Homework Set 3

This problem set is due **Monday February 20**
Analysis (in hard copy) by **class time**; Programming by **11:59PM**.
Solutions should be submitted in PDF form using LaTeX.
Note that the analysis problems *follow* the programming problems in this problem set.

---

1. Implement a Priority Queue using a (dynamically allocated) array-based binary max-heap data structure. Your priority queue must support the following public operations, where `T` is a `typedef` for an `int`, or is a template type parameter.

```
typedef int T;
const int INITSIZE = 1024;

// Priority Queue
class PQueue {
 public:
  PQueue();                            // Empty Constructor
  PQueue(int size);                    // Construct with size initial allocation
  PQueue(T * array, int n);            // Build PQueue from unsorted array size n
  ~PQueue();                           // Destructor
  void Clear();                        // Remove all elements from PQueue
  void Insert(const T & el);           // Insert element el into PQueue
  bool Dequeue(T & el);                // Remove and return max element
  bool FindMax(T & el) const;          // Return max element, leaving in place
  bool Empty() const;                  // Is BST empty?
  int Count() const;                   // Number of elements in PQueue (heap)
  bool Element(int i, T & el) const; // Return element at index i
  int Parent(int i) const;             // Compute parent index of element i
  int Left(int i) const;               // Compute left child index of element i
  int Right(int i) const;              // Compute right child index of element i
  void Breadth() const;                // Breadth first traversal, printing elements
  T * Sort(int newsize);               // Sort heap in ascending order, returning
                                       // sorted array; leaving empty priority queue
 private:                              // with allocation of newsize elements
  T * A;
  int sizeA;
  int sizeHeap;
}
```

Notes

(a) At construction time, the priority queue will allocate the heap storage based on either the passed size or the default INITSIZE. Note, however, that an Insert on a "full" priority queue should not fail, but should allocate a new array at twice the size of the old, copy from the old to the new, and then free the original array.

(b) Functions returning a value of an element do so through the reference parameter instead of the return value.

(c) Boolean return values in Dequeue, FindMax, and Element are used to indicate a valid index.

(d) Parent, Left, and Right should return -1 if the computed element is outside the valid index range for the current heap size.

(e) Normally, you would never expose the implementation details of a Priority Queue ADT by having public functions for computing tree-relative indices and retrieving an element by index, but these help our ability to drive test cases.

(f) The constructor giving a pointer to an unsorted array and the Sort() function are twisting beyond a pure Priority Queue to allow us to do heapsort without defining a new class. The building of the heap for heapsort would happen as a result of invoking the constructor (after creating and initializing the array of elements to be sorted), and the Sort function performs the other half of the heapsort algorithm in place in A. But an ascending sorted array is no longer a max-heap, so the Sort function passes back the pointer to the now-sorted array, and allocates a new array for an empty priority queue following the sort. (Kind of a hack, but it will shorten your development time!)

(g) I have not specified the helper functions you will need within this class, like HeapifyUp, and HeapifyDown, and BuildHeap, as I don't see a need for those to be public. Your implementation may define them as protected or private.

For completing this problem, I want you to implement an appropriate driver.cpp test program. This time, the driver will not be supplied to you, but you should be able to use significant parts of the infrastructure from homework 2 for this. Note that your analogue to a current "node" in the driver is just an integer index. Your driver should support the following operations:

- create, destroy
- insert, insertfile
- dequeue, clear
- empty, count
- root, parent, left, right
- key

- breadth
- quit

Like our BST driver program, create and destroy should take no arguments; insert should allow both one or a list of elements to be inserted. The root command should set the current index to 0 (with OK output if the PQ is non-empty, while the parent, left, and right commands change the current index appropriately with an output of the element value, while the key command retrieves the value of the current index (using the Element function). We only need the breadth first traversal, whose output characteristics should be the same as in our BST. And we will test/exercise the Sort external to this driver program.

Include with your submission a selection of text scripts and expected results. This need not be as comprehensive as the tests we generated as part of homework 2, but should give fairly good coverage. If everyone follows the same specification from above, I should be able to execute any of your inputs on any of your implementations and get the same result.

2. Experimentally analyze the heapsort algorithm and compare it to our other $O(n \lg n)$ algorithms for sorting.

3. Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for small enough n. Make sure your bounds are tight, and justify your answers.

   (a) $T(n) = 2T(n/2) + n^3$

   (b) $T(n) = 16T(n/4) + n^2$

   (c) $T(n) = T(n-1) + n$

   (d) $T(n) = 3T(n/2) + n \lg n$

   (e) $T(n) = 2T(n/2) + \frac{n}{\lg n}$ (ok to just show upper bound)

4. What are the minimum and maximum number of elements in a heap of height h?

5. Show that an n-element heap has height exactly $\lfloor \lg n \rfloor$ (Hint: you can use your answer to the previous question to give a succinct argument.)