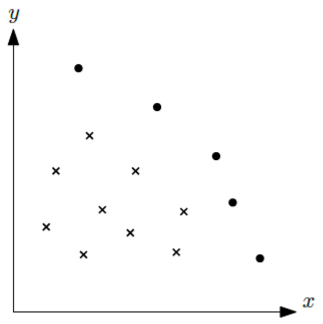
DESIGN AND ANALYSIS OF ALGORITHMS
Homework 1

Aman Choudhary
MTech Coursework, CSA 2020
Sr No: 17920

October 10, 2020

1 (Skyline points)

Let P be a set of n points in the plane. A point $p(p_x, p_y)$ *dominates* another point $q(q_x, q_y)$ if and only if $p_x > q_x$ and $p_y > q_y$. Any point in P which is not dominated by any other point in P is a *skyline* point. The goal here is to compute all the skyline points of P. As a motivating example, think of each cricketer as a point in the plane, and the x-axis (resp., y-axis) to be the runs scored (resp., wickets taken). Computing skyline points is a popular approach to capture a potential set of good points (in this case, all-rounders in cricket).



Here is an algorithm to compute the skyline points.

```
skyline($P$):
1. if $|P| = 1$, then return $P$.
2. divide $P$ into the left and right halves $P_\ell$ and $P_r$ by the median $x$-coordinate.
3. discover the point $p$ with the maximum $y$-coordinate in $P_r$.
4. prune all points from $P_\ell$ that are dominated by $p$.
5. return the concatenation of skyline($P_\ell$) and skyline($P_r$).
```

Prove that the above algorithm runs in $O(n + n \log h)$ time, where h is the number of skyline points in P. (*Hint: Define $T(n, h)$ to be the running time of the algorithm on an input of size n and output size h. Write a recurrence in terms of $T(n, h)$ and establish an important base case of $T(n, 1) \leq cn$, where c is a constant independent of n.*)

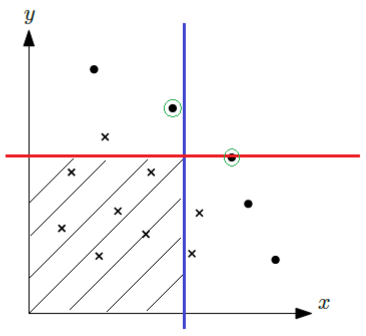
Solution:

Let us first write the recurrence relation for the problem. Let T(n,h) is the time to compute skyline(P).

1. Time taken to divide P into the left and right halves P_ℓ and P_r by the median x-coordinate $\leq$ c$_1$n (Median Finding runs in linear time in worst case. After finding the median, the points in left part and right part could simply be identified and copied to separate lists).

2. Time taken to *discover* the point p with the maximum y-coordinate in $P_r \leq$ c$_2 \left(\frac{n}{2}\right)$ (A single scan through the points of P_r is sufficient).

3. Time taken to *prune* all points from P_ℓ that are dominated by $p \leq$ c$_3 \left(\frac{n}{2}\right)$ (A single scan through the points of P_ℓ is sufficient).



4. Let us assume that the number of points in skyline(P_ℓ) $= h_1$.
   Let us assume that the number of points in skyline(P_r) $= h_2$.

   We will now establish that $h_1 + h_2 = h$, and moreover, the concatenation of skyline(P_ℓ) and skyline(P_r) is simply a process of joining the tail of one output list to the head of another, and takes constant time.

   Fig. 2 shows that the blue line (chosen arbitrarily for illustration purposes) divides P into sets P_ℓ and P_r. The red line is drawn corresponding to point p discovered in step 3, which is used to prune points in P_ℓ. The shaded region shows the points that are pruned.

   The point circled green in P_ℓ is the last skyline point in P_ℓ. Let us call it **a**. Similarly, the point encircled green in P_r is the first skyline point in P_r. Let us call it **b**. The claim is that neither **a** dominates **b**, nor **b** dominates **a**. This is because though **a** is at a higher level than **b**, it lies to the left of **b**. Similarly, even though **b** lies to the right of **a**, it is lower than **a**. Hence, after every partitioning of points, the skyline points are also separated neatly into respective sets. Therefore, during the merge of skyline(P_ℓ) and skyline(P_r), no skyline point in either set gets removed. The merge is simply joining one list to another, and takes constant time.

5. Let us assume that the number of points that are pruned from $P_\ell = n_1$. So, the size of $P_\ell = \dfrac{n}{2} - n_1$.
   Therefore, time taken to compute skyline(P_ℓ) = T $\left(\dfrac{n}{2} - n_1, h_1\right)$. Also time taken to compute skyline(P_r) = T $\left(\dfrac{n}{2}, h - h_1\right)$. The total cost of work before divide (point 1, 2 and 3), and during conquer (point 4) is = (1) + (2) + (3) + (4) $\leq$ c$_1 n + c_2\left(\dfrac{n}{2}\right) + c_3\left(\dfrac{n}{2}\right) + k = O(n)$.

6. Hence, the final recurrence is,

   $$T(n,h) = T\left(\frac{n}{2} - n_1, h_1\right) + \mathrm{T}\left(\frac{n}{2}, h - h_1\right) + O(n).$$

   We want to show that the solution is $T(n,h) = O(n \log h)$. The substitution method requires us to prove that $T(n,h) \leq dn \log h$ for an appropriate choice of the constant $d > 0$. We start by assuming that this bound holds for all positive $n' < n$, and all positive $h' < h$. We also pick a constant c such that the function described by the $O(n)$ term is bounded above by cn for all $n > 0$.

   $$\begin{aligned} T(n,h) &\leq d\left(\frac{n}{2} - n_1\right)\log(h_1) + c\left(\frac{n}{2}\right)\log(h - h_1) + cn \\ &\leq d\left(\frac{n}{2}\right)\log(h_1) + c\left(\frac{n}{2}\right)\log(h - h_1) + cn \\ &\leq d\left(\frac{n}{2}\right)(\log(h_1) + log(h - h_1)) + cn \\ &\leq d\left(\frac{n}{2}\right)\log(h_1(h - h_1)) + cn \qquad \text{...(1)} \end{aligned}$$

   To find the worst case of algorithm, we need to maximize the R.H.S of (1). To do so, we need to maximize the product $(h_1(h - h_1))$. We differentiate this product with respect to h_1 and equate it to 0.

   $$f(h_1) = h_1(h - h_1)$$

   $$\frac{\mathrm{d}f}{\mathrm{d}h_1} = h_1(-1) + (h - h_1)$$

   $$\frac{\mathrm{d}f}{\mathrm{d}h_1} = h - 2h_1$$

   Putting, $\dfrac{\mathrm{d}f}{\mathrm{d}h_1} = 0$, we get $h_1 = h/2$. Now, we compute $\dfrac{\mathrm{d}^2 f}{\mathrm{d}h_1^2}$ which turns out to be -2, which is $<$ 0, thus confirming that $h_1 = h/2$ is indeed a point of maxima. Substituting this is in (1),

$$\begin{aligned}
T(n,h) &\leq d\left(\frac{n}{2}\right)\log\left(\frac{h^2}{4}\right) + cn \\
&\leq d\frac{n}{2}\log(h^2) - d\frac{n}{2}\log(4) + cn \\
&\leq d\frac{n}{2}(2\log h) - d\frac{n}{2}(2) + cn \\
&\leq dn\log h - dn + cn \\
&\leq dn\log h - n(d-c) && \text{...(2)}
\end{aligned}$$

For (2) to be true, the following condition must hold,

$$\begin{aligned}
&d - c \geq 0 \\
&d > c && \text{...(3)}
\end{aligned}$$

Since, the above relation can be fulfilled with an appropriate choice of constants, c and d. We conclude that $T(n,h) \leq dn\log h$ where c $>$ 0 and $d > 1$ from condition (3).

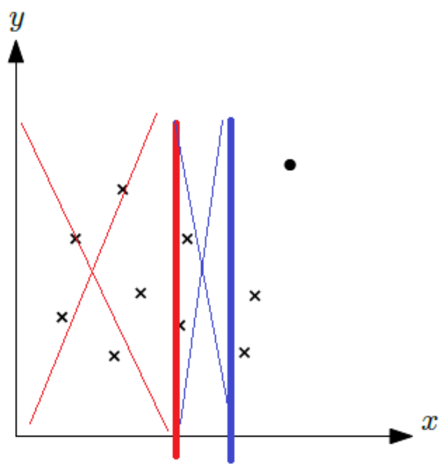
7. Mathematical induction now requires us to show that our solution holds for the boundary conditions. Now, we define some boundary cases,

$$T(n,h) = \begin{cases} O(1), & \text{if } n = 0, h = 0 \text{ (case of } P_\ell\text{, when it gets entirely pruned by p)} \\ O(1), & \text{if } n = 1, h = 1 \text{ (step 1 in skyline(P))} \\ O(n), & \text{if } n > 1, h = 1 \text{ (Proved later in point 8)} \end{cases}$$

We now make a distinction between the base case of recurrence and base case of our inductive proof. We have defined the base case of recurrence above, and now we choose the boundary of our inductive proof as $n > 1$ and $h > 1$. This is because, $T(n,h) \leq dn\log h$ has undefined value when $h = 0$, and 0 when $h = 1$. We can get around these troublesome boundaries by taking advantage of asymptotic notation which requires us only to prove $T(n,h) \leq dn\log h$ for $n \geq n_0$ and $h \geq h_0$, where n_0 and h_0 are constants that we get to choose.

Hence, $T(n) \leq dn\log h$ **, where** $d > 1$ **,** $n \geq n_0 = 2$ **,** $h \geq h_0 = 2$

8. All that remains, is that we prove $T(n,1) = O(n)$. If, we examine this carefully, it is the case, when there is only one skyline point dominating all other points. Moreover, this point will always end up being in P_r since it dominates all other points and so, its x-coordinate (and also y-coordinate) is more than all other points. So, at every step, this point will completely keep pruning its P_ℓ. For example, in first step, the single skyline point prunes the red area, and the blue area in the next step.



So, we write the recurrence relation as, $T(n) = T(n/2) + cn$. This is true because, at each step the problem size reduces by half. Also, as we already established in point 5, the total cost of work before divide (point 1, 2 and 3), and during conquer (point 4) of the algorithm is $O(n)$. Now, we solve the recurrence as follows, We guess that the solution is $T(n) = O(n)$. The substitution method requires us to prove that $T(n) \leq dn$ for an appropriate choice of the constant d $>$ 0. We start by assuming that this bound holds for all positive $n' < n$. We also pick a constant c such that the function described by the $O(n)$ term is bounded above by cn for all $n > 0$. Hence,

$$\begin{aligned}
T(n) &= T(n/2) + cn \\
T(n) &\leq d(n/2) + cn \\
&\leq n(d/2 + c) \\
&\leq dn \qquad \text{...(4)}
\end{aligned}$$

For (4) to be true, the following condition must hold,

$$\begin{aligned}
&d/2 + c \leq d \\
&d \geq 2c \qquad \text{...(5)}
\end{aligned}$$

Thus, we conclude that $T(n) \leq dn$ where c $>$ 0 and d follows condition (5). Hence $T(n) = O(n)$.

9. The last minor detail is the time taken to scan the input which takes $O(n)$ time.
   Hence, we conclude that the overall time complexity of the algorithm is $O(n \log h + n)$.

2 (Alternate median Finding algorithm)

1. Let T(n) = $\sum_{i=1}^{k} T(a_i) + O(n)$, where $a_i > 0$ for i = 1,...,k and $\sum_{i=1}^{k} a_i < 1$. Use the substitution method to prove that

$$T(n) = \begin{cases} O(n), & \text{if } \sum_{i=1}^{k} a_i < 1 \\ O(n \log n), & \text{if } \sum_{i=1}^{k} a_i = 1 \end{cases}$$

Solution:

(a) We guess that the solution is $T(n) = O(n)$. The substitution method requires us to prove that $T(n) \leq dn$ for an appropriate choice of the constant d > 0. We start by assuming that this bound holds for all positive $n' < n$. We also pick a constant c such that the function described by the $O(n)$ term is bounded above by cn for all $n > 0$. Hence,

$$\begin{aligned} T(n) &= T(a_1 n) + T(a_2 n) + ... + T(a_k n) + O(n) \\ T(n) &\leq da_1 n + da_2 n + ... + da_k n + cn \\ &\leq n\left(da_1 + da_2 + ... + da_k + c\right) \\ &\leq n\left(d\left(\sum_{i=1}^{k} a_i\right) + c\right) \\ &\leq dn \qquad\qquad ...(1) \end{aligned}$$

For (1) to be true, the following condition must hold,

$$\begin{aligned} & d\left(\sum_{i=1}^{k} a_i\right) + c \leq d \\ & d\left(1 - \sum_{i=1}^{k} a_i\right) \geq c \\ & d \geq \frac{c}{\left(1 - \sum_{i=1}^{k} a_i\right)} \qquad\qquad ...(2) \end{aligned}$$

Thus, we conclude that $T(n) \leq dn$ where c $>$ 0 and d follows condition (2). Hence $T(n) = O(n)$.

(b) We guess that the solution is $T(n) = O(n \log n)$. The substitution method requires us to prove that $T(n) \leq dn \log n$ for an appropriate choice of the constant d $>$ 0. We start by assuming that this bound holds for all positive $n' < n$. We also pick a constant c such that the function described by the $O(n)$ term is bounded above by cn for all $n > 0$. Also, let us assume, $a_i = \frac{1}{b_i}$, because for every i, $a_i \leq 1$. Moreover, $\sum_{i=1}^{k} \frac{1}{b_i} = \sum_{i=1}^{k} a_i = 1$.

$$
\begin{aligned}
T(n) &= T(a_1 n) + T(a_2 n) + ... + T(a_k n) + O(n) \\
T(n) &= T\left(\frac{n}{b_1}\right) + T\left(\frac{n}{b_2}\right) + ... + T\left(\frac{n}{b_k}\right) + O(n) \\
&\leq d\left(\frac{n}{b_1}\right)\log\left(\frac{n}{b_1}\right) + d\left(\frac{n}{b_2}\right)\log\left(\frac{n}{b_2}\right) + ... + d\left(\frac{n}{b_k}\right)\log\left(\frac{n}{b_k}\right) + cn \\
&\leq dn\left(\sum_{i=1}^{k} \frac{1}{b_i} \log \frac{n}{b_i}\right) + cn \\
&\leq dn\left(\sum_{i=1}^{k} \frac{\log n - \log b_i}{b_i}\right) + cn \\
&\leq dn\left(\sum_{i=1}^{k} \frac{\log n}{b_i} - \sum_{i=1}^{k} \frac{\log b_i}{b_i}\right) + cn \\
&\leq dn\left(\log n \sum_{i=1}^{k} \frac{1}{b_i} - \sum_{i=1}^{k} \frac{\log b_i}{b_i}\right) + cn \\
&\leq dn\left(\log n \ (1) - \sum_{i=1}^{k} \frac{\log b_i}{b_i}\right) + cn \\
&\leq dn \log n - dn \sum_{i=1}^{k} \frac{\log b_i}{b_i} + cn \\
&\leq dn \log n - n\left(d \sum_{i=1}^{k} \frac{\log b_i}{b_i} - c\right) \\
&\leq dn \log n \qquad\qquad ...(3)
\end{aligned}
$$

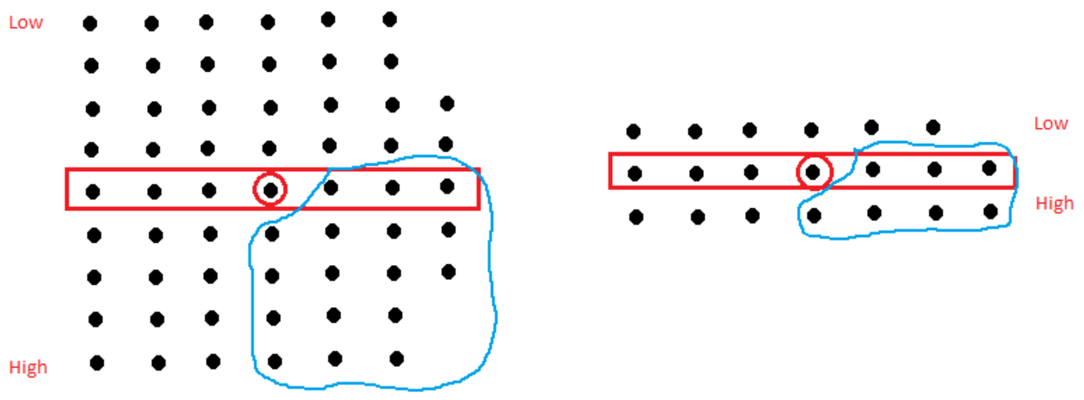
For (3) to be true, the following condition must hold,

$$
\begin{aligned}
&\left(d \sum_{i=1}^{k} \frac{\log b_i}{b_i} - c\right) \geq 0 \\
&d \geq \frac{c}{\sum_{i=1}^{k} \frac{\log b_i}{b_i}} \\
&d \geq \frac{c}{\sum_{i=1}^{k} a_i \log \frac{1}{a_i}} \qquad\qquad ...(4)
\end{aligned}
$$

Thus, we conclude that $T(n) \leq dn \log n$ where c $>$ 0 and d follows condition (4). Hence $T(n) = O(n \log n)$.

2. In the median finding algorithm described in the lecture, the input elements are divided into groups of 5. The goal of this exercise is to understand if there is anything special about this number 5. Will the algorithm work in linear time if they are divided into groups of 9? If yes, then prove it. Will the algorithm work in linear time if they are divided into groups of size 3? Make use of the recurrence proven above.

   Solution:



   (a) Case: **When elements are divided into groups of 9**

   To analyze the running time of our algorithm, we first determine a lower bound on the number of elements that are greater than the partitioning element x (shown in red circle). Fig 4(a) helps us to visualize this bookkeeping. At least half of the medians (the ones discovered for each column, and shown in red rectangular box), are greater than or equal to the median-of-medians x. Thus, at least half of the $\lceil n/9 \rceil$ groups contribute at least 5 elements that are greater than x, except for the one group that has fewer than 9 elements if 9 does not divide n exactly, and the one group containing x itself. Discounting these two groups, it follows that the number of elements greater than x is at least,

   $$5\left(\frac{1}{2}\left\lceil\frac{n}{9}\right\rceil - 2\right) \geq \left(\frac{5n}{18} - 10\right)$$

   Similarly, at least $\left(\frac{5n}{18} - 10\right)$ elements are less than x. Thus, in the worst case, after discarding one partition of the array, our algorithm proceeds recursively on at most, $n - \left(\frac{5n}{18} - 10\right) = \left(\frac{13n}{18} + 10\right)$ elements. We can write recurrence relation for the above as follows,

   $$T(n) = T\left(\left\lceil\frac{n}{9}\right\rceil\right) + T\left(\frac{13n}{18} + 10\right) + O(n)$$

   $$T(n) \leq T\left(\frac{n}{9} + 1\right) + T\left(\frac{13n}{18} + 10\right) + O(n)$$

   In the above recurrence the additional constant 1, cannot substantially affect the solution to the recurrence. When n is large, the difference between $\left(\frac{n}{9} + 1\right)$ and $\left(\frac{n}{9}\right)$ is not large. The same can be said about the constant 10.

   Thus, the above recurrence is similar to, $T(n) = \sum_{i=1}^{k} T(a_i) + O(n)$, where $a_i > 0$ for i = 1,...,k and $\sum_{i=1}^{k} a_i < 1$, because a_1 = (1/9) and a_2 = (13/18) and their sum is (15/18) which is less than 1.

   We guess that the solution is $T(n) = O(n)$. The substitution method requires us to prove that $T(n) \leq dn$ for an appropriate choice of the constant d > 0. We start by assuming that this bound holds for all positive $n' < n$. We also pick a constant c such that the function described by the $O(n)$ term is bounded above by cn for all $n > 0$. Hence,

$$\begin{aligned}
T(n) &\leq d\left(\frac{n}{9}+1\right) + d\left(\frac{13n}{18}+10\right) + cn \\
&\leq d\left(\frac{15n}{18}+11\right) + cn \\
&\leq \frac{15dn}{18} + 11d + cn \\
&\leq dn - \frac{3dn}{18} + 11d + cn \\
&\leq dn - \left(\frac{3dn}{18} - 11d - cn\right) \\
&\leq dn \qquad\qquad \text{...(1)}
\end{aligned}$$

For (1) to be true, the following condition must hold,

$$\left(\frac{3dn}{18} - 11d - cn\right) \geq 0$$

$$d \geq 6c\left(\frac{n}{n-66}\right) \qquad\qquad \text{...(2)}$$

Since, we assumed our constants c and d to be positive quantities, we maintain the same by choosing n strictly $>$ 66. For our convenience, we choose n=122. As a result, $n/(n-66) \leq 2$. So, choosing appropriate $c > 0$ and $d \geq 12c$ will satisfy inequality (2).

Hence, we have established that $T(n) \leq dn$ for $n \geq n_0 = 122$. We conclude that $T(n) = O(n)$ when median finding algorithm works with groups of 9.

(b) Case: **When elements are divided into groups of 3**

To analyze the running time of our algorithm, we first determine a lower bound on the number of elements that are greater than the partitioning element x (shown in red circle). Fig 4(b) helps us to visualize this bookkeeping. At least half of the medians (the ones discovered for each column, and shown in red rectangular box), are greater than or equal to the median-of-medians x. Thus, at least half of the $\lceil n/3 \rceil$ groups contribute at least 2 elements that are greater than x, except for the one group that has fewer than 3 elements if 3 does not divide n exactly, and the one group containing x itself. Discounting these two groups, it follows that the number of elements greater than x is at least,

$$2\left(\frac{1}{2}\left\lceil\frac{n}{3}\right\rceil - 2\right) \geq \left(\frac{n}{3} - 4\right)$$

Similarly, at least $\left(\frac{n}{3} - 4\right)$ elements are less than x. Thus, in the worst case, after discarding one partition of the array, our algorithm proceeds recursively on at most, $n - \left(\frac{n}{3} - 4\right) = \left(\frac{2n}{3} + 4\right)$ elements. We can write recurrence relation for the above as follows,

$$T(n) = T\left(\left\lceil\frac{n}{3}\right\rceil\right) + T\left(\frac{2n}{3} + 4\right) + O(n)$$

When n is large, the difference between $\left(\frac{2n}{3} + 4\right)$ and $\left(\frac{2n}{3}\right)$ is insignificant. Hence, we can safely say that after every step the problem size reduces to $(2/3)rd$ of original one. Also, we can rewrite $\left\lceil\frac{n}{3}\right\rceil$ as $\frac{n}{3}$. Hence our recurrence relation is now,

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n)$$

Thus, the above recurrence is similar to, $T(n) = \sum_{i=1}^{k} T(a_i) + O(n)$, where $a_i > 0$ for i = 1,...,k and $\sum_{i=1}^{k} a_i = 1$, because a_1 = (1/3) and a_2 = (2/3). Hence, we already have an upper bound for the above recurrence i.e. $T(n) = O(n \log n)$. We just have to show, that the above problem cannot be solved in linear time.

To do so, we assume that the solution is $T(n) = O(n)$. The substitution method requires us to prove that $T(n) \leq dn$ for an appropriate choice of the constant d $>$ 0. We start by assuming that this bound holds for all positive $n' < n$. We also pick a constant c such that the function described by the $O(n)$ term is bounded above by cn for all $n > 0$. Hence,

$$\begin{aligned} T(n) &\leq d\left(\frac{n}{3}\right) + d\left(\frac{2n}{3}\right) + cn \\ &\leq dn + cn \\ &\leq dn \qquad \text{...(3)} \end{aligned}$$

For (3) to be true, the following condition must hold,

$$dn + cn \leq dn$$

$$cn \leq 0$$

As these contradicts our assumptions that c and n must be positive, we hence conclude that the median finding algorithm cannot be solved in linear time when elements are divided into groups of 3.

NOTES

1. Ideas and different approaches pertaining to solving of first problem were discussed with:
   Shashank Singh (M.Tech Coursework, CSA 2020).
2. CLRS 3rd edition was extensively used as reference material.