EC330 Applied Algorithms and Data Structures for Engineers
Fall 2018

Homework 4

Out: October 5, 2018
Due: October 17, 2018

This homework has a written part and a programming part. Both are due at 8:59 am on October 17. You should submit both parts on Blackboard. For the written part, you should submit a single PDF file containing either typeset answers or scanned copies of hand-written answers. Make sure you write your answers clearly. For the programming part, your code should be easy to read and understand, and demonstrate good code design and style. Your program must compile and run on the lab computers.

1. "Does it sort?" [20 pt]

For the following algorithm:

- **i.** Provide the result when the algorithm is run on the array [8, 7, …, 3, 2, 1].
- **ii.** Does the algorithm correctly sort all arrays A of size *n* containing only positive integers, for $n \geq 8$?
- **iii.** If the answer to the previous question is "yes", provide a short explanation and a worst-case (asymptotic) running time. If "no" give a small counterexample.

a)
```
sortA(Array A[0..n-1])
   for i = 0 to n/2
      for j = n/2+1 to n-1
         if (A[i] <= A[j])
            swap(A[i], A[j])
   return A;
```

b)
```
sortB(Array A[i..j]) // call sortB(A[0..n-1]) to sort A[0..n-1]
   if (i == j)
      return
   mid = (i + j)/2
   sortB(A[i..mid])
   sortB(A[mid+1..j])
   if (A[i] < A[mid+1])
      swap(A[i], A[mid+1])
   return A;
```

2. Programming [80 pt]

Make sure to write your name and BU ID in a comment at the top of the program, along with your collaborator's name and BU ID, if any. To use the compiler on the lab computers, run "*module load gcc*" first. **Read the instructions carefully**!

a) **[40 pt]** In this problem, you are tasked with sorting a string in *increasing* order based on the number of occurrences of characters. If there is a tie, output them based on *alphabetical order*, e.g., 'a' before 'e'. You can assume that all the characters are lower-case letters (so a total of 26 possible types of characters). Below are some example inputs and the corresponding expected outputs.

Input1: "engineers"
Output2: "girsnneee"

Input2: "engineering"
Output2: "rggiieeennn"

Implement *sortByFreq* in *sort.h*. You are allowed to use *only* the libraries included in *sort.h* (you may not need to use all of them). You are welcome to implement your own data structure or use built-in ones like *int[]*. You *cannot* use any of the built-in sort functions including those provided by the <*algorithm*> library and will *need to implement your own* (you should write this as a separate function and call it from *sortByFreq*).

In the *written* part of your submission, *state and justify the time and space complexity* of your algorithm (in terms of *n* which is the length of the input string).
Your code will be graded not only on correctness but also on efficiency and memory footprint. For efficiency and memory consideration, you should expect long string inputs (i.e. large *n*). You can benchmark your implementation against our sample solution *sort2a.o*. To link the object file, you will need to modify *sort.h* so that it contains only the function *declaration* of *sortByFreq*.

The *Problem2a.cpp* file is given as a sample test program for your implementation. You can compile with the test program using:
> *g++ -std=c++11 (-o myProgram) Problem2a.cpp*

Submit only *sort.h*. Document your code clearly.

b) **[40 pt]** Recall in an earlier lecture, I claimed that we could have a *divide-and-conquer* algorithm that is faster than $O(n^2)$ for finding the closest pair of points on a 2D plane. We will learn to implement this algorithm in this homework.

Closest pair of points:
Input: A set of n points on the 2D plane, $\{p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)\}$
Output: The closest pair of points p_i and p_j $(i \neq j)$ such that the distance between them, that is,

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2},$$

is the smallest.

For simplicity, let us assume that n is a power of two, and all the x and y coordinates are distinct.

Here is how the algorithm works:

- Find a value x for which exactly half of the points have $x_i < x$ (or $x_i \leq x$) and half have $x_i > x$. Split the points into two groups L and R using x.
- Recursively find the closest pair in L and in R. Say these pairs are (p_L, q_L) in L, and (p_R, q_R) in R, with distances d_L and d_R respectively. Let d be the smaller of these two distances.
- We still need to check if there is a point in L and a point in R such that the distance between them is less than d. To this end, we can discard all points with $x_i < x - d$ or $x_i > x + d$ and sort the remaining points by the y-coordinate.
- Go through this list of points, and for each point, compute its distance to the *seven* subsequent points in the list. Let (p_M, q_M) be the closest pair found this way.
- The final answer is the closest pair among the three pairs (p_L, q_L), (p_R, q_R) and (p_M, q_M).

In the *written* part of your submission, *write down and solve* the recurrence relation for this algorithm.

Implement this algorithm as *findClosestPair* in *closest.h*. Think about what to do when the number of points in a partition becomes small as the recursion progresses. As opposed to part a), you can use any function from the C++ Standard Library for this problem. The *Problem2b.cpp* file is given as a sample test program for your implementation. You can compile with the test program using:

> *g++ -std=c++11 (-o myProgram) Problem2b.cpp*

Bonus [10 pt]: Improve your code to bring the running time down to $O(n \log n)$. Implement it in a new function called *findClosestPairOptimal* (you can reuse your utility functions for the original code as much as possible). Justify your improvement in the written part of your submission. *Hint*: Recall the recurrence relation I gave in lecture for this problem.

Submit only *closest.h*. Document your code clearly.