Assignment 1

Sheryl Mathew (11627236)

September 18, 2018

1 Questions

1.1 In Lesson 1, algorithm's complexity is measured on input size instead of input values. Please indicate the input size for an algorithm that solves the following problem: Given: a number n and two primes p, q Question: is it the case that n = p . q?

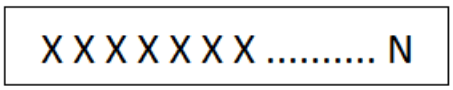
Let :
Size of n = S_n
Size of p = S_p
Size of q = S_q

Total input size,

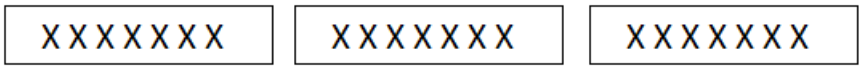
$$\begin{aligned} S_{total} &= S_n + S_p + S_q \\ &= \log_2(S_n)+1 + \log_2(S_p) +1+ \log_2(S_q)+1 \\ &= \log_2(S_n)+ \log_2(S_p)+ \log_2(S_q)+3 \end{aligned}$$

1.2 In Lesson 3, we learned linear-time selection algorithm where the input array of numbers are cut into groups of size 5. Show that, when the group size is 7, the algorithm still runs in linear time.

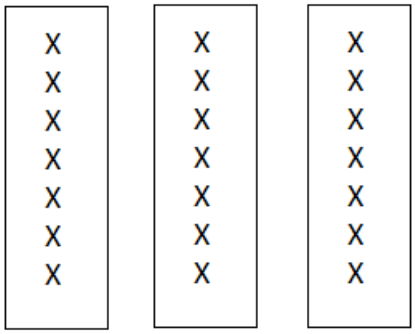
Step 1: Consider an array with n elements



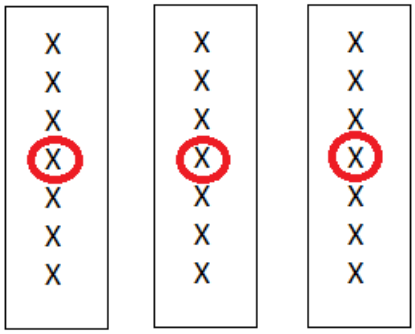
Step 2: Divide the n elements into groups of 7. Therefore there will be $\frac{n}{7}$ groups. This takes linear time.



Step 3: Shift the horizontal groups to vertical groups. This taken linear time.

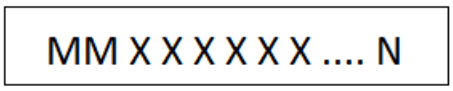


Step 4: Sort the elements of each group in ascending order. This taken linear time. Then find the median of each group. The time taken here is a constant. Now we have $\frac{n}{7}$ medians

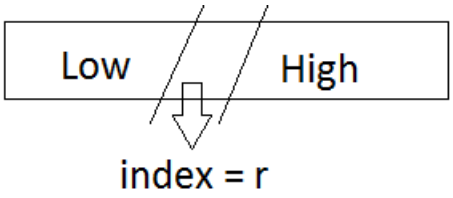


Step 5: Sort the $\frac{n}{7}$ medians in ascending order and find the median from the $\frac{n}{7}$ medians. The Median of medians (MM) will be $\frac{n}{14}$ element from the $\frac{n}{7}$ medians. This takes some computation time.

Step 6: Swap the MM with the first element in the array. This takes linear time.



Step 7: Partition the updated array. This takes linear time.



Let i be the i^{th}

1. If i=r : Smallest i^{th} element will be MM. This takes constant time.
2. If i<r : Recursively find the smallest i^{th} element from Low. This takes some computation time.
3. If i>r : Recursively find the smallest i^{th} element from High. This takes some computation time.

Worst Time complexity,

$T_w(n)$= Step 1 + Step 2 + Step 3 + Step 4 + Step 5 + Step 6 + Step 7
= Step 5 + Step 7 + O(n) [Step 1 = Step 2 = Step 3 = Step 4 = Step 6 = O(n) ie They all take linear time]

Step 5,
$T_w(n) = T_w(\frac{n}{7})$
Step 7,
$T_w(n)$ = max of [$T_w|Low|$ and $T_w|High|$]

$|Low| \geq 4\ (\frac{n}{14})$
$|High| \geq n - 4\ (\frac{n}{14})$
$\geq \frac{10n}{14}$

$T_w(n)$= max of $[4(\frac{n}{14}), \frac{10n}{14}]$
$= T_w(\frac{10n}{14})$

$T_w(n) = T_w(\frac{n}{7}) + T_w(\frac{10n}{14}) + O(n)$

By induction hypothesis,

$T_w(n) = T_w(\frac{n}{7}) + T_w(\frac{10n}{14}) + a.n$

$\geq c.(\frac{n}{7}) + c.(\frac{10n}{14}) + a.n$

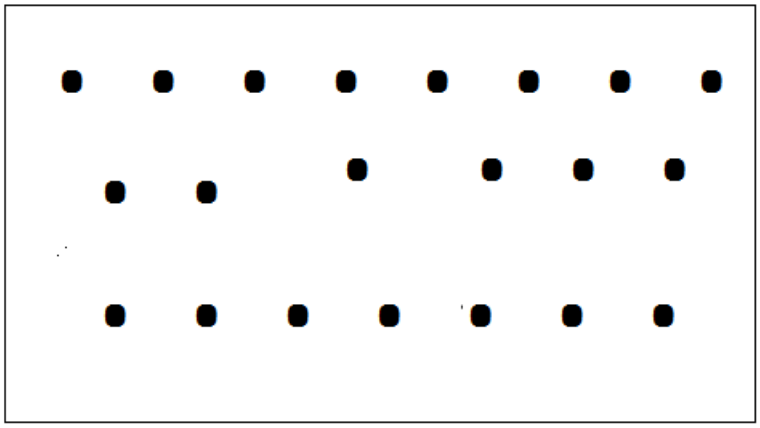
$\geq c.(\frac{2n}{14}) + c.(\frac{10n}{14}) + a.n$

$\geq c.(\frac{12n}{14}) + a.n$

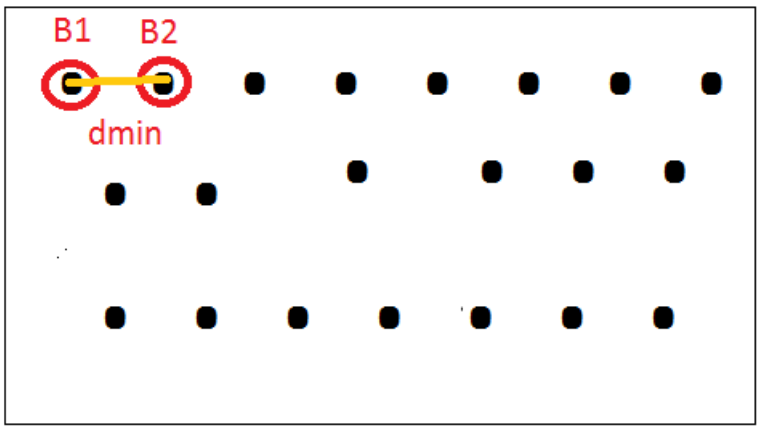
$\geq c.(\frac{6n}{7})$

1.3 In Lesson 2, we learned closest pair algorithm that runs in O(nlogn) time. However, that algorithm can be improved further when additional assumption is made. Here is one. Suppose that there are n^2 bugs sitting on a piece of paper of size n by n. Any two bugs must stay away by at least 1. Each bug is given as a pair of coordinates. Design a linear-time algorithm that finds the closest pair of bugs.

Step 1: Consider a paper of size n x n with n^2 bugs randomly placed.



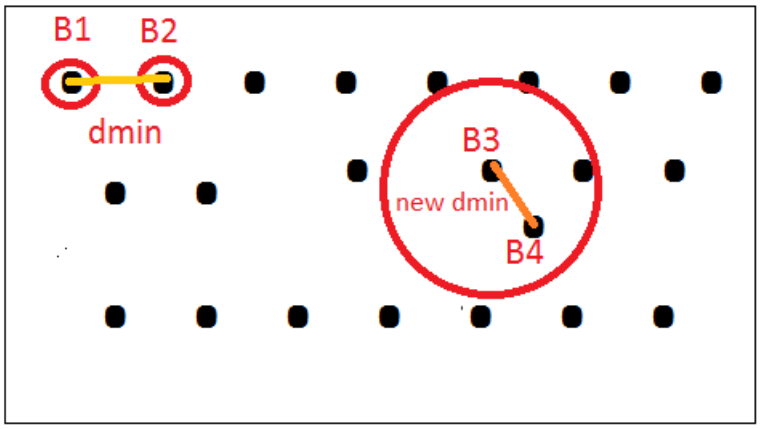
Step 2: Consider any one bug on the paper. Let that be called B_1. Find the closest bug to B_1. Let that be called B_2. Find the distance between B_1 and B_2. Consider this distance to be the minimum distance d_{min}



Step 3: Consider another bug named B_3. Draw a circle of radius d_{min} keeping B_3 as the center.

Case 1: If there is any bugs within the radius, then find the closest bug to B_3. Let that be called B_4. Find the distance between B_3 and B_4. Now this distance will become the new d_{min}

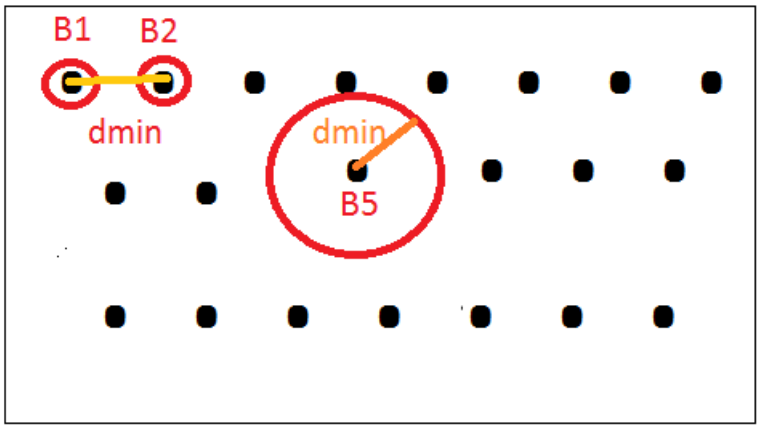
Case 2: If there is no bugs within the radius, then move on to the next bug.



Step 4: Consider the next bug B_5. Draw a circle of radius d_{min} keeping B_5 as the center.

Case 1: If there is any bugs within the radius, then find the closest bug to B_5. Let that be called B_6. Find the distance between B_5 and B_6. Now this distance will become the new d_{min}

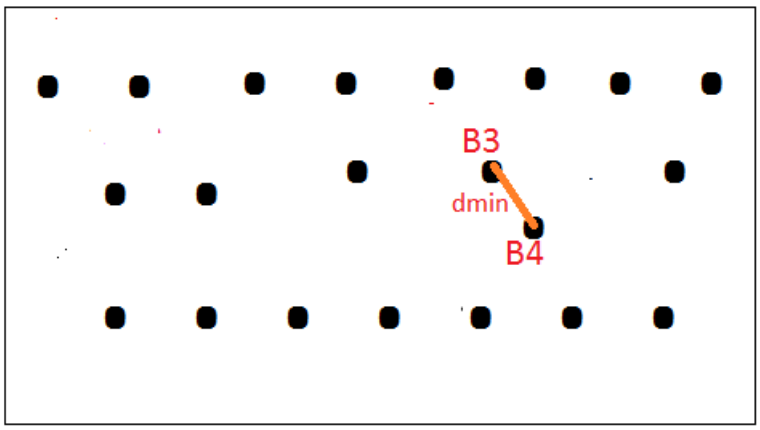
Case 2: If there is no bugs within the radius, then move on to the next bug.



Step 5: Consider the next bug B_7. Repeat Step 3.

Step 6: Repeat Step 3 for all the remaining bugs.

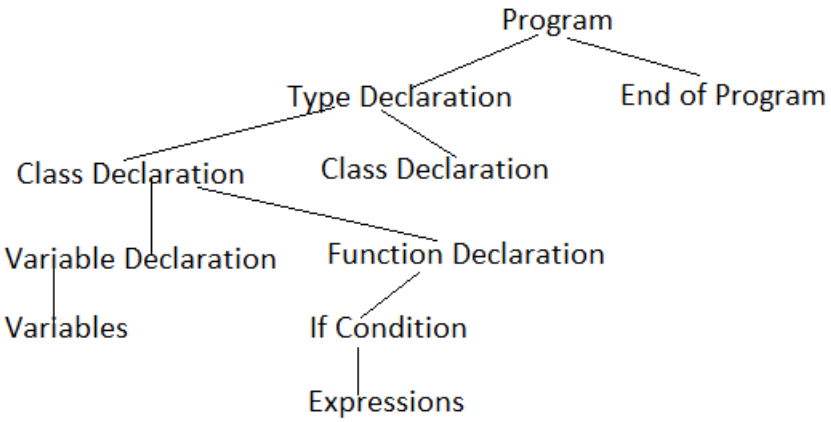
Step 7: The pair of bugs with the minimum distance d_{min} after considering all the n^2 bugs are the closest pair or bugs. Therefore here B_3 and B_4 are the closest pair of bugs



Conclusion: The time complexity is linear [O(n^2)] since all the n^2 bugs are visited only once.

1.4 Write an algorithm to decide the similarity between two C programs.

Step 1: Create a parse tree of both the C programs with each node a declaration, a variable, a function etc.



Step 2: Parse both the tress from the top to the bottom simultaneously.
Step 3: While parsing if we encounter the exact same nodes on both the trees then we can say both the programs are similar.
Step 4: While parsing if we encounter different nodes on both the trees then we can say both the programs are dissimilar.
Step 5: Find the time and space complexity of both the programs and check whether they are same or different.