CS 3100 – *Data Structures and Algorithms*

Project #6 - Array-based Heap

Learning Objectives

- Implement a data structure to meet given specifications.
- Design, implement, and use a priority queue implemented as an array-based heap.

Overview

Your task for this assignment is to build a priority queue implemented as a MAX-HEAP of integers that uses an expandable array for data storage.

The MaxHeap class

Your MaxHeap class should contain an **array** (not a vector) of integers. The initial size of the array should be #defined in your MaxHeap.h file as #define HEAP_MIN_SIZE 20. If the heap becomes full, the array size should be doubled. If the heap becomes less than half full (but larger than HEAP_MIN_SIZE) the size of the array should be halved.

As usual, your heap should be defined in two files: MaxHeap.h and MaxHeap.cpp. Your class should support the following operations:

- `void MaxHeap::offer(int value)` – Insert a new value into the heap. Duplicate keys are allowed.
- `int MaxHeap::poll()` – Removes and returns the maximum value in the heap. If the heap is empty, this method should throw an exception.
- `bool MaxHeap::isEmpty() const` – Returns `true` if the heap is empty, and `false` otherwise.
- `int MaxHeap::peek() const` – Returns the maximum value in the heap without removing it. If the heap is empty, this method should throw an exception.
- `vector<int> MaxHeap::sorted() const` – Creates and returns a vector of integers containing the heap elements sorted in largest to smallest order. If the heap is empty, this method should return an empty vector. *A good approach for implementing this method would be to copy the heap contents into a new vector, and then perform heapsort on the copy.*
- Your heap class should include the following two constructors:
  - `MaxHeap::MaxHeap()` - Creates an empty MaxHeap of size HEAP_MIN_SIZE.
  - `MaxHeap::MaxHeap(int * values, int count)` - Creates a new heap by copying the array `values` and then using the heapify algorithm to convert the values to a maxheap. This method must use heapify and must run in *O(n)* time.

- As usual, your class should have an appropriate copy constructor and should overload `operator=` to produce an independent copy of your heap.

- Your MaxHeap class should also overload `operator<<` such that `cout << myHeap;` prints the heap in the order in which it is stored in the array (NOT in sorted order).

- In addition to the methods listed above, you may create any additional private methods that help in your class implementation. Some possibilities include `int getParent(int); int getLeftChild(int);` and `int getRightChild(int);`

- You will also need to create a test harness that creates multiple heaps and thoroughly tests all of the above methods and requirements. Additionally, in order to work with my test harness, your method declarations **must match the method signatures given above exactly.**

Turn in and Grading

Please zip your entire project directory into a single file called Project6.zip.

This project is worth 50 points, distributed as follows:

Task	Points
`MaxHeap::MaxHeap()` correctly creates an empty heap. `MaxHeap::MaxHeap(int * values, int count)` correctly creates a heap in *O(n)* time by heapifying a copy of the array `values`.	5
`MaxHeap::offer` inserts items into the heap, maintaining the MAX-HEAP property at all times.	5
`MaxHeap::poll` removes and returns the maximum value in the heap, maintaining the MAX-HEAP property at all times, and throws and exception when called on an empty heap.	5
Your heap doubles in size when full. Your heap is never smaller than HEAP_MIN_SIZE, and never more than half-empty unless it cannot be halved without becoming smaller than HEAP_MIN_SIZE.	5
`MaxHeap::isEmpty` returns true when the heap is emtpy, and false otherwise.	2
`MaxHeap::peek` returns the maximum value in the heap without modifying the heap contents.	3
`MaxHeap::sorted` correctly creates and returns a sorted vector from the heap contents.	5
`MaxHeap::operator=` and your copy constructor produce independent copies of your heap.	5
`operator<<` is correctly overloaded for your heap class as described above.	5
Code is well organized, well documented, and properly formatted. Variable names are clear, and readable. Classes are declared and implemented in seperate (.cpp and .h) files.	5
Appropriate use of public and private class member data. No global variables or unnecessary member variables. Efficient and well-designed code. No memory leaks.	5