Running Time of Graham-Scan

Although Graham-Scan-Core consists of two nested loop, in fact it runs in linear time! To see this, consider the types of operations and how many each type of operation the algorithm needs to do. There are three types of operations this algorithm does: *push*, *pop*, and *checking-orientation*, and each operation takes $\Theta(1)$ time. Now let's consider the times of each type will need to execute. First, notice that #(*push*) $= n$, as each point will be pushed to the stack exactly once. Second, #(*pop*) $\leq n$, as #(*push*) $\leq$ #(*pop*) starting from an empty stack (a point must be pushed into the stack prior to pop). Third, notice that right after *checking-orientation* it is either a *push* or a *pop* operation (i.e., it's not possible to have two *checking-orientation* operations next to each other). This implies that #(*checking-orientation*) $\leq$ #(*push*) + #(*pop*) $\leq 2n$. Combined, we have that Graham-Scan-Core takes $\Theta(n)$ time.

The preprocessing step of Graham-Scan takes $\Theta(n \log n)$, as it's dominated by the sorting step. The entire running time of Graham-Scan therefore takes $\Theta(n \log n)$ time.

Divide-and-Conquer Algorithm for Convex Hull

We now design a divide-and-conquer algorithm for convex hull problem. As usual, we define recursive function, say, CHDC (P), which takes a set of points P as input and returns $CH(P)$, represented as the list of vertices of the convex polygon in counter-clockwise order.

function CHDC ($P = \{p_1, p_2, \cdots, p_n\}$)
  if $n \leq 3$: resolve this base case and return the convex hull;
  C_1 = CHDC ($P[1..n/2]$);
  C_2 = CHDC ($P[n/2+1..n]$);
  return combine (C_1, C_2);
end algorithm;

The "combine" function in above pseudo-code is missing. Before design an algorithm for "combine", we first make sure that it suffices to only consider points in C_1 and C_2. Formally, we can write $CH(P) = CH(C_1 \cup C_2)$. This can be proved using the Property 2 of convex hull. Intuitively, any point *inside* C_1 or C_2 with be inside $CH(P)$, and therefore won't be *on* $CH(P)$.

We can use, for example, Graham-Scan to calculate $CH(C_1 \cup C_2)$, which takes $\Theta(m \log m)$, where $m = |C_1 \cup C_2|$. Can we do better? Yes. We will design an $\Theta(m)$ time algorithm to calculate $CH(C_1 \cup C_2)$. The idea is to take advantage of that, C_1 and C_2 are already sorted (in counter-clockwise order along the polygon), and then to use Graham-Scan-Core to calculate $CH(C_1 \cup C_2)$. Recall that, Graham-Scan-Core is linear-time algorithm for convex hull; it just requires that all points in its input is sorted in counter-clockwise order w.r.t. the first point in its input.

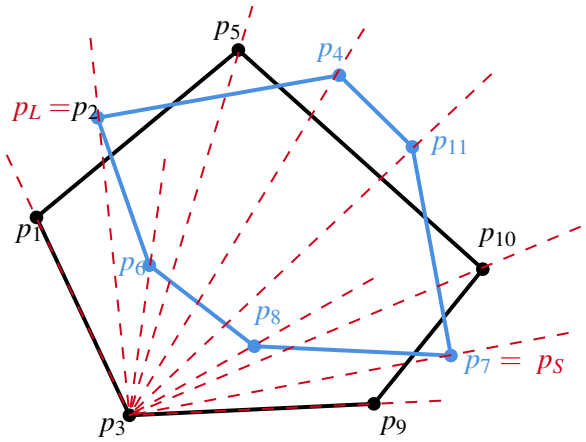
The pseudo-code for combine procedure is given below.

function combine (C_1, C_2)

find p_* in $C_1 \cup C_2$ with smallest y-coordinate;
assume that $p_* \in C_1$; otherwise exchange C_1 and C_2;
rewrite C_1 into C_1' with circular shifting so that p_* is the first point of C_1'; (*Note: now points in C_1' is sorted w.r.t. p_* in counter-clockwise order.*)
find p_S in C_2 with smallest angle (i.e., angle $\angle p_S p_* p_\infty$, where $p_\infty = (+\infty, p_*.y)$;
find p_L in C_2 with largest angle (i.e., angle $\angle p_L p_* p_\infty$, where $p_\infty = (+\infty, p_*.y)$;
let C_{2a} be the sublist of C_2 from p_S to p_L in counter-clockwise order; (*Note: now points in C_{2a} is sorted w.r.t. p_* in counter-clockwise order.*)
let C_{2b} be the sublist of C_2 from p_S to p_L in clockwise order; (*Note: now points in C_{2b} is sorted w.r.t. p_* in counter-clockwise order.*)
C_2' = merge-two-sorted-arrays (C_{2a}, C_{2b}); (*Note: merging is by the angle w.r.t. p_* and p_∞; after it points in C_2' is sorted w.r.t. p_* in counter-clockwise order.*)
C' = merge-two-sorted-arrays (C_1', C_2'); (*Note: merging is by the angle w.r.t. p_* and p_∞; after it points in C', i.e., points in $C_1 \cup C_2$, is sorted w.r.t. p_* in counter-clockwise order.*)
return Graham-Core-Scan (C');

end algorithm;

See figure below for explaining above combine step.



$C_1 = (p_5, p_1, p_3, p_9, p_{10})$ $\quad C_2 = (p_4, p_2, p_6, p_8, p_7, p_{11})$

$C_1' = (p_3, p_9, p_{10}, p_5, p_1)$ $\quad C_{2a} = (p_7, p_{11}, p_4)$

$C_{2b} = (p_8, p_6, p_2)$

$C_2' = (p_7, p_8, p_{11}, p_4, p_6, p_2)$

$C' = (p_3, p_9, p_7, p_{10}, p_8, p_{11}, p_4, p_5, p_6, p_2, p_1)$

Figure 1: Example of combining C_1 and C_2.

Clearly each step of above combine takes linear time. Therefore, the entire running time of combine is $\Theta(|C_1| + |C_2|)$.

To obtain the running time of CHDC, we write its recursion $T(n) = 2T(n/2) + \Theta(|C_1| + |C_2|)$. In worst case, $|C_1| + |C_2| = \Theta(n)$, and therefore $T(n) = 2T(n/2) + \Theta(n)$, which gives $T(n) = \Theta(n \log n)$. In this (worst) case the running time of this divide-and-conquer algorithm is the same with Graham-Scan. However,

if the size of $|C_1| + |C_2|$ is in the order of $o(n)$, we will have improved running time. For example, if $|C_1| + |C_2| = \Theta(n^{0.99})$, then $T(n) = \Theta(n)$. This might happen in practical cases, although in worst case divide-and-conquer runs in $\Theta(n \log n)$ time.