Priority Queue
(Heap Implementation)

Purpose:

This worksheet is concerned with priority queues implemented using heaps. The several variants of heaps are the prototypical most efficient implementations of the abstract data type priority queues. Priority queues are useful in many applications. Typical operations for priority queues are more efficient when implemented using heaps compared to a list implementation. In particular, heaps are crucial in several efficient graph algorithms. We can also use heaps for sorting. Although heaps could in principle be implemented using pointers and dynamic memory allocation, they are usually implemented in arrays for reasons of efficiency.

A min heap is a specialized tree-based data structure that satisfies the *min heap property:* if *B* is a child node of *A*, then key(*A*) ≤ key(*B*). This implies that an element with the smallest key is always in the root node. Alternatively, if the comparison is reversed, the largest element is always in the root node, which results in a *max heap*.

Work on as many problems as you can; the remaining problems you can use for practice and to test your understanding of heaps.

1) Implement a C routine for inserting elements into a min heap, one element at a time. We start with an empty heap and insert one element at a time. We assume that there is an upper limit on the number of elements that will be in the heap. Write a corresponding main program.
2) Write a C function that removes the smallest element from the heap. Because of the heap property, the smallest element will always be at the root of the heap. Once removed, the heap property has to be re-established. This can be done by removing the very last element of the heap, placing it at the root node and adjusting the values in the nodes from the root all the way to a leaf.
3) Modify your C routine in Part 1 and 2 for inserting and deleting an element into a max heap so that it maintains the max heap property, i.e. if *B* is a child node of *A*, then key(*A*) ≥ key(*B*).
4) Assume that all elements are given a priority; we can apply a faster algorithm for building a heap. Implement the C function `Heapify()` that turns an array into a min heap with implicit tree structure.
5) Write a C function that searches for an element in the min heap and increases its value. Once the value has changed, the min heap property may no longer be satisfied; your function must re-establish the min heap property.
6) Write a C function that searches for an element in the min heap and decreases its value. Once the value has changed, the min heap property may no longer be satisfied; your function must re-establish the min heap property.
7) Write a C function that uses a heap to find the k-smallest element in an array.

Please note that from this worksheet, you need to submit the solution of the following question (8)

8) Write a program that will read the below data from the file and use a heap to sort the data. You can do sort as follows;
   - Build a max heap from the input data, so that the largest item is stored at the root of the heap.
   - Replace the first (largest) item of the heap with the last (smallest) item of the heap followed by reducing the size of heap by 1 and heapifying the root of the tree.
   - Data in the File: 3,2,5,8,16,1,9