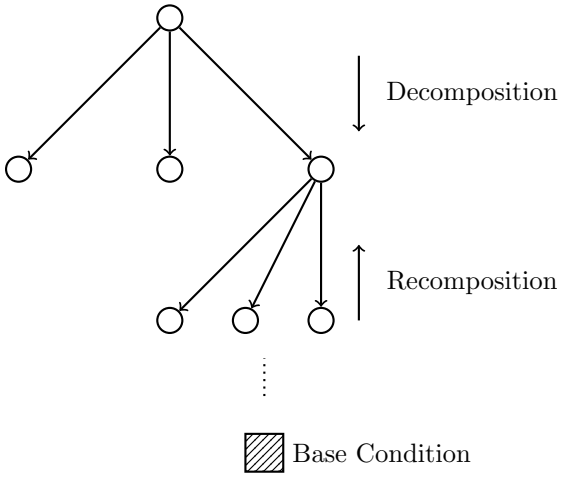
# FADML Scribe

Balaji M

20th January 2025

## 1 Divide and Conquer Paradigm - RECAP




In general, a divide-and-conquer algorithm has the following format.

(1) If the size of instance I is 'small', then solve the problem using a straightforward method and return the answer, also known as base conditions. Otherwise, continue to the next step.
(2) Divide the instance I into p sub-instances $I_1$, $I_2$,...,$I_p$ of approximately the same size. (Decomposition)
(3) Recursively call the algorithm on each sub-instance $I_j$, $1 \leq j \leq p.$, to obtain p partial solutions.
(4) Combine the results of the p partial solutions to obtain the solution to the original instance I. Return the solution of instance I. (Recomposition)

A good algorithm balances between the optimal split point, decomposition,and recomposition.

## 2 Median Finding Algorithm

Given an array A = A[1,...,n] of n numbers and an index i ($1 \leq i \leq n.$), find the median of A. If $n$ is odd, then setting $i = \frac{n+1}{2}$ gives the median of $A$. If $n$ is even, then the median will be average of the cases $i = \lceil \frac{n+1}{2} \rceil$ and $i = \lfloor \frac{n+1}{2} \rfloor$. Median shall be found by sorting the array and obtaining the median using the above formula. However, even the best sorting algorithm takes $\Theta(nlogn)$. time. However, with the help of a deterministic algorithm, this shall be found in $O(n)$ time.

The steps of 'Median of Medians' algorithm given below:

(1) Divide the n items into groups of 5.If 5 does not divide p, then discard the remaining elements.
(2) Find the median of each group of 5. Sort the five elements and $3^rd$element is the median of group. Let the set of medians be M.
(3) Use `median` recursively to find the median (call it $x$) of these $\lceil n/5 \rceil$ medians.

$$\text{mm} \leftarrow \text{median}(M, \lfloor n/2 \rfloor)$$

(4) Partitions the elements in A into three arrays: $A_1$, $A_2$, and $A_3$, which, respectively, contain those elements less than, equal to, and greater than mm. (5) This step decides where the median is present among the three arrays, depending on the outcome of this test.The algorithm either returns the median,

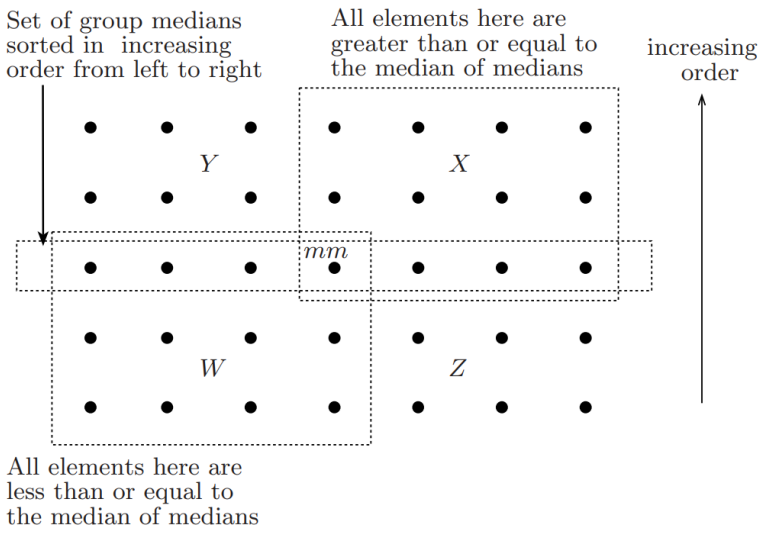



Figure 1: Nature of elements w.r.t Median of Medians .

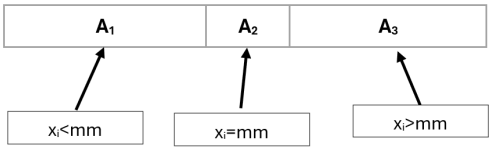



Figure 2: Splitting of array into three parts based on mm

or recurses on either $A_1$ or $A_3$.
The amount of data trimmed everytime is atleast

$$3 \cdot \lceil \lfloor n/5 \rfloor /2 \rceil = \frac{3}{2} \cdot \lfloor n/5 \rfloor \leq \frac{3}{2} \cdot \frac{n-4}{5}$$

Thus the new length of data called for next iteration is

$$n - \frac{3}{2} \cdot \frac{n-4}{5} = 0.7n + 1.2 \leq \frac{3}{4}n$$

The time complexity is given by

$$T(n) = O(n) + T\left(\lfloor n/5 \rfloor\right) + T(0.7n + 1.2)$$

It shall be inferred that the no. of elements in $A_1$ and $A_3$ would not exceed roughly 0.7*n, which is a constant factor.

$$T(n) \leq O(n) + T\left(\lfloor n/5 \rfloor\right) + T\left(\frac{3}{4}n\right)$$

This implies that T(n) is of the order $O(n)$. Thus, it is fair to say that

$$T(n) \leq k \cdot n$$

The above equation is of the form The recurrence relation is given by:

$$T(n) = T(c_1 n) + T(c_2 n) + bn$$

The solution to this recurrence equation is:

$$k \geq \frac{b}{1 - c_1 - c_2}$$

When $c_1$=0.75 and $c_2$=0.2,

$$k \geq 20b$$

**Pseudocode**

**Algorithm 1:** Median of Medians

**Input:** An array $A[1..n]$ of $n$ elements.
**Output:** The median element in $A$.

1. `median`($A$, $n/2$)

**Procedure** `median`($A$, $n/2$)

1. $n \leftarrow |A|$
2. **if** $n < 44$ **then** sort $A$ and **return** $(A[\lfloor n/2 \rfloor])$
3. Let $q = \lfloor n/5 \rfloor$. Divide $A$ into $q$ groups of 5 elements each. (If 5 does not divide $q$, then discard the remaining elements.)
4. Sort each of the $q$ groups individually and extract its median. Let the set of medians be $M$.
5. mm $\leftarrow$ `median`($M$, $\lfloor q/2 \rfloor$) {mm is the median of medians}
6. Partition $A$ into three arrays:
   $A_1 = \{\, a \mid a < \text{mm}\}$, $A_2 = \{\, a \mid a = \text{mm}\}$, $A_3 = \{\, a \mid a > \text{mm}\}$.
7. **case**
   $|A_1| \geq n/2$**: return** `median`($A_1$, $n/2$)
   $|A_1| + |A_2| \geq n/2$**: return** mm
   $|A_1| + |A_2| < n/2$**: return** `median`($A_3$, $n/2 - |A_1| - |A_2|$)
8. **end case**

**Why groups of 5?**
When the groups are of even size, it becomes necessary to add more row to compute the median as compared to their odd counterparts.
When n=3, the trimmed portion is

$$n - \frac{n-2}{3} = \frac{2n}{3} + \frac{2}{3}$$

This is not considered ideal due to the reason that $\frac{n}{3}$ elements are involved in sorting for median and $\frac{2n}{3}$ is involved in decomposition. Thus, there is no trimming of data here. For odd integers greater than 5, even they are asymptotically same as that of that 5, the constant value keeps increasing.
The sorting cost is given by

$$O\left(\frac{n}{k} \cdot k \log k\right)$$

When n=1000, k=5,7,9,11...., sorting cost is given by,

| **k** | $\mathbf{1000 \cdot \log_2(k)}$ |
|---|---|
| 5 | 2321.92 |
| 7 | 2807.35 |
| 9 | 3169.92 |
| 11 | 3459.43 |
| 13 | 3700.43 |
| 15 | 3906.89 |

Table 1: Approximate values of $1000 \cdot \log_2(k)$.

Thus, 5 is considered as the ideal split among others.

## 3 Closest Pair Algorithm

Let S be a set of n points in the plane. The goal is to find a pair of points p and q in S whose mutual distance is minimum. In other words, we want to find two points $p_1 = (x_1, y_2)$ and $p_2 = (x_2, y_2)$ in S with the property that the distance between them defined by $d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ is minimum among all pairs of points in S. Here $d(p_1, p_2)$ is referred to as the Euclidean distance between $p_1$ and $p_2$.

The steps of the algorithm are as follows:
(1) Sort the points in S by increasing x-coordinate, which is done only once throughout the algorithm.
(2) The point set $S$ is divided about a vertical line $L$ into two subsets $S_l$ and $S_r$ such that $|S_l| = \lfloor |S|/2 \rfloor$ and $|S_r| = \lceil |S|/2 \rceil$. The points to the left are part of $S_l$ and points to the right are part of $S_r$.
(3) Recursively, the minimum separations $\delta_l$ and $\delta_r$ of the two subsets $S_l$ and $S_r$, respectively, are computed. (4) For the combine step, the smallest separation $\delta$' between a point in $S_l$ and a point in $S_r$ is also computed. Finally, the desired solution is the minimum of $\delta_l$, $\delta_r$, and $\delta$'.

**Computation of $\delta$'**
The naïve method which computes the distance between each point in $S_l$ and each point in $S_r$ requires $\Omega(n^2)$ in the worst case. However, it shall be performed in better time complexity. The new algorithm proposes that rather comparing all the n points for each point. It's enough to compare only the 7 neighbouring points. The proof of the same is discussed below:

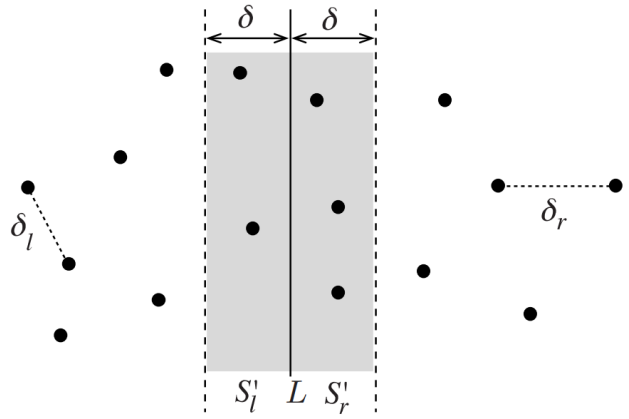



Figure 3: Points at distance of $\delta$ from vertical line L .

(1) Let $\delta = \min\{\delta_l, \delta_r\}$. If exists, the closest pair $p_l$ in $S_l$ and some point $p_r$ in $S_r$, they must be within distance $\delta$ of the dividing line $L$. Let $S'_l$ and $S'_r$ denote, respectively, the points in $S_l$ and $S_r$ within distance $\delta$ of $L$, then $p_l$ must be in $S'_l$ and $p_r$ must be in $S'_r$.
(2) Let there be $\delta'$ such that $\delta' \leq \delta$, then there is a pair of points where $d(p_l, p_r) = \delta$. This indicates that the maximum possible vertical distance between the pair is $\delta$.
(3) This indicates that the pair of point is bound to be within a rectangle of area $\delta$ x $2\delta'$

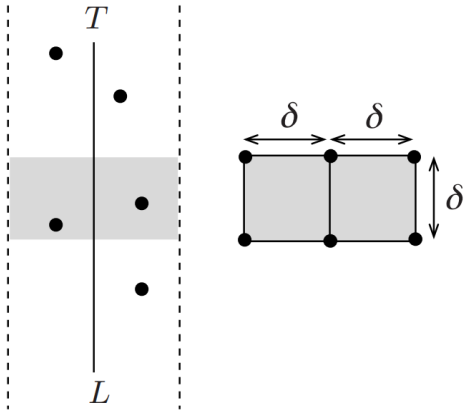



Figure 4: Pair of points with in the rectangle

(4) Let T be the set of points within the two vertical strips. By geometry, the maximum number of points with distance of $\delta$ from each other could be 8. (4 in $S_l$ and 4 in $S_r$)
(5) The maximum number is attained when one point from $S_l$ coincides with one point from $S_r$ at the intersection of $L$ with the top and bottom of the rectangle.
(6) Thus, each point in $T$ needs to be compared with at most seven points in $T$. Neighbours are found by sorting the points in $T$ by increasing y-coordinate.

**Time Complexity of Algorithm**

$$T(n) = \begin{cases} c, & \text{if } n \leq 3, \\ T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + \Theta(n), & \text{if } n > 3. \end{cases}$$

The $\Theta(n)$ is attained by the Mergesort algorithm, where sorting is every recursion is replaced by merging.

**Pseudocode**

The Pesudocode for this algorithm is attached in the next page.

**Algorithm 2:** CLOSESTPAIR

**Input:** A set $S$ of $n$ points in the plane.
**Output:** The minimum separation realized by two points in $S$.

1. Sort the points in $S$ in nondecreasing order of their x-coordinates.
2. $(\delta, Y) \leftarrow cp(1, n)$
3. **return** $\delta$

**Procedure** cp(low, high)

1. **if** $(\text{high} - \text{low} + 1) \leq 3$ **then**
2. compute $\delta$ by a straightforward method.
3. Let $Y$ contain the points in nondecreasing order of y-coordinates.
4. **else**
5. $\text{mid} \leftarrow \frac{\text{low}+\text{high}}{2}$
6. $x_0 \leftarrow x\big(S[\text{mid}]\big)$
7. $(\delta_l, Y_l) \leftarrow cp(\text{low}, \text{mid})$
8. $(\delta_r, Y_r) \leftarrow cp(\text{mid} + 1, \text{high})$
9. $\delta \leftarrow \min\{\delta_l, \delta_r\}$
10. $Y \leftarrow \text{Merge}(Y_l, Y_r)$ in nondecreasing order of y-coordinates.
11. $k \leftarrow 0$
12. **for** $i \leftarrow 1$ to $|Y|$ {Extract $T$ from $Y$}
13. **if** $\big|x(Y[i]) - x_0\big| \leq \delta$ **then**
14. $k \leftarrow k + 1$
15. $T[k] \leftarrow Y[i]$
16. **end if**
17. **end for** {$k$ is the size of $T$}
18. $\delta' \leftarrow 2\,\delta$ {Initialize $\delta'$ to any number greater than $\delta$}
19. **for** $i \leftarrow 1$ to $(k - 1)$ {Compute $\delta'$}
20. **for** $j \leftarrow (i + 1)$ to $\min\{i + 7, k\}$
21. **if** $d(T[i], T[j]) < \delta'$ **then** $\delta' \leftarrow d(T[i], T[j])$
22. **end for**
23. **end for**
24. $\delta \leftarrow \min\{\delta, \delta'\}$
25. **end if**
26. **return** $(\delta, Y)$

# 4 Convex Hull Algorithm

Given a set $S$ of $n$ points in the plane, find CH($S$), the convex hull of $S$.

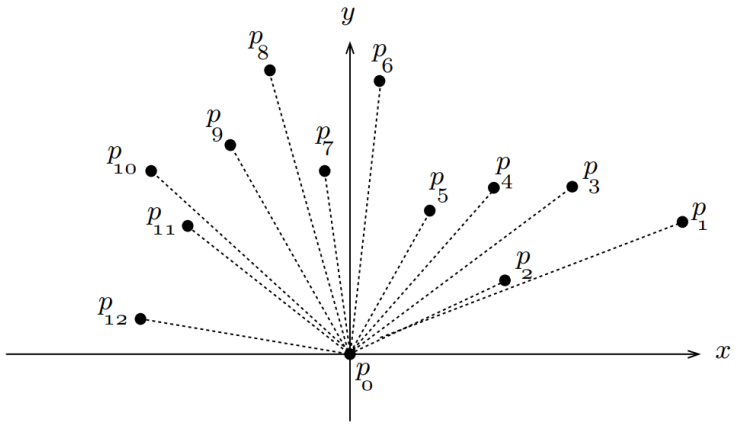



Figure 5: A set of point S for which convex hull is calculated

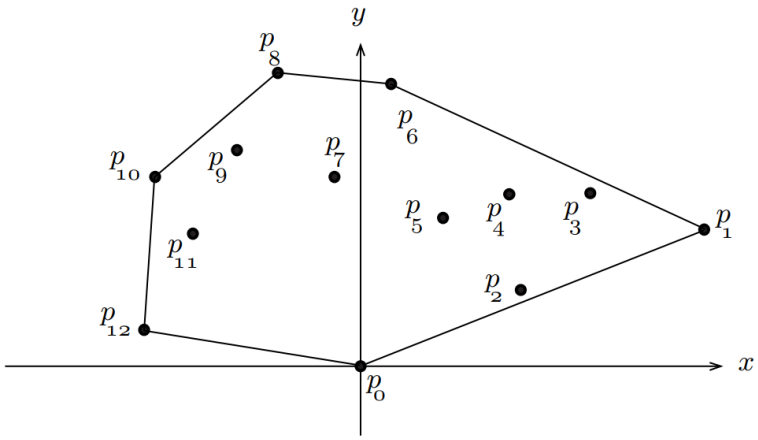



Figure 6: Final convex hull

The Graham Scan algorithm for Convex Hull is given by:

(1) The point with the minimum $y$-coordinate is found, call it $p_0$.
(2) If there are two or more points with the minimum $y$-coordinate, $p_0$ is chosen as the rightmost one.
(3) The coordinates of all points are transformed such that $p_0$ is at the origin. The points in $S \setminus \{p_0\}$ are then sorted by polar angle about the origin $p_0$.
(4) If two points $p_i$ and $p_j$ form the same angle with $p_0$, then the one that is closer to $p_0$ precedes the other in the ordering. Let the sorted list be $T = \{p_1, p_2, \ldots, p_{n-1}\}$, where $p_1$ and $p_{n-1}$ form the least and greatest angles with $p_0$, respectively.
(5) Let there be a scan with respect to event point based on the order of points in the sorted list $T$, and the sweep line status being implemented using a stack $St$. At any moment, let the stack content be $St = (p_{n-1}, p_0, \ldots, p_i, p_j)$ (i.e., $p_i$ and $p_j$ are the most recently pushed points), and let $p_k$ be the next point to be considered.

(6) If the triplet $p_i, p_j, p_k$ forms a left turn, then $p_k$ is pushed on top of the stack and the sweep line is moved to the next point. If $p_i, p_j, p_k$ form a right turn or are collinear, then $p_j$ is popped off the stack and the sweep line is kept at point $p_k$.

**Time Complexity**
The sorting step costs $O(n \log n)$ time. In the while loop, each point is pushed exactly once and is popped at most once. Moreover, checking whether three points form a left turn or a right turn amounts to computing their signed area in $\Theta(1)$ time. Thus, the cost of the while loop is $\Theta(n)$. Due to this, the algorithm has a time complexity of $O(n \log n)$.

**Pseudocode**

**Algorithm 3:** CONVEXHULL - GRAHAM SCAN

**Input:** A set $S$ of $n$ points in the plane.
**Output:** $CH(S)$, the convex hull of $S$ stored in a stack $St$.

1. Let $p_0$ be the rightmost point with minimum $y$-coordinate.
2. $T[0] \leftarrow p_0$
3. Let $T[1..n-1]$ be the points in $S \setminus \{p_0\}$ sorted in increasing polar angle about $p_0$. If two points $p_i$ and $p_j$ form the same angle with $p_0$, then the one that is closer to $p_0$ precedes the other in the ordering.
4. `push`$(St, T[n-1])$; `push`$(St, T[0])$
5. $k \leftarrow 1$
6. **while** $k < n - 1$
7. Let $St = (T[n-1], \ldots, T[i], T[j])$, where $T[j]$ is on top of the stack.
8. **if** $T[i], T[j], T[k]$ is a left turn **then**
9. `push`$(St, T[k])$
10. $k \leftarrow k + 1$
11. **else** `pop`$(St)$
12. **end if**
13. **end while**

# 5 Reductions

Reductions shall be used to establish lower bounds for algorithms of a given problem by comparing the complexity of classes of computational problems. Such comparisons will be made by describing transformations from one problem to another. A transformation is simply a function that maps instances of one problem to instances of another problem.

Let $A$ be a problem whose lower bound is known to be $\Omega(f(n))$, where $n = o(f(n))$, e.g., $f(n) = n \log n$. Let $B$ be a problem for which we wish to establish a lower bound of $\Omega(f(n))$. We establish this lower bound for problem $B$ as follows:

1. Convert the input to $A$ into a suitable input to problem $B$.
2. Solve problem $B$.
3. Convert the output into a correct solution to problem $A$.

In order to achieve a linear time reduction, Steps 1 and 3 above must be performed in time $O(n)$. In this case, we say that the problem $A$ has been reduced to the problem $B$ in linear time, and we denote this by writing $A \propto_n B$.