CMPSC24 Final Exam
Spring 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. All questions must be asked in writing using the provided yellow paper or scratch sheet.**

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!

All coding problems carry style points.

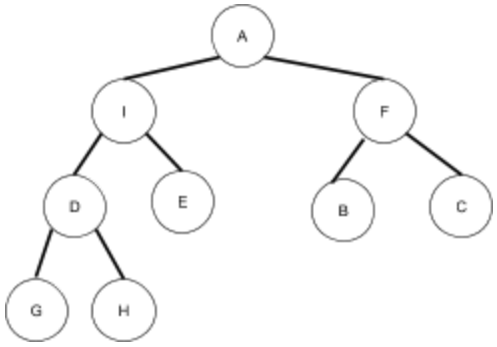
Part 1 [20 points] Provide answers to each of the following questions in your answer sheet

1. [5 pts] In your answer sheet write the name(s) of ALL the Standard template library (STL) classes that match each of the provided descriptions. Choose among the STL classes: **`set, priority_queue, vector, stack, queue.`**

i. Is a balanced Binary Search Tree
ii. Support(s) **`push(), pop()`** and **`top()`** operations where the first item that is pushed is the first to be popped out.
iii. Provide(s) sublinear (Big-O) running times for both **`insert`** and **`min`**
iv. Allow(s) accessing all the elements in it without having to delete any element
v. Running time for insert and search is O(log N) where N is the number of items in the data structure.

2. [5 pts] A preorder traversal on a BST gives the following output 500, 400, 200, 300, 600. Draw the BST.

3. [10 pts] Consider the following binary **min-heap** with keys labeled A through I. You are not given the ordering scheme used to insert the keys into the **heap**. Do not assume alphabetical ordering. Instead use the properties of min-heaps to answer the following questions:



i. (1 pt) Is B less than A ? (Yes/No/Cannot be determined)
ii. (1 pt) Is H greater than or equal to A ? (Yes/No/Cannot be determined)
iii. (1 pt) Is G greater than or equal to I ? (Yes/No/Cannot be determined)
iv. (1 pt) Is G greater than or equal to F? (Yes/No/Cannot be determined)
v. (1 pt) What is the key with the minimum value? Write "N/A" if this cannot be determined
vi. (1 pt) What is the key with the maximum value? Write "N/A" if this cannot be determined
vii. (2 pts) Draw an array representation of the binary Heap
viii. (2 pts) You are given the following information about a new element with label Z:

I< Z< D (Z is greater than I and less than D),

Z<E and Z< G (Z is less than E and Z is less than G)

Use the provided ordering and the properties of a min-Heap to insert Z into the given heap and draw the resulting binary heap as a tree or an array

Part 2: [40 points] C++ concepts, data structures and templates.

The class **Card** provided below is used to represent a card that comprises of a suit and a value. The member variable **suit** is a character among **{'c', 'd', 's', 'h'},** which stand for clubs, diamonds, spades and hearts respectively. The member variable **value** is a number between 1 and 13, where 1 represents an ace, 11-13 stand for jack, queen and king respectively.

```
class Card{

    public:
        Card(const string& s); //initializes the suit and value
/*Assume the string parameter s is always formatted as a single character for
the suit, followed by a space and one or two characters to represent the value
of the card e.g. "h 3" stands for 3 of hearts, "s 10" stands for 10 of
spades*/

        char getSuit() const; //returns the suit
        int getValue() const; //returns the value

    private:
        char suit;
        int value;
};
```

You are also given the following information:

s.erase(pos, len);

Erases part of the string **s** reducing its length.

pos: Index of the first character to erase

len: Number of characters to erase

4. [6 pts] Based on the definition of the **Card** class, **write "Error" if each of the provided functions results in a compile time error and give the reason for the error.** If the functions cause runtime and/or logic errors but NOT compile time errors OR do not result in any error at all, write "**Correct**". Assume that the functions are embedded in an otherwise correct and complete C++ program.

i)
```
int main(){
      Card c;
      cout<<c.getSuit()<<endl;
  }
```

ii)
```
int main(){
      Card c("h 9");
      cout<<c.getValue()<<endl;
   }
```

```
iii) int main(){
            Card c("d 4");
            Card b = c;
      }

iv) int main(){
            Card c("d 4");
            c.suit= 's';
            cout<<c.getSuit()<<endl;
      }

v) char Card::getSuit()const{
             suit= 'c';
             return suit;
       }

vi) Card::Card(const string& s){
            suit = s[0];
            s.erase(0,2);// erase the first two characters of s
                         //starting at position 0
            value = s[0];
    }
```

Assume that all the methods of the class have been correctly implemented for the following questions:
5. [4 pts] Implement the overloaded equality operator (==) for the **`Card`** class as a non member function.

6. [30 pts] Assume that an ordering is defined on cards. The ordering of cards is determined first by its suit and then by the value just like in PA02 and PA03:

- The ordering least to greatest is: clubs, diamonds, spades, hearts. Thus a club of any value is less than a diamond of any value.
- The ordering within each suit is determined by the value from least to greatest as follows: ace, 2, 3, . . . 10, jack, queen, king

Answer the following questions:

i. [10 pts] Implement the **<** operator for the **`Card`** class as a non-member function. Your code should be under 12 lines to be considered concise.
ii. [10 pts] Insert the following card values in the provided order into a BST:
h 9, c a, d j, s k, h 10, s 5, c 2 , then:

a. Draw the resulting tree in your answer sheet
b. Delete the element "d j" from the tree and draw the resulting tree

iii. [10 pts] Implement the function **`playgame`** that takes two vector of **`Card`** objects as inputs. Each vector represents a player's hand (list of **`Card`** objects). The first input corresponds to Alice's hand and the second one Bob's hand. Your function **`playgame()`** should implement the game described below and return the name of the winner, which is either the string "**`Alice`**" or "**`Bob`**". If the game ends in a tie, the function should return the string "**`Tie`**".

The game proceeds in rounds until both players have exhausted their cards. In each round each player should select the minimum value card in their own hand without having any knowledge of the other player's cards. Whoever has the minimum card among the two selected cards will receive a point for that round. If the two cards are equal then neither player receives a point. Each player should then remove the card they selected from their hand. The game stops when both players have exhausted their cards.

Use any appropriate data structure from the STL to provide an efficient implementation for the game.
Assume that:

- ***the two players have equal number of cards***
- ***you have a correct implementation of the operators < and == for the Card class.***

Part 3: [10 points] Running Time

7. [10 pts] Find the Big-O running time of each of the following code as a function of the input size N. Justify your answer by writing an expression for the primitive operations in terms of N and deducing the Big O expression from it.

i.[2pts]

```
int foo(int N){
      int x=0;
      for(int i=0; i<N; i+=2) {
            for(int j = N; j>0; j=j/2) {
                  x++;
            }
      }
      return x;
}
```

ii. [2 pts]

```
int foo(int N){
      int x=0;
      for(int i=0; i<N; i+=2) {
            for(int j = 0; j<N/2; j++) {
                  x++;
            }
      }
      return x;
}
```

iii. [2 pts]

```
stack<int>& foo(int N){
      stack<int> s;
      for(int i=1; i<N; i=i*2) {
            s.push(i);
      }
      return s;
}
```

iv. [2 pts]

```
int foo(vector<int> v, int N){ //Assume v has N elements
      priority_queue<int> s;
      for(int i=0; i<N; i++) {
            s.push(v[i]);
      }
      s.pop();
      return s.top();
}
```

v. [2 pts]

```
bool foo(vector<int>& v, int N, int elem){ //Assume v has N elements
      set<int> s;
      int i;
      for(int i=0; i<N; i++) {
            s.insert(v[i]);
      }
      return s.find(elem);
      // find searches for elem in s and returns true if the element
      // was present otherwise returns false.

}
```

8. [5 pts] Suppose that you have analyzed the running time of two algorithms and found that algorithm A has a complexity of $O(2^N)$ and algorithm B has a complexity of $O(N^2)$, which of the following deductions can you make about the two algorithms. Select all the options that apply.

A. Algorithm B is always faster than algorithm A for any input size
B. Algorithm A is always faster than algorithm B for any input size
C. The running time of Algorithm A is bounded by $c * 2^N$ for some constant c and large values of N
D. The running time of Algorithm B cannot be more than $c * N^4$ for some constant c and large values of N
E. We can determine the exact running times of the two algorithms for any specific input size N using the given Big-O complexity.

9. [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 in a balanced binary search tree
B. **Deleting** a value from a sorted array
C. Both are equally fast

ii. (1 pt) **Inserting a value**

A. **Inserting a value** in a heap
B. **Inserting a value** in a stack
C. Both are equally fast

iii. (1 pt) **Finding the minimum** element:

A. **Finding the minimum** element in a balanced BST
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 element that was last inserted**:

A. **Finding the element that was last inserted in a stack**
B. **Finding the element that was last inserted in a heap**
C. Both are equally fast

v. (1 pt) **Searching for an element**:

A. **Searching for an element in a sorted array**
B. **Searching for an element in a balanced BST**
C. Both are equally fast

Part 4 [20 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;//pointer to the parent
    Node(const int& d){ data = d; left = 0; right = 0; parent=0;}
};
```

```
class BST{
  public:
    BST(){root = 0;}            // constructor
   ~BST();                     // destructor
    bool insert(const int value);
    // true on successful insertion, false if node was already present
    bool search(const int value) const;
    // returns true if the given value is present in the BST
    int getSum() const{
       return getSum(root);
    }
    friend bool operator==(const BST& b1, const BST& b2);
private:
    Node* root; // pointer to the root of the tree
    int getSum(Node* r) const;
    void linearizeInorder(Node* r, vector<int>& v);
    void linearizePreorder(Node* r, vector<int>& v);
};
```

```
bool operator==(const BST& b1, const BST& b2);
```

For all the questions assume that:

- the default constructor, **`insert()`** and **`search()`** methods have been correctly implemented.
- No duplicates are inserted into the BST

10. [5 pts] Provide an implementation of the function **`getSum()`**

```
int getSum(Node* n);
```

This function takes a pointer to the root of a BST and finds the sum of all the nodes in that tree. 3pts for code that is readable, concise and well-commented.

11. [5 pts] You have a choice between implementing the **`linearizeInorder`** or **`linearizePreorder`** functions. Both functions take a pointer to the root of a BST and a reference to a vector as inputs.

- **`linearizeInorder`** performs an in-order traversal of the BST and inserts each key into the vector. For example if b1 is the BST depicted in Figure 1 and w is a vector of integers that is initially empty then after calling **`linearizeInorder(b1.root, w), w`** will contain the key values of b1 in order {20, 80, 90, 100, 200}
- **`linearizePreorder`** performs a preorder traversal of the BST and inserts each key into the vector. For example if b1 is the BST depicted in Figure 1 and w is a vector of integers that is initially empty then after calling **`linearizePreorder(b1.root, w), w`** will contain the key values of b1 in pre-order {100, 80, 20, 90, 200}

12. [10 pts] Implement the overloaded == operator. The assignment operator must return true if the two input BSTs have the same keys even if they don't have identical structures. Examples of two BSTs that are equal is shown in Figure 1 and BSTs that are not equal are shown in Figure 2. If both trees are empty the operator should return true. You may use any of the methods declared in the class BST on page 7 or add your own private helper functions. You may also use any STL container class (other than **`set`**). 3 out of 10 pts will be awarded for code that is readable, concise and well commented.

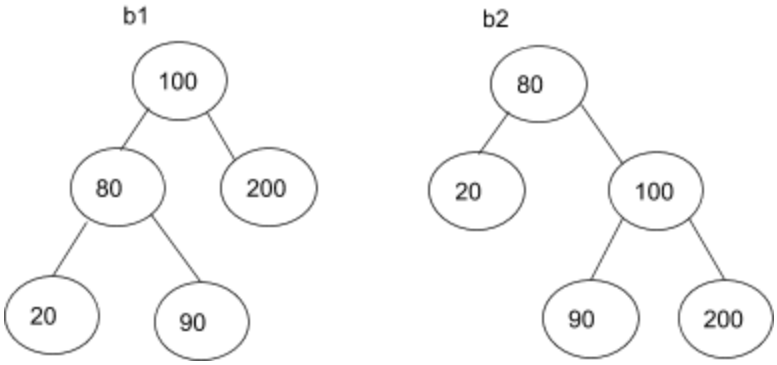


Figure 1: Example of two trees that are equal

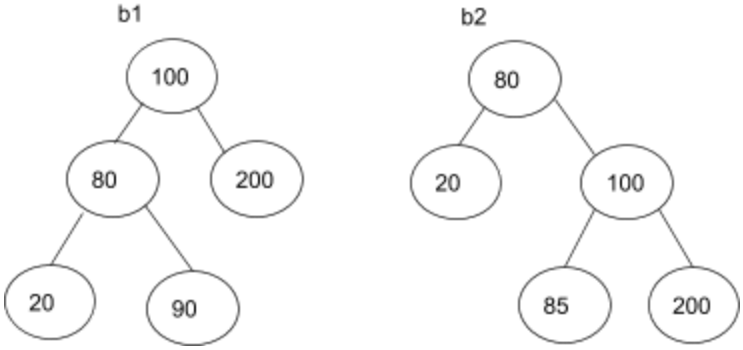


Figure 2: Example of two trees that are NOT equal

The End

Scratch Paper