COMP 2611 – Data Structures

Lab #8

Part 1: Max Priority Queues

Download the `Lab8-Part1-Files.zip` file from myeLearning. The zipped file contains a Dev-C++ project, `MaxPriorityQueue.dev`. The project contains the .h and .cpp files for building and manipulating a max-heap from Lab #7. The `MaxPriorityQueue.h` file contains the function prototypes for a max-priority queue which uses an underlying max-heap:

```
MaxPriorityQueue * initMaxPQ (int A[], int numElements);
```

Creates a max-priority queue from the array *A* containing *numElements* elements and returns the address of the max-priority queue created. *initMaxPQ* calls *initMaxHeapFromArray* and *buildMaxHeap* to create the underlying max-heap.

```
int maximumPQ (MaxPriorityQueue * maxPQ);
```

Returns the value of the highest priority element in the max-priority queue. This involves returning the largest value from the underlying max-heap.

```
int extractMaximumPQ (MaxPriorityQueue * maxPQ);
```

Removes the highest priority element from the max-priority queue and returns its value. This involves deleting the largest value from the underlying max-heap.

```
void insertMaxPQ (MaxPriorityQueue * maxPQ, int priority);
```

Inserts a new element in the max-priority queue with the given priority. This involves inserting a new key in the underlying max-heap.

```
void increasePriority (MaxPriorityQueue * maxPQ, int i, int newPriority);
```

Increases the priority of element *i* in the max-priority queue to the new priority. This involves changing the key of node *i* in the underlying max-heap to *newPriority*, if *newPriority* is greater than the existing key. The same procedure when inserting a new value in a max-heap must be followed to ensure that the key goes up the max-heap if it is bigger than its parent.

```
void displayMaxPQ (MaxPriorityQueue * maxPQ);
```

Displays the values in the max-priority queue by displaying the values in the underlying max-heap.

(a) Write the code for the functions above in `MaxPriorityQueue.cpp`.

(b) A max-priority queue is to be populated from the elements in the following array, *A*:

10	60	5	25	70	65	45	50	15	80

Draw the max-priority queue *maxPQ* as a binary tree after **each** of the following operations takes place, **in the order given** (i.e., 5 binary trees must be drawn):

```
(i)    maxPQ = initMaxPQ (A, 10);
(ii)   int x = maximumPQ (maxPQ);
(iii)  insertMaxPQ (maxPQ, 75);
(iv)   x = extractMaximumPQ (maxPQ);
(v)    increasePriority (maxPQ, 8, 90);
```

(c) Run the code in `UsingMaxPriorityQueue.cpp` and verify that your results in (b) correspond to the output generated by the program.

Part 2: Some Exam-Type Questions

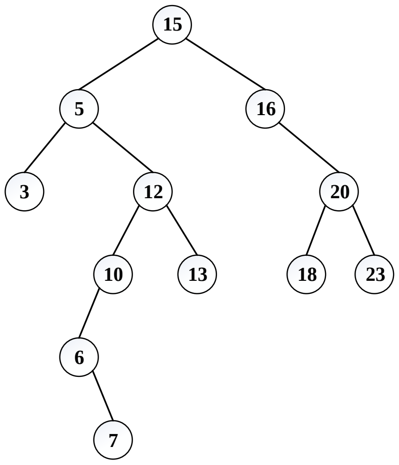
1. Assume that a node in a binary tree is defined as follows:

```
struct BTNode {
      int data;
      BTNode * left;
      BTNode * right;
      BTNode * parent;
};
```

a) A binary tree is said to be a *full binary tree* if every non-leaf node has exactly two non-empty subtrees. Write a **recursive** function with the following prototype which returns *true* if the binary tree passed as a parameter is a full binary tree, and *false* otherwise.

```
bool isFullBT (BTNode * bt);
```

b) Draw separate diagrams to show the binary search tree below *after* **each** of the following nodes has been deleted (assume that the deletions are independent of each other and each one starts from the same tree given in the diagram): 16, 15, 5.



c) For each of the following sequences, state whether it could be the sequence of values examined in searching for the number **36** in a binary search tree. If it cannot, state why.

7 25 42 40 33 34 39 36
92 22 91 24 89 20 35 36
95 20 90 24 92 27 30 36

2. A *Set* is a data structure which stores a collection of unique values. There is no requirement for the values to be sorted. You are required to implement a *Set* using a linked list.

Assume that a node in a linked list is declared as:

```
struct LLNode {
   int data;
   LLNode * next;
};
```

a) Declare a struct for a *Set*.

b) Using any of the usual functions provided by a linked list, write functions to:

i. Insert an element in a *Set*
ii. Determine if a *Set* contains an element
iii. Delete an element from a *Set*
iv. Find the intersection of two *Set*s
v. Find the union of two *Set*s

3. Suppose that the array *A* contains the values shown below:

▨	5	3	17	10	84	19	6	22	9
0	1	2	3	4	5	6	7	8	9

Draw the binary tree corresponding to *A* **before** the *buildMaxHeap* function is called and after **each** element is processed by the *buildMaxHeap* function.

End of Lab #8