# Week 4 Applied Sheet

**Objectives:** The applied sessions, in general, give practice in problem solving, in analysis of algorithms and data structures, and in mathematics and logic useful in the above.

**Instructions to the class:** You should actively participate in the class.

**Instructions to Tutors:** The purpose of the applied class is not to solve the practical exercises! The purpose is to check answers, and to discuss particular sticking points, not to simply make answers available.

**Supplementary problems:** The supplementary problems provide additional practice for you to complete after your applied class, or as pre-exam revision. Problems that are marked as **(Advanced)** difficulty are beyond the difficulty that you would be expected to complete in the exam, but are nonetheless useful practice problems as they will teach you skills and concepts that you can apply to other problems.

## Problems

**Problem 1.** In the seminars, a probability argument was given to show that the average-case complexity of Quicksort is $O(n\log(n))$. Use a similar argument to show that for an input array of $n$ elements the average-case complexity of Quickselect with random pivots is $O(n)$.

**Problem 2.** Devise an algorithm that, given a box with $n$ different locks and $n$ corresponding keys, matches the keys and locks in average-case time complexity $O(n\log n)$. Each lock matches only one key, and each key matches only one lock. You can try a key in a lock to determine whether the key is larger than, smaller than or fits the lock. However, you cannot compare two keys or two locks directly.

**Problem 3.** Devise an algorithm that given a sequence of $n$ unique integers and some integer $1 \leq k \leq n$, finds the $k$ closest numbers to the median of the sequence. Your algorithm should run in $O(n)$ time. You may assume that Quickselect runs in $\Theta(n)$ time (achievable by using median of medians to find good pivots).

**Problem 4.** One common method of speeding up sorting in practice is to sort using a fast sorting algorithm like Quicksort or Mergesort until the subproblems sizes are small and then to change to using insertion sort since insertion sort is fast for small, nearly sorted lists. Suppose we perform Mergesort until the subproblems have size $k$, at which point we finish with insertion sort. What is the worst-case running time of this algorithm?

**Problem 5.** A subroutine used by Quicksort is the partitioning function which takes a list and rearranges the elements such that all elements $\leq p$ come before all elements $> p$ where $p$ is the pivot element. Suppose one instead has $k \leq n$ pivot elements and wishes to rearrange the list such that all elements $\leq p_1$ come before all elements that are $> p_1$ and $\leq p_2$ and so on..., where $p_1, p_2, ..., p_k$ denote the pivots in sorted order. The pivots are not necessarily given in sorted order in the input.

(a) Describe an algorithm for performing $k$-partitioning in $O(nk)$ time. Write pseudocode for your algorithm.

(b) Describe a better algorithm for performing $k$-partitioning in $O(n\log(k))$ time. Write pseudocode for your algorithm.

(c) Is it possible to write an algorithm for $k$-partitioning that runs faster than $O(n\log(k))$?

**Problem 6.** Suppose for an array of unique elements Bob implements Quicksort by selecting the average element of the sequence (or the closest element to it) as the pivot. Recall that the average is the sum of all of the elements divided by the number of elements. What is the worst-case time complexity of Bob's implementation? Describe a family of inputs that cause Bob's algorithm to exhibit its worst-case behaviour.

**Problem 7.** Consider a generalisation of the median finding problem, the *weighted median*. Given $n$ unique elements $a_1, a_2, ...a_n$ each with a positive weight $w_1, w_2, ..., w_n$ all summing up to 1, the weighted median is the

element $a_k$ such that

$$\sum_{a_i<a_k} w_i \leq \frac{1}{2} \qquad \text{and} \qquad \sum_{a_i>a_k} w_i \leq \frac{1}{2}$$

Intuitively, we are seeking the element whose cumulative weight is in the middle (around $\frac{1}{2}$). Explain how to modify the Quickselect algorithm so that it computes the weighted median. Give pseudocode that implements your algorithm.

## Supplementary Problems

**Problem 8.** Write pseudocode for a version of Quickselect that is iterative rather than recursive.

**Problem 9.** Write an algorithm that given two sorted sequences $a[1..n]$ and $b[1..m]$ of unique integers finds the $k^{\text{th}}$ order statistic of the union of $a$ and $b$

1. Your algorithm should run in $O(\log(n)\log(m))$ time.
2. **(Advanced)** Your algorithm should run in $O(\log(n)+\log(m))$ time.