CMPSC 465 Data Structures & Algorithms
Spring 2022 Chunhao Wang and Mingfu Shao

Assignment 03

Due: Friday 11:59 am, Feb. 04, 2022

Instructions: You may work in groups of up to three people to solve the homework. You must write your own solutions and explicitly acknowledge everyone whom you have worked with or who has given you any significant ideas about your solutions. You may also use books or online resources to help solve homework problems. All consulted references must be acknowledged. The acknowledgements need to be made by answering Problem 1 below.

You are encouraged to solve the problem sets on your own using only the textbook and lecture notes as a reference. This will give you the best chance of doing well on the exams. Relying too much on the help of group members or on online resources will hinder your performance on the exams.

Submissions being late in 2 hours will be accepted with a 20% penalty. Submissions late more than 2 hours will receive 0. There will be no exceptions to this policy, as we post the solutions soon after the deadline.

For the full policy on assignments, please consult the syllabus.

Formatting: Start a new page for each problem.

Describing an Algorithm: Please make sure you use plain wording to explain your algorithm. It is always a good practice to start with a summary of the high-level idea of your algorithm to ease graders understand your solution quickly. Then, explain your algorithm, using plain wording and including enough details.

The use of pseudo-code is optional, and it is your decision. No matter you use it or not, above description in words is always required. The pseudo-code has its own advantage in explaining structured (i.e., if-else, for-loop, recursive functions, etc) algorithms and in putting details in the right place. If you think pseudo-code better explains your algorithm, and/or helps graders understand your solution, and/or contains more details not included in the plain-wording description, then use pseudo-code. If you think everything is already clearly explained in the description with words, then you don't need to include pseudo-code. An algorithm that is only written in pseudo-code (i.e., missing above plain-wording description) is not acceptable, as it is extremely hard to read just pseudo-code without any explanation.

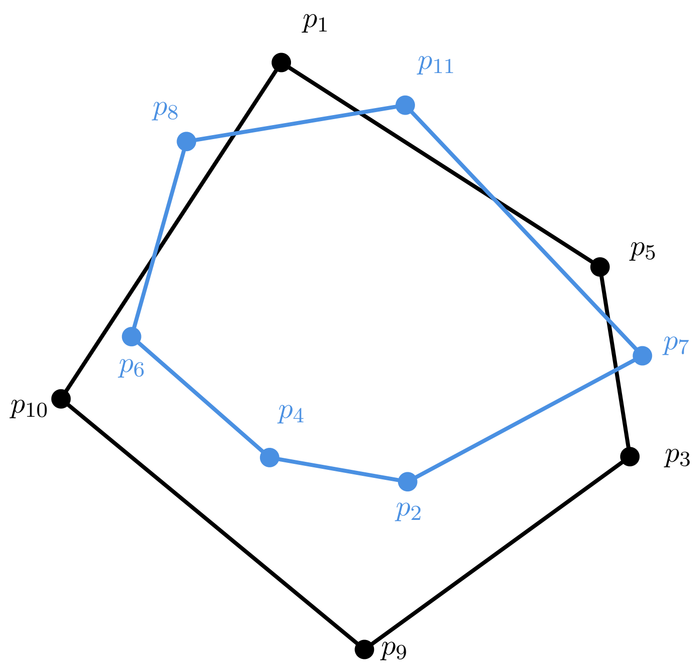
Here is a general situation that may help you decide whether to use pseudo-code or not. An algorithm could be "designed from scratch", i.e., you will need to come up with the step-by-step procedure. This usually involves in implementing a function with clear input and output. In this case, including pseudo-code usually helps. All algorithm we've seen so far (e.g., merge-two-sorted-arrays, merge-sort, etc) falls in this category. Second, an algorithm could also be "transformed into another algorithm", i.e., you use an existing algorithm to solve this problem. In this case you usually don't need to include pseudo-code but to describe how to transform one problem into the other. We will see such examples soon.

0. (0 pts.) Acknowledgements. The assignment will receive a 0 if this question is not answered.

1. If you worked in a group, list the members of the group. Otherwise, write "I did not work in a group."
2. If you received significant ideas about your solutions from anyone not in your group, list their names here. Otherwise, write "I did not consult anyone except my group members".
3. List any resources besides the course material that you consulted in order to solve the material. If you did not consult anything, write "I did not consult any non-class materials."

1. (15 pts.) Run the combine step of the divide-and-conquer algorithm for convex hull on the instance given below. You are given $C_1 = (p_1, p_{10}, p_9, p_3, p_5)$ and $C_2 = (p_8, p_6, p_4, p_2, p_7, p_{11})$.

1. Find the lowest point p^* in $C_1 \cup C_2$.
2. Transform C_1 into C_1' so that points in C_1' is sorted in increasing angle w.r.t. p^*.
3. Partition C_2 into two lists C_{2a} and C_{2b} so that each list is sorted in increasing angle w.r.t. p^*.
4. Give list C_2' by merging C_{2a} and C_{2b} so that each points in C_2' is sorted in increasing angle w.r.t. p^*.
5. Give list C' by merging C_1' and C_2' so that each points in C' is sorted in increasing angle w.r.t. p^*.
6. Run Graham-Scan-Core algorithm to find convex hull of C'. Show stack operations at each step (to deal with each point). For example, you need to write like "For A: push A; pop B", which indicates when you process point A, push A into stack and also pop B out.



2. (10 pts.) Given n points $p_1, p_2, ..., p_n$ in 2D-plane and a slope a, we want to find two *closest* lines with functions $y = ax + b_1$ and $y = ax + b_2$, $b_1 \leq b_2$, so that all n points are in or on between these two lines. Construct an algorithm with the time complexity $O(n)$ and justify your algorithm. Describe your algorithm (you do not need to give a pseudo code) and analyze the running time of your algorithm.

3. (10 pts.)

1. Let P be a set of n points $(p_1, p_2, \cdots, p_n)$ in a plane. A point $p_i \in P$ is *maximal* if no point in P is both above and to the right of p_i. Give a set P with 6 points such that there is only one point on the

convex hull of P but this point is not maximal, and there is only one point that is maximal but not on the convex hull of P. Include a visual illustration of these 6 points and the convex hull. Specifically, point out which is the point that is on the convex hull but not maximal, and which is the maximal point but not on the convex hull.

2. Design an instance of convex hull problem with 6 points, such that if Graham-Scan algorithm runs on your instance, the sequence of stack operations is (push, push, push, push, pop, push, pop, pop, push). Include a visual illustration of these 6 points and the convex hull.

4. (10 pts.) You are given n points $P = \{p_1, p_2, \cdots, p_n\}$ on 2D plane, represented as their coordinates. You are informed that the convex hull of P contains $O(\sqrt{\log n})$ points in P. Design an algorithm to compute the convex hull of P in $O(n \cdot \sqrt{\log n})$ time. Describe your algorithm and analyze its running time. You may assume that no three points in P are on the same line.

5. (0 pts.) *(NOTE: you don't need to submit your solution for this problem.)* Given two *sorted* arrays A and B of size m and n respectively, and an integer k, $1 \le k \le m + n$, design an algorithm to find the k-th smallest number in A and B. Describe your algorithm and analyze the running time of your algorithm. Your algorithm should run in $O(\log(m + n))$ time.

Solution: We cannot afford merging A and B, as it takes linear time rather than the desired logarithmic time. The idea of the algorithm is to reduce k by half using a single comparison. Specifically, each time we compare $A[mid_1 - 1]$ and $B[mid_2 - 1]$, where $mid_1 = \min(m, k/2)$ and $mid_2 = \min(n, k/2)$. If $A[mid_1 - 1] > B[mid_2 - 1]$, we eliminate all the elements before $B[mid_2]$, because it is impossible for the k-th smallest value to be located before and including $B[mid_2 - 1]$; otherwise, we eliminate all the elements before $A[mid_1]$.

Define function find-kth-smallest(A, m, B, n, k) return the k-th smallest number of $A[0 \cdots m)$ and $B[0 \cdots n)$. We assume A and B start with index 0. We also assume A and B represent "pointers" to the array, i.e., we will use $A + 3$ to represents the array shifted 3 elements. The pseudocode is shown below:

```
function: find-kth-smallest($A, m, B, n, k$)
if $m == 0$ then
    return $B[k-1]$
end
if $n == 0$ then
    return $A[k-1]$
end
if $k == 1$ then
    return $\min(A[0], B[0])$
end
$mid_1 = \min(m, k/2)$, $mid_2 = \min(n, k/2)$;
if $A[mid_1 - 1] > B[mid_2 - 1]$ then
    return find-kth-smallest($A, m, B + mid_2, n - mid_2, k - mid_2$)
end
return find-kth-smallest($A + mid_1, m - mid_1, B, n, k - mid_1$)
```

In above algorithm, at each recursive level, either k is halved, or one array is completely eliminated (and then in the next recursive call the algorithm terminates). The running time is $T(k) = T(k/2) + \Theta(1) \Rightarrow T(k) = \Theta(\log k)$.