CSC373 Worksheet 1 Solution

July 23, 2020

1.

```
Strassen_Algorithm(A,B):
    n = A.rows
    let C be a new n x n matrix

    if n == 1
        C_11 = A_11 * B_11

    else partition as in step 3 of strassen's algorithm

        p1 = Strassen_Algorithm(A_11, B_12) -
             Strassen_Algorithm(A_11, B_22)

        p2 = Strassen_Algorithm(A_11, B_22) +
             Strassen_Algorithm(A_12, B_22)

        p3 = Strassen_Algorithm(A_21, B_11) +
             Strassen_Algorithm(A_22, B_11)

        p4 = Strassen_Algorithm(A_22, B_21) -
             Strassen_Algorithm(A_22, B_11)

        p5 = Strassen_Algorithm(A_11, B_11) +
             Strassen_Algorithm(A_11, B_22) +
             Strassen_Algorithm(A_22, B_11) +
             Strassen_Algorithm(A_22, B_22)

        p6 = Strassen_Algorithm(A_12, B_21) +
             Strassen_Algorithm(A_12, B_22) -
             Strassen_Algorithm(A_22, B_21) -
             Strassen_Algorithm(A_22, B_22)

        p7 = Strassen_Algorithm(A_11, B_11) +
             Strassen_Algorithm(A_11, B_12) -
             Strassen_Algorithm(A_21, B_11) -
             Strassen_Algorithm(A_21, B_12)

        C_11 = p5 + p4 - p2 + p6
        C_12 = p1 + p2
        C_21 = p3 + p4
```

```
            C_22 = p5 + p1 - p3 - p7

        return C

```

Notes:

- Strassen's method for matrix multiplication
  - Reduces the time complexity of matrix multiplication from $O(n^3)$ to $O(n^{\log_2 7}) = O(n^{2.81})$
  - Has four steps

1) Divide the input matrics A and B and output matrix C into $n/2 \times n/2$ submatrices

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

2) Create 10 matrices, $S_1, S_2, ..., S_{10}$ each of which is $n/2 \times n/2$ and is the sum or difference of two matrices created in step 1

$S_1 = B_{12} - B_{22}$
$S_2 = A_{11} + A_{12}$
$S_3 = A_{21} + A_{22}$
$S_4 = B_{21} - B_{11}$
$S_5 = A_{11} + A_{22}$
$S_6 = B_{11} + B_{22}$
$S_7 = A_{12} - A_{22}$
$S_8 = B_{21} + B_{22}$
$S_9 = A_{11} - A_{21}$
$S_{10} = B_{11} + B_{12}$

3) Recursively multiply $n/2 \times n/2$ matrices seven times to compute the following $n/2 \times n/2$ matrices

$P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}$
$P_2 = S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}$
$P_3 = S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}$
$P_4 = A_{22} \cdot S_4 = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}$
$P_5 = S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}$
$P_6 = S_7 \cdot S_8 = A_{12} \cdot B_{21} + A12 \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}$
$P_7 = S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}$

4) Construct the four $n/2 \times n/2$ submatrices of the product C

$C_{11} = P_5 + P_4 - P_2 + P_6 = A_{11} \cdot B_{11} + A_{12} + B_{12}$
$C_{12} = P_1 + P_2 = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$
$C_{21} = P_3 + P_4 = A_2 1 \cdot B_1 1 + A_{22} \cdot B_{21}$
$C_{22} = P_5 + P_1 - P_3 - P_7 = A_{22} \cdot B_{22} + A_{21} \cdot B_{12}$

Example: Use Strassen's algorithm to compute the matrix product

$$\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix} \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}$$

* **STEP 1**

  $A_{11} = 1, A_{12} = 3, A_{21} = 7, A_{22} = 5$

  $B_{11} = 6, B_{12} = 8, B_{21} = 4, B_{22} = 2$

* **STEP 2**

  $S_1 = B_{12} - B_{22} = 4 - 2 = 2$

  $S_2 = A_{11} + A_{12} = 1 + 3 = 4$

  $S_3 = A_{21} + A_{22} = 7 + 5 = 12$

  $S_4 = B_{21} - B_{11} = 4 - 6 = -2$

  $S_5 = A_{11} + A_{22} = 1 + 5 = 6$

  $S_6 = B_{11} + B_{22} = 6 + 2 = 8$

  $S_7 = A_{12} - A_{22} = 3 - 5 = -2$

  $S_8 = B_{21} + B_{22} = 8 + 2 = 10$

  $S_9 = A_{11} - A_{21} = 3 - 5 = -2$

  $S_{10} = B_{11} + B_{12} = 6 + 4 = 10$

* **STEP 3**

  $P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22} = 1 \cdot 4 - 1 \cdot 2 = 2$

  $P_2 = S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22} = 1 \cdot 2 + 3 \cdot 2 = 8$

  $P_3 = S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11} = 6 \cdot 7 + 6 \cdot 5 = 72$

  $P_4 = A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11} = 5 \cdot 4 - 5 \cdot 6 = -10$

  $P_5 = S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} = 48$

  $P_6 = S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22} = -20$

  $P_7 = S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12} = -20$

- **STEP 4**

  $C_{11} = P_5 + P_4 - P_2 + P_6 = 48 - 10 - 8 - 20 = 10$

  $C_{12} = P_1 + P_2 = 10$

  $C_{21} = P_3 + P_4 = 62$

  $C_{22} = P_5 + P_1 - P_3 - P_7 = 48 + 2 - 72 + 20 = -2$

- Is not preferred in practical purposes

  1) The constants used in Strassen's method are high and for a typical application Naive method works better.
  2) For Sparse matrices, there are better methods especially designed for them.
  3) The submatrices in recursion take extra space.
  4) Because of the limited precision of computer arithmetic on noninteger values, larger errors accumulate in Strassen's algorithm than in Naive Method

References:

1) GeeksForGeeks, Divide and Conquer — Set 5 (Strassen's Matrix Multiplication), link

- Regular matrix multiplication
  -
- The master method for solving recurrences
  - provides 'cookbook' method for solving recurrences of the form

$$T(n) = aT(n/b) + f(n)$$

  - depends on the following theorem
    - Let $a \leq 1$ and $b > 1$ be constants, let $f(n)$ be a function and let $T(n)$ be defined on the nonnegative integers by the recurrence

      $T(n) = aT(n/b) + f(n)$

      where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

      1. If $f(n) = O(n^{\log_b a - \epsilon}) for some constant \epsilon > 0, then T(n) = \Theta(n^{\log_b a})$
      2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$
      3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$, and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Example:

$T(n) = 9T(n/3) + n$

Here, $a = 9$, $b = 3$, and $f(n) = n = O(n^{\log_3 9-1})$ where $\epsilon = 1$.

Thus, $T(n) = \Theta(n^{\log_3 9})$ or $T(n) = \Theta(n^2)$

Example 2:

$T(n) = T(2n/3) + 1$

Here, $a = 1$, $b = 3/2$, $f(n) = 1 = \Theta(n^{\log_{3/2} 1})$.

Thus, $T(n) = \theta(\lg n)$

Example 3:

$T(n) = T(n/4) + n \lg n$

Here $a = 1$, $b = 4$, and $f(n) = n \lg n$ has asymptotic lowerbound of $f(n) = \Omega(n^{\log_4 3+\epsilon}) = \Omega(n)$ where $\epsilon \approx 0.2$

Furthermore,

$$\begin{aligned}
af(n/b) &= (3n/4) \lg n/4 \\
&= (3/4)n \lg n/4 \\
&= (3/4)n \lg n/4 \\
&= 3/4n \lg n - \lg 4 \\
&< 3/4n \lg n \\
&= cf(n)
\end{aligned}$$

where $c = 3/4$.

Thus, $T(n) = \Theta(n \lg n)$

Example 4:

$T(n) = 2T(n/2) + n \lg n$

Here, $a = 2$, $b = 2$, $f(n) = n \lg n$.

2. Let $n = 3^m$ where m is an element of $\mathbb{Z}^+ \cup \{0\}$

Then we know the time it takes to multiply $n \times n$ matrices in 3×3 matrices is $T(n) = kT(\frac{n}{3}) + \Theta(n^2)$.

Now, I need to look for the upper bound of k in $T(n) = \Theta(n^{\log_3 k})$ satisfying $O(n^{\lg 7}) \approx O(n^{2.81})$.

And using master's master's theorem, we can write that the upper limit of k is 21.

Improved Solution:

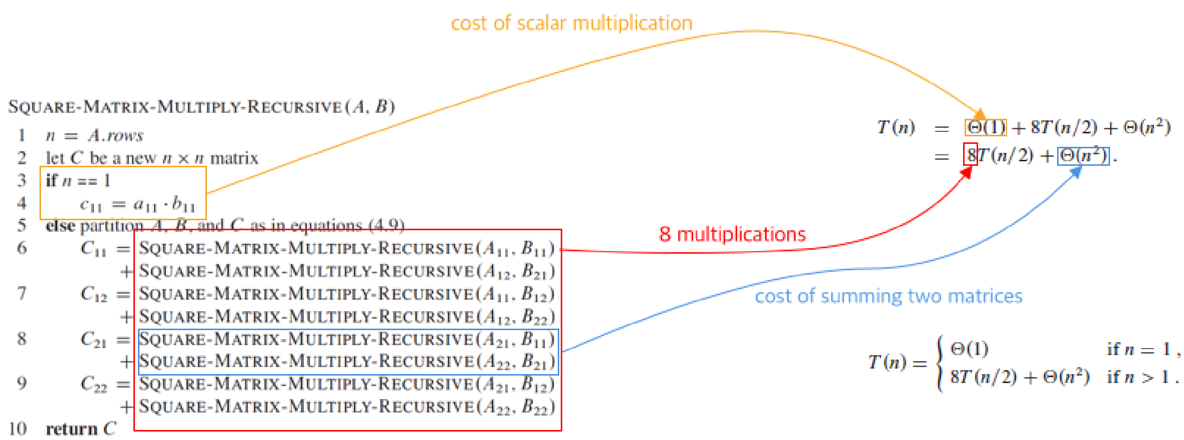
Let $n = 3^m$ where m is an element of $\mathbb{Z}^+ \cup \{0\}$

Then we know the time it takes to multiply $n \times n$ matrices in 3×3 matrices is $T(n) = kT(\frac{n}{3}) + \Theta(n^2)$.

Now, I need to look for the upper bound of k in $T(n) = \Theta(n^{\log_3 k})$ satisfying $O(n^{\lg 7}) \approx O(n^{2.81})$.

And using master's master's theorem, we can write that the upper limit of k is 21 (Following the first condition $f(n) = \mathcal{O}(n^{\log_3 k - \epsilon})$ where $\epsilon \approx 0.81$).

Notes:



- $T(n)$ represents the time it takes to multiply two $n \times n$ matrices.
- At base case scalar multiplication is performed. So, $T(1) = \Theta(1)$.
- 8 represents the number of recursive calls on the function SQUARE-MATRIX-MULTIPLY-RECURSIVE
- $\Theta(n^2)$ represents the addition of two $\frac{n}{2} \times \frac{n}{2}$ matrices

3. - 68×68 matrices using $132,464$ multiplications:
     - Has recurrence of form $T(n) = 132,464T(\frac{n}{68}) + \Theta(n^2)$
     - Has $a = 132,464, b = 68, f(n) = \Theta(n^2)$
     - Since $f(n) = \Theta(n^{\log_b a - \epsilon})$ where $\epsilon \approx 0.80$, case 1 of master's theorem applies and $T(n) = \Theta(n^{\log_b a}) \approx \Theta(n^{2.80})$
   - 70×70 matrices using $143,640$ multiplications:

- Has recurrence of form $T(n) = 143,640T(\frac{n}{70}) + \Theta(n^2)$
- Has $a = 143,640, b = 70, f(n) = \Theta(n^2)$
- Since $f(n) = \Theta(n^{\log_b a - \epsilon})$ where $\epsilon \approx 0.80$, case 1 of master's theorem applies and $T(n) = \Theta(n^{\log_b a}) \approx \Theta(n^{2.80})$

- 72×72 matrices using $155,424$ multiplications:
  - Has recurrence of form $T(n) = 155,424(\frac{n}{72}) + \Theta(n^2)$
  - Has $a = 155,424, b = 72, f(n) = \Theta(n^2)$
  - Since $f(n) = \Theta(n^{\log_b a - \epsilon})$ where $\epsilon \approx 0.80$, case 1 of master's theorem applies and $T(n) = \Theta(n^{\log_b a}) \approx \Theta(n^{2.80})$

They all have the same asymptotic running time (오잉?!).

In comparison to Strassen method (which has $\Theta(n^{\lg 7}) \approx \Theta(n^{2.81})$), the above three divide and conquer algorithms are a bit faster.

Correct Solution:

We need to find the divide and conquer method that yields the best asymptotic running time.

Using master's method, we have:

- $T(n) = 132,464T(\frac{n}{68}) + \Theta(n^2) \rightarrow T(n) \approx \Theta(n^{2.7951284873613815})$
- $T(n) = 143,640T(\frac{n}{70}) + \Theta(n^2) \rightarrow T(n) \approx \Theta(n^{2.795122689748337})$
- $T(n) = 155,424T(\frac{n}{72}) + \Theta(n^2) \rightarrow T(n) \approx \Theta(n^{2.795147391093449})$

Based on the above, the second method $T(n) = 143,640T(\frac{n}{70})$ has the best asymptotic running time.

In comparison to Strassen method (which has $\Theta(n^{\lg 7}) \approx \Theta(n^{2.81})$), the above three divide and conquer algorithm is a bit faster.

4. Consider three multiplications

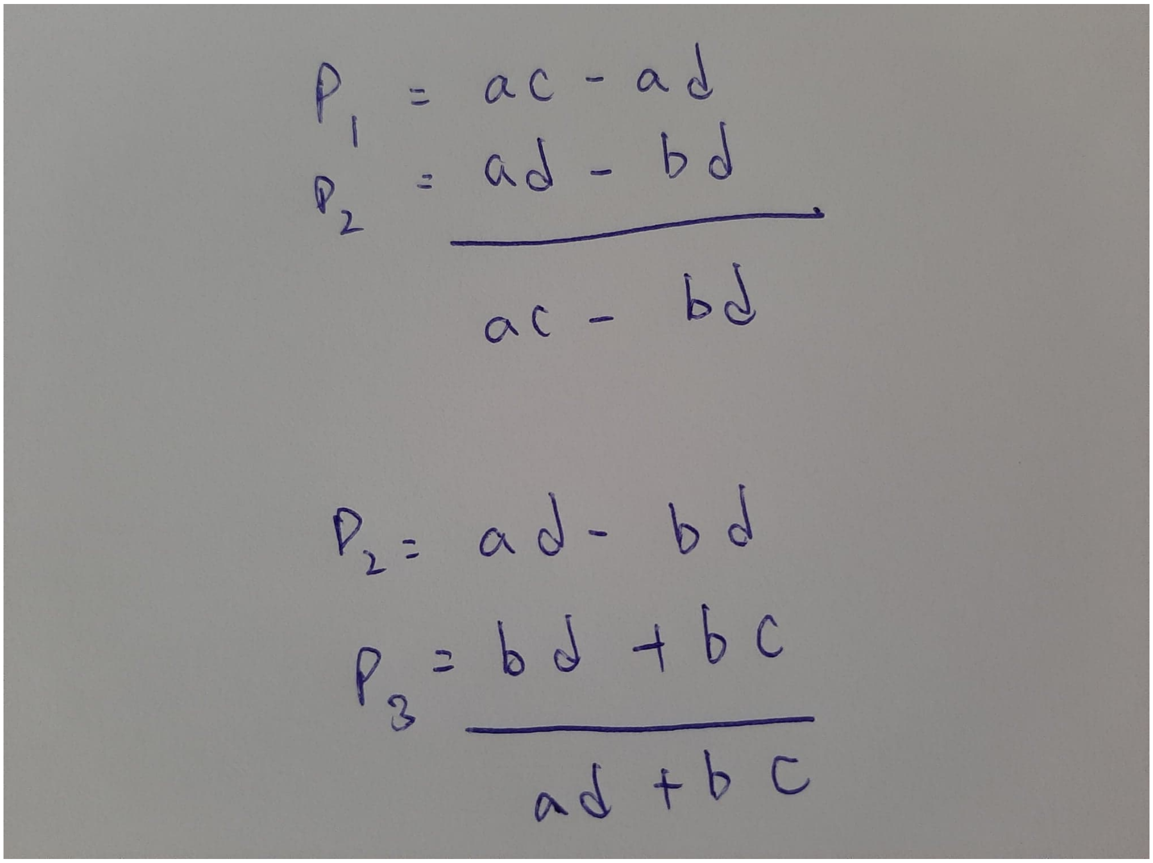
1. $P_1 = ac - ad$
2. $P_2 = ac - ad$
3. $P_3 = bd + bc$

Then, we have

$P_1 + P_2 = (ac - ad) + (ad - bd) = ac - bd$
$P_2 + P_3 = (ad - bd) + (bd + bc) = ad + bc$

Notes:

- I arrived to solution by working backward, by laying the solution, the known pieces, and then finding the missing ones.



5. a) Here, $a = 2, b = 2, f(n) = 4$.

   Since $f(n) = n^{\log_2 2+3} = n^{\log_b a+\epsilon}$ where $\epsilon = 3$, and $af(\frac{n}{b}) = 2\left(\frac{n^4}{16}\right) = \frac{n^4}{8} \leq cn^4$ where $c = \frac{1}{8}$ for sufficiently large n, the case 3 of master's theorem applies.

   Thus, $T(n)$ has upper bound of $\mathcal{O}(n^4)$ and lower bounds of $\Omega(n^4)$, or $\Theta(n^4)$.

   b) Here $a = 1, b = \frac{10}{7}, f(n) = n$.

   Since $f(n) = 1 = n^{0+1} = n^{\log_{10/7}(1)+1} = n^{\log_b(a)+\epsilon}$, where $\epsilon = 1$, and $af(\frac{n}{b}) = \frac{7n}{10} \leq cn^4$ where $c = \frac{7}{10}$ for sufficiently large n, the case 3 of master's theorem applies.

   Thus, $T(n)$ has upper bound of $\mathcal{O}(n)$ and lower bounds of $\Omega(n^4)$, or $\Theta(n)$.

c) Here we have $a = 16, b = 4, f(n) = n^2$.

Since $f(n) = n^2 = n^{\log_4 16} = n^{\log_b a}$, case 2 of master's theorem applies.

Thus, $T(n)$ has upper bound of $\mathcal{O}(n^2 \lg n)$ and lower bounds of $\Omega(n^2 \lg n)$.

d) Here we have $a = 7, b = 3, f(n) = n^2$.

Since $n^2 = n^{\log_3 7+\epsilon} = n^{\log_b a+\epsilon}$ where $\epsilon \approx 0.23$, and $af(\frac{n}{b}) \leq cn^2$ where $c = \frac{7}{9}$, the case 3 of master's theorem applies.

Thus, $T(n)$ has upper bound of $\mathcal{O}(n^2)$ and lower bounds of $\Omega(n^2)$, or $\Theta(n^2)$.

e) Here we have $a = 7, b = 2, f(n) = n^2$.

Since $f(n) = n^2 = n^{\log_2(7)-\epsilon}$, where $\epsilon \approx 0.81$, case 1 of master theorem applies.

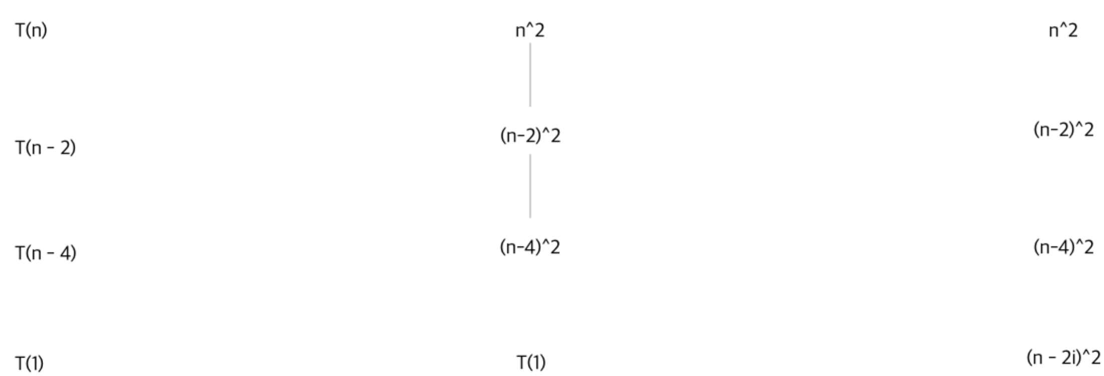
Thus, $T(n)$ has upper bound of $\mathcal{O}(n^{\log_2 7})$ and lower bound of $\Omega(n^{\log_2 7})$, or $\Theta(n^{\log_2 7})$.

f) Here we have $a = 2, b = 4, f(n) = \sqrt{(n)}$.

Since $f(n) = \sqrt{n} = n^{\log_4 2} = n^{\log_b a}$, case 2 of master's theorem applies.

Thus $T(n)$ has upper bound of $\mathcal{O}(\sqrt{n} \lg n)$, and lower bound of $\Omega(\sqrt{n} \lg n)$, or $\Theta(\sqrt{n} \lg n)$.

g) **<u>Solution:</u>**



Using recurrence tree method, we can see that the tree has depth of $\frac{n}{2}$, level cost of $(n - 2i)^2$ where $i = 0, 1, ..., n - 1$, and bottom level cost of $T(1)$ or $\Theta(1)$.

So, the total cost of $T(n)$ is:

$$
\begin{align}
T(n) &= \sum_{i=0}^{\frac{n}{2}-1}(n-2i) + \Theta(1) \tag{1}\\
&= \frac{n^2}{2} - 2\sum_{i=0}^{\frac{n}{2}-1} i + \Theta(1) \tag{2}\\
&= \frac{n^2}{2} - \left(\frac{n}{2}\right)\left(\frac{n}{2}-1\right) + \Theta(1) \tag{3}\\
&= \left(\frac{n^2}{2}\right) - \left(\frac{n^2}{4} - \frac{n}{2}\right) + \Theta(1) \tag{4}\\
&= \frac{n^2}{4} + \frac{n}{2} + \Theta(1) \tag{5}\\
&= \Theta(n^2) \tag{6}
\end{align}
$$

And to verify $T(n)$, I will use subtitution method.

Let the guess be $T(n) \leq cn^3$.

Then,

$$
\begin{align}
T(n) &= T(n-2) + n^2 \tag{7}\\
&\leq c(n-2)^3 + n^2 \tag{8}\\
&= c(n^3 - 6n^2 + 12n - 8) + n^2 \tag{9}\\
&= c(n^3 - 5n^2 + 12n - 8) - n^2(c-1) \tag{10}\\
&\leq c(n^3 + 12n - 8) - n^2(c-1) \tag{11}\\
&= cn^3 - n^2(c-1) && \text{[Since } n^3 \text{ dominates } n\text{]} \tag{12}\\
&\leq cn^3 \tag{13}
\end{align}
$$

and the boundary holds as long as $c \geq 1$.

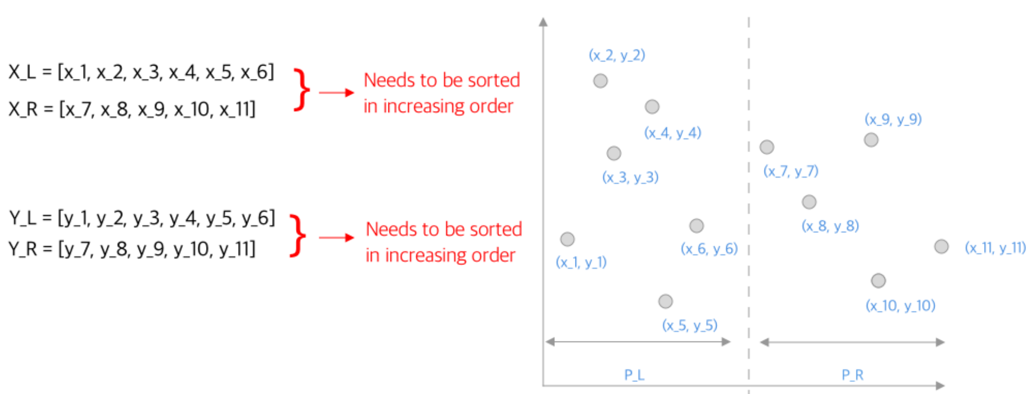
6. **Notes:**

- Computational Geometry
  - Is the study of algorithm for solving geometric problems
  - Has applications in
    1. Computer graphics
    2. Robotics
    3. VLSI design

4. Computer-aided design
5. Molecular modeling
6. Metallurgy
7. Manufacturing (!!!)
8. Textile layout
9. Forestry
10. Statistics

- Finding the closest pair of paths
  - "closest" - the distance between two points in euclidean space $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$, or $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
  - Brute force method takes $\Theta(n^2)$ Time
  - Divide and conquer method takes $T(n) = 2T(\frac{n}{2}) + \mathcal{O}(n)$ time, or $\mathcal{O}(n \lg n)$ time.
- Finding closest pair of paths using divide and conquer algorithm
  1. Divide
     - Find a vertical line l that bisects the point set P into two sets P_L and P_R
       * $|P_L| = \lceil |P|/2 \rceil, |P_R| = \lfloor |P|/2 \rfloor$.

         In other words, take the norm of all points, and divide it into two halves
     - Divide the array X into X_L and X_R, and Y into Y_L and Y_R
       * X_L : x-coordinate points from P_L
       * X_R : x-coordinate points from P_R
       * Y_L : y-coordinate points from P_L
       * Y_R : y-coordinate points from P_R



  2. Having divided P into P_L and P_R
     - Make one recursive call to find the closest pair of points in P_L
       * The inputs are P_L, X_L, Y_L
       * The returned value is defined as δ_L
     - Make the other recursive call to find the other closest pair of points in P_R

* The inputs are P_R, X_R, Y_R
* The returned value is defined as δ_R

7. •