1. (CLRS 6.1-1) What are the minimum and maximum number of elements in a heap of height $h$?

**Solution:** The minimum number of elements is $2^h$ and the maximum number of elements is $2^{h+1} - 1$.

2. (CLRS 6.1-4) Where in a min-heap might the largest element reside, assuming that all elements are distinct?

**Solution:** Since the parent is greater or equal to its children, the smallest element must be a leaf node.

3. (CLRS 6.1-5) Is an array that is in sorted order a min-heap?

Yes.

4. (CLRS 6.2-4) What is the effect of calling MIN-HEAPIFY($A$, $i$) for $i > size[A]/2$?

**Solution:** No effect. All nodes at index $i > size[A]/2$ are leaves.

5. (CLRS 6.5-3) Write pseudocode for the procedures HEAP-EXTRACT-MIN, HEAP-DECREASE-KEY and HEAP-INSERT that implement a min-priority queue with a min-heap.

**Solution:**

```plaintext
HEAP-MINIMUM(A)
   return A[1]

HEAP-EXTRACT-MIN(A)
   if heap-size[A] < 1
      then error ‘‘heap underflow’’
   min <- A[1]
   MIN-HEAPIFY(A,1)
   return min
```
HEAP-DECREASE-KEY(A,i,key)
    if key > A[i]
        then error ‘‘new key is larger than current key’’
    A[i] <- key
    while i > 1 and A[parent(i)] > A[i]
        do exchange A[i] <-> A[parent(i)]
        i <- parent(i)

MIN-HEAP-INSERT(A,key)
    heap-size[A] <- heap-size[A] + 1
    A[heap-size[A]] <- +inf
    HEAP-DECREASE-KEY(A,heap-size[A],key)

6. (CLRS 6.5-8) Give an \( O(n \lg k) \)-time algorithm to merge \( k \) sorted lists into one sorted list, where \( n \) is the total number of elements in all the input lists. (Hint: use a min-heap for \( k \)-way merging.)

Solution: The straightforward solution is to pick the smallest of the top elements in each list, repeatedly. This takes \( k - 1 \) comparisons per element, in total \( O(k \cdot n) \).

As the hint suggests, the idea for the “improved” solution is to keep the smallest element from each list in a heap; each element is augmented with the index of the lists where it comes from. We can perform a DeleteMin on the heap to find and delete the smallest element and insert the next element from the corresponding list.

Analysis: It takes \( O(k) \) to build the heap; for every element, it takes \( O(lg k) \) to DeleteMin and \( O(lg k) \) to insert the next one from the same list. In total it takes \( O(k + n \lg k) = O(n \lg k) \).

7. (CLRS 9.3-6) Give an \( O(n \lg k) \) algorithm to find the \( k - 1 \) elements in a set that partition the set into (approx.) \( k \) equal-sized sets \( A_1, A_2, \ldots A_k \) such that all elements in \( A_i \) are smaller than all elements in \( A_{i+1} \).

Solution: For simplicity, assume that \( k \) is a power of 2.

\[
k\text{-PARTITION}(A, p, r, k)\]
if \( k > 1 \) then
    q = SELECT(A, (p+r)/2)
    output q
    k\text{-PARTITION}(A, p, (p+r)/2, k/2)
    k\text{-PARTITION}(A, (p+r)/2+1, r, k/2)
End.

Analysis: \( T(n, k) = 2T(n/2, k/2) + \Theta(n) \), and \( T(n/k, 1) = 1 \) has solution \( T(n) = \Theta(n \lg k) \).
8. (CLRS 9-1) Given a set of \( n \) numbers, we wish to find the \( i \) largest in sorted order using a comparison-based algorithm. Find the algorithm that implements each of the following methods with the best asymptotic worst-case running time, and analyze the running times of the algorithms on terms of \( n \) and \( i \).

(a) Sort the numbers, and list the \( i \) largest.

**Solution:** Use Mergesort, or Quicksort with median as pivot. It takes \( O(n \lg n) \) to sort and \( O(i) \) to list, in total \( O(n \lg n) \).

(b) Build a max-priority queue from the numbers, and call EXTRACT-MAX \( i \) times.

**Solution:** Building a heap takes \( O(n) \), and EXTRACT-MAX costs \( O(\lg n) \). In total this algorithm takes \( O(n + i \lg n) \).

(c) Use a SELECT algorithm to find the \( i \)th largest number, partition around that number, and sort the \( i \) largest numbers.

**Solution:** This takes \( O(n) \) to select the \( i \)th largest and partition around it, and \( O(i \lg i) \) to sort, in total \( O(n + i \lg i) \).