Lab 2-2: Lab Heaps

Goal: Master implementation of binary heap and heap sort

Answer the following questions (**they must be typed**) then submit to PolyLearn as **a PDFdocument**. You will be submitting two files to Canvas – this pdf and a .java file.

You can submit multiple files in a single assignment submission by clicking on the **+Add Another File** link that appears in the assignment submission window or modal. See the following video - https://www.youtube.com/watch?v=zQEdGYt-2O8 .

1. Is it possible to find an arbitrary element in a heap in less than linear time in the size of the heap using the standard implementation?

No, because with the standard implementation there is no way of identifying which subtree the element could be in like you can with a binary search tree. So, you would have to check the entire single heap in order to find the element. It would be like doing a linear search.

2. Write pseudo code for an algorithm that finds the k smallest elements of an unsorted list of n integers in O(n + (k*log n)) time.

```
Construct a MinHeap using Bottom- up method (O(n));
Result[k];
int i <- 0;
while i < k do
    j <- remove min item from MinHeap; (O(log n))
    Result[i] <- j;
    i <- i + 1;
return Result;
```

3. Give pseudo code for an O(n*log k) time algorithm that merges k sorted lists with a total of n elements into one sorted list. O(k*n) operations that would be required of the algorithm that simply merges the sorted lists.

```
Construct a MinHeap using the first index of the k sorted lists
Result[k]
while minheap is not empty do
    j <- remove min item from heap
    Result.append(j)
    If j was not the last element of its array then
        Insert next element of the array into min heap
Return Result
```

4. Given an array based MinHeap containing n elements and a real number x, efficiently determine whether the kth smallest element in the heap is greater than or equal to x. Your algorithm should be O(k) in the worst case, independent of the size of the heap. Hint: You do not have to find the kth smallest element; you need only determine its relationship to x.

```
For k -1 times do
    Remove min from minheap
Return (heap[0] >= x )
```

5. **Application: Median Maintenance**: Consider the following situation. You are presented with a sequence of numbers, one by one. Assume for simplicity they are all distinct. Each time you receive a new number your program must respond with the median (recall the median is the middle number, if there are 11 numbers it would be the 6th smallest or if 12 then either the 6th or 7th smallest) of all numbers received so far. In the next part of the course we will see a way to determine the median of a set of n numbers in O(n) time. However in this particular situation it can be done in O($log_2 n$) for each median that must be returned. Clearly describe how can this be done? Define any data structures required by your algorithm. You may use a picture to help clarity how your algorithm works.

```
Initialize a minheap
Initialize a maxheap
For i in arr do
    If maxheap is empty then
        Add i to maxheap
    Else if i < root of maxheap then
        Add i to maxheap
    Else do
        Add to min heap
    If abs(size of minheap – size of maxheap) >= 1 then
        remove root from bigger heap
        insert removed root to smaller heap
    If size of minheap + size of maxheap is even then
        median = (root of maxheap + root of minheap) / 2
    else
        median = root of the bigger heap
Return median
```

Lab 2-2 B: Heap implementation and heap sort

Goal: Be able to implement a Heap and use it in a heapsort utility function

Implement a Heap class to contain Integer

Eventually you may need to implement a Class **MyHeap (a min heap)** that contains arbitrary objects. But for this lab implement a min heap that contains integers. A MyHeap object will have a **capacity**. The capacity is the maximum number of items the heap can hold. It will also have a size. The heap **size** is the current number of objects in the heap

- Implement class MyHeap of integers that has:
  - Constructor creating an empty heap with default capacity = 50
    public MyHeap()
  - Constructor creating an empty heap with the capacity given as a parameter
    public MyHeap(int capacity)
  - Public method **buildHeap** that has a single explicit argument "**array of int**" and builds a heap using the **MyHeap object that is the implicit parameter**. It should return true if the build was successful and false if the capacity of the MyHeap object is not large enough to hold the "array of int" argument. **buildHeap** **must use** bottom up heap construction.
  - **boolean insert**(int) inserts the int argument into the heap.. Returns true if successful and false if not
  - int **findMin**(), returns the minimum value in the heap
  - int **deleteMin**(), deletes and returns the minimum value in the heap
  - boolean **isEmpty**(), returns true if the heap is empty, false otherwise
  - boolean **isFull**(), returns true if the number of items in the heap is equal to the capacity of the heap
  - Your **build, insert, and delete** methods should use one of public methods "void **driftDown**(int index)" or "void **driftUp(int index)**" . **Normally these would be private but make them public for testing purposes.**
  - Finally for testing purposes your MyHeap class should also contain the following:
    - **getHeapCap**() returns an integer that is maximum number of a entries **the heap** can hold.
    - **getHeapSize**() -- returns an integer that is the number of elements in the heap

Implement HeapSort

Implement Heap Sort as a method in the MyHeap class.

public static int[] heapSortDecreasing(int[]) the method is static since it is a meant to be a standalone sorting method for which the user should not need to know anything about heaps (information hiding).

This method should perform the sorting as discussed in the screencast, namely **in place**. It should build a heap from data in the int[] that is the explicit argument. Thus the storage for the heap where you put the data for processing will no longer be a heap after you have sorted the data using the same array. **It is not necessary to restore the heap to its original state.**

This takes an array of integers and returns an array with the integers in **decreasing** order. Note for both the input array and output array the data is stored beginning at index 0 but in the heap the data is stored in indices beginning at 1.

Submit to PolyLearn the class: **MyHeap**