# COL351: Analysis and Design of Algorithms

Ragesh Jaiswal, CSE, IITD

## Course Overview

- Basic graph algorithms
- Algorithm Design Techniques:
  - Greedy Algorithms
  - Divide and Conquer
  - Dynamic Programming
  - Network Flows
- Computational Intractability

## Divide and Conquer

- You have already seen multiple examples of Divide and Conquer algorithms:
  - Binary Search
  - Merge Sort
  - Quick Sort
  - Multiplying two $n$-bit numbers in $O\left(n^{\log_2 3}\right)$ time.

## Divide and Conquer
### Main Idea

- Main Idea: *Divide the input into smaller parts. Solve the smaller parts and combine their solution.*

# Divide and Conquer
## Merge Sort

**Problem**

Given an array of unsorted integers, output a sorted array.

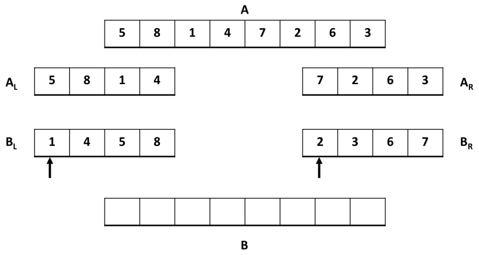
**Algorithm**

```
MergeSort($A$)
    - If ($|A| = 1$) return($A$)
    - Divide $A$ into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return($B$)
```

## Divide and Conquer
### Merge Sort

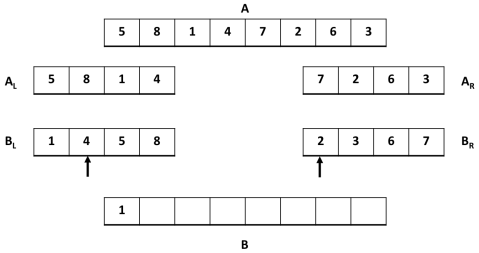
**Algorithm**

```
MergeSort($A$)
    - If ($|A| = 1$) return($A$)
    - Divide $A$ into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return($B$)
```



## Algorithm

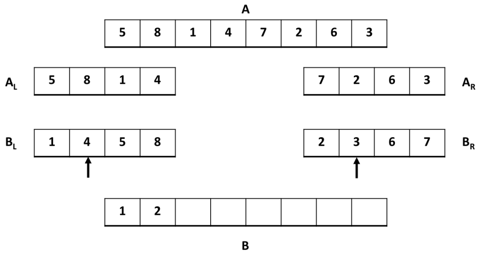
```
MergeSort($A$)
    - If ($|A| = 1$) return($A$)
    - Divide $A$ into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return($B$)
```



# Divide and Conquer
## Merge Sort

### Algorithm

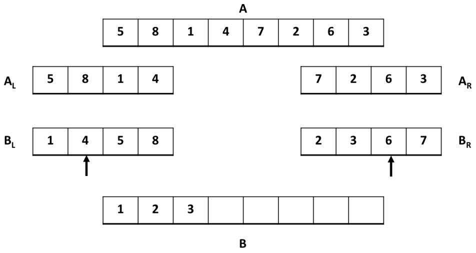
```
MergeSort($A$)
    - If ($|A| = 1$) return($A$)
    - Divide $A$ into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return($B$)
```



# Divide and Conquer
## Merge Sort

### Algorithm

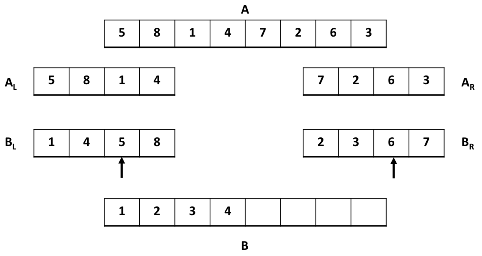
```
MergeSort(A)
    - If (|A| = 1) return(A)
    - Divide A into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return(B)
```



# Divide and Conquer
## Merge Sort

### Algorithm

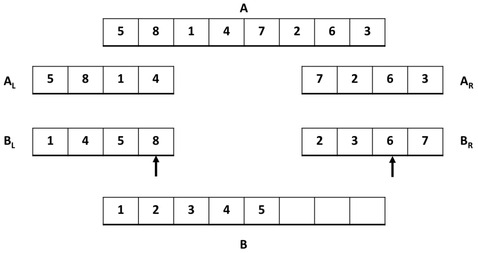
```
MergeSort($A$)
    - If ($|A| = 1$) return($A$)
    - Divide $A$ into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return($B$)
```



# Divide and Conquer
## Merge Sort

### Algorithm

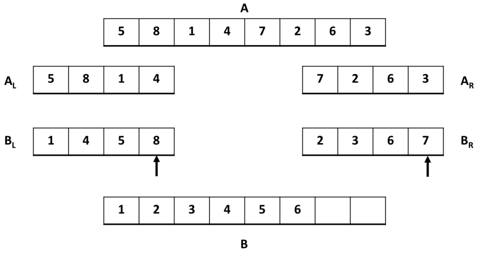
```
MergeSort(A)
    - If (|A| = 1) return(A)
    - Divide A into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return(B)
```



# Divide and Conquer
## Merge Sort

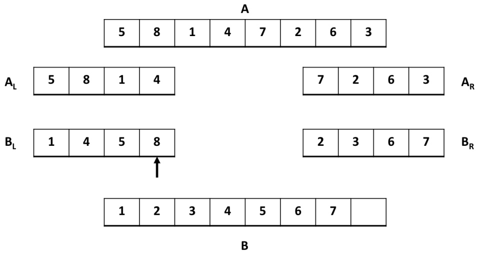
### Algorithm

```
MergeSort($A$)
    - If ($|A| = 1$) return($A$)
    - Divide $A$ into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return($B$)
```



## Algorithm

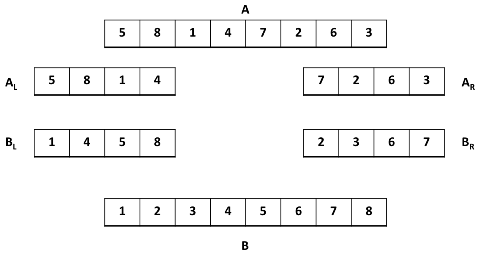
```
MergeSort($A$)
    - If ($|A| = 1$) return($A$)
    - Divide $A$ into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return($B$)
```



## Divide and Conquer
### Merge Sort

**Algorithm**

```
MergeSort($A$)
    - If ($|A| = 1$) return($A$)
    - Divide $A$ into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return($B$)
```



## Divide and Conquer
### Merge Sort

**Algorithm**

```
MergeSort($A$)
    - If ($|A| = 1$) return($A$)
    - Divide $A$ into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return($B$)
```

- How do we argue correctness?

## Divide and Conquer
### Merge Sort

**Algorithm**

```
MergeSort($A$)
    - If ($|A| = 1$) return($A$)
    - Divide $A$ into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return($B$)
```

- How do we argue correctness?
- Proof of correctness of Divide and Conquer algorithms are usually by induction.
    - Base case: This corresponds to the base cases of the algorithm. For the MergeSort, the base case is that the algorithm correctly sorts arrays of size 1.
    - Inductive step: In general, this corresponds to correctly combining the solutions of smaller subproblems. For MergeSort, this is just proving that the Merge routine works correctly. This may again be done using induction and is left as an exercise.

# Divide and Conquer
## Merge Sort

**Algorithm**

```
MergeSort($A$)
    - If ($|A| = 1$) return($A$)
    - Divide $A$ into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return($B$)
```

- Let $n$ be a power of 2 (e.g., $n = 256$)
- Let $T(n)$ denote the worst case running time for the algorithm.
- Claim 1: $T(1) \leq c$ for some constant $c$.

## Divide and Conquer
### Merge Sort

**Algorithm**

```
MergeSort($A$)
    - If ($|A| = 1$) return($A$)
    - Divide $A$ into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return($B$)
```

- Let $n$ be a power of 2 (e.g., $n = 256$)
- Let $T(n)$ denote the worst case running time for the algorithm.
- Claim 1: $T(1) \leq c$ for some constant $c$.
- Claim 2: $T(n) \leq 2 \cdot T(n/2) + cn$ for all $n \geq 2$.

# Divide and Conquer
## Merge Sort

### Algorithm

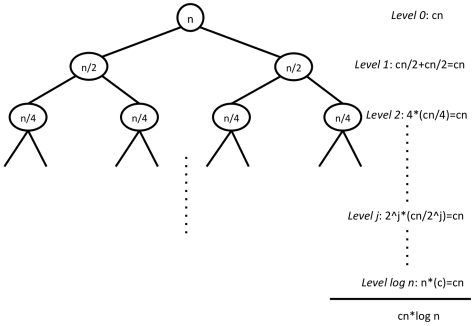
```
MergeSort($A$)
    - If ($|A| = 1$) return($A$)
    - Divide $A$ into two equal parts $A_L$ and $A_R$
    - $B_L \leftarrow$ MergeSort($A_L$)
    - $B_R \leftarrow$ MergeSort($A_R$)
    - $B \leftarrow$ Merge($B_L, B_R$)
    - return($B$)
```

- Let $n$ be a power of 2 (e.g., $n = 256$)
- Let $T(n)$ denote the worst case running time for the algorithm.
- Claim 1: $T(1) \leq c$ for some constant $c$.
- Claim 2: $T(n) \leq 2 \cdot T(n/2) + cn$ for all $n \geq 2$.
- $T(n) \leq 2 \cdot T(n/2) + cn$ for $n \geq 2$ and $T(1) \leq c$ is called a *recurrence relation* for the running time $T(n)$.
- How do we solve such recurrence relation to obtain the value of $T(n)$ as a function of $n$?

- Let $n$ be a power of 2 (e.g., $n = 256$)
- Let $T(n)$ denote the worst case running time for the algorithm.
- Claim 1: $T(1) \leq c$ for some constant $c$.
- Claim 2: $T(n) \leq 2 \cdot T(n/2) + cn$ for all $n \geq 2$.
- $T(n) \leq 2 \cdot T(n/2) + cn$ for $n \geq 2$ and $T(1) \leq c$ is called a *recurrence relation* for the running time $T(n)$.
- How do we solve such recurrence relation to obtain the value of $T(n)$ as a function of $n$?
  - Unrolling the recursion: Rewrite $T(n/2)$ in terms of $T(n/4)$ and so on until a pattern for the running time with respect to all levels of the recursion is observed. Then, combine these and get the value of $T(n)$.

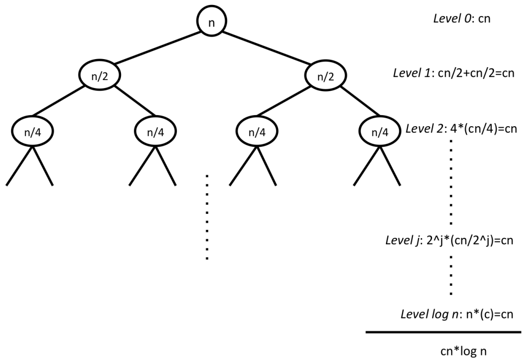
# Divide and Conquer
## Merge Sort

- Recurrence relation for Merge Sort: $T(n) \leq 2 \cdot T(n/2) + cn$ for $n \geq 2$ and $T(1) \leq c$.
- How do we solve such recurrence relation to obtain the value of $T(n)$ as a function of $n$?
  - Unrolling the recursion: Rewrite $T(n/2)$ in terms of $T(n/4)$ and so on until a pattern for the running time with respect to all levels of the recursion is observed. Then, combine these and get the value of $T(n)$.



# Divide and Conquer
## Merge Sort

- Recurrence relation for Merge Sort: $T(n) \leq 2 \cdot T(n/2) + cn$ for $n \geq 2$ and $T(1) \leq c$.
- How do we solve such recurrence relation to obtain the value of $T(n)$ as a function of $n$?
- So, the running time $T(n) \leq cn \cdot \log n = O(n \log n)$.



## Divide and Conquer
### Solving recurrence relations

- Question: Suppose there is a divide and conquer algorithm where the recurrence relation for running time $T(n)$ is the following:

  $$T(n) \leq 2T(n/2) + cn^2 \text{ for } n \geq 2, \text{ and } T(1) \leq c.$$

  What is the solution of this recurrence relation in big-oh notation?

## Divide and Conquer

### Master theorem

**Theorem**

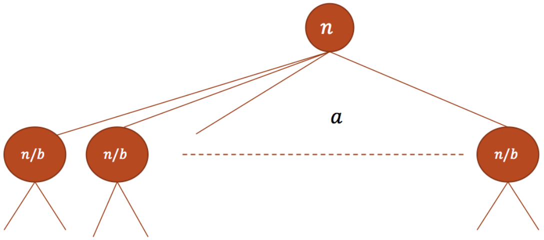
Master Theorem: Let

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k \quad \text{and} \quad T(1) \leq c,$$

Then

$$T(n) = \begin{cases} ? & \text{if } a < b^k \\ ? & \text{if } a = b^k \\ ? & \text{if } a > b^k \end{cases}$$

# Divide and Conquer
## Master theorem



$c \cdot n^k$

$c \cdot a \cdot (n/b)^k$

$c \cdot n^k \cdot (a/b^k)^2$

$c \cdot n^k \cdot (a/b^k)^i$

$$c \cdot n^k \cdot (1 + r + r^2 + \dots + r^{\log(n)/\log(b)}), r = a/b^k$$

# Divide and Conquer
## Master theorem

**Theorem**

Master Theorem: Let

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k \quad \text{and} \quad T(1) \leq c,$$

Then

$$T(n) = \begin{cases} O(n^k) & \text{if } a < b^k \\ O(n^k \cdot \log_b n) & \text{if } a = b^k \\ O\left(n^{\log a / \log b}\right) & \text{if } a > b^k \end{cases}$$

## Divide and Conquer
### Master theorem

**Problem**

Multiplying two $n$-bit numbers: Given two $n$-bit numbers, $A$ and $B$, Design an algorithm to output $A \cdot B$.

**Problem**

Multiplying two $n$-bit numbers: Given two $n$-bit numbers, $A$ and $B$, Design an algorithm to output $A \cdot B$.

- Solution 1: Use long multiplication.
- What is the running time of the algorithm that uses long multiplication?

**Problem**

Multiplying two $n$-bit numbers: Given two $n$-bit numbers, $A$ and $B$, Design an algorithm to output $A \cdot B$.

- Solution 1: Use long multiplication.
- What is the running time of the algorithm that uses long multiplication? $O(n^2)$
- Is there a faster algorithm?

**Problem**

Multiplying two $n$-bit numbers: Given two $n$-bit numbers, $A$ and $B$, Design an algorithm to output $A \cdot B$.

- Solution 1: Algorithm using long multiplication with running time $O(n^2)$.
- Solution 2: (Assume $n$ is a power of 2)
  - Write $A = A_L \cdot 2^{n/2} + A_R$ and $B = B_L \cdot 2^{n/2} + B_R$.
  - So, $A \cdot B = (A_L \cdot B_L) \cdot 2^n + (A_L \cdot B_R + A_R \cdot B_L) \cdot 2^{n/2} + (A_R \cdot B_R)$
  - Main Idea: Compute $(A_L \cdot B_L)$, $(A_R \cdot B_R)$, and $(A_L + B_L) \cdot (A_R + B_R) - (A_L \cdot B_L) - (A_R \cdot B_R)$.

## Problem

Multiplying two $n$-bit numbers: Given two $n$-bit numbers, $A$ and $B$, Design an algorithm to output $A \cdot B$.

## Algorithm

```
Karatsuba($A, B$)
    - If $(|A| = |B| = 1)$ return$(A \cdot B)$
    - Split $A$ into $A_L$ and $A_R$
    - Split $B$ into $B_L$ and $B_R$
    - $P \leftarrow$ Karatsuba$(A_L, B_L)$
    - $Q \leftarrow$ Karatsuba$(A_R, B_R)$
    - $R \leftarrow$ Karatsuba$(A_L + A_R, B_L + B_R)$
    - return$(2^n \cdot P + 2^{n/2} \cdot (R - P - Q) + Q)$
```

- What is the recurrence relation for the running time of the above algorithm?

# Divide and Conquer
## Master theorem

### Problem

Multiplying two $n$-bit numbers: Given two $n$-bit numbers, $A$ and $B$, Design an algorithm to output $A \cdot B$.

### Algorithm

```
Karatsuba($A, B$)
    - If $(|A| = |B| = 1)$ return$(A \cdot B)$
    - Split $A$ into $A_L$ and $A_R$
    - Split $B$ into $B_L$ and $B_R$
    - $P \leftarrow$ Karatsuba($A_L, B_L$)
    - $Q \leftarrow$ Karatsuba($A_R, B_R$)
    - $R \leftarrow$ Karatsuba($A_L + A_R, B_L + B_R$)
    - return$(2^n \cdot P + 2^{n/2} \cdot (R - P - Q) + Q)$
```

- Recurrence relation: $T(n) \leq 3 \cdot T(n/2) + cn$; $T(1) \leq c$.
- What is the solution of this recurrence relation from the Master Theorem?

## Divide and Conquer
### Master theorem

**Theorem**

Master Theorem: Let

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k \quad \text{and} \quad T(1) \leq c,$$

Then

$$T(n) = \begin{cases} O(n^k) & \text{if } a < b^k \\ O(n^k \cdot \log_b n) & \text{if } a = b^k \\ O\left(n^{\log a / \log b}\right) & \text{if } a > b^k \end{cases}$$

- Recurrence relation: $T(n) \leq 3 \cdot T(n/2) + cn$; $T(1) \leq c$.
- What is the solution to the above recurrence relation?

## Divide and Conquer
### Master theorem

**Theorem**

Master Theorem: Let

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k \quad \text{and} \quad T(1) \leq c,$$

Then

$$T(n) = \begin{cases} O(n^k) & \text{if } a < b^k \\ O(n^k \cdot \log_b n) & \text{if } a = b^k \\ O\left(n^{\log a / \log b}\right) & \text{if } a > b^k \end{cases}$$

- Recurrence relation: $T(n) \leq 3 \cdot T(n/2) + cn$; $T(1) \leq c$.
- What is the solution to the above recurrence relation?
  $T(n) \leq O(n^{\log_2 3})$

# Divide and Conquer
## Solving recurrence relation

- Consider the recurrence relation for the running time of the `MergeSort` algorithm:

$$T(n) \leq 2 \cdot T(n/2) + cn \text{ for all } n \geq 2 \text{ ; } T(2) \leq c$$

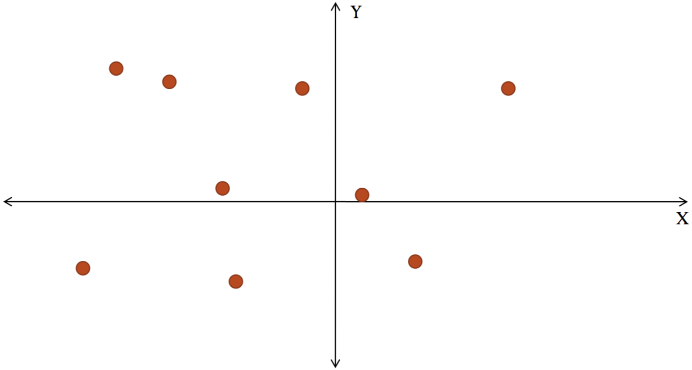
- Another way to solve the recurrence relation is *substitution*:
  1. Guess the bound on $T(n)$, and
  2. Show that this bound holds using induction.
- Let our guess be $T(n) \leq cn \log n$ for all $n \geq 2$. We will now prove this by induction
- Base case: $T(n) \leq cn \log n$ when $n = 2$ since we are given that $T(2) \leq c$.
- Inductive step: Suppose the bound holds for $n = 2, ..., k-1$, we will show that the bound also holds for $n = k$.
  - We know $T(k) \leq 2T(k/2) + ck$.
  - So, using induction hypothesis, we get: $T(k) \leq 2c(k/2)\log(k/2) + ck = ck \log k$.

# Divide and Conquer

## Closest pair of points on a plane

**Problem**

You are given $n$ points on a two dimensional plane. Each point $i$ is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.



## Problem

You are given $n$ points on a two dimensional plane. Each point $i$ is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Brute-force algorithm: Consider all pairs and pick closest.
    - Running time:

# Divide and Conquer
## Closest pair of points on a plane

### Problem

You are given $n$ points on a two dimensional plane. Each point $i$ is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Brute-force algorithm: Consider all pairs and pick closest.
    - Running time: $O(n^2)$

## Problem

You are given $n$ points on a two dimensional plane. Each point $i$ is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Divide and Conquer: *(Divide based on $X$-axis)*
    - Consider the *left-half* points $P_L$ and *right-half* points $P_R$.

## Divide and Conquer
### Closest pair of points on a plane

**Problem**

You are given $n$ points on a two dimensional plane. Each point $i$ is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Divide and Conquer: *(Divide based on $X$-axis)*
  - Consider the *left-half* points $P_L$ and *right-half* points $P_R$.
  - Recursively find the closest pair of points $(i_l, j_L)$ in $P_L$, and $(i_R, j_R)$ in $P_R$.

# Divide and Conquer
## Closest pair of points on a plane

**Problem**

You are given $n$ points on a two dimensional plane. Each point $i$ is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Divide and Conquer: *(Divide based on $X$-axis)*
  - Consider the *left-half* points $P_L$ and *right-half* points $P_R$.
  - Recursively find the closest pair of points $(i_l, j_L)$ in $P_L$, and $(i_R, j_R)$ in $P_R$.
  - Consider all pair of points $(p, q)$ such that $p$ belongs to $P_L$ and $q$ belongs to $P_R$.

# Divide and Conquer
## Closest pair of points on a plane

**Problem**

You are given $n$ points on a two dimensional plane. Each point $i$ is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Divide and Conquer: *(Divide based on $X$-axis)*
  - Consider the *left-half* points $P_L$ and *right-half* points $P_R$.
  - Recursively find the closest pair of points $(i_l, j_L)$ in $P_L$, and $(i_R, j_R)$ in $P_R$.
  - Consider all pair of points $(p, q)$ such that $p$ belongs to $P_L$ and $q$ belongs to $P_R$.
  - Pick the closest pair among $(i_L, j_L), (i_R, j_R)$, and $(p, q)$.

# Divide and Conquer
## Closest pair of points on a plane

**Problem**

You are given $n$ points on a two dimensional plane. Each point $i$ is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Divide and Conquer: *(Divide based on X-axis)*
  - Consider the *left-half* points $P_L$ and *right-half* points $P_R$.
  - Recursively find the closest pair of points $(i_l, j_L)$ in $P_L$, and $(i_R, j_R)$ in $P_R$.
  - Consider all pair of points $(p, q)$ such that $p$ belongs to $P_L$ and $q$ belongs to $P_R$.
  - Pick the closest pair among $(i_L, j_L), (i_R, j_R)$, and $(p, q)$.
- What is the running time of the above algorithm?

# Divide and Conquer
## Closest pair of points on a plane

- Divide and Conquer: *(Divide based on $X$-axis)*
  - Consider the *left-half* points $P_L$ and *right-half* points $P_R$.
  - Recursively find the closest pair of points $(i_l, j_L)$ in $P_L$, and $(i_R, j_R)$ in $P_R$.
  - Consider all pair of points $(p, q)$ such that $p$ belongs to $P_L$ and $q$ belongs to $P_R$.
  - Pick the closest pair among $(i_L, j_L), (i_R, j_R)$, and $(p, q)$.
- Let $x = x^*$ be a line along the $Y$-axis dividing the points into $P_L$ and $P_R$.
- Let $d$ be the distance between the closest pair of points in $P_L$ and $P_R$.
- Claim 1: For any pair of points $(p, q)$ such that $x(p) < x^* - d$ and $x(q) \geq x^*$, the distance between $p$ and $q$ is $\geq d$.
- Claim 2: For any pair of points $(p, q)$ such that $x(p) \leq x^*$ and $x(q) > x^* + d$, the distance between $p$ and $q$ is $\geq d$.

# Divide and Conquer

## Closest pair of points on a plane

- Divide and Conquer: *(Divide based on $X$-axis)*
  - Consider the *left-half* points $P_L$ and *right-half* points $P_R$.
  - Recursively find the closest pair of points $(i_l, j_L)$ in $P_L$, and $(i_R, j_R)$ in $P_R$.
  - Consider all pair of points $(p, q)$ such that $p$ belongs to $P_L$ and $q$ belongs to $P_R$.
  - Pick the closest pair among $(i_L, j_L), (i_R, j_R)$, and $(p, q)$.
- Let $x = x^*$ be a line along the $Y$-axis dividing the points into $P_L$ and $P_R$.
- Let $d$ be the distance between the closest pair of points in $P_L$ and $P_R$.
- <u>Claim 1</u>: For any pair of points $(p, q)$ such that $x(p) < x^* - d$ and $x(q) \geq x^*$, the distance between $p$ and $q$ is $\geq d$.
- <u>Claim 2</u>: For any pair of points $(p, q)$ such that $x(p) \leq x^*$ and $x(q) > x^* + d$, the distance between $p$ and $q$ is $\geq d$.
- This means that for pairs of points across the line $x = x^*$, we can throw any point in $P_L$ that has small $X$-coordinate and any point in $P_R$ that has large $X$-coordinate.
- Do these claims help in improving the running time?

## End