Soham Chatterjee
Email: soham.chatterjee@tifr.res.in
Course: Algorithms

Problem 1 P3 (15 marks)

Solve the recurrences:

   (i) $T(n) = 2T(n/2) + n\log n,$

   (ii) $T(n) = 7T(n/3) + n^2,$

   (iii) $T(n) = \sqrt{n}T(\sqrt{n}) + n.$

Solution:

(i) We have the recurrence relation $T(n) = 2T\left(\frac{n}{2}\right) + n\log n$. So

$$
\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + n\log n \\
&= 4T\left(\frac{n}{4}\right) + \frac{n}{2}\log\frac{n}{2} + n\log n \le 2^2 T\left(\frac{n}{2^2}\right) + 2n\log n \\
&= 2^3 T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\log\frac{n}{2^2} + 2n\log n \le 2^3 T\left(\frac{n}{2^3}\right) + 3n\log n \\
&\cdots \\
&= 2^k T\left(\frac{n}{2^k}\right) + \frac{n}{2^k}\log\frac{n}{2^k} + (k-1)n\log n \le 2^k T\left(\frac{n}{2^k}\right) + kn\log n \\
&\cdots \\
&\le 2^{\log n} T(1) + \log n(n\log n) \le nT(n) + n\log^2 n = n(T(1) + \log^2 n) = O(n\log^2 n)
\end{aligned}
$$

So we claim $T(n) \le cn(T(1)+\log^2 n)$ for all $n \ge n_0$ for some c which we will choose accordingly. Now $n_0 = 2$. So for $n = 2$ we have $T(2) = 2T(1) + 2\log 2 = 2T(1) + 2 = 2(T(1) + 1) \le c2(T(1) + \log^2 2)$. Hence the base case follows. Now let $T(n) = cn\log^2 n$ is true for all $n = 2,\ldots,k-1$. Now

$$
T(k) = 2T\left(\frac{k}{2}\right) + k\log k \le 2c\frac{k}{2}\left(T(1) + \log^2\frac{k}{2}\right) + k\log k = ck\left(T(1) + \log^2\frac{k}{2}\right) + k\log k
$$

Now $\log^2\frac{k}{2} = (\log k - 1)^2 = \log^2 k - 2\log k + 1$. So we have

$$
ck\left(T(1) + \log^2\frac{k}{2}\right) + k\log k = ck(T(1)+\log^2 k) - 2ck\log k + ck + k\log k = ck(T(1)+\log^2 k) + (1-2c)k\log k + ck
$$

If $c \ge 1$ we have $1 - 2c \le -1$. So we have

$$
(1-2c)k\log k + ck \le ck - k\log k \le 0
$$

Here the last inequality follows if $c \le \log k$. Since $k \ge 2$ we have $\log k \ge 1$. So take $c = 1$. Then $(1-2c)k\log k + ck \le 0$. Therefore

$$
T(k) = k(T(1) + \log^2 k) + (1-2)k\log k + k \le k(T(1) + \log^2 k)
$$

Hence by mathematical induction we have for all $n \ge 2$, $n \in \mathbb{N}$ we have $T(n) \le n(T(1) + \log^2 n)$. Now

$$
n(T(1) + \log^2 n) = n(T(1)\log^2 n + \log^2 n) = (1 + T(1))n\log^2 n = O(n\log^2 n)
$$

Hence we have $T(n) = O(n\log^2 n)$.

(ii) We have the recurrence relation $T(n) = 7T\left(\frac{n}{3}\right) + n^2$. So

$$T(n) = 7T\left(\frac{n}{3}\right) + n^2$$

$$= 7^2 T\left(\frac{n}{3^2}\right) + \frac{n^2}{9} + n^2$$

$$= 7^3 T\left(\frac{n}{3^3}\right) + \frac{n^2}{3^4} + \frac{n^2}{3^2} + n^2 = 7^3 T\left(\frac{n}{3^3}\right) + n^2 \sum_{i=1}^{3} \frac{1}{3^{2i}}$$

$$\cdots$$

$$= 7^n T\left(\frac{n}{3^k}\right) + n^2 \sum_{i=1}^{k} \frac{1}{9^i}$$

$$\cdots$$

$$= 7^{\log_3 n} T(1) + n^2 \sum_{i=1}^{\log_3 n} \frac{1}{9^i} \leq n^{\log_7 3} T(1) + \frac{9}{8} n^2 \leq T(1)n^2 + \frac{9}{8} n^2 = \left(T(1) + \frac{9}{8}\right) n^2$$

So we claim $T(n) = (T(1) + c)n^2$ for some $c \geq 2$ and $n \geq n_0$ where $n_0 \in \mathbb{N}$. So take $n_0 = 3$. Then $T(3) = 7T(1) + 9 \leq 9T(1) + 18 \times 9 = (T(1) + c)9$. Hence this follows for the base case. Now suppose $T(n) = (T(1) + c)n^2$ for all $n = 3, \ldots, k-1$. Then for $n = k$

$$T(k) = 7T\left(\frac{k}{3}\right) + k^2 \leq 7(T(1) + c)\frac{k^2}{3^2} + k^2 = k^2\left(\frac{7(T(1) + c)}{9} + 1\right)$$

We want

$$\frac{7(T(1) + c)}{9} + 1 \leq T(1) + c \iff 7(T(1) + c) + 1 \leq 9(T(1) + c) \iff 1 \leq 2(c + T(1))$$

this is indeed true since $c \geq 2$. Hence we have $T(k) \leq (c + T(1))k^2$. Hence by mathematical induction we have $T(n) \leq (c + T(1))n^2$ for all $n \geq 4$ with $n \in \mathbb{N}$. Now $(c + T(1))n^2 = O(n^2)$. Hence $T(n) = O(n^2)$.

(iii) We have the recurrence relation

$$T(n) = \sqrt{n}T(\sqrt{n}) + n \iff \frac{T(n)}{n} = \frac{T(\sqrt{n})}{\sqrt{n}} + 1$$

Now denote $F(n) = \frac{T(n)}{n}$. Then we have the new recurrence relation

$$f(n) = f(\sqrt{n}) + 1$$

Now suppose $n = 2^{2^k}$. Then

$$f\left(2^{2^k}\right) = f\left(\sqrt{2^{2^k}}\right) + 1 = f\left(2^{2^{k-1}}\right) + 1$$

$$= f\left(2^{2^{k-2}}\right) + 2$$

$$\cdots$$

$$= f\left(2^{2^0}\right) + k$$

$$= f(2) + k$$

Now $f(2) = \frac{T(2)}{2}$ which is a constant. So there exists $n_0 \in \mathbb{N}$ such that $f(2) \leq \log\log n$ for all $n \geq n_0$. So for large k we have

$$f\left(2^{2^k}\right) = f(2) + k \leq 2k$$

Hence we claim $f(n) = O(\log_2 \log_2 n)$. For $n = n_0$ we already have $f(n_0) \leq 2\log_2 \log_2 n$. So let for $n = n_0, \ldots, t-1$ we have $f(n) \leq c\log_2 \log_2 n$ for some $c \in \mathbb{N}$. Certainly seeing the $n = n_0$ we have $c \geq 2$ but we will choose c appropriately later. Now for $n = t$

$$f(t) = f(\sqrt{t}) + 1$$
$$\leq c\log_2 \log_2(\sqrt{t}) + 1$$
$$= c\log_2\left(\frac{1}{2}\log_2 t\right) + 1$$
$$= c\log_2\frac{1}{2} + c\log_2\log_2 t + 1$$
$$= c\log_2\log_2 t - c + 1 \leq 2\log_2\log_2 t$$

So if we choose $c = 2$ then we are done. Hence by mathematical induction $f(n) = O(\log_2\log_2 n)$ for all $n \geq n_0$. Now we have $f(n) = \frac{T(n)}{n}$ and $f(n) = O(\log_2\log_2 n)$. Hence we have

$$T(n) = O(n\log_2\log_2 n)$$

■

Problem 2 P4   (5 marks)

Give the best upper bounds you can on the nth Fibonacci number F_n, where $F_n = F_{n-1} + F_{n-2}$ and $F_1 = F_2 = 1$

Solution: We have the recurrence relation $F(n) = F_{n-1} + F_{n-2}$ with $F_1 = F_2 = 1$. So we can represent this with matrices like following:

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^2\begin{bmatrix} F_{n-2} \\ F_{n-3} \end{bmatrix} = \cdots \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2}\begin{bmatrix} F_2 \\ F_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1}\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Denote $\overline{F}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $\overline{F}_k = \begin{bmatrix} F_{k+1} \\ F_k \end{bmatrix}$ and $A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$. Therefore we have $\overline{F}_n = A^n\overline{F}_0$.

Now clearly A has full rank and $\forall\, k \in \mathbb{N}, \overline{F}_k \in \mathbb{R}^2$. So we will find the eigenvalues of A to find an eigenbasis.

$$\det(A - tI) = \det\begin{bmatrix} 1-t & 1 \\ 1 & -t \end{bmatrix} = -t(1-t) - 1 = t^2 - t - 1$$

So if $t^2 - t - 1 = 0$ then

$$t = \frac{1 \pm \sqrt{1+4}}{2} = \frac{1 \pm \sqrt{5}}{2}$$

So denote $\varphi = \frac{1+\sqrt{5}}{2}$ and $\psi = \frac{1-\sqrt{5}}{2}$. Now let $X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ be an eigenvector corresponding to φ. Then

$$AX = \begin{bmatrix} x_1 + x_2 \\ x_1 \end{bmatrix} = \varphi\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Therefore $x_1 = \varphi x_2$. Therefore take $x_2 = 1$ then we have $x_1 = \varphi$. So $X = \begin{bmatrix} \varphi \\ 1 \end{bmatrix}$. Similarly we have $Y = \begin{bmatrix} \psi \\ 1 \end{bmatrix}$ is an eigenvector of A corresponding to ψ.

Now we want to express $\overline{F}_0$ as a linear combination of X, Y. Notice

$$\frac{1}{\sqrt{5}}(X - Y) = \frac{1}{\sqrt{5}}\begin{bmatrix} \varphi - \psi \\ 0 \end{bmatrix} = \frac{1}{\sqrt{5}}\begin{bmatrix} \frac{1=\sqrt{5}}{2} - \frac{1-\sqrt{5}}{2} \\ 0 \end{bmatrix} = \frac{1}{\sqrt{5}}\begin{bmatrix} \sqrt{5} \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \overline{F}_0$$

Therefore

$$\overline{F}_n = A^n\overline{F}_0 = A^n\left(\frac{1}{\sqrt{5}}(X - Y)\right) = \frac{1}{\sqrt{5}}(AX - AY) = \frac{1}{\sqrt{5}}(\varphi^n X - \psi^n Y) = \frac{1}{\sqrt{5}}\varphi^n\begin{bmatrix} \varphi \\ 1 \end{bmatrix} - \frac{1}{\sqrt{5}}\psi^n\begin{bmatrix} \psi \\ 1 \end{bmatrix}$$

Therefore $F_n = \frac{\phi^n - \psi^n}{\sqrt{5}}$.

[I knew about How to solve Linear Recurrences using Matrices]   ■

Consider two sets A and B, each having n integers in the range from 0 to $10n$. We wish to compute the Cartesian sum of A and B, defined by

$$C = \{x + y \colon x \in A, y \in B\}$$

Note that the integers in C are in the range 0 to $20n$. We want to find the elements in C and the number of times each element of C is realized as a sum of elements in A and B. Give an algorithm that solves the problem in $O(n \log n)$ time, and prove correctness.

Solution: Given A, B we create two polynomials $p_A(x) = \sum\limits_{k \in A} x^k$ and $p_B(x) = \sum\limits_{k \in B} x^k$. Since all entries of A and B ranges form 0 to $10n$. We have $\deg p_A \le 10n$ and $\deg p_B \le 10n$. Hence now we can use the algorithm for polynomial multiplication to calculate $p = p_A \cdot p_B$. Now $\deg p \le 20n$. For any term x^k in p, $\exists\, a \in A$ and $b \in B$ such that $a + b = c$ since p is the product of p_A and p_B. Let $S_k := \{(a, b) \in A \times B$ such that $a + b = k\}$. Then the coefficient of x^k in p is $|S_k|$ since

$$\text{Coeff}(x^k) = \sum_{i=0}^{k} \text{Coeff}_A(x^i) \cdot \text{Coeff}_B(x^{k-i})$$

where $\text{Coeff}_A(x^i)$ is the coefficient of x^k in p_A and $\text{Coeff}_B(x^{k-i})$ is the coefficient of x^{k-i} in p_B. Hence for any $i \in \{0, \ldots, k\}$, $\text{Coeff}_A(x^i) \cdot \text{Coeff}_B(x^{k-i})$ will contribute to $\text{Coeff}(x^k)$ iff both of them are 1 iff $(i, k - i) \in A \times B$. So

$$\text{Coeff}(x^k) = |\{(a, b) \in A \times B \mid a + b = k\}|$$

So now we will describe the algorithm. We denote the polynomial multiplication algorithm of two polynomials S, T by Polynomial-Multiplication(S, T). So the algorithm will be:

Algorithm 1: Cartesian-Sum

Input: $A = \{a_i \mid i \in [n],\ a_i \in \mathbb{Z},\ 0 \le a_i \le 10n\}$, $B = \{b_i \mid i \in [n],\ b_i \in \mathbb{Z},\ 0 \le b_i \le 10n\}$
Output: $C = \left\{ (c, k_c) \colon \exists\, a \in A,\ b \in B \text{ st } c = a + b,\ k_c = \left| \{(a, b) \in A \times B \mid a + b = c\} \right| \right\}$

1 **begin**
2 $\quad$ Create two arrays S_A and S_B of length $10n + 1$ with all elements 0
3 $\quad$ **for** $i = 1, \ldots, n$ **do**
4 $\quad\quad$ $S_A[A[i]] \longleftarrow 1$ $\qquad\qquad\qquad$ // Creates the polynomial p_A
5 $\quad\quad$ $S_B[B[i]] \longleftarrow 1$ $\qquad\qquad\qquad$ // Creates the polynomial p_B
6 $\quad$ $S \longleftarrow$ Polynomial-Multiplication(S_A, S_B)
7 $\quad$ $C \longleftarrow \{(k, S[k]) \colon S[k] \ne 0\}$
8 $\quad$ **return** C

Time Complexity: To create S_A and S_B it takes $O(n)$ time. Also for the for loop in each iteration it takes constant time and the loop runs for n iterations so the for loop in total takes $O(n)$ time.

Now in class we did the Polynomial-Multiplication algorithm which multiplies two degree $n - 1 = O(n)$ polynomial in $O(n \log n)$ time. Since the arrays S_A and S_B represents the polynomials p_A and p_B respectively and $\deg p_A, \deg p_B \le 10n = O(n)$ the Polynomial-Multiplication algorithm takes $O(n \log n)$ time to multiply them. Therefore in Line 6 it takes $O(n \log n)$ time.

Now $p = p_A \cdot p_B$. So $\deg p \le 20n$. So S is of length at most $20n = O(n)$. Therefore to create C it takes $O(n)$ time.

Therefore total time taken by the algorithm is $O(n) + O(n) + O(n \log n) + O(n) = O(n \log n)$. Hence time complexity of the algorithm is $O(n \log n)$. $\blacksquare$

Problem 4 P6 (20 marks)

Define $[n] := \{1, 2, \ldots, n\}$. You are given n, and oracle access to a function $f : [n] \times [n] \to [n] \times [n]$ that takes as input two positive integers of value at most n, and returns two positive integers of value at most n. Let $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$ be the first and second coordinates of $f(x_1, x_2)$, respectively. You are also told that f_i is monotone nondecreasing in coordinate i when coordinate $3 - i$ is kept fixed, and monotone nonincreasing in coordinate $3 - i$ when coordinate i is kept fixed. That is, given $x_1 \le x_1' \in [n]$ and $x_2 \le x_2' \in [n]$, $f_1(x_1, x_2) \le f_1(x_1', x_2)$, and $f_1(x_1, x_2) \ge f_1(x_1, x_2')$. Similarly, $f_2(x_1, x_2) \ge f_2(x_1', x_2)$, and $f_2(x_1, x_2) \le f_2(x_1, x_2')$.

The problem is to find a fixed point of the function, i.e., values $x_1, x_2 \in [n]$ so that $f(x_1, x_2) = (x_1, x_2)$. Give an algorithm that given n and oracle access to such a function f, finds a fixed point of f in time $O(\text{poly}(\log n))$. You must also give a proof of correctness, and running time analysis.

Solution: We will solve this algorithm for any $n \times m$ block where $f : [n] \times [m] \to [n] \times [m]$ with the properties same as given in the question. ∎

Problem 5 P7 (15 marks)

A palindrome is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, `civic`, `racecar`, and `aibohphobia`. Give an efficient algorithm, with proof of correctness and run-time analysis, to find the longest palindrome that is a subsequence of a given input string. For example, given the input string `character`, your algorithm should return `carac`.

Solution: We will solve this using dynamic programming. So we will create an array LPS of $n \times n$ size where for any $i, j \in [n]$ with $i \le j$, $LPS[i, j]$ denotes the longest palindromic subsequence starting in the subarray starting at i and ending at j. Let the given string is S. Now for any $i, j \in [n]$ with $i \le j$ we have:

- If $S[i] = S[j]$ then $LPS[i, j] = 2 + LPS[i + 1, j - 1]$

- If $S[i] \ne S[j]$ then $LPS[i, j] = \max\{LPS[i + 1, j], LPS[i, j - 1]\}$

This is the main recursion relation in order to find the longest palindromic subsequence. For edge cases we have

- If $i = j$ then $LPS[i, i] = 1$

- If $j = i + 1$ and $S[i] = S[j]$ then $LPS[i, j] = 2$.

In order to have the algorithm by the recursion relation we have to move by finding all the entries of LPS in each diagonal parallel to principle diagonal and then move to the next diagonal parallel to principle diagonal. There are n diagonals parallel to principle diagonal. In each diagonal d there are $n + 1 - d$ many points. And for any diagonal d the ith point on the diagonal has coordinates $(i, d + i - 1)$. So with this we describe the algorithm:

Time Complexity: The first two for loops take $O(n)$ time each. In the second for loop in each iteration we are running over $n + 1 - d = O(n)$ many iterations each of which takes a constant time. Hence it takes $O(n^2)$ time. Therefore the above algorithm takes $O(n^2)$ time to find the longest palindromic sequence. ∎

Problem 6 P8 (25 marks)

The purpose of this question is to extend the closest-points algorithm seen in the first lecture, to give an $O\left(n \log^2 n\right)$ algorithm for finding the closest pair of points in 3 dimensions. All points in this question are in $\mathbb{R}^3$.

(a) (5 marks) Prove that, if all points are at least distance δ apart, a cube with each dimension of size 2δ contains at most a constant (say k) number of points.

(b) (10 marks) You are now given 2 sets of points S_1 and S_2, each containing n points. The distance between any pair of points in S_1 is at least δ, and further, each point in S_1 has z-coordinate in $[0, \delta]$. Similarly, the distance between any pair of points in S_2 is at least δ, and each point in S_2 has z-

Algorithm 2: LONGEST-PALINDROMIC-SUBSEQUENCE(S)

Input: A string S of length n.

Output: T where $T \subseteq S$ is the Longest SPalindromic ubsequence of S and l is length of T.

1 **begin**

2     $n \longleftarrow$ LENGTH(S)

3   Create an array LPS of size $n \times n$

4   **for** $i \in [n]$ **do**

5        $LPS[i,i] = \big(1,[S[i]]\big)$

6   **for** $i \in [n-1]$ **do**

7   **if** $S[i] == S[i+1]$ **then**

8           $LPS[i,i+1] = \big(2,[S[i],S[i+1]]\big)$

9   **else**

10          $LPS[i,i+1] = \big(1,[S[i]]\big)$

11   **for** $d = 2,\ldots,n$ **do**

12   **for** $i \in [n+1-d]$ **do**

13           $j \longleftarrow d+i-1$ **if** $S[i] == S[j]$ **then**

14              $LPS[i,j] = \big(2 + LPS[i-1,j-1][1],[S[i]]++LPS[i-1,j-1][2]++[S[d+i-1]]\big)$

15   **else**

16   **if** $LPS[i+1,j][1] \geq LPS[i,j-1][1]$ **then**

17                 $LPS[i,j] = LPS[i+1,j]$

18   **else**

19                 $LPS[i,j] = LPS[i,j-1]$

20   **return** $LPS[1,n][2]$

> coordinate in $[-\delta, 0]$.
>
> Extend the algorithm discussed in class to give an $O(n\log n)$-time algorithm for finding the closest pair of points in $S_1 \cup S_2$. Note that, by the first part of the question, any cube with each dimension at most 2δ, contains at most $2k$ points from $S_1 \cup S_2$.
>
> (c) (10 marks) Given a set S of n points in $\mathbb{R}^3$, now give an $O\left(n\log^2 n\right)$-time algorithm to find the closest pair of points.

Solution:

(a) Divide the cube into small cubes with each dimension $\frac{\delta}{4}$. There are total 64 such small cubes. So there can be at most 64 points inside the cube with each dimension 2δ. Cause if there are more than 64 points in side the cube then there exists at least one small cube containing at least 2 points.

   Since any two points are at least δ distance apart in each of this cube, if there are two points inside one small cube then their distance will be at most $\frac{\delta}{4}\sqrt{3} < \delta$. This is not possible. Hence contradiction. Therefore there can be at most 64 points inside a cube with each dimension 2δ. So a cube with each dimension of size 2δ contains at most a constant number of points.

(b) We assume the points are in general position. So first in $S_1 \cup S_2$ we sort the points with respect to their x-coordinates. Each points in $S_1 \cup S_2$ has z-coodinate in $[-\delta, \delta]$. So now after sorting the points of $S_1 \cup S_2$ we start from left most point and for any point $p_i \in S_1 \cup S_2$ we take the cube with two diagonal points $(x_i, y_i - \delta, -\delta)$, $(x_i + 2\delta, y_i + \delta, \delta)$ which is a cube with each dimension 2δ i.e. in the left most side of the cube the point p_i is in the middle vertical line (Like in the picture)
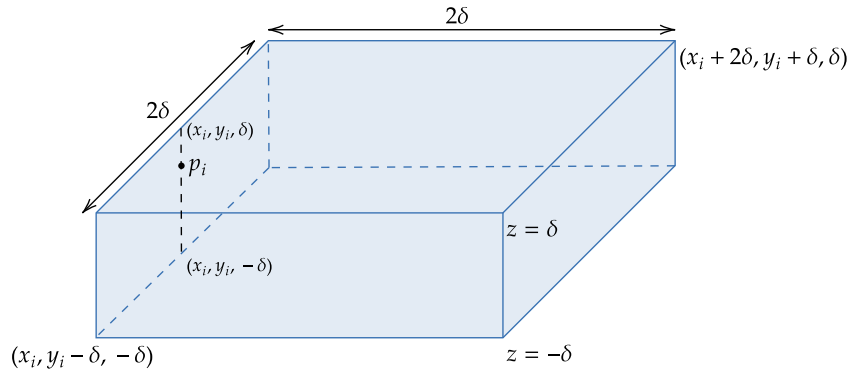
   Now by part (a) we know there are at most 64 points in this cube. So we can check for the next 64. Hence we just check the next 64 points in the sorted $S_1 \cup S_2$ and see if there are any points with distance less than δ. Hence the algorithm is:

   In the algorithm instead od checking for a point with distance less than δ we are checking pair of points with distance less than or equal to δ this is because we don't have any pair of points in S_1 or S_2 which are at least δ distance. So if there is no pair of points in $S_1 \cup S_2$ with distance less than δ we still get a pair of points with distance equal to δ and that pair points have the shortest distance.

   Time Complexity: Here to sort the points in $S_1 \cup S_2$ it takes $O(n\log n)$ time since there are total $2n$ points in $S_1 \cup S_2$. Now the for loop runs for $2n$ iterations and in each iterations it is checking for constant number of points and for each point it is doing constant time operations. So each iteration of the loop takes $O(1)$ time and the for loop takes in total $2nO(1) = O(n)$ time. Hence the whole algorithm takes $O(n\log n)$ time.

(c) Like in the case of $\mathbb{R}^2$ algorithm we will first divide the set of all points into two roughly equal parts which we can do by sorting the points with respect to their z-coordinates. Let $\bar{z}$ be the $\left\lfloor \frac{n}{2} \right\rfloor^{th}$ highest z-coordinate. We we partition S into points with z-coordinate $< \bar{z}$ and $\geq z$. Call these sets S_U and S_D respectively. Then run the algorithm recursively on each of those sets. And thus we get a closest pair of points and their distance

---

Algorithm 3: FIND-CLOSEST-2-SETS(S_1, S_2, δ)

> **Input:** Pair of sets S_1, S_2 each with n points with in each set any two points have distance at least δ and further z−coordinate of points in S_1 in $[0, \delta]$ and of points in S_2 in $[-\delta, 0]$.
>
> **Output:** Closest pair of ponts, (P_i, P_j, δ_0) where $\delta_0 = d(P_i, P_j)$ with $P_i, P_j \in S_1 \cup S_2$

1 **begin**
2      $T \longleftarrow S_1 \cup S_2$ sorted by x−coordinate
3   **for** $i = 1, \ldots, 2n$ **do**
4          $U \longleftarrow$ Next 64 points in T
5   **for** $\hat{P} \in U$ **do**
6   **if** $d(P, \hat{P}) \leq \delta$ **then**
7                  $\delta \longleftarrow d(P, \hat{P})$
8                  $(P_1, P_2) \longleftarrow (P, \hat{P})$

9   **return** (P_1, P_2, δ)

---

in each of the smaller sets. So we take the pair of points with the distance equal to the minimum of those two distances. Call this distance δ. So we only need to check the points from S with z−coordinate $\in [\bar{z} - \delta, \bar{z} + \delta]$. Call this set of points as $T = S_U^T \sqcup S_D^T$. So

$$S_U^T = \{P_i \in S_U : z_i \in [\bar{z}, \bar{z} + \delta]\} \qquad S_D^T = \{P_i \in S_D : z_i \in [\bar{z} - \delta, \bar{z}]\}$$

Now the situation is almost like the situation in part (b). So we have two sets of points S_U^T, S_D^T each set has at most $\frac{n}{2}$ points and any two points in any one of the sets has distance at least δ. Then we need to find the closest pair of points in the union of those two sets.

The only change from the situation in part (b) is the now the points in S_U^T have z−coordinate in $[\bar{z}, \bar{z} + \delta]$ instead of $[0, \delta]$ and similarly the points in S_D^T have z−coordinate in $[\bar{z} - \delta, \bar{z}]$ instead of $[-\delta, 0]$. Also like in the case of part (b) we don't have to worry about not having a pair of points with their distance $= \delta$. So we have to only look for points with distance less than δ. Therefore we need to do slight change in the algorithm above and then we are good to go.

Where we describe he FIND-CLOSEST-2-SETS-MOD algorithm:

Time Complexity: Let the algorithm takes $T(n)$ to find closest pair of points from n points. To sort the set of points with respect to their z−coordinate takes $O(n \log n)$ time. Now forming the sets S_U and S_D takes $O(n)$ time. Now $|S_U|, |S_D| \leq \frac{n}{2}$. Hence in lines 8,9 it takes $T\left(\frac{n}{2}\right)$ time. Now line 11-14 it takes constant time. Then again forming the sets S_U^T, S_D^T takes linear time. Then to run the algorithm FIND-CLOSEST-2-SETS-MOD takes $O(n \log n)$ time. Therefore total time taken is

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

Hence by Problem 1: P3 (i) we have $T(n) = O(n \log^2 n)$. Hence our algorithm takes $O(n \log^2 n)$ to find closest pair of points in $\mathbb{R}^3$.

$\blacksquare$

[I discussed the solution with Shubham]

Problem 7 P9   **(10 marks)**

This problem relates to one of the questions asked in class. For any $p, q \geq 1$, and any points x, y, and $z \in \mathbb{R}^2$, prove or disprove the following:

$$\|x - y\|_p \leq \|x - z\|_p \Leftrightarrow \|x - y\|_q \leq \|x - z\|_q$$

That is, prove or disprove that y is closer to x than z in the L_p distance metric if and only if it is closer to x

---

Algorithm 4: FIND-CLOSEST(S)

Input: Set of n points, $S = \{P_i(x_i, y_i, z_i) \mid x_i, y_i, z_i \in \mathbb{R}, \; \forall \; i \in [n]\}$.

Output: Closest pair of ponts, (P_i, P_j, δ) where $\delta = d(P_i, P_j)$

1 **begin**
2   **if** $|S| \leq 10$ **then**
3   Solve by Brute Force (Consider every pair of points)
4    $S^z \longleftarrow S$ sorted by z–coordinate
5    $\bar{z} \longleftarrow \lfloor \frac{n}{2} \rfloor$ highest z–coordinate
6    $S_U \longleftarrow \{P_i \mid z_i > \bar{z}, \; \forall \; i \in [n]\}$
7    $S_D \longleftarrow \{P_i \mid z_i \leq \bar{z}, \; \forall \; i \in [n]\}$
8    $(P_1^L, P_2^L, \delta^L) \longleftarrow$ FIND-CLOSEST(S_U)
9    $(P_1^R, P_2^R, \delta^R) \longleftarrow$ FIND-CLOSEST(S_D)
10    $\delta_{min} \longleftarrow \min\{\delta^L, \delta^R\}$
11   **if** $\delta_{min} < \delta^L$ **then**
12      $P_1 \longleftarrow P_1^R, P_2 \longleftarrow P_2^R$
13   **else**
14      $P_1 \longleftarrow P_1^L, P_2 \longleftarrow P_2^L$
15    $S_U^T \longleftarrow \{P_i \mid z_i - \bar{z} \leq \delta_{min}\}$
16    $S_D^T \longleftarrow \{P_i \mid \bar{z} - z_i \leq \delta_{min}\}$
17    $(P_1', P_2', \delta) \longleftarrow$ FIND-CLOSEST-2-SETS-MOD$\left(S_U^T, S_D^T, \bar{z}, \delta_{min}\right)$
18   **if** $\delta < \delta_{min}$ **then**
19   **return** (P_1', P_2', δ)
20   **else**
21   **return** (P_1, P_2, δ_{min})

Algorithm 5: FIND-CLOSEST-2-SETS-MOD(S_1, S_2, z, δ)

Input: Pair of sets S_1, S_2 each with n points with in each set any two points have distance at least δ and further z–coordinate of points in S_1 in $[z, z + \delta]$ and of points in S_2 in $[z - \delta, z]$.

Output: Closest pair of ponts, (P_i, P_j, δ_0) where $\delta_0 = d(P_i, P_j)$ with $P_i, P_j \in S_1 \cup S_2$

1 **begin**
2    $T \longleftarrow S_1 \cup S_2$ sorted by x–coordinate
3   **for** $i = 1, \ldots, 2n$ **do**
4      $U \longleftarrow$ Next 64 points in T
5   **for** $\hat{P} \in U$ **do**
6   **if** $d(P, \hat{P}) < \delta$ **then**
7          $\delta \longleftarrow d(P, \hat{P})$
8          $(P_1, P_2) \longleftarrow (P, \hat{P})$
9   **return** (P_1, P_2, δ)

> in the L_q distance metric As usual, $\|x - y\|_p = ((x_1 - y_1)^p + (x_2 - y_2)^p)^{1/p}$.

Solution: This question reduces to showing for any two points $x, y \in \mathbb{R}^2$ if the following is true of not

$$\|x\|_p \leq \|y\|_p \iff \|x\|_q \leq \|y\|_q$$

This is not true. Take $p = 1$, $q = 2$. And $x = (1, 0)$ and $y = (0.55, 0.55)$. Then $\|x\|_1 = \|x\|_2 = 1$. But $\|y\| = 1.1$, $\|y\| = 0.55\sqrt{2} < 1$. Hence $\|x\|_1 < \|y\|_1$ but $\|x\|_2 > \|y\|_2$. Hence the above claim is not true. ∎