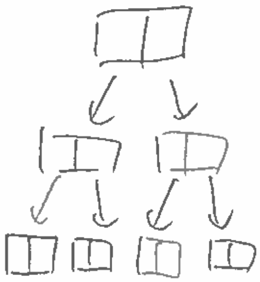
What is "Divide and conquer" means? Week2

Algorithm to solve problem in following way.



① Divide the problem set
② Conquer by recurrsion
③ Combine solutions of subproblem and get the final solution.

e.g. Merge sort
Karatsuba Multiplication
Inverse counting
Strassen's algorithm

Inverse Counting

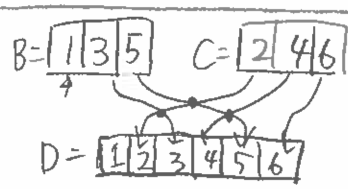
A = [1, 2, 3, 4, 5] ⟶ Two inverse
B = [1, 4, 2, 3, 5]

Easiest is bruteforcing it. But $O(n^2)$.

```
for i in range(len(B)):
    for j in range(i+1, len(B)):
        if B(i) > B[j]:
            count_inv += 1
```

Cleaver way is to use merge sort.

e.g.



⇒ During the merge step, we usually counting a number of inversions.

⇒ Every time I copied number from C, it's mean there is inversion.
In example in left, "2" skipped 3 and 5 (Two inversion) and "4" skipped 5 (One inversion)
Thus, there are three inversion in total.

Strassen's algorithm

what is the problem of matrixes multiplication?

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix}$$

i.e.

$$\underbrace{Z_{ij} = \sum_{k=0}^{n} X_{ik} \cdot Y_{kj}}_{\Theta(n)} \Rightarrow \text{Three nested for loop} \Rightarrow O(n^3)$$

$\Rightarrow$ How we can get close to $O(n^2)$?

How about dividing the matrix in to four submatrices?

$$Z = \underbrace{\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)}_{X} \cdot \underbrace{\left(\begin{array}{c|c} E & F \\ \hline G & H \end{array}\right)}_{Y} = \begin{pmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{pmatrix}$$

Divided and calculate as same as $X \cdot Y$, but have to do 8 times for each recurrsion.
$\Rightarrow$ End up in $O(n^3)$
$\Rightarrow$ Oh no! Same as straight forward approach. $\Rightarrow$ Problem

How we can reduce the recursive call from 8 to lower?

$\Rightarrow$ Strassen's Algorithm only has to call 7.
And makes the complexity from cubic to sub-cubic.
$O(n^{2.8})$

Closest pair

$A = [\ a(x,y), b(x,y), c(x,y) \cdots n(x,y)]$

⇒ Find the closest pair (Euclidean distance)

Brute force

⇒ Check all possible pair by two nested for loops.

1D version

① Sort points $O(n \cdot \log n)$
② Go through the points linearly and keep the shortest distance. $O(n)$

$O(n \cdot \log n)$ as whole. Better than straight forward $O(n^2)$.

2D version

why not sort by both the x-values and the y-values.

① Make copies of points sorted
by x-coordinate (P_x) and
by y-coordinate (P_y) } $O(n \cdot \log n)$

② Use second Divid and conqur algorithm

(2-1) Make it half
Q = left half of P
R = right half of P
form Q_x, Q_y, R_x, R_y (Sorted version of Q and R)

(2-2) $(p_1, q_1) = \text{Closest pair}(Q_x, Q_y)$

(2-3) $(p_2, q_2) = \text{Closest pair}(R_x, R_y)$

(2-4) $\delta = \min(d(p_1, q_1), d(p_2, q_2))$

(2-5) $(p_3, q_3) = \text{ClosestSplitPair}(P_x, P_y, \delta)$

(2-6) Return best of $(p_1, q_1)(p_2, q_2)(p_3, q_3)$

Want to be
① $O(n)$ time
② Correct whenever splitpair is smaller than $(p_1, q_1)(p_2, q_2)$

How we can solve ClosestSplitPair by $O(n)$?

⇒ Deriving is Difficult (>w<) Just use the ready made code.

Master method

Method to evaluate the speed of recurrsion expressed as follow.

$$T(n) \le a \cdot T\left(\frac{n}{b}\right) + O(n^d)$$

Recurrsion — Conquer

T(n) will be upper bounded by one of following three.

Number of recurrsive call for each iteration

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d \quad (\text{Case } 1) \\ O(n^d) & \text{if } a < b^d \quad (\text{Case } 2) \\ O(n^{\log_b a}) & \text{if } a > b^d \quad (\text{Case } 3) \end{cases}$$

e.g. Merge sort

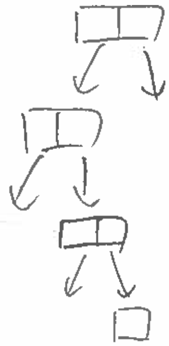
- Always subproblem increase by factor of 2
  $a=2$
- Always size of subproblem is half of original size n.
  $b=2$
- Conquerring part (Merge) is a linear time subroutine $O(n)$
  $d=1$

⇒ $a=2$
$b^d = 2^1 = 2$

$a = b^d$

Therefore, running time is
$O(n^d \log n) = O(n \cdot \log(n))$

e.g. Binary tree



- Only track the subset which has keyword
  $a = 1$
- Always subproblem is half the input.
  $b = 2$
- Conqurring Part is a const. operation to check the middle value is bigger or less than the value we are searching for.
  $d = 0$

$a = 1$
$b^d = 2^0 = 1$

$a = b^d$ (Case 1)

$O(n^d \cdot \log n) \Rightarrow O(\log n)$

e.g. Karatsuba

Version 1 w/o gauss's trick

$a = 4$: Four recurrsive for each iteration
$b = 2$: Divided in half for each iteration
$d = 1$: Adding and substructing is linear

$a = 4$
$b^d = 2$

$a > b^d$ (Case 3)

$O(n^{\log_b a}) = O(n^{\log_2 4})$
$= O(n^2)$

Version 2 with gauss's trick

$a = 3$
$b = 2$
$d = 1$

$a = 3$
$b^d = 2$

$a > b^d$ (case 3)

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

e.g. Strassen's algorithm

$a = 7$: Only 7 multiplication will be called
$b = 2$: n will be half
$d = 2$: Number of elements in matrix is still governed by quadratic number.

$7 > 2^2$ (Case 3) $O(n^{\log_2 7}) \fallingdotseq O(n^{2.81})$

e.g. Case 2 example

$T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + O(n^2)$

2 recurrsive — half the size — Out side recurrsion use quadratic.

$a = 2$
$b = 2$ $\quad a < b^d$ (Case 2)
$d = 2$

$O(T(n)) = O(n^d) = O(n^2)$