# Exercise Sheet 5
# COMS10017 Algorithms 2020/2021

## 1 Heap Sort

Consider the following array $A$:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 9 | 10 | 14 | 8 | 7 | 2 | 1 | 7 |

1. Interpret $A$ as a binary tree as in the lecture (on heaps).
2. Run Create-Heap() on the initial array. Give the sequence of node exchanges. Draw the resulting heap.
3. What is the worst-case runtime of Heapify()?
4. Explain how heap sort uses the heap for sorting. Explain why the algorithm has a worst-case runtime of $O(n \log n)$.

## 2 Merge Sort

Illustrate how the Mergesort algorithm sorts the following array using a recursion tree:

$$11 \quad 7 \quad 2 \quad 5 \quad 9 \quad 6 \quad 1$$

## 3 Quick Sort

Consider an array $A$ of length $n$ so that $A[i] = n - i$. For example, for $n = 10$ we are given the following array:

$$A = 10 \quad 9 \quad 8 \quad 7 \quad 6 \quad 5 \quad 4 \quad 3 \quad 2 \quad 1\ .$$

The goal is to sort $A$ in non-decreasing order which in this case is equivalent to reversing it. The pivot plays a central role in Quicksort. Consider the following options as a choice for the pivot:

1. The right-most position.
2. The element at position $\lceil n/2 \rceil$.
3. The left-most position.

For each of these options, what is the runtime of Quicksort on $A$? State your answers using $\Theta(.)$-notation. Justify your answers.

# 4 Circularly Shifted Arrays

Suppose you are given an array $A$ of length $n$ of **distinct** (all integers are different) sorted integers that has been circularly shifted by $k$ positions to the right. For example, $[35, 42, 5, 15, 27, 29]$ is a sorted array that has been circularly shifted by $k = 2$ positions, while $[27, 29, 35, 42, 5, 15]$ has been shifted by $k = 4$ positions. Describe an $O(\log n)$ time algorithm that allows us to find the maximum element.

# 5 Optional and Difficult Questions

Exercises in this section are intentionally more difficult and are there to challenge yourself.

## 5.1 Closest Pair of Points (hard!)

The input consists of two arrays of $n$ real numbers $X, Y$ and represent $n$ points with coordinates $(X[0], Y[0]), (X[1], Y[1]), \ldots, (X[n-1], Y[n-1])$. Give a divide-and-conquer algorithm that finds the pair of points that are closest to each other, i.e., the output consists of a two indices $i, j$ such that $(X[i], Y[i])$ and $(X[j], Y[j])$ are the two closest points.

*Hint:* This algorithm is similar to the algorithm given for the Maximum Subarray problem. The combine step is tricky here. It is easy to give a combine step that runs in $O(n^2)$ time. How can we get a combine step that runs in $O(n)$ time?