Divide-and-Conquer

1 Definitions

1. **Divide** up problem into several subproblems of the same kind.
2. **Conquer** (solve) each subproblem recursively.
3. **Combine** solutions to subproblems into overall solution.

2 Merge Sort

2.1 Overview

1. Divide array into two halves
2. Recursively sort each half (mergesort each half)
3. Merge two halves to make sorted whole.

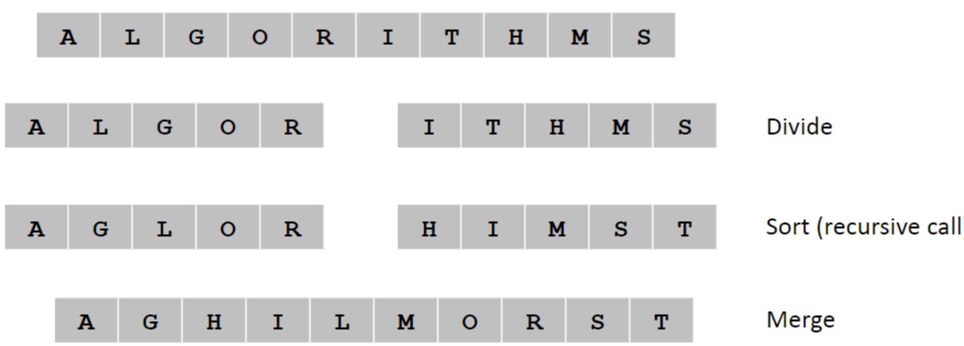

Figure 1: Merge Sort Overview

2.2 Merge Analysis

1. Scan two pre-sorted list left to right.
2. Keep track of smallest element in each sorted half.
3. Insert the smallest of two elements into auxiliary array.

4. Update the smallest element in each sorted half, repeat until done.

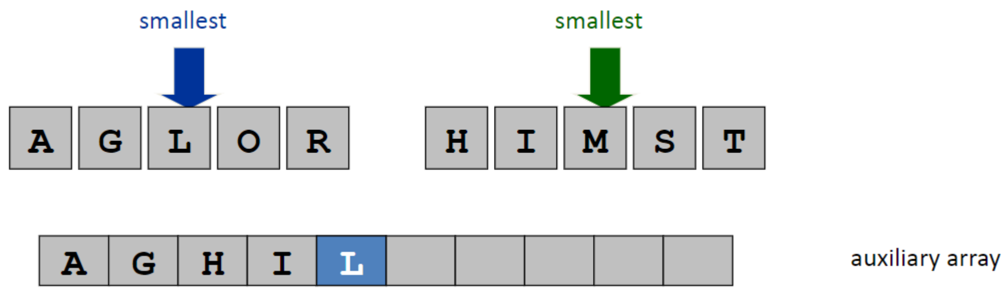

Figure 2: Merge Analysis

2.3 Recurrence

Define $T(n)$ as the max number of compares to merge sort a list of size $\leq n$. Then the merge sort recurrence could be expressed as:

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases} \tag{1}$$

2.3.1 n as a power of 2

The solution of this recurrence when n is a power of 2, from proposition we have:

$$T(n) = n \log_2 n \tag{2}$$

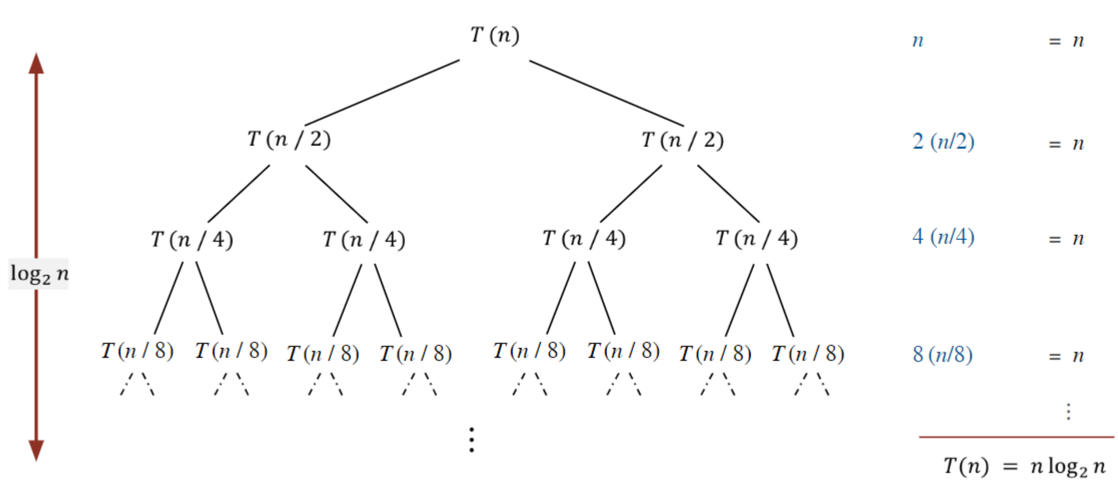

Figure 3: Recurrence, n as power of 2

If we increase n by 2, we get:

$$
\begin{aligned}
T(2n) &= 2T(n) + 2n \\
&= 2n\log_2 n + 2n \\
&= 2n(\log_2 n + \log_2 2 - 1) + 2n \\
&= 2b(\log_2 (2n) - 1) + 2n \\
&= 2n\log_2 (2n)
\end{aligned}
$$

2.3.2 n not as a power of 2

Claim: If $T(n)$ satisfies the following recurrence, then $T(n) <= n \lceil \log_2 n \rceil$.

$$
T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}
$$

Pf. (by induction on n)

- Base case: $n = 1$.
- Define $n_1 = \lfloor n/2 \rfloor$, $n_2 = \lceil n/2 \rceil$.
- Induction step: assume true for 1, 2, ... , $n-1$. Strong induction

$$
\begin{aligned}
T(n) &\leq T(n_1) + T(n_2) + n \\
T(n) &\leq n_1\lceil \log_2 n_1 \rceil + n_2\lceil \log_2 n_2 \rceil + n \quad \text{(by ind. hyp.)} \\
T(n) &\leq n_1\lceil \log_2 n_2 \rceil + n_2\lceil \log_2 n_2 \rceil + n \\
T(n) &\leq n\lceil \log_2 n_2 \rceil + n \\
T(n) &\leq n(\lceil \log_2 n \rceil - 1) + n \\
T(n) &\leq n\lceil \log_2 n \rceil \blacksquare
\end{aligned}
$$

$n_2 = \lceil n/2 \rceil$
$<= \lceil 2^{\lceil \log 2\, n \rceil} / 2 \rceil$
$= 2^{\lceil \log 2\, n \rceil} / 2$
$\log_2 n_2 <= \lceil \log_2 n \rceil - 1$

3 Closest Pair

3.1 Problem Definition

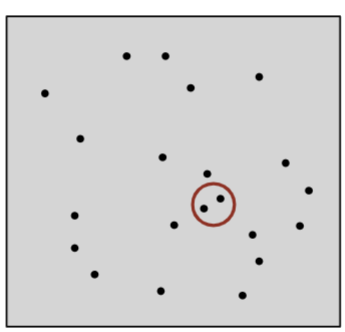

Figure 4: Closet Pair of Points Definition

Given n points in the plane, find a pair of points with the smallest Euclidean distance between them. **Euclidean distance** is a measure of the true straight line distance between two points in Euclidean space. For example, if we have two points (x_1, y_1) and (x_2, y_2), then the euclidean distance is calculated as:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3}$$

3.2 Divide-and-Conquer Algorithm

3.2.1 Divide

Draw vertical line L so that $n/2$ points on each side.

3.2.2 Conquer

Find closest pair in each side recursively.

3.2.3 Merge

Find closest pair with one point in each side. Then return the best of 3 solutions.

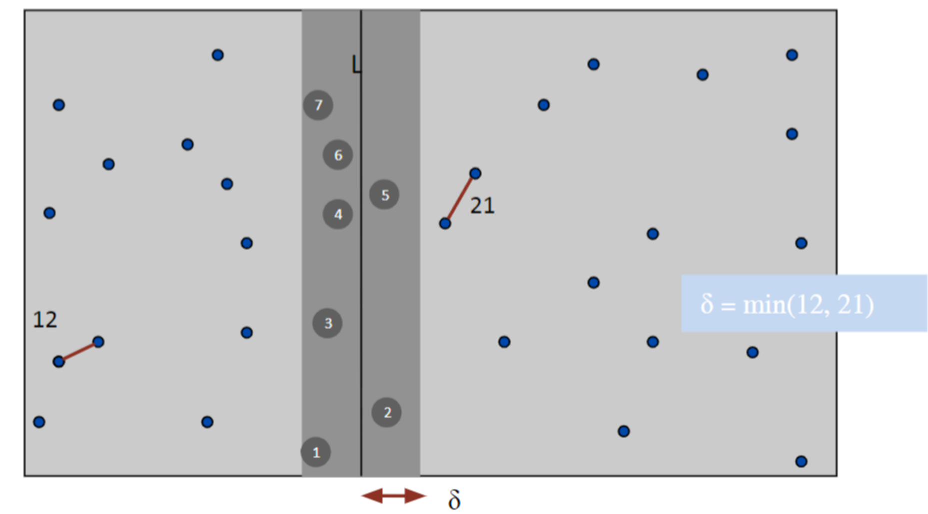

Figure 5: Merge Step

To find the closest pair with one point in each side, we assume that distance $\leq \delta$, where:

$$\delta = min(\text{left.min}, \text{right.min}) \tag{4}$$

Based on the observation, we only need to consider points within δ of line L. We sort points in 2δ strip by their y coordinates. We only check distances of those within 11 positions in sorted list, which could be proved below:

Def. Let s_i be the point in the 2δ-strip, with the i^{th} smallest y-coordinate.

Claim. If $|i - j| \ge 12$, then the distance between s_i and s_j is at least δ.

Pf.

- No two points lie in same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\ge 2(\frac{1}{2}\delta)$. ▪

The KT book used 16 instead of 12 boxes.

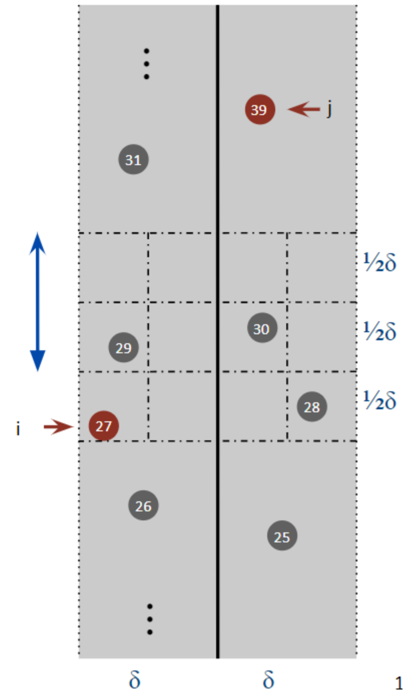

Figure 6: 11 neighbours proof

3.3 Implementation

3.3.1 $O(n \log^2 n)$

CLOSEST-PAIR $(p_1, p_2, \ldots, p_n)$

Compute separation line L such that half the points are on each side of the line. ← $O(n \log n)$

$\delta_1 \leftarrow$ CLOSEST-PAIR (points in left half).
$\delta_2 \leftarrow$ CLOSEST-PAIR (points in right half). ← $2\,T(n/2)$

$\delta \leftarrow \min\{\delta_1, \delta_2\}$.

Delete all points further than δ from line L. ← $O(n)$

Sort remaining points by y-coordinate. ← $O(n \log n)$

Scan points in y-order and compare distance between each point and next 11 neighbors. If any of these distances is less than δ, update δ. ← $O(n)$

RETURN δ.

Figure 7: $O(n \log^2 n)$ Algorithm

The recurrence solution could be expressed as:

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n \log n) & \text{otherwise} \end{cases} \tag{5}$$

For the recursive steps, we keep dividing n into 2 parts, so there will be $\log n$ levels. For each level, the merge will take runtime $O(n \log n)$. Therefore, the total runtime will be $O(n \log^2 n)$.

3.3.2 $O(n \log n)$

This algorithm could be improved if we don't sort points in strip from scratch each time. So each recursive returns two lists with all points sorted by x, y coordinates, and then we sort by merging two pre-sorted lists.

```
x_sorted_all <- sort points by x-axis
CLOSEST-PAIR (p_1, p_2, ..., p_n, x_sorted_all)
________________________________________________

Compute separation line L such that half the points                          <- O(n)
are on each side of the line, and obtain points on each side sorted by x

(δ_1 , y_sorted_1)← CLOSEST-PAIR (points in left half, x_sorted_left).
(δ_2 , y_sorted_2)← CLOSEST-PAIR (points in right half, x_sorted_right).     <- 2 T(n / 2)

δ ← min { δ_1 , δ_2 }.

Merge y_sorted_1 and y_sorted_2 to obtain a y_sorted list.                   <- O(n)

Delete all points further than δ from line L.                                <- O(n)

Scan points in y-order and compare distance between
each point and next 11 neighbors. If any of these
distances is less than δ, update δ.                                          <- O(n)

RETURN δ.
________________________________________________
```

Figure 8: $O(n \log n)$ Algorithm

The recurrence solution could be expressed as:

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n) & \text{otherwise} \end{cases} \tag{6}$$

4 Master Theorem

The recipe for solving common divide-and-conquer recurrences could be expressed as:

$$T(n) = aT(\frac{n}{b}) + f(n) \tag{7}$$

Where:

1. $T(0) = 0$ and $T(1) = \Theta(1)$
2. $a \geq 1$ is the number of subproblems, also called the branching factor.
3. $b \geq 2$ is the factor by which the subproblem size decreases.
4. $f(n) \geq 0$ is the work to divide and combine subproblems (merge).
5. a^i is the number of subproblems at level i.
6. $k = \log_b n$ levels
7. n/b^i is the size of subproblem at level i

If $f(n)$ is $\Theta(n^d)$, then we can use master method:

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases} \tag{8}$$

There are several conditions we can not use master theorem:

1. $f(n)$ is not a polynomial, for example, $f(n) = 2^n$
2. b can not be expressed as a constant, for example, $T(n) = aT(\sqrt{n}) + f(n)$