ALGORITHM DESIGN USING DIVIDE & CONQUER METHOD: IV

Partha P Chakrabarti

Indian Institute of Technology Kharagpur

Overview of Algorithm Design

1. Initial Solution
   a. Recursive Definition – A set of Solutions
   b. Inductive Proof of Correctness
   c. Analysis Using Recurrence Relations
2. Exploration of Possibilities
   a. Decomposition or Unfolding of the Recursion Tree
   b. Examination of Structures formed
   c. Re-composition Properties
3. Choice of Solution & Complexity Analysis
   a. Balancing the Split, Choosing Paths
   b. Identical Sub-problems
4. Data Structures & Complexity Analysis
   a. Remembering Past Computation for Future
   b. Space Complexity
5. Final Algorithm & Complexity Analysis
   a. Traversal of the Recursion Tree
   b. Pruning
6. Implementation
   a. Available Memory, Time, Quality of Solution, etc

1. Core Methods
   a. Divide and Conquer
   b. Greedy Algorithms
   c. Dynamic Programming
   d. Branch-and-Bound
   e. Analysis using Recurrences
   f. Advanced Data Structuring
2. Important Problems to be addressed
   a. Sorting and Searching
   b. Strings and Patterns
   c. Trees and Graphs
   d. Combinatorial Optimization
3. Complexity & Advanced Topics
   a. Time and Space Complexity
   b. Lower Bounds
   c. Polynomial Time, NP-Hard
   d. Parallelizability, Randomization

Master Theorem

Let $a \ge 1$ and $b > 1$ be constants, let $f(n)$ be a function, and

Let $T(n)$ be defined on nonnegative integers by the recurrence

$\boxed{T(n) = aT(n/b) + f(n)}$, where we can replace n/b by $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

$T(n)$ can be bounded asymptotically in three cases:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $\boxed{T(n) = \Theta(n^{\log_b a})}$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$,

   and if, for some constant $c < 1$ and all sufficiently large n,

   we have $a \cdot f(n/b) \le c\, f(n)$, then $T(n) = \Theta(f(n))$.

Multiplication of Two n-bit Numbers

$x = 101001$ n-bit

$y = 101010$ n-bit

$$
\begin{array}{r}
\cancel{101001\cdot 0} \\
1010010\cdot 1 \\
\cancel{10100100\cdot 0} \\
101001000\cdot 1 \\
\cancel{1010010000\cdot 0} \\
10100100000\cdot 1 \\
\hline
11010111010
\end{array}
$$

Result

$O(n^2)$

x	x_1	x_2
y	y_1	y_2
	$\leftarrow \frac{n}{2} \rightarrow$	$\leftarrow \frac{n}{2} \rightarrow$

$x = x_1 * 2^{n/2} + x_2$

$y = y_1 * 2^{n/2} + y_2$

$x \cdot y = 2^n \underbrace{x_1 y_1}_{1} + 2^{n/2}(\underbrace{x_1 y_2}_{2} + \underbrace{x_2 y_1}_{3}) + \underbrace{x_2 y_2}_{4}$

$T(n) = 4T(n/2) + O(n)$

$O(n^2)$

$A = x_1 y_2 + x_2 y_1$

$= \underbrace{(x_1 + x_2)(y_1 + y_2)}_{4} - x_1 y_1 - x_2 y_2$

$xy = 2^n \underbrace{x_1 y_1}_{1} + 2^{n/2} \cdot A + \underbrace{x_2 y_2}_{2}$

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

$= O(n^{\log_2 3}) = O(n^{1.59})$

Strassen's Algorithm for Matrix Multiplication

Let $A = n \times n$ matrix, $B: n \times n$

$$\left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \times \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$= \left[\begin{array}{c|c} ae \pm bg & af \pm bh \\ \hline ce \pm dg & cf \pm dh \end{array}\right]$$

$$T(n) = 8T\left(\frac{n}{2}\right) + O(n^2)$$

$$= O(n^3)$$

$P_1 = a(f-h)$, $P_2 = (a+b)h$

$P_3 = (c+d)e$, $P_4 = d(g-e)$

$P_5 = (a+d)(e+h)$ $P_6 = (b-d)(g+h)$

$P_7 = (a-c)(e+f)$

$$\left[\begin{array}{c|c} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ \hline P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{array}\right]$$

$$T(n) = \boxed{7}\,T(n/\boxed{2}) + O(n^2)$$

$$O(n^{\log_2 7}) = O(n^{2.81})$$

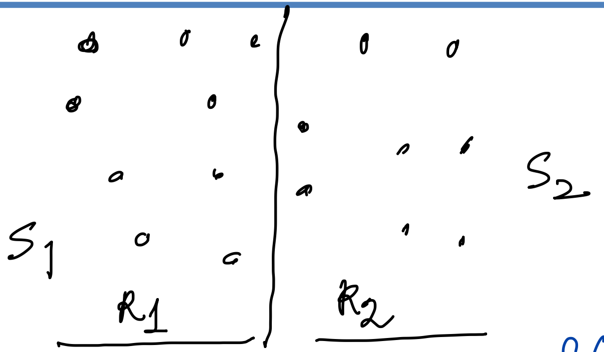
Closest Pair of Points

Given a set of n points in a 2-D plane, find the closest pair of points. [General version is in some d-D]

Straightforward Method : $O(n^2)$

```
closestpair (S)
{ Let S = {<x1,y1>, <x2,y2>, ..., <xn,yn>}
  || split S into 2 disjoint non-empty        O(1)
     subsets S1 & S2                          O(n)
  R1 = closestpair (S1)
  R2 = closestpair (S2)
  Let R = min (R1, R2)                        f(n)
  R3 = combine (S1, S2, R)              <-    O(n)
  return (R3)
}
```

equal

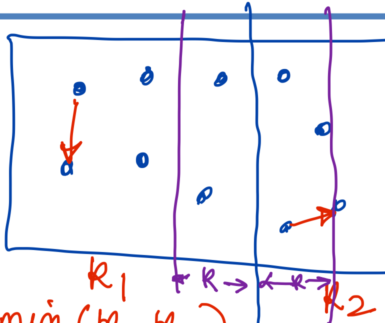


$$T(n) = T(n_1) + T(n_2) + f(n)$$
$$= 2T(n/2) + f(n) \leftarrow$$

1. Divide into 2 equal parts
2. Divide randomly [check for av. case]

→ Median along x-axis
- → sorting $O(n \log n)$
- → Median Finding $O(n)$

Closest Pair of Points: Strip Combine



$k = \min(k_1, k_2)$

Examine only those points along this $2k$ strip on the boundary.

$Z = \text{strip}(S_1, k) \cup \text{strip}(S_2, k)$

Sort(Z) by the y-axis

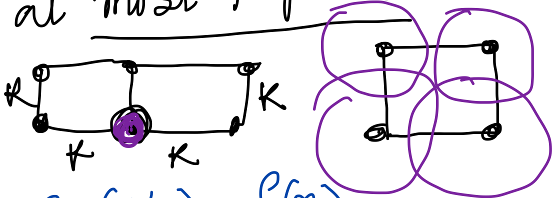
$Z = \{q_1, q_2, \dots, q_r\}$

For each q_i we need to check which points are there within k, distance and if so find the min

Does this comparison require every point in Z to be compared with many or $O(n)$ other points in Z?

$\boxed{\text{NO}}$

Theorem: For each q_i we need to check at most 7 points



$T(n) = 2T(n/2) + f(n)$

case 1: $f(n) = O(n \log n)$

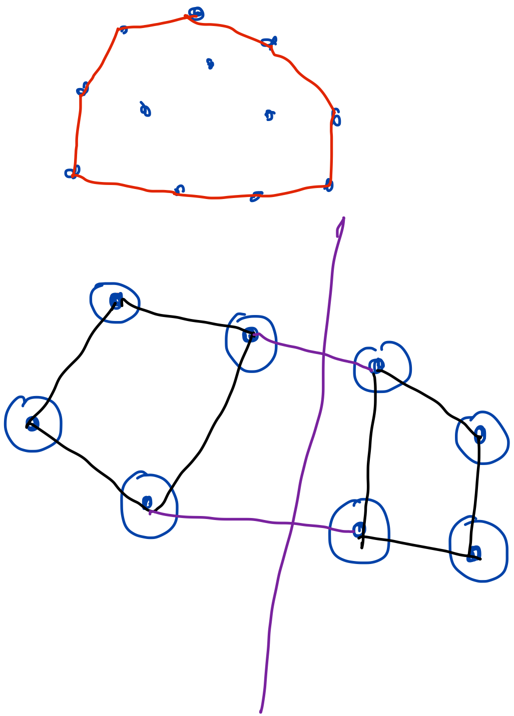
$= O(n \log n)$

case 2: Median finding in $O(n)$

X-axis & Y-axis sorting is done once globally : $O(n)$ $\longrightarrow f(n) = O(n)$

Higher or d-dimensions $O(n \log^{d-1} n)$

Finding the Convex Hull



convexHull(S)
{
split S into S_1, S_2 (O(n)) median

H_1 = convexHull(S_1)
H_2 = convexHull(S_2)
H_3 = combine(H_1, H_2) $\rightarrow O(n)$
return(H_3)
}

$$\boxed{T(n) = 2T(n/2) + O(n)}$$

$O(n \log n)$

Summary

Balancing the split

↳ Decomposition → $f_1(n)$

Recursion → a, b

Recomposition → $f_2(n)$

$$T(n) = a\,T(n/b) + f_1(n) + f_2(n)$$

optimizing the ratio of a/b and the f_1 & f_2 or $\max(f_1(n), f_2(n))$

Classical Problems

MEDIAN FINDING in $O(n)$ TIME

↳ which we shall discuss in a subsequent class.

Thank you

Any Questions?