CS 340
Assignment 3
Total: 100pts

Dr. Malek Mouhoub

Exercise 1 : *exercise 7.41 page 310* [8pts]

1. Prove that any comparison-based algorithm to sort 4 elements requires 5 comparisons.
2. Give an algorithm to sort 4 elements in 5 comparisons.

Exercise 2 : *exercise 7.42 page 310* [8pts]

1. Prove that 7 comparisons are required to sort 5 elements using any comparison-based algorithm.
2. Give an algorithm to sort 5 elements with 7 comparisons.

Exercise 3 : *exercise 7.48 page 310 (first 2 questions)* [8pts]

We are given an array that contains N numbers. We want to determine if there are two numbers whose sum equals a given number K. For instance, if the input is 8, 4, 1, 6 and K is 10, then the answer is yes (4 and 6). A number may be used twice. Do the following:

1. Give an $O(N^2)$ algorithm to solve this problem.
2. Give an $O(N \log N)$ algorithm to solve this problem. (*Hint: Sort the items first. After that is done, you can solve the problem in linear time.*)

Exercise 4 [8pts]

Suppose that the splits at every level of quicksort are in the proportion 1- a to a, where 0 < a < 1/2 is a constant. Show that the minimum depth of a leaf in the recursion tree is approximately - log(n) / log(a) and the maximum depth is approximately -log(n)/log(1-a) (don't worry about integer round-off).

Exercise 5 [8pts]

The running time of quicksort can be improved in practice by taking advantage of the fast running time of insertion sort when its input is "nearly" sorted. When quicksort is called on a subarray with fewer than k elements, let it simply return without sorting the subarray. After the top-level call to quicksort returns, run insertion sort on the entire array to finish the sorting process. Argue that this sorting algorithm runs in O(nk + n log(n/k)) expected time. How should k be picked, both in theory and in practice ?

Programming Part : Sorting Algorithms [60pts]

The goal here is a comparative experimental study, in terms of running time, of the following 4 sorting algorithms.

- Insertion sort.
- Mergesort.
- Quicksort.
- Quicksort as shown in exercise above (we will call this algorithm Quick-insertion).

1. Implement the 4 algorithms above within the same program. The input and output are described as follows.

   Input: 2 parameters N (number of random naturals to sort) and K (used by Quick-insertion).

   Output: Display the list of N randomly generated naturals. For each algorithm display the list of sorted numbers + corresponding running time.

2. Find the pair of values (N, K) where :

   (a) Quick-insertion outperforms all the other algorithms.

   (b) Insertion sort outperforms all the other algorithms.

   (c) Quicksort outperforms all the other algorithms.

Marking scheme

1. Readability : 10pts
2. Compiling and execution process : 10pts
3. Correctness : 40pts

Hand in

Written Part

- Submit electronic version : pdf(preferred) or word via URCourses. Your file should be named assign3.pdf (or assign3.doc). At the top of the file you should write your first name, last name and ID #.

Programming Part

Single file submission

Using URCourses, submit the file containing the C++ code of the programming part **assign3.cpp**. At the top of this file add the following comments :

1. Your first name, last name and ID #,
2. the compiling command you have used,
3. an example on how to execute the program,
4. and other comments describing your program.

Multiple files submission

Submit all files required by the programming part:

1. README (file including your name and ID #; and explaining the compilation and execution of your program including the format of input and other details)
2. headers (.h)
3. implementations (.cpp)
4. the Makefile :
   - should be named "**makefile**". In the makefile, the generated executable should be named : "**assign3**"

You can give any name to your source files. The marker will only have to run "**make**" to compile your program and "**assign3**" to execute it.