# Algorithm Analysis and Data Structures

## Quiz 2

Submitted By: Aman Pandita

Question 2: Perform Quick Sort on the following array, taking the pivot as the middle element every time. Show the intermediate output after each timestep

arr = [2, 1, 5, 3, 4, 6]

**Answer:**

The initial array is **[2, 1, 5, 3, 4, 6].**

1. First partitioning (pivot: 3):
   Left sub-array: **[2, 1]** , Pivot: **3**, Right sub-array: **[5, 4, 6]**

   Intermediate array: **[2, 1, 3, 5, 4, 6]**

2. Sort the left sub-array (pivot: 1):
   Left sub-array: **[]** , Pivot: **1** , Right sub-array: **[2]**

   Intermediate array: **[1, 2, 3, 5, 4, 6]**

3. Sort the right sub-array (pivot: 4):
   Left sub-array: **[]**, Pivot: **4**, Right sub-array: **[5, 6]**

   Intermediate array: **[1, 2, 3, 4, 5, 6]**

## Question 2

Consider a Divide and Conquer algorithm that divides each problem into 2 sub-problems of size 3n/4 each. Write the recurrence relation for this algorithm, and compute the time complexity using Master Theorem.

### Answer:

The recurrence relation for the given Divide and Conquer algorithm can be written as:

$T(n) = 2T(3n/4) + f(n)$

Where $T(n)$ is the time complexity of the problem of size n, and $f(n)$ is the cost of dividing the problem and combining the results of sub-problems.

Using the Master Theorem to solve this recurrence relation, we compare the function $f(n)$ with $n^{\log\_b(a)}$, where $a = 2$ and $b = 4/3$.

The Master Theorem states that if $f(n) = O(n^c)$ where $c < \log\_b(a)$, then $T(n) = \Theta(n^{\log\_b(a)})$.

In this case, if $f(n)$ is $O(n^c)$ where $c < \log(4/3)*(2)$, then the time complexity:

***$T(n) = \Theta(n^{\log (4/3)(2)})$.***

The value of log (4/3) *(2) is approximately 2.41. Therefore, if $c < 2.41$, the time complexity of the algorithm is ***$\Theta(n^{2.41})$.***

## Question 3

Given a sequence of numbers: 19, 6, 8, 11, 4, 5

a) Draw a binary min-heap (in a tree form) by inserting the above numbers and reading them from left to right.

b) Show a tree that can be the result after the call to deleteMin() on the above heap.

c) Show a tree after another call to deleteMin().

### Answer:

a) Binary Min-Heap: The min-heap is constructed by inserting the numbers in the given order.
The initial heap after inserting all the numbers will look like this:

```
        4
      /    \
    6        5
   / \      /
  19 11    8
```

b) After deleteMin(): The min-heap after calling deleteMin() (removing 4) will be restructured to maintain the heap property. The heap may look like this:

```
    5
   / \
  6   8
 / \
19 11
```

c) After another deleteMin(): Another call to deleteMin() (removing 5) will restructure the heap again. The heap may look like this:

```
    6
   / \
  11 8
 /
19
```

## Question 4

What is the big-Oh time complexity of getting a sorted array out of a max heap? Justify your answer.

### Answer:

Let's calculate time step by step:

**Build a Max Heap:** This operation has a time complexity of O(n), where nn is the number of elements in the array.

**Sort the Heap: This step has multiple sub steps:**

1. **Removing the Maximum Element:** This operation is O(1) — it's just removing the root of the heap.
2. **Heapify:** After each removal, the heap property may be violated, so we need to perform heapify operation to maintain the max heap property.
   Heapify has a time complexity of O (log n).

Since we perform the heapify operation n times, the total time complexity for this step is n * O (log n) = O (n log n).

Combining these steps, the overall time complexity of getting a sorted array out of a max heap is O (n + n log n). However, the (n log n) term dominates, so the final time complexity is:

<u>***O(n log n).***</u>