pts) Which of the following operations is/are NOT supported by a stack data structure? *SELECT ALL THAT APPLY*

A. Inserting an element into the stack
B. Deleting the element that was last inserted into the stack
C. Printing the elements of the stack in sorted order
D. Searching for an element in the stack
E. None of the above

6. **[10 pts] The following code that uses the stack data structure of the C++ Standard Template Library. This data structure has a copy constructor, copy assignment operator and de-constructor that allows for deep copy/deletion of data.**

```
#include <stack>

void foo(){

    stack<int> a;
    stack<int> b;

     a.push(10);
     a.push(20);
     a.push(30);

    stack<int> *p = new stack<int>(a);
    stack<int> *q;

      q = p;
      b = a;

      while(!q->empty()){
             cout<<q->top()<<" ";
             q->pop();
      }
      cout<<endl;

      while(!b.empty()){
             cout<<b.top()<<" ";
             b.pop();
       }
       cout<<endl;
       return;
}
```

Answer the following questions about the given code:

i. (3 pts) Suppose that the function **`foo()`** is called and the code within the function is executed until it reaches the return statement (not yet executed). Complete each row of the table provided in your answer sheet.

ii. (3 pts) What is the output of the above code when the function **`foo()`** is called?

iii. (2 pts) The **copy-assignment** operator of **`stack`** is invoked as a result of which of the following statements. The variables **`a, b, p and q`** are defined in the function **`foo().`** *SELECT THE SINGLE BEST ANSWER*

A. q = p;
B. b = a;
C. Both A and B
D. None of the above

iv. (2 pts) The **copy-constructor** of **`stack`** is invoked as a result of which of the following statements, assuming **`a`** is an existing instance of **`stack .`** *SELECT ALL THAT APPLY*

A. stack<int> *p = new stack<int>(a);
B. stack<int> y = a;
C. stack<int>* y = new stack<int>;
D. None of the above

7. **[5 pts]** The function given below takes as input an array of integers of size `N`, where `N>0` and sorts the elements in the array in ascending order. Answer the questions that follow above the given code

```
void sortNaive(int* a, int N){
    //Precondition: unsorted array a
    //Post condition: a is sorted array in ascending order

    for(int i =0; i<N; i++){
       int minElem = a[i];
       int index=i;
       for(int j = i+1; j<N;j++){
            //Code in the inner loop
             if(a[j]<minElem){
                   minElem = a[j];
                   index = j;
             }
       }
       int tmp = a[i];
       a[i] = a[index];
       a[index]=tmp;
    }

}
```

i. (1 pt) What is worst case Big-O running time of executing the following statements for any SINGLE iteration of the inner loop?

```
if(a[j]<minElem){
      minElem = a[j];
      index = j;
}
```

ii. (2 pts) How many times is the code in the inner loop executed when the function **`sortNaive`** is called on an array with N elements? Write an expression in terms of N (Do not provide the Big O running time here, just the total number of iterations).

iii (1 pt) What is the Big O running time of the given code in terms of the array size N?

iv (1pt) If the input array is already sorted, does the Big-O complexity of the given code increase, decrease or remain the same?

8. **[5 pts]** Write the pseudo code or actual code for an algorithm that uses one of the data structures you have learned so far to improve the running time of the sorting function from the previous question. Assume that the data structure you are using has been implemented for you. What is the Big O running time of your algorithm? Justify your answer.

See next page for part 2

Part 2 [30 points] Binary Search Trees

Consider the definition of the class **`Node`** and class **`BST`** used in the construction of a binary search tree. Note that the provided BST stores an integer in each node and does not insert duplicates

```
class Node{
  public:
    int data; // data element
    Node* left; // pointer to the left child
    Node* right; //pointer to the right child
    Node* parent;
    Node(const int& d){ data = d; left = 0; right = 0; parent=0;}
};

class BST{
  public:
    BST(){root = 0;}            // constructor
   ~BST(){deleteTreeRecursively(root);}  // de-constructor
    bool insert(const int value);
    // true on successful insertion, false if node was already present

    //Implement this function
    bool search(const int value) const;

private:
    Node* root; // pointer to the root of the tree

    //Implement either of the following functions
    void deleteTreeRecursively(Node* n);
    Node* successor(Node* n) const;
};
```

Assume that

- the constructor and `insert()` method have been correctly implemented.
- No duplicates are inserted into the BST

1. **[15 pts]** Implement the **`search()`** method of the BST that returns true if a given value is present in the BST otherwise returns false. Your search method should run with a complexity of O(H) where H is the height of the tree. You may use a helper function as long as you provide its implementation (assume your function is already declared as a private method of the class). 5 pts will be awarded for code that is readable, concise and well commented.

2. [15 pts] You have a choice between implementing one of the following private methods. Clearly indicate in your answer sheet which one you are implementing by writing the complete definition of the function:

```
void deleteTreeRecursively(Node* n);
```

This function takes a pointer to the root of a sub-tree within the BST and recursively deletes all the nodes in that subtree (including the original node). Your implementation MUST be recursive. Non-recursive implementations will not receive credit. 5 pts for code that is readable, concise and well-commented.

OR

```
Node* successor(Node* n)const;
```

This function takes as input a pointer to node in the BST and returns the address of the node with the next larger value in the BST. For example if the BST stores the integers 80, 70, 50, 10 , 20, 100 and the successor method was called passing in the address of the node with value 50, the function should return the address of the node with value 70. If there is no succssor, return null. 5 pts for code that is readable, concise and well-commented.

The End

Scratch Paper