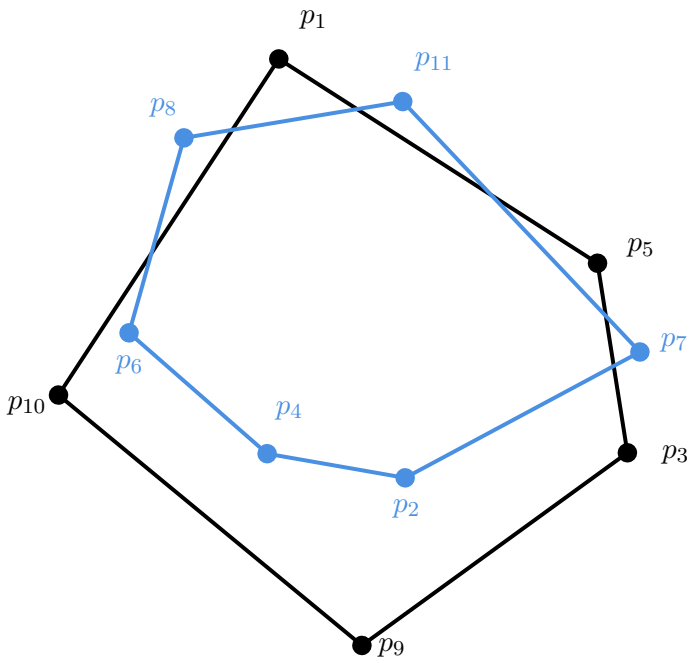
CMPSC 465 Spring 2022 | Data Structures & Algorithms | Chunhao Wang and Mingfu Shao | Solution 03

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.



Solution:

1. p_9 is the point with lowest y-coordinate.
2. $C'_1 = (p_9, p_3, p_5, p_1, p_{10})$
3. $C_{2a} = (p_7, p_{11}, p_8)$, $C_{2b} = (p_2, p_4, p_6)$. You may put p_7 and p_6 in either C_{2a} or C_{2b}.
4. $C'_2 = (p_7, p_2, p_{11}, p_8, p_4, p_6)$
5. $C' = (p_9, p_3, p_7, p_5, p_2, p_1 1, p_1, p_8, p_4, p_6, p_{10})$.
6. Elements in the stack are listed from stack bottom to top.
   For p_9, push p_9, stack: p_9;
   For p_3, push p_3, stack: p_9, p_3;
   For p_7, push p_7, stack: p_9, p_3, p_7;
   For p_5, push p_5, stack: p_9, p_3, p_7, p_5;
   For p_2, push p_2, stack: p_9, p_3, p_7, p_5, p_2;
   For p_{11}, pop p_2, push p_{11}, stack: $p_9, p_3, p_7, p_5, p_{11}$;
   For p_1, push p_1, stack: $p_9, p_3, p_7, p_5, p_{11}, p_1$;
   For p_8, push p_8, stack: $p_9, p_3, p_7, p_5, p_{11}, p_1, p_8$;
   For p_4, push p_4, stack: $p_9, p_3, p_7, p_5, p_{11}, p_1, p_8, p_4$;
   For p_6, pop p_4, push p_6, stack: $p_9, p_3, p_7, p_5, p_{11}, p_1, p_8, p_6$;
   For p_{10}, pop p_6, push p_{10}, stack: $p_9, p_3, p_7, p_5, p_{11}, p_1, p_8, p_{10}$;

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.

Solution: For given n points, let $p_i = (p_{ix}, p_{iy})$ and let $P^* = \{p_i^* : i = 1, 2, ..., n\}$ be a set of duals of p_i. Then, consider the upper- and lower-envelop of P^*, denoted as $UE(P^*)$ and $LE(P^*)$, respectively.
We now are interested in lines with the fixed slope a. Since a line $y = ax + b$ corresponds to $(a, -b)$ in the dual plane, the desired lines will be on the line $x = a$ in the dual plane. In other words, if we let $l_1 : y = ax + b_1$ and $l_2 : y = ax + b_2$, then $l_1^* = (a, -b_1), l_2^* = (a, -b_2)$ are on the vertical line $x = a$ in the dual plane.
If some point l^* lies on $UE(P^*)$, it means that l^* is above all lines p_i^* in P^*. Thus, by the propery of the duality, all points p_i is above the line l. Same goes true for the lower-envelope. Therefore, if we find two points on the line $x = a$ in the dual plane that one is on the upper-envelop and the other one is on the lower-envelop, their duals have a slope a and all points p_i are contained between them. As we need to find two cloesest lines, therefore, l_1^* = the intersection of $UE(P^*)$ and $x = a$, and l_2^* = the intersection of $LE(P^*)$ and $x = a$ in the dual plane. In other words, $l_1^* = (a, \max_{1 \leq i \leq n}(p_{ix}a - p_{iy}))$ and $l_2^* = (a, \min_{1 \leq i \leq n}(p_{ix}a - p_{iy}))$. Transforming to the original plane gives $b_1 = -\max_{1 \leq i \leq n}(p_{ix}a - p_{iy}))$ and $b_2 = -\min_{1 \leq i \leq n}(p_{ix}a - p_{iy})$.
From this argument, it suffices to construct an algorithm to find the maximum and minimum of $p_i.xa - p_i.y$. It takes $O(n)$ as we need to iterate once and get the maximum and minimum value of them.

3. (10 pts.)

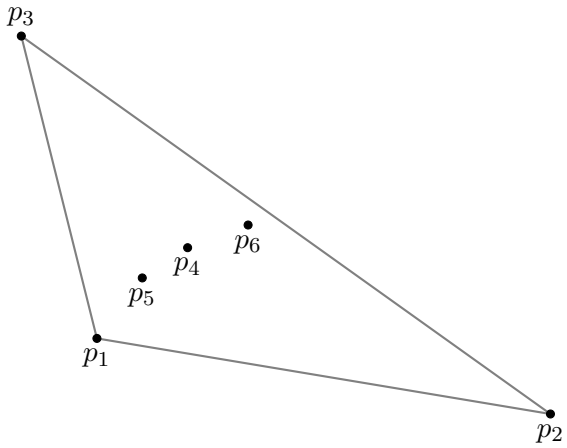
1. Let P be a set of n points $(p_1, p_2, \cdots, p_n)$ in a plane. A point $p_i \in P$ is *maximal* if no point in P is both above and to the right of p_i. Give a set P with 6 points such that there is only one point on the convex hull of P but this point is not maximal, and there is only one point 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.

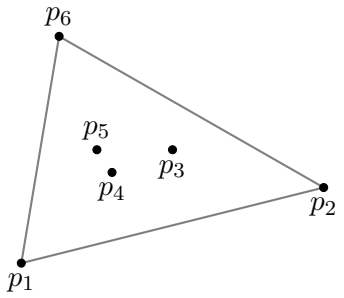
2. Design an instance of convex hull problem with 6 points, such that if Graham-Scan algorithm runs on your instance, the sequence of stack operations is (push, push, push, push, pop, push, pop, pop, push). Include a visual illustration of these 6 points and the convex hull.

Solution: *Both of these parts can have multiple viable solutions*. Below are simply example answers for these two parts, respectively.

1. p_1 is the point that is on the convex hull but not maximal. p_6 is the maximal point that is not on the convex hull.
   Thinking process: As we know from class, we need at least 3 points to form a convex hull. So, to find these 6 points, we start with the simplest setting i.e. to find 4 points with one maximal but non-convex point and one convex but non-maximal point. Then, we add 2 other points that don't break the requirements we fulfilled with those 4 points earlier.



2. Thinking process: Based on the Gramham-Scan-Core pseudo-code covered in class and the stack operation sequence, we can deduct the positional relationship between these 6 points.



   - Since the 4th operation is push, $p_2 \to p_3 \to p_4$ is "turning left".
   - Since the 5th operation is pop, $p_3 \to p_4 \to p_5$ is "turning right".
   - Since the 6th operation is push, $p_2 \to p_3 \to p_5$ is "turning left".
   - Since the 7th operation is pop, $p_3 \to p_5 \to p_6$ is "turning right".
   - Since the 8th operation is pop, $p_2 \to p_3 \to p_6$ is "turning right".
   - Since the 9th operation is push, $p_1 \to p_2 \to p_6$ is "turning left".

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. You may assume that no three points in P are on the same line.

Solution: We first find the point in P with smallest y-coordinate, denoted as p_1^*. Clearly, p_1^* must be on the convex hull of P. Let $C = (p_1^*)$ store the list of points on the convex hull of P found so far. Let $p_0^* = p_1^* - (1, 0)$ be a virtual point (not within P) be the point on the left of p_1^*. We then iteratively find all other points on the convex hull of P. In the k-th iteration, $k = 2, 3, \cdots$, for each point $p \in P \setminus \{p_0^*, p_1^*\}$, we compute the angle $\angle p p_{k-1}^* p_{k-2}^*$. We then find the point that maximizes this angle, denoted as p_k^*. Clearly, p_k^* must be on the convex hull of p as well (*Proof.* line $p_k^* p_{k-1}^*$ is an indicator: all other points locate at one side of this line). Therefore, we add p_k^* to C and continue to next iteration. The algorithm terminates when we find p_k^* equals to p_1^*.

Each iteration finds a new point on the convex hull and takes linear time. The total number of iterations equals to the number of points on the convex hull. Hence, the running time is $O(n \cdot s)$, where s is the number of points on the convex hull. This algorithm is *output-sensitive*. When there are $O(\sqrt{\log n})$ number of points on the convex hull, as described in this problem, this algorithm runs in $O(n \cdot \sqrt{\log n})$ time.

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 eliminated. The running time is $T(k) = T(k/2) + \Theta(1)$, which gives $T(k) = \Theta(\log k)$.

Rubrics:

Problem 1, 15 pts

1. 1 point : identified p* correctly.
2. 2 points : C_1' is correct
3. 3 points : C_{2a} and C_{2b} are correct.
4. 2 points : C_2' is correct.
5. 2 points : C_1' is correct. theorem/recurrence
6. 5 points : stack operations at each step are correct
7. 3 points : At least 7 operations are correct.
8. 2 points : at least 5 operations are correct.
9. 1.5 points : I don't know how to answer this question.

Problem 2, 10 pts

1. 5 points : Provided a proper proof or justification of an algorithm (Although they don't use the duality to justify, they would get full scores if it's correct)
2. 3 points : Provided $O(n)$ algorithm
3. 2 points : Analyzed the running time correctly
4. 1 point : I don't know how to answer this question.

Problem 3, 10 pts

1.
   - 2 points: Provided a correct visual illustration of the 6 points.
   - 1 point: Drew the convex hull in the visual illustration.
   - 1 point: Correctly pointed out which point is on the convex hull but not maximal.
   - 1 point: Correctly pointed out which point is maximal but not on the convex hull.
   - 0.5 points : I don't know how to answer this question.
2.
   - 3 points: The 6 points have the correct positional relationships. (See "thinking process" in the solution for the positional relationships)
   - 2 points: Drew the convex hull in the visual illustration i.e. p_1, p_2, and p_6 are on the convex hull; the other points are inside the convex hull.
   - 0.5 points : I don't know how to answer this question.

Problem 4, 10 pts

1. 10 points: Provided $O(n\sqrt{logn})$ algorithm.
2. 4 points: Incorrect design but provided appropriate running time analysis (e.g. the running time is $O(n \cdot s)$, where s is the number of points on the convex hull).
3. 1 point : I don't know how to answer this question.