Adavanced Data Structures and Algorithms @ IIIT Sri City
Fall 2019

Assignment 1
Due: August 10, 2019, 5pm

Please answer each of the following problems. Refer to the course webpage for the **collaboration policy,** as well as for **helpful advice** for how to write up your solutions.

Note on notation: On a few problems in this homework set, we use "big-O" notation to state the problem. For all of the problems where O appears, it is fine to to take the definition of "$O(n)$" very informally to mean "grows (at most) roughly linearly in n:" for example, $100 \cdot n$ grows roughly linearly with n, so $100 \cdot n = O(n)$. Similarly, $100 \cdot n + 100 = O(n)$. But n^2 does *not* grow roughly linearly with n, it grows much faster, so $n^2 \neq O(n)$. And $\sqrt{n}$ grows much more slowly than n, so we would still say $\sqrt{n} = O(n)$. We use similar notation for other functions, like $O(\log(n))$.

1. **What do you want from this course?** (4 points) What skills do you hope to learn, what topics do you think well cover that will stick with you five, or ten years down the road? Write *at least three sentences* describing how you expect to benefit from taking this class. The reason for this question is twofold. First, it will help me understand what you want. Second, keep your answer to this question in mind throughout the semester as motivation if the going gets tough!

2. **New friends.** (16 points) Each of n users spends some time on a social media site. For each $i = 1, \ldots, n$, user i enters the site at time a_i and leaves at time $b_i \geq a_i$. You are interested in the question: how many distinct pairs of users are ever on the site at the same time? (Here, the pair (i, j) is the same as the pair (j, i)).

   Example: Suppose there are 5 users with the following entering and leaving times:

User	Enter time	Leave time
1	1	4
2	2	5
3	7	8
4	9	10
5	6	10

   Then, the number of distinct pairs of users who are on the site at the same time is three: these pairs are $(1, 2)$, $(4, 5)$, $(3, 5)$.

   (a) (3+3 pts) Given input $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ as above, there is a straightforward algorithm that takes about[1] n^2 time to compute the number of pairs of users who are ever on the site at the same time. (a) Give this algorithm and explain why it takes time about n^2. (b) Write the code in C programming language for this algorithm.

   (b) (5+5 pts) (a) Give an $O(n \log(n))$-time algorithm to do the same task and analyze its running time. (**Hint:** consider sorting relevant events by time). (b) Write the code in C programming language for this algorithm.

[1] Formally, "about" here means $\Theta(n^2)$, but you can be informal about this.

3. **Proof of correctness.** (4+6 points) Consider the following algorithm that is supposed to sort an array of integers. (a) Provide a proof that this algorithm is correct. (**Hint**: you may want to use more than one loop invariant.), and (b) Write the code in C and validate through different inputs.

```
# Sorts an array of integers.
Sort(array A):
    for i = 1 to A.length:
        minIndex = i
        for j = i + 1 to A.length:
            if A[j] < A[minIndex]:
                minIndex = j
        Swap(A[i], A[minIndex])

# Swaps two elements of the array.  You may assume this function is correct.
Swap(array A, int x, int y):
    tmp = A[x]
    A[x] = A[y]
    A[y] = tmp
```

4. **Needlessly complicating the issue.** (20 points)

   (a) (3pts) Give a linear-time (that is, an $O(n)$-time) algorithm for finding the minimum of n values (which are not necessarily sorted).

   (b) (2pts) Argue that any algorithm that finds the minimum of n items must do at least n operations in the worst case.

   (c) (3pts) Write the C program for this linear-time algorithm.

   Now consider the following recursive algorithm to find the minimum of a set of n items.

Algorithm 1: `findMinimum`

Input: List $A = [a_1, \ldots, a_n]$ of n items
Output: $\min_i \{a_1, \ldots, a_n\}$
if n=*1* **then**
  return ________
$A_1 = A[0 : n/2]$
$A_2 = A[n/2 : n]$
return $\min($`findMinimum`(A_1), `findMinimum`$(A_2))$

   (c) (3pts) Fill in the blank in the pseudo-code: what should the algorithm return in the base case? Briefly argue that the algorithm is correct with your choice.

   (d) (3pts) Analyze the running time of this recursive algorithm. How does it compare to your solution in part (a)?

(d) (6pts) Write the C program for this recursive algorithm.

5. **Recursive local-minimum-finding.** (20 points)

(a) Suppose A is an array of n integers (for simplicity assume that all integers are distinct). A *local minimum* of A is an element that is smaller than all of its neighbors. For example, in the array $A = [1, 2, 0, 3]$, the local minima are $A[1] = 1$ and $A[3] = 0$.

i. (2 points) Design a recursive algorithm to find a local minimum of A, which runs in time $O(\log(n))$.

ii. (2 points) Prove formally that your algorithm is correct.

iii. (2 points) Formally analyze the runtime of your algorithm.

iv. (4 points) Write C program for your algorithm.

(b) Let G be a square $n \times n$ grid of integers. A *local minimum* of A is an element that is smaller than all of its direct neighbors (diagonals don't count). For example, in the grid

$$G = \begin{bmatrix} 5 & 6 & 2 \\ 6 & 1 & 4 \\ 3 & 2 & 3 \end{bmatrix}$$

some of the local minima are G[1][1] = 5 and G[2][2] = 1. You may once again assume that all integers are distinct.

i. (2 points) Design a recursive algorithm to find a local minimum in $O(n)$ time.

ii. (2 points) We are not looking for a formal correctness proof, but please explain why your algorithm is correct.

iii. (2 points) Give a formal analysis of the running time of your algorithm.

iv. (4 points) Write C program for your algorithm.