CMPT 225 - Midterm 2
November 17, 2017
Time: 45 minutes

DO NOT begin until told to do so.
Fill in your name and student ID below.
Turn off your phone and put it in your bag.
Keep your student ID card on your desk at all times.
You can't have anything other than a pen/pencil and your student ID card on you during the test.
Before you begin, please check that your exam consists of **7** consecutively numbered questions and **2** pages of appendix.

Read the questions carefully and more than once.
Good luck!

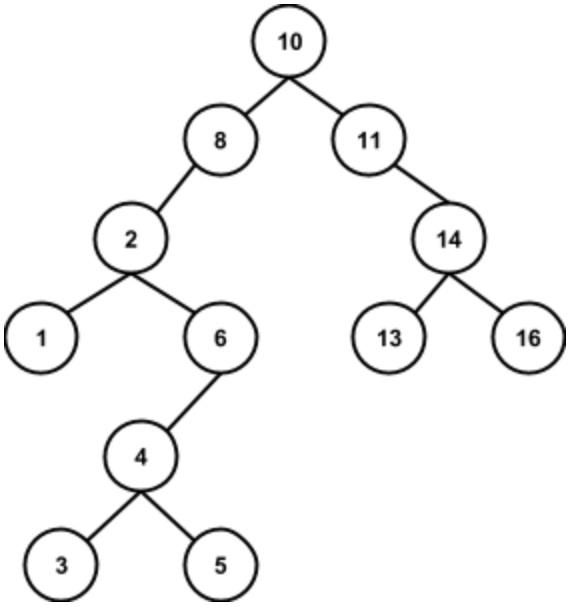
NAME: …….....

STUDENT ID: ……

SCORE:

Question	**Marks**
Question 1 (5 marks)	
Question 2 (5 marks)	
Question 3 (5 marks)	
Question 4 (10 marks)	
Question 5 (5 marks)	
Question 6 (10 marks)	
Question 7 (10 marks)	

1. Give the in-order traversal of the following tree (5 marks)

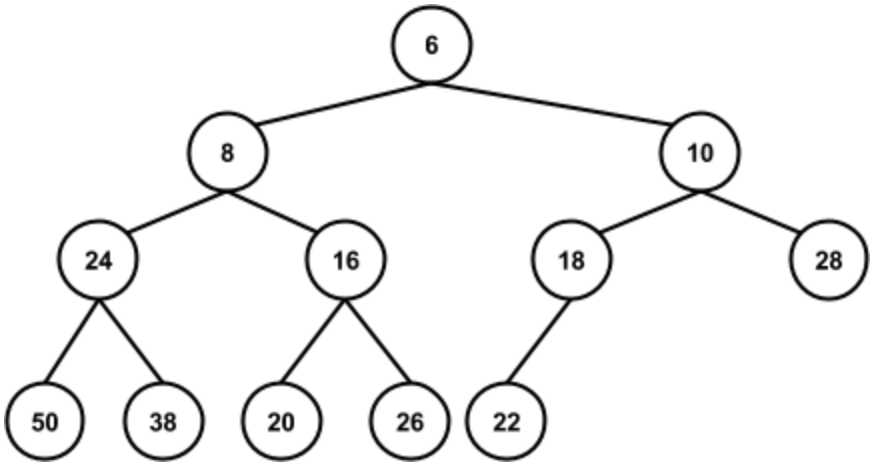


1, 2, 3, 4, 5, 6, 8, 10, 11, 13, 14, 16

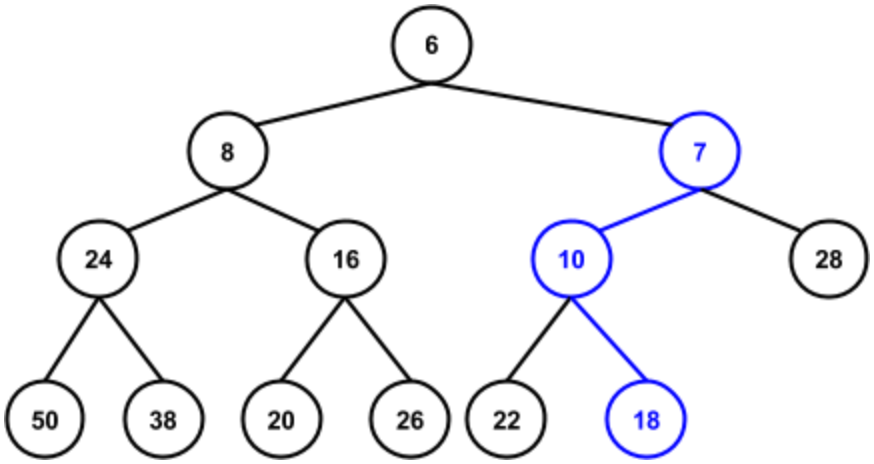
2. True/False? (5 marks)

a. the above tree is a min-heap F
b. the above tree is complete F
c. the above tree is proper F
d. the above tree is a binary search tree T
e. the height of the above tree is 5 T

3. What will the following **heap** look like after you insert 7 into it? (5 marks)



Answer:



4. What is the time complexity of the following in big-O notation in the worst case scenario? You do not need to justify your answers. (10 marks)

a. insertion into a binary search tree O(n)

b. removeMin() in a heap-based priority queue O(lg n)

c. bottom-up construction of a heap from a list of items O(n)

d. insertion into a priority queue that is based on a sorted list O(n)

e. getting the min() from a priority queue that is based on a sorted list O(1)

5. What is the time complexity of the following functions in big-O notation? You do not need to justify your answers.

(3 marks)

```
void mystery(int n){
        if (n>1)
                mystery(n/2);
        for (int i=0; i<n; i++)
                cout << "*";
        cout << endl;
}
```

O(n)

```
void mystery2(int n){
        for (int i=0; i<n; i++)
                for (int j=i+2; j<i+7; j++)
                        cout <<"*";
        cout << endl;
}
```

(2 mark)

O(n)

6. Give the code for the insert() function in the HeapPriorityQueue. The rest of the class implementation is provided in appendix A for your reference. (10 marks)

Same as other version!

7. Describe (in pseudo-code or in fewer than five sentences) an algorithm that finds the kth smallest element of a vector of n distinct integers in O(n + k log(n)) time. (10 marks)

Same as other version!

Apendix A. Implementation of HeapPriorityQueue

```
template <typename E, typename C>
class HeapPriorityQueue {
public:
        int size() const { return T.size(); }                   // number of elements

        bool empty() const { return size() == 0; }              // is the queue empty?

        void insert(const E& e) {/* You will fill this.*/}      // insert element

        const E& min() { return *(T.root()); };                 // minimum element

        void removeMin(){                                       // remove minimum
                if (size() == 1)                                // only one node?
                T.removeLast();                                 // . . .remove it
                else {
                        Position u = T.root();                  // root position
                        T.swap(u, T.last());                    // swap last with root
                        T.removeLast();                         // . . .and remove last
                        while (T.hasLeft(u)) {                  // down-heap bubbling
                                Position v = T.left(u);
                                if (T.hasRight(u) && isLess(*(T.right(u)), *v))
                                        v = T.right(u);         // v is u's smaller child
                                if (isLess(*v, *u)) {           // is u out of order?
                                        T.swap(u, v);           // . . .then swap
                                        u = v;
                                }
                                else break;                     // else we're done
                }
        }
private:
        VectorCompleteTree T;         // priority queue contents - Implemented in the next page
        C isLess;                     // less-than comparator
                                      // shortcut for tree position
        typedef typename VectorCompleteTree::Position Position;
};
```

Apendix A (continued). Implementation of VectorCompleteTree used in HeapPriorityQueue

```
template <typename E>
class VectorCompleteTree {
private:                                                    // member data
        std::vector V;                                      // tree contents
public:                                                     // publicly accessible types
        typedef typename std::vector::iterator Position;    // a position in the tree
protected:                                                  // protected utility functions
        Position pos(int i)                                 // map an index to a position
        { return V.begin() + i; }
        int idx(const Position& p) const                    // map a position to an index
        { return p − V.begin(); }
public: VectorCompleteTree() : V(1) {}                      // constructor
        int size() const { return V.size() − 1; }
        Position left(const Position& p) { return pos(2*idx(p)); }
        Position right(const Position& p) { return pos(2*idx(p) + 1); }
        Position parent(const Position& p) { return pos(idx(p)/2); }
        bool hasLeft(const Position& p) const { return 2*idx(p) <= size(); }
        bool hasRight(const Position& p) const { return 2*idx(p) + 1 <= size(); }
        bool isRoot(const Position& p) const { return idx(p) == 1; }
        Position root() { return pos(1); }
        Position last() { return pos(size()); }
        void addLast(const E& e) { V.push back(e); }
        void removeLast() { V.pop back(); }
        void swap(const Position& p, const Position& q) { E e = *q; *q = *p; *p = e; }
};
```