Week 4 Tutorial Sheet

(To be completed during the Week 4 tutorial class)

Objectives: The tutorials, 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. The preparation problems are not assessed. We strongly recommend that everyone tries to solve the preparation problems before the tutorial as you will benefit the most from the tutorial if you come properly prepared. However, you can still attend the tutorial if you have not solved those problems beforehand.

Instructions to Tutors: The purpose of the tutorials 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 tutorial, 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.

Implementation checklist

It will be most beneficial for your learning if you have completed this checklist **before** the tutorial.

By the end of week 4, write Python code for:

1. **Quicksort** (see Problem 6)
2. **Quickselect** (see Problem 7)

Tutorial Problems

Problem 1. (Preparation) What are the worst-case time complexities of Quicksort assuming the following pivot choices:

(a) Select the first element of the sequence.

(b) Select the minimum element of the sequence.

(c) Select the median element of the sequence.

(d) Select an element that is greater than exactly 10% of the others.

Describe a family of inputs that cause Quicksort to exhibit its worst case behaviour for each of these pivot choices.

Problem 2. Suppose that 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 3. In the lectures, 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 the average-case complexity of Quickselect is $O(n)$.

Problem 4. 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 $O(n)$ time (achievable by using median of medians).

Problem 5. 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 6. Write a Python function that implements the Randomised Quicksort algorithm (Quicksort with random pivot selection).

Problem 7. Implement Quickselect with random pivot choice. Modify your implementation from problem 6 so that it uses Quickselect to choose a median pivot.

Problem 8. Compare your solutions to problem 6 and 7, and see which one performs better for randomly generated lists.

Problem 9. 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 $O(n\log(k))$?

Supplementary Problems

Problem 10. Implement the median-of-medians algorithm for Quickselect as described in the lecture notes. Use this algorithm to select a pivot for Quicksort and compare its performance against the previous pivot-selection methods.

Problem 11. Write pseudocode for a version of Quickselect that is iterative rather than recursive.

Problem 12. 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

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

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.

Problem 13. (Advanced) Write an algorithm that given two sorted sequences $a[1..n]$ and $b[1..m]$ of unique integers finds the k^{th} order statistic of the union of a and b

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