Heap Data Structure Practice Problems

1. What are the minimum and maximum numbers of elements in a heap of height h?
Solution: A heap is a complete binary tree, which means that all levels of the tree are fully filled except possibly the last level.

- The maximum number of elements in a heap of height h occurs when the heap is completely filled. Thus, the maximum number of elements is:

$$\text{Max elements} = 2^{h+1} - 1$$

- The minimum number of elements in a heap of height h occurs when all levels except the last one are completely filled and the last level has only one element. Since the first $h-1$ levels are fully filled, there are $2^h - 1$ nodes there and one node at level h. Thus, the minimum number of elements is:

$$\text{Min elements} = 1 + (2^h - 1) = 2^h$$

2. Where in a max-heap might the smallest element reside, assuming that all elements are distinct?
Solution: In a max-heap, the smallest element is guaranteed to reside in one of the leaf nodes because all internal nodes must be greater than or equal to their children. The leaf nodes are at the lowest level or in the second-to-last level, depending on the number of elements. Thus, the smallest element will reside at one of the leaf nodes, which are the children of the bottom-most non-leaf nodes.

3. At which levels in a max-heap might the kth largest element reside, for $2 \le k \le \frac{n}{2}$, assuming that all elements are distinct?
Solution: In a max-heap, the largest element is at the root, which is at the first level. The second largest element is one of the children of the root, located at the second level. Assume the kth largest element is denoted by t. The element t cannot reside at levels $k+1$ or below. This is because all nodes on the path from the root to t are larger than t (due to the properties of a max-heap), and if there are more than k nodes on this path, then t cannot be the kth largest element. Therefore, the kth largest element resides between levels 2 and k.

4. Is an array sorted in ascending order always a min-heap?
Solution: Yes, since the array A is sorted in ascending order, for each $i < j$, we have $A[i] \le A[j]$. In particular, for every index i, its children at indices $2i+1$ and $2i+2$ (if they exist) satisfy $A[i] \le A[2i+1]$ and $A[i] \le A[2i+2]$. This ensures that the min-heap property holds throughout the array, meaning that any sorted array in ascending order is always a valid min-heap.

5. Rewrite the following remove function using recursion.

Solution: The remove operation in a heap is typically used to remove the root (the minimum or maximum element). The following is a recursive implementation of the remove function for a min-heap:

Algorithm 1 Remove from Min-Heap

```
procedure REMOVE
    temp ← $A[0]$
    $A[0] \leftarrow A[n-1]$
    $i \leftarrow 0$
    while True do
        $s \leftarrow i$        ▷ Store the index of the smaller child if $A[i]$ is not less than both children
        $l \leftarrow 2i+1$        ▷ Left child's index
        $r \leftarrow 2i+2$        ▷ Right child's index
        if $l < n$ and $A[l] < A[s]$ then
            $s \leftarrow l$
        end if
        if $r < n$ and $A[r] < A[s]$ then
            $s \leftarrow r$
        end if
        if $s \neq i$ then
            Swap($A[i], A[s]$)
            $i \leftarrow s$
        else
            break        ▷ If $s$ didn't change, $A[i]$ was smaller than both children and we are done
        end if
    end while
    return temp
end procedure
```

Algorithm 2 Remove from Min-Heap (Recursive)

```
procedure REMOVE(A, n, i)
    if $i \geq n$ then
        return None        ▷ Base case if index is out of bounds
    end if
    temp ← $A[0]$
    $l \leftarrow 2i+1$        ▷ Left child's index
    $r \leftarrow 2i+2$        ▷ Right child's index
    $s \leftarrow i$        ▷ Index of the smaller child if $A[i]$ is not less than both children
    if $l < n$ and $A[l] < A[s]$ then
        $s \leftarrow l$
    end if
    if $r < n$ and $A[r] < A[s]$ then
        $s \leftarrow r$
    end if
    if $s \neq i$ then
        Swap($A[i], A[s]$)
        return REMOVE(A, n, s)        ▷ Recurse to the child node
    end if
    return temp
end procedure
```

6. Show how to implement a first-in, first-out queue with a priority queue. Show how to implement a stack with a priority queue.

Solution: - To implement a queue using a priority queue, we can assign priorities based on the arrival time of elements (e.g., a lower priority for earlier elements). We can dequeue elements in the order of their arrival.

Implementation:

```
enqueue(element):
    priority = current_time
    priority_queue.insert(element, priority)

dequeue():
    return priority_queue.extract_min()
```

- To implement a stack using a priority queue, we can assign a priority based on the order of insertion, where the most recently added elements have the highest priority.

Implementation:

```
push(element):
    priority = current_time
    priority_queue.insert(element, priority)

pop():
    return priority_queue.extract_max()
```

7. A d-ary heap is like a binary heap, but non-leaf nodes have d children instead of two.

a. **Describe how to represent a d-ary heap in an array.**

In a d-ary heap, each node can have up to d children. The children of a node at index i can be represented by the following indices:

$$\text{Left child index: } d \cdot i + 1, \text{ to } d \cdot i + d$$

The parent of a node at index i is located at index:

$$\text{Parent index: } \left\lfloor \frac{i-1}{d} \right\rfloor$$

b. **Using Θ-notation, express the height of a d-ary heap of n elements in terms of n and d.**

The height of a heap with n elements and d children per node is:

$$\text{Height} = \lceil \log_d n \rceil = \Theta(\log_d n)$$