7 Divide-and-Conquer

Thursday, October 26, 2023 12:03 PM

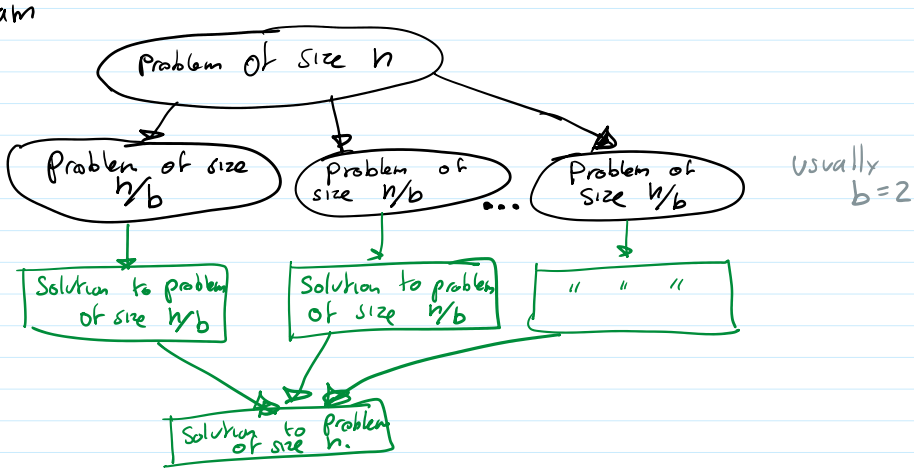
The basic schema:

1.- The problem is divided into several subproblems.
   - Ideally of the same size
   - Usually 2

2.- The subproblems are solved.
   - usually recursively

3.- The solutions to the subproblems are combined to get a solution to original problem.

Diagram



usually $b=2$

NOTE: In this class we will concentrate on single processor solution.-

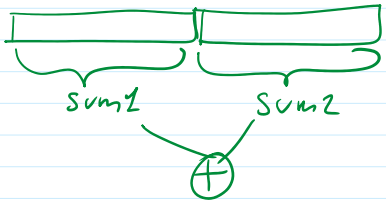
NOTE: Some of the most important algorithms in computer Science are of this kind.

Applying Divide-and-conquer is good, but not always gives us a better solution.

- Algorithm 1.- Sum an Array.

sum(A[0..n-1])

divide by 2 → A[0.. $\lfloor \frac{n}{2} \rfloor$ -1]
A[$\lfloor \frac{n}{2} \rfloor$ .. n-1]



$$+ \begin{matrix} \text{Sum}\left(A[0 \ldots \lfloor \frac{n}{2} \rfloor - 1\right) \\ \text{Sum}\left(A[\lfloor \frac{n}{2} \rfloor \ldots n-1]\right) \end{matrix}$$

Base case. $\text{Sum}(A[a..a]) = A[a]$;

- Analysis framework:

- input of size n
- divided into b instances of size $\frac{n}{b}$
- a of these instances need to be solved.
- the cost of reconvination is $f(n)$
- lets assume n is a power of b

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

- "General Divide-and-conquer recurrance"

- THE MASTER THEOREM.

previous results concerning General divide and conquer recurrance.

★ the solution to a recurrance $T(n) = a \cdot T(n/b) + f(n)$ is:

If $f(n) \in \Theta(n^d)$ where $d \geq 0$ then

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

- Analysis #1. divide-and-conquer sum

basic operation sum $+$

$$A(1) = 0$$
$$A(n) = 2 \cdot A\left(\frac{n}{2}\right) + 1$$

Appying M.T.

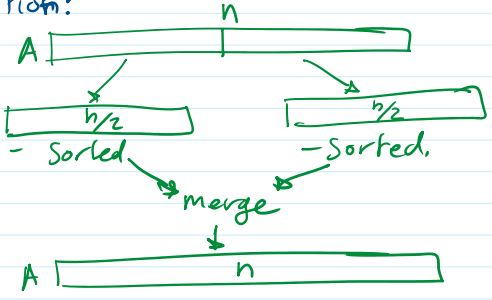
$d = 0$
$a = 2$
$b = 2$

as $a > b^d$ $\because 2 > 2^0$ then

$$A(n) \in \Theta\left(n^{\log_b a}\right) = n^{\log_2 2} = n$$
$$A(n) \in \Theta(n)$$

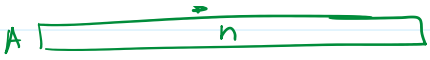
- Algorithm #2 : Merge-Sort

Intuition:



base-case $n=1$

$n = 13$



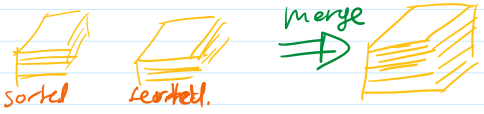

```
FUNCTION MergeSort( A[0...n-1] )
  IF n>1 THEN
     Copy A[0... ⌊n/2⌋-1] to B[0.. ⌊n/2⌋-1]
     Copy A[⌊n/2⌋... n-1] to C[0.. ⌈n/2⌉-1]
     MergeSort(B)
     MergeSort(C)
     Merge(B, C, A)
```

n = 13
A[0 ... 12]

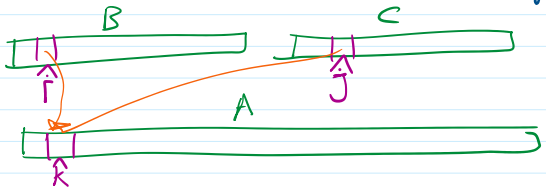
B[0...5] C[0...5]
6 6

Copy A[0..5] to B[0...5]
Copy A[6...12] to C[0...6]
7 7

How to Merge?



1.- compare elements at top of piles. move smaller one to new pile




```
FUNCTION Merge (B[0...p-1], C[0...q-1], A[0... p+q-1])
  i←0; j←0; k←0;
  WHILE i<p and j<q DO
     IF B[i] ≤ C[j] THEN
        A[k]←B[i]; i←i+1
     ELSE
        A[k]←C[j]; j←j+1
     k←k+1
  IF i=p THEN
     Copy C[j..q-1] to A[k... p+q-1]
  ELSE
     Copy B[i..p-1] to A[k.. p+q-1]
```

MergeSort is cool
- ⊕ Stable.
- ⊖ needs extra storage (linear)

Analysis:

basic operation < comparison

$$C(n) = 2 \cdot C(\tfrac{n}{2}) + T_{merge}(n)$$

$T^{worst}_{merge}(n) = n-1$

$C_{worst}(n) = 2 \cdot C_{worst}(n/2) + n-1$

Appy M.S.

$b = 2$
$a = 2$
$d = 1$ $\quad f(n) = n-1 \in \Theta(n^d)$

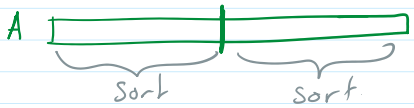
then $a = b^d \quad \ldots \quad 2 = 2^1$

Then $C_{worst}(n) \in \Theta(n^d \cdot \log n) = n \cdot \log n$

- Algorithm #2 : Quicksort.

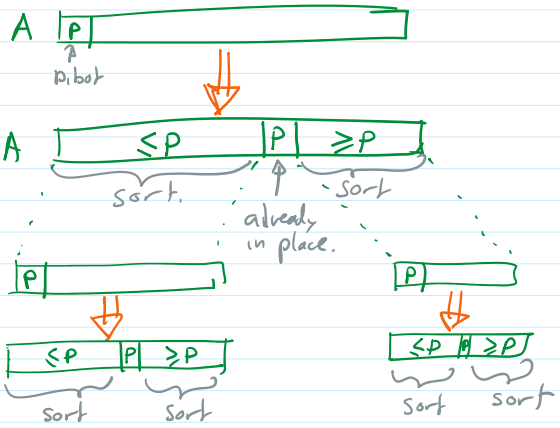
"Tony"
C.A.R. Hoare. '60

- Russian to english translator.

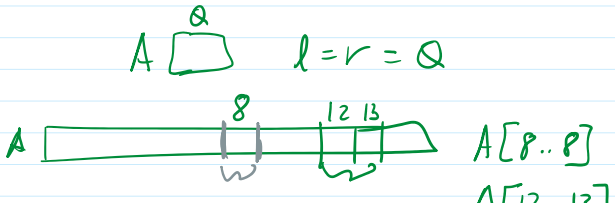


He did not wanted to merge.

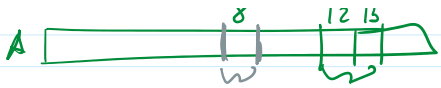
The solution is Partition.



ALGORITHM QuickSort(A[l...r])
IF l < r THEN
  s ← partition(A[l...r])
  Quicksort(A[l..s-1])

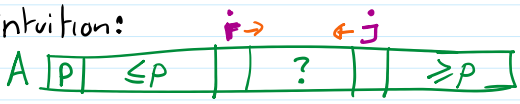


$s \leftarrow$ partition $(A[l..r])$
Quicksort $(A[l..s-1])$
Quicksort $(A[s+1..r])$



= A new way of partitioning (Hoare Partition)

Intuition:



if $A[i] \leq p$ then $i \leftarrow i+1$.
if $A[j] \geq p$ then $j \leftarrow j-1$.

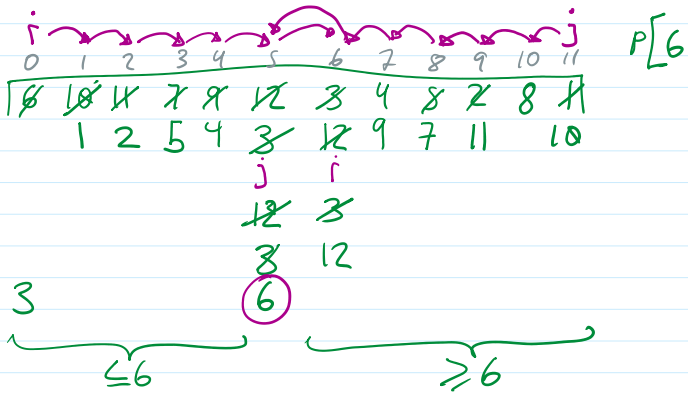
eventually: $A[i] > p$ AND $A[j] < p$
Swap $A[i], A[j]$

when do we stop? when $i = j$ $j < i$ $j \leq i$

ALGORITHM: Hoare Partitioning $(A[l..r])$

```
p←A[l]
i←l; j←r
REPEAT
    REPEAT i←i+1 UNTIL A[i]≥p
    REPEAT j←j-1 UNTIL A[j]≤p
    Swap (A[i], A[j])
UNTIL j≤i
Swap (A[i], A[j]) //undo last swap.
Swap (A[l], A[j])
return j
```

Trace



Analysis.

basic operation key comparisons.

Notice. partitioning makes n or $n+1$ comparisons.

$$C_{best}(n) = 2 \cdot C_{best}(n/2) + n$$

($C_{best}(n/2)$: sort each subarray; n: partitioning)

Apply M.S.
$a = 2$
$b = 2$
$f(n) = n \in \Theta(n^1)$
$d = 1$

so $a = b^d$

Then: $C_{best}(n) \in \Theta(n \cdot \log n)$

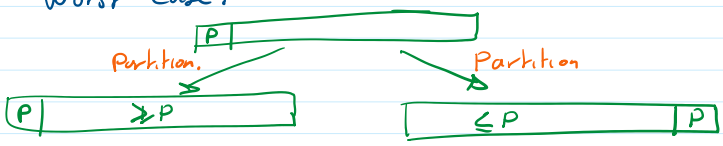
$b = c$

$f(n) = n \in \Theta(n^1)$

$d = 1$

Then: $C_{best}(n) \in \Theta(n \cdot \log n)$

Worst-case:



$$C_{worst}(n) = (n+1) + n + n-1 + n-2 \dots 3 = \frac{(n+1)(n+2)}{2} \in \Theta(n^2).$$

Scenarios:
- Quicksort on already sorted array
- Quicksort an array sorted in decreasing order.

- True power can only be assesed $C_{Avg}(n)$

$C_{avg}(n) \approx 1.39\, n \log n \in \Theta(n \log n)$

- Selection of the pivot is crucial.
  - randomize selection,
  - pick 3, and keep the middle one.

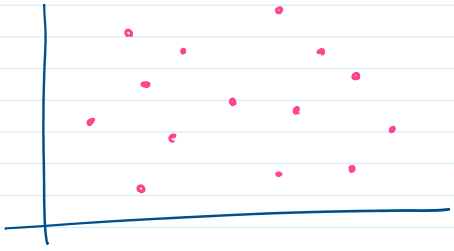
- Other improvements
  - Switch to insertion sort for small subarrays $n \leq 5 \dots 15$
  - 3 way partitioning (using 2 pivots)

20% to 30% speedups.

⊕ in-place no need for extra memory
⊖ Not_stable

- ALGORITHM #3 Closest-pair.

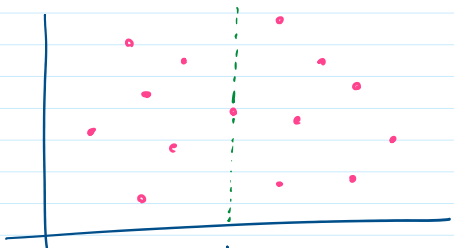
Problem



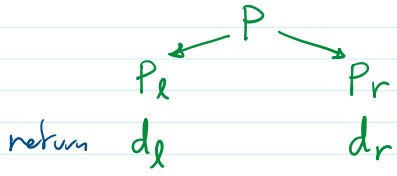
Find the closest two points.

Lets apply divide-and-conquer:

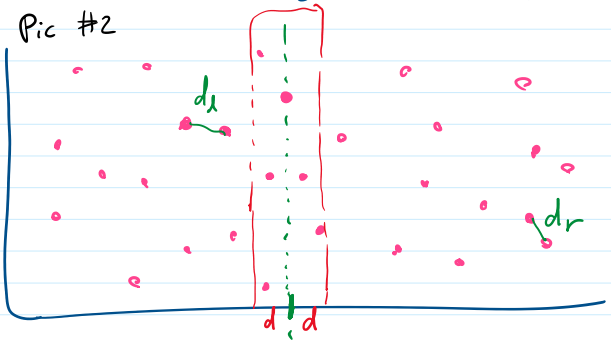
Intuition.:



1) Split points by median.

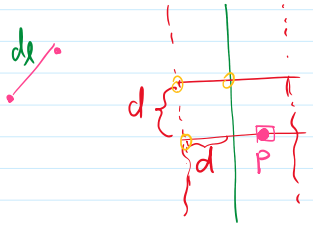


2) How to combine sub-solutions to build complete solution.



Let d = smallest of d_ℓ, d_r.

- the only way for a smallest pair to exists is if extremes are in each side of the partition.
- we can even narrow further:



at most 8 points to compare to p

ALGORITHM

Eff Closest Pair. (P: collection of points sorted by x value
Q: collection of points sorted by y value)

copy first $\lceil n/2 \rceil$ points of P to P_ℓ
copy same $\lceil n/2 \rceil$ points from Q to Q_ℓ
copy remaining $\lfloor n/2 \rfloor$ points of P to P_r
copy remainy $\lfloor n/2 \rfloor$ points from Q to Q_r
$d_\ell \leftarrow$ EffClosestPair(P_ℓ, Q_ℓ)
$d_r \leftarrow$ Eff Closest Pair (P_r, Q_r)
merging solutions
$d \leftarrow \min(d_\ell, d_r)$
$m \leftarrow P[\lceil n/2 \rceil - 1].x$

Base case
$|P| = 2$
$|P| = 3$

should be here
return distance
return smallest distance
by brute force.

$d \leftarrow \min(dl, dr)$

$m \leftarrow P[\lceil n/2 \rceil - 1].x$

Copy from Q all points for which $|x - m| < d$ into $S[0 \ldots ns-1]$

$dmin \leftarrow d^2$

FOR $i \leftarrow 0$ TO $ns-2$ DO

  $k \leftarrow i+1$

  WHILE $k < ns$ and $(S[k].y - S[i].y)^2 < dmin$ DO

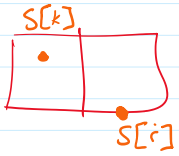
    $d' = (S[k].x - S[i].x)^2 + (S[k].y - S[i].y)^2$

    IF $d' < dmin$ THEN

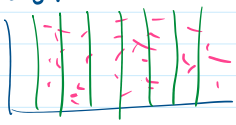
      $dmin \leftarrow d'$

    $k \leftarrow k+1$

RETURN $\sqrt{dmin}$



Analysis:



$T(n) = 2 \cdot T(n/2) + f(n)$ ← cost of searching the middle strip.

- $f(n) \in \Theta(n)$

Apply Master Theorem

$a = 2$
$b = 2$
$d = 1$

$T(n) \in \Theta(n \log n)$

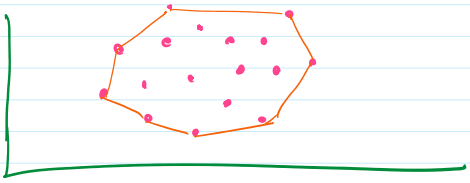
(assuming points are nicely split)

Worst-Case Scenario: 😟

Presorting is not an issue:
- Sorting and Eff closest pair have the same order of growth.

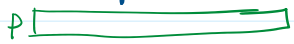
ALGORITHM #4:

Convex Hull.

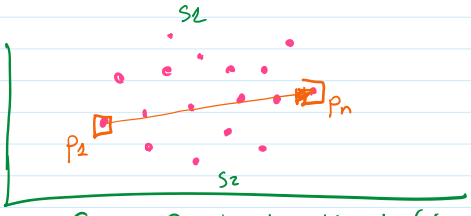


idea: Sort the points by x-value.

idea: Sort the points by x-value.

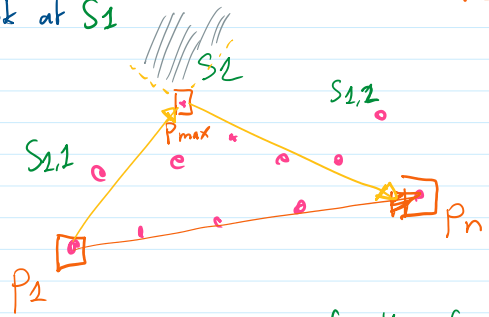


Gain 2 points!!
the extremes of P are points in the Convex hull.



S_1 = Points to the left of $\overrightarrow{p_1 p_n}$
S_2 = points to the right of $\overrightarrow{p_1 p_n}$

Let us look at S_1

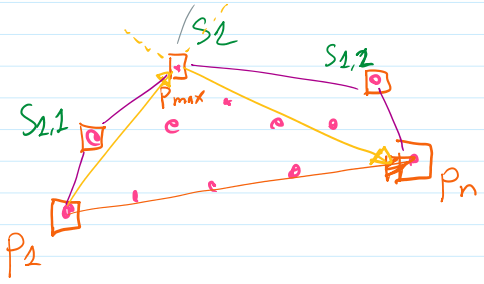


Identify p_{max} : point further from the line $\overrightarrow{p_1 p_n}$.

Consider $\overrightarrow{p_1 p_{max}}$ and $\overrightarrow{p_{max} p_n}$:

- p_{max} is in the convex hull
- points inside triangle are not in the convex hull
- there is no points to the left of both $\overrightarrow{p_1 p_{max}}$ and $\overrightarrow{p_{max} p_n}$

Then repeat same process for $S_{1,1}$, $S_{1,2}$.



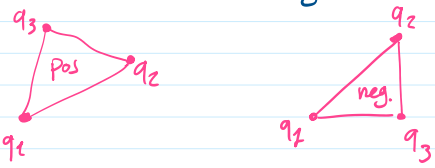
Repeat similar process for lower Hull.

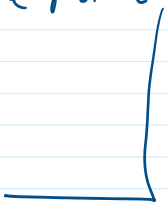
Base Cases. $|S| = 0$ previous line in the hull
$|S| = 1$ that point is in the hull.

- How to compare 3 points:
  Given q_1, q_2, q_3 points.
  We can obtain information on the triangle $\triangle q_1 q_2 q_3$ by looking at the determinant of the matrix

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1 \cdot y_2 + x_3 \cdot y_1 + x_2 \cdot y_3 - x_3 \cdot y_2 - x_2 \cdot y_1 - x_1 \cdot y_3$$

- the determinant is twice the area of the triangle.
- if q_3 is to the left of $q_1 q_2$ this area will be positive otherwise it will be negative.



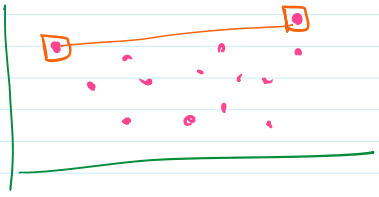


Analysis of QuickHull:

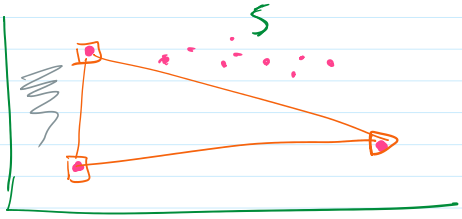
best:

is $\Theta(n \log n)$

worst:



is $\Theta(n^2)$



— EOF —