15/01/24
Monday

Insertion Sort $\quad \Theta(n^2) = cn^2$

$$2 \times \frac{10^7 \times 10^7}{10^{10}} \frac{\text{data}}{\text{ins/sec}} = 2 \times 10^4 \text{ sec} \quad (5.5 \text{ hrs.})$$

Merge Sort $\quad \Theta(n \log n) = c \cdot n \cdot \log n$

$$\frac{50 \times 10^7 \times \log(10^7)}{10^7 \text{ ins/sec}} = 50 \times 7 \text{ sec} \quad (20 \text{ mins})$$

1000x slower computer, still faster.

Constraint

1. Speed (main concern)
2. Memory
3. Quality – 1) ML
   2) Data Driven Algorithm

* Synthetically Standardize.

* Save memory, batch process

Assignment

Hand Gesture

Gesture x Orientation ⟶ Different Meaning.

Holistic Measure

Mahan ..... / Manhattan distance.

DDW

22/01/24
Monday

$\log n \quad \sqrt{n} \quad n \quad n\log n \quad n^2 \quad n^3 \quad \underline{2^n \quad n!}$
avoid

* n^c - polynomial
* 2^n - exponent
* $n!$ - all possible combination, bruteforce

"Bruteforce is not an algorithm"

#not realistically solvable.

Insertion Sort

In place sorting

Every iteration → array is sorted. (loop invariant)

Line 1 $\underbrace{\underbrace{2 \to n}_{n-1} \quad \underbrace{n+1}_{\text{stop check}}}_{n}$ itr + boundary cond.

$\sum_{j=2}^{n} t_j$

↑ because of A[i] > key based on j

* $t_j = 1$

Best case $\Theta(n)$ Already sorted. $(t_j = 1)$

Worst case $\Theta(n^2)$ Reverse sorted. $t_j = j$

Avg case:

$t_j = j/2$

Chapter 3

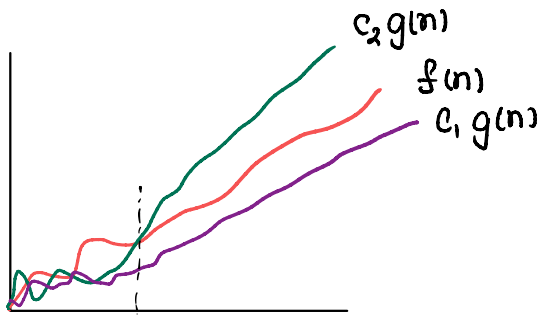
Average Case

$f(n) = \Theta(g(n))$

$c_1 g(n) < f(n) < c_2 g(n)$

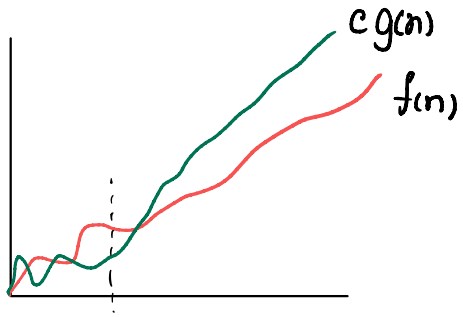
Upper & Lower Bound.

Tight bound



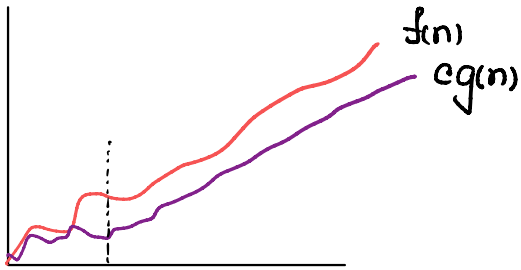
$f(n) = O(g(n))$

Upper Bound



$f(n) = \Omega(g(n))$

Lower Bound.



$O \rightarrow$ upper bound

but not tight

$2n^2 = O(n^2)$ tight

$2n = O(n^2)$ not tight

$n^2/2 = \omega(n)$

$n^2/2 \neq \omega(n^2)$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ $\quad g(n) \gg f(n)$ small o

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ $\quad f(n) \gg g(n)$ ω

Chapter 4

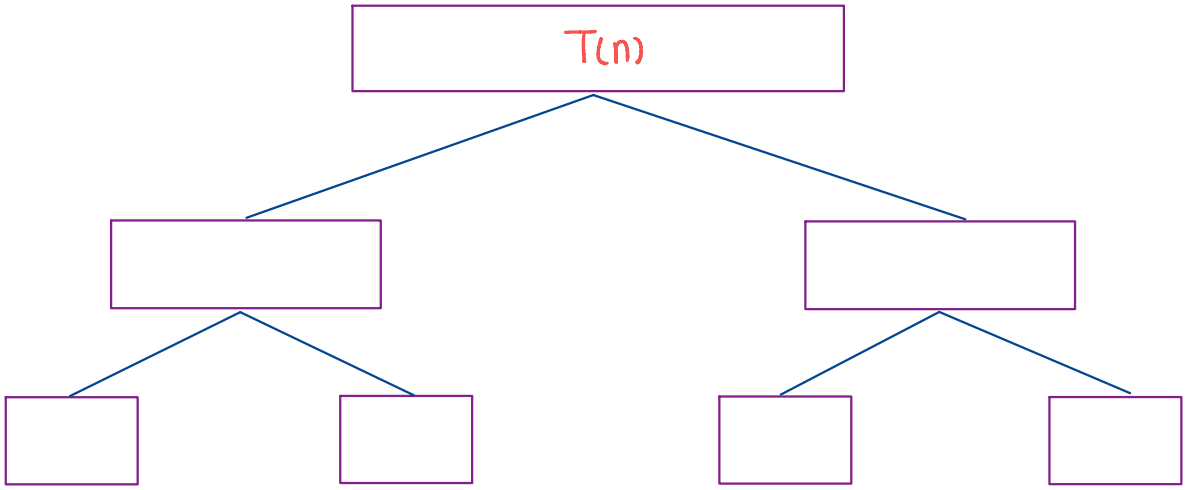
Divide & Conquer

Divide Cost = D(n) D(1) (half)

Conquer Cost = C(n) {Merging cost} Θ(n)

Calculation Cost = f(n) {fixed cost}

$$T(n) = D(n) + C(n) + f(n)$$



Recursion: Solving a problem with same function but different dataset.

Example: Binary Search.

* Sorting two sorted arrays take n iterations.

$$T(n) = 2T(n/2) + \Theta(n) + \Theta(1)$$

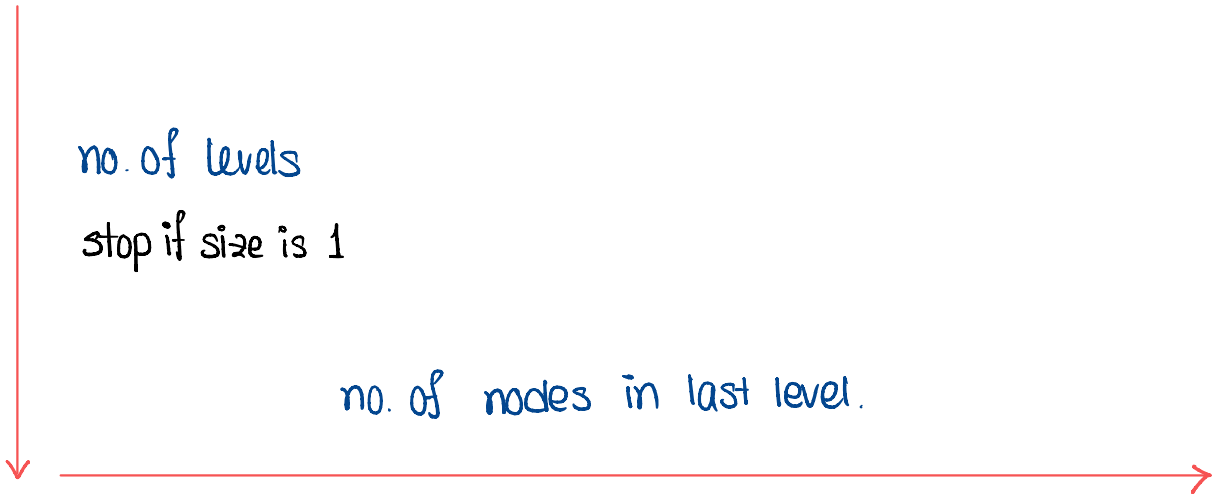
D C f

f(n)

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

Master's Method :

$T(n) = 3T(\frac{n}{4}) + n^2$



$\Theta(n^{\log_4 3})$

$= \Theta(3^{\log_4 n})$

$\log_{(\text{division})} n$

Upper bound $O(n^2)$

div ↑ depth ↓

No. of problems in last level · factor ^{i} (3^i)

"See things you didn't see"

$T(n) = 3\ T(n/4) + n^2$

$a \qquad b$

$T(n) = T(n/3) + T(\frac{2n}{3}) + \Theta(n)$

$\frac{2}{3}$ longer than $\frac{1}{3}$

depth using $b. = \frac{3}{2}$

$\log_{3/2} b$ $\qquad\qquad$ $O(n \log n)$

Merge sort

$\log_2 n \qquad b = 2$

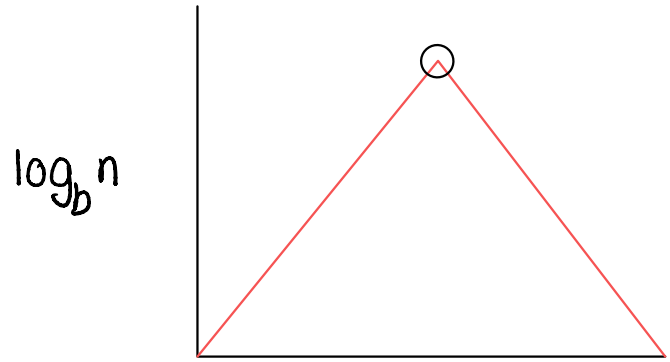
Last depth 2^i

$= 2^{\log_2 n}$

$= n$

Every Recursion can be represented by

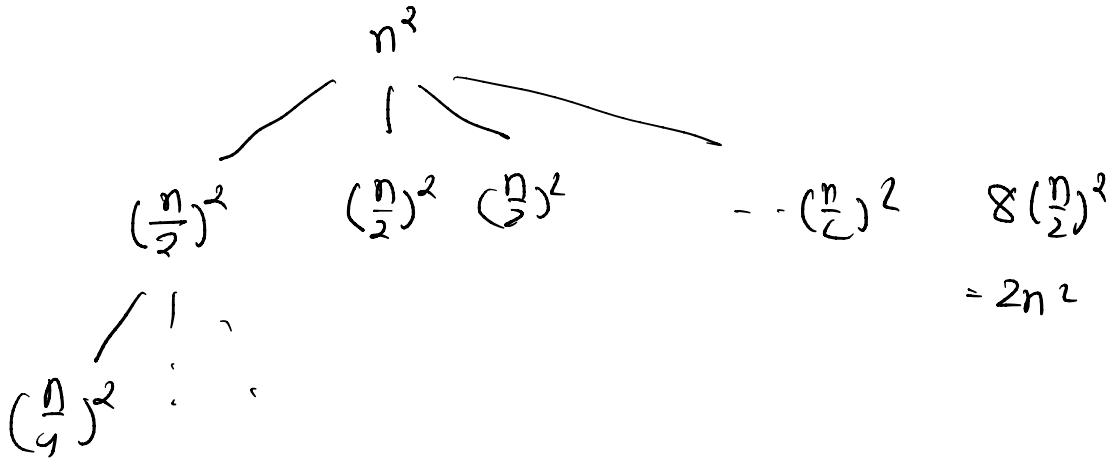
$$T(n) = a \cdot T(n/2) + f(n)$$



$$a^{\log_b n} = n^{\log_b a}$$

* Bottom level cost $\gg$ Top level cost

$$T(n) = 8T(n/2) + n^2$$



(increasing)

$\therefore n^2 + 2n^2 + 4n^2 + 16n^2 + \ldots$ last level cost will dominate

① $f(n) = O(n^{\log_b a - \varepsilon})$

→ cost of bottom level

↑ cost of top level

← Subtracting ε is still bigger polynomially bigger.

Bottom level determines cost

② $f(n) = \Theta(n^{\log_b a})$ 1st clause of masters method

$$T(n) = \Theta(f(n) \cdot \log_2 n)$$

③ $f(n) = \Omega(n^{\log_b a + \varepsilon})$

$$T(n) = \Theta(f(n))$$

* Cost determined by top
* Cost has to monotonically decrease.

$$a f(\frac{n}{b}) < c f(n)$$

↑ cost of the layer immediate below top. (2nd)

Top cost

* decreasing cost of level

Assignment : Sheet. 15 Marks

Explaining these.

29/01/24
Monday

Divide & Conquer

Binary Search

"Any multiplication can't be reduced"

$$\begin{array}{r} 1\ 2\ 3\ 4 \\ \times\ 5\ 6\ 7\ 8 \\ \hline \end{array}$$

n^2 multiplication (nm)

4 X 4 Multiplications

- Claimed by Russian Mathematician.
- Proved wrong by Japanese student Karabatsu just after 15 days.

$a_L a_R$ A n digits

$b_L b_R$ B m digits

$10^n a_L + a_R$

$$\begin{array}{l} a_L a_R \\ b_L b_R \\ \hline a_R b_R + a_L b_R + a_R b_L + a_L b_L \end{array}$$

still not reduced. same property

$$a_L b_L + a_R b_R + \underbrace{(a_L + a_R)}_{P}\underbrace{(b_L + b_R)}_{Q} - a_L b_L - a_R b_R$$

sum

$$\boxed{a_L b_L} + \boxed{a_R b_R} + \boxed{PQ} - a_L b_L - a_R b_R$$

We need 3 Multiplications.

$T(n) = 3T(n/2) + \Theta(n)$ | Half size, Three multiplication.

↑ Addition

$O(n^{\log_2 3}) = O(n^{1.58})$

better than $O(n^2)$

Strassen's Algorithm

7 Eq^n } Self Study

$T(n) = 7T(n/2) + n^2$

Karabatsu → Overhead of shift (10^n), addition

↳ Not ideal for multiplication of fewer digits

↳ Slower in terms of speed $\approx O(n^2)$

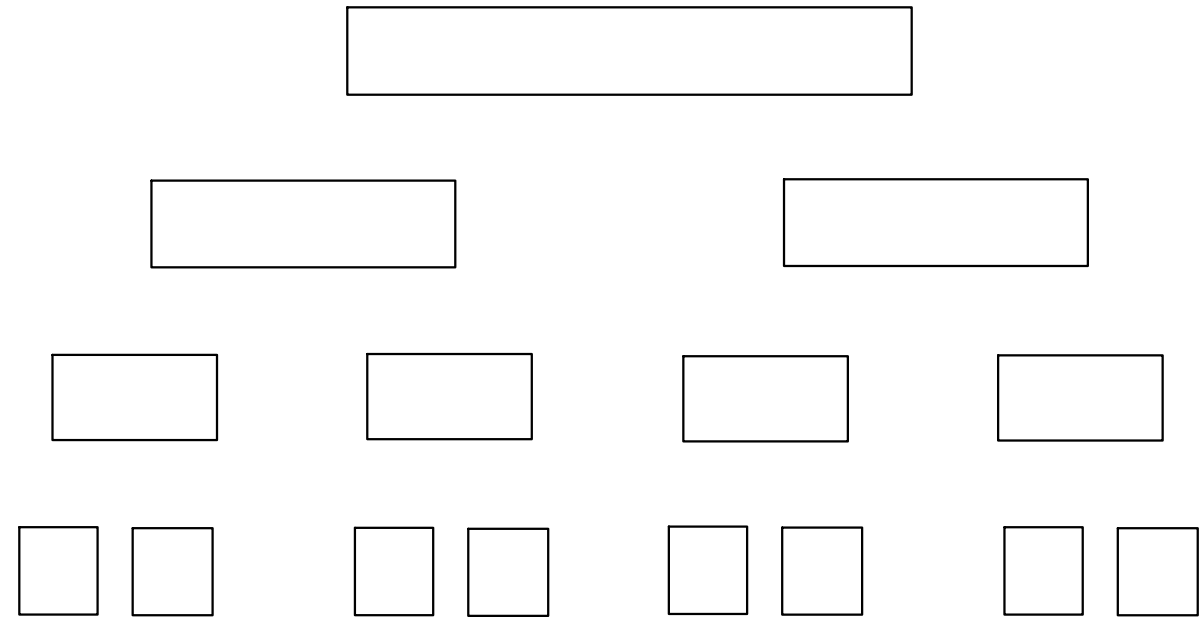
Digits need to be multiplication of 2/ Power of 2.

Zero padding required.

Just theoretical approach → Introduction of divide & conquer

"Karabatsu"

Merge Sort



1 element

Sorting task not reduced. (Again sort 2,4,8,16 elements)

Save time → this time sub-arrays are sorted.

Merge two sorted arrays - $\Theta(n)$

Merge (A,B) - Proof of exam.

Two markers, compare and add lower.

```
i=0, j=0, k=0
if   A[i] < B[j],
        Arr[k] = A[i]
        k++, i++
else  Arr[k] = B[j]
        k++, j++
```

Compare a single element.

Copy remaining

"There's no magic"

- K-H on basics of an optical mouse.

Quick Sort

Partion and calling

Only division, no merging / conquer

```
Quicksort (A,p,r)
if p<r
 q= Partition (A,p,r)
 Quicksort (A,p,q-1)
 Quicksort (A,q+1,r)
```

p - start index

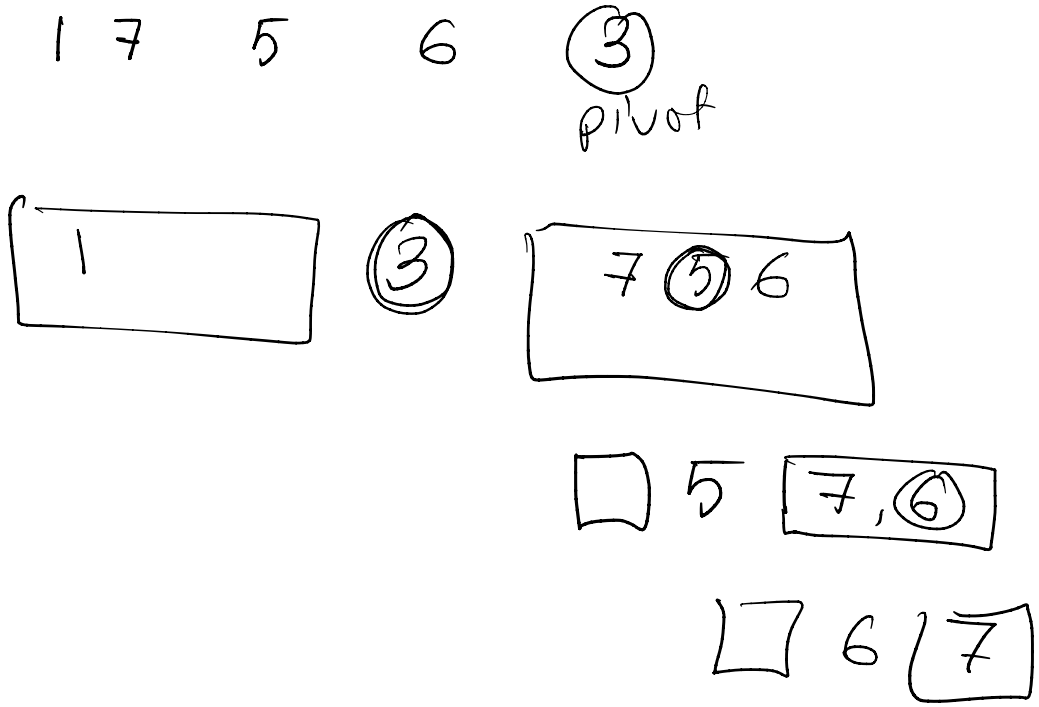
q - last index

} q dropped.

↳ Thats why merging is not necessary

q is in the right position.

pivot → with whom I compare everybody



No merging

Division is not cheap.

Compare with pivot.

$$T(n) = T(q-1) + T(n-q) + D(n) + C(n)$$

$$q-1+n-q = n-1$$

Divide Cost = Partition Cost

Conquer Cost = 0 (None)

Quicksort

$$T(n) = T(q-1) + T(n-q) + \Theta(n)$$

Clusturing → Optimized divide and conquer

Expectation Maximization

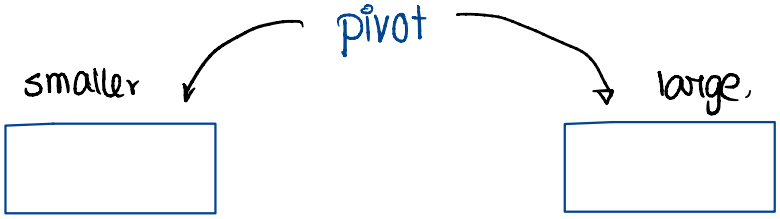
2 divide & conquer in your research domain & Dynamic Programming

2 Algo in area of research or Dynamic Programming Recent / Most used Tools.

Assignment : Due 6th week.

Max sub-array → DNC or greedy → better (sln variation)

Sum of max sub-array



Worst case already sorted.	$O(n^2)$	≈ Insertion
Best case, middle partition	$O(n \log n)$	≈ Merge
Average case, 1 vs all	$O(n \log n)$	

Avg Case : $T(n) = T(\frac{n}{10}) + T(\frac{9n}{10}) + \Theta(n)$

finish quickly ↑ longer tree.

$= T(\frac{n}{10}) + T(\frac{n}{10/9}) + \Theta(n)$

$\therefore$ Max level $= \log_{10/9} n$

$\therefore O(n \log n)$

Best and Worst case doesn't happen regularly.
Practically quick-sort is very "stable".

Insertion sort - Small data

Quicksort - $(n \log n)$ for any size of data.

05/02/24
Monday

Master's Theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$
$$\| $$
$$D(n) + C(n)$$

No. of nodes in last level $a^{\log_b n} = n^{\log_b a}$.

$$\text{Total} = \theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$

1. Bottom level cost > Top Level Cost

$f(n) = O(n^{\log_b a - \epsilon})$

$g(n) = O(n^{\log_b a})$

$n^{\log_b a}$: bottom level cost

ϵ: polynomially big

O: upper bound

2. Equally distributed.

$f(n) = \theta(n^{\log_b a})$

$g(n) = \theta(n^{\log_b a} \cdot \log n)$

3. Bottom level cost < Top level Cost (monotonically decreasing from top)

$af\left(\frac{n}{b}\right) \leq cf(n)$

$g(n) = \theta(c(f(n))$

$c < 1$, $\Sigma \neq 1$

3 → Top dominated.

Proof:

Case 1

$$g(n) = O\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j} \right)^{\log_b a - \varepsilon} \right)$$

$$= O\left(n^{\log_b a - \varepsilon} \cdot \sum_{j=0}^{\log_b a - 1} \left(\frac{a}{b^{\log_b a - \varepsilon}} \right)^j \right)$$

$$= O\left(n^{\log_b a - \varepsilon} \cdot \sum_{j=0}^{\log_b a - 1} \left(\frac{a}{a \cdot b^{-\varepsilon}} \right)^j \right)$$

$$= O\left(\log_b \cdot a^{-\varepsilon} \cdot \sum_{j=0}^{\log_b a - 1} b^{\varepsilon j} \right)$$

$$= O\left[n^{\log_b a - \varepsilon} \cdot \frac{(b^\varepsilon)^{\log_b n} - 1}{b^\varepsilon - 1} \right]$$

keep, others constant

$$= O\left[n^{\log_b a - \varepsilon} \cdot \frac{n^\varepsilon - 1}{b^\varepsilon - 1} \right]$$

$$= O\left(n^{\log_b a} \right)$$

$$T(n) = \Theta\left(n^{\log_b a} \right) + g(n)$$

Case II

$$g(n) = O\left(\sum_{j=0}^{\log_b - 1} a^j \left(\frac{n}{b^j} \right)^{\log_b a} \right)$$

$$= O\left(n^{\log_b a} \cdot \sum_{j=0}^{\log_b a - 1} \left(\frac{a}{b^{\log_b a}} \right)^j \right)$$

$$= O\left(n^{\log_b a} \cdot \sum_{j=0}^{\log_b a - 1} \left(\frac{a}{a} \right)^j \right)$$

$$= O\left(n^{\log_b a} \cdot \sum_{j=0}^{\log_b a - 1} 1 \right)$$

$$= O\left(n^{\log_b a} \cdot \log_b n \right)$$

$$g(n) = \Theta\left(n^{\log_b a} \cdot \log_b n \right)$$

$$= \Theta\left(n^{\log_b a} \cdot \log_b n \right)$$

Case III

$$a f\left(\frac{n}{b}\right) < c f(n) \longrightarrow \frac{a}{c} f\left(\frac{n}{b^2}\right) < f\left(\frac{n}{b}\right)$$

replacing n by n/b

$$\Rightarrow \frac{a}{c} f\left(\frac{n}{b}\right) < f(n)$$

$$\Rightarrow \frac{a^2}{c^2} f\left(\frac{n}{b^2}\right) < f(n)$$

$$\vdots \qquad \vdots$$

$$\left(\frac{a}{c}\right)^j f\left(\frac{n}{b^j}\right) < f(n)$$

$$\Rightarrow a^j f\left(n/b^j\right) < c^j f(n)$$

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b}\right)^j$$

$$\leq \sum_{j=0}^{\log_b n - 1} c^j f(n) + O(1)$$

$$\leq f(n) \sum_{j=0}^{\log_b n - 1} c^j + O(1)$$

$$= f(n) \cdot \frac{1}{1-c} + O(1)$$

$$= O\left(f(n)\right)$$

12/02/24
Monday

Dynamic Programming (DP)

- sub-problems
- reuse of sub-problems. (not done in D&C)
- Restricted vs AI (more options, explore, path finding)
- optimization problem → Gives a single solution min cost, max value.
- Single objective (single goal / factor / constraint)
  (Time / Path / Cost / Quality but not together)
  (Can't get best of all features if they aren't aligned) → Minimum things

Fibonacci Number

$$F(n) = F(n-1) + F(n-2)$$

$$F(n) = F(n-1) + F(n-2)$$

Subproblem $= F(n-2) + F(n-3) + F(n-3) + F(n-4)$

Not optimization → Divide and Conquer.

Not entirely DP → Reuse subproblem but doesn't optimize.

Characterize the problem

i. Divide and Conquer

ii. Recursive

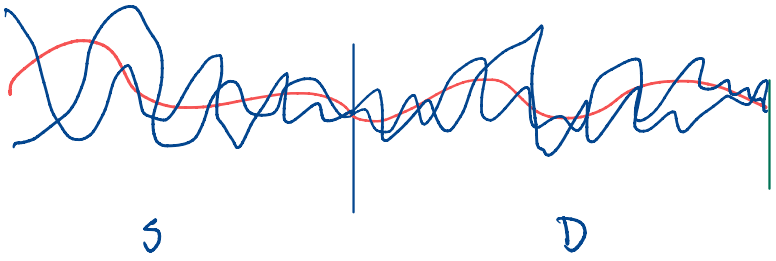
iii. Reuse subproblem

* (Optimal substructure) * Tries all combination (BF) picks best

* DP doesn't remember all combinations/paths, gives best single output.

Airlines Examples

$(S)_{best} + (D)_{best}$



best of S + best of D

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$$

Optimal substructure Eq^n

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) \qquad O(2^n)$$

Kinda bruteforce

Memoization

Recursion + DP bad

Bottom up → DP, no recursion required.

Combination problem minimized, Overall best ← best of all combination

DP → Memory vs Time

Possible within the substructure.

01/03/24
Friday

Dynamic Programming

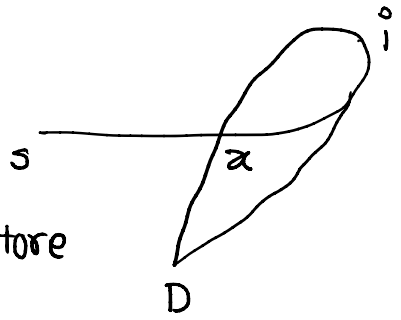
Combining subproblem

Optimal Substructure

* Longest Simple path

After combining -

longest but not simple



All combinations not possible to store

* loops

Optimal substructure fails

DP is hard.

Matrix Chain Multiplication / Paranthesis

Sequence important

Based on dimension the number of dimension is determined.

A_0 A_1

$P_0 \times P_1$ $P_1 \times P_2$

$$C = P_0 \times P_1 \times P_2$$

Combinatorial problem

$A_1\ A_2\ A_3$ - $(A_1 A_2) A_3$, $A_1 (A_2 A_3)$

$A_1 A_2 A_3 A_4$ - $A_1 (A_2 A_3 A_4), (A_1 A_2)(A_3 A_4), (A_1 A_2 A_3) A_4$

2 1 2

5

"Combination is hard to understand"

$A_1 A_2 A_3 A_4 A_5$ - $A_1 (A_2 A_3 A_4 A_5), (A_1 A_2)(A_3 A_4 A_5), (A_1 A_2 A_3)(A_4 A_5),$

5 2 2

$(A_1 A_2 A_3 A_4) A_5$ = 14

5

Exponential problem - grows quickly Catalan Number

Catalan Number

All possible combination.

$$\sum_{K=1}^{n-1} P(K)\, P(n-K) \qquad n \geq 2$$

Grows exponentially - not easily tractable.

Cut all position k, multipucations needed - $P_{i-1} P_k P_j$

Subproblems already calculated.

We don't do combination.

Complexity of problem.

Change problem - It gets hard.

DP ✓ Combination ✗ Same time

Cost Matrix & Solution Matrix

$A_1 (A_2 A_3) = 0 + 2625 + 30 \times 35 \times 5$

$(A_1 A_2) A_3 = 15{,}750 + 0 + 30 \times 15 \times 5$

Algo isn't a theory, its a technique

LCS Edit Distance ✓

BLAST ×

Interpretation of Table

DTW from Internet (Dynamic Time Warping)

0/1 Knapsack ↳ Move & Match

GPU Computer Doesn't help DP

Quasi - Polynomial $O($ item no $\times$ sack size $)$

$k \times 2^n$

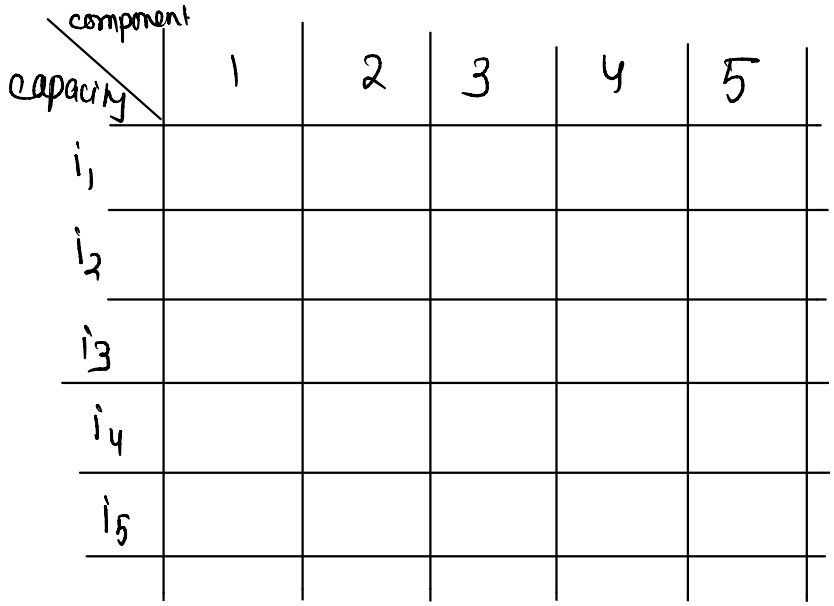
0/1 Knapsack - hard problem

but weak, easy to solve.

Knapsack problem - hard (0/1)

Fractional Knapsack problem is easy

All or nothing
Taking or not taking that element.
Within capacity



Depend on data size, not input size / value.

Good algo depend on input size

0/1 Knapsack - Input capacity
DP - Doesn't depend on input size.
Sequential process, can't be parallel.

#Online

02/03/24
Saturday

Longest Common Subsequence (LCS)

First algo in bio-informatics, then BLAST.

EDITH Distance Algo

Divide & Conquer Problem

Match from x or y - max(c[i,j-1], c[i-1,j])

Proof - if you want

Computer representation - 2 tables (data, direction)

Sequence consecutive match

Table is important, follow sequence

Signal matching - Warping & Matching
Sequential matching

dtw(i,j) = cost(i,j)

medium - an intuitive approach to dtw

Diagonally → Progressive Match (most diagonal match)

Horizontal/Vertical → Single Match - element (not good)

Will not always produce good match.

O(mxn) → expensive

DTW allows matching betw any dimension (big, small)

Use fast DTW - Saves time, but quality not ensured
Longer sequence zipped down.

Complexity O(n)

Interpolate to match - Bad result

Constant DTW - Skip matching beyond diagonal,
No interpolation for match
Progressive sequence
Matching skipped - Faster
Good.

LSTM >> DTW

CTC : ML alternative of HMM
tweaking needed, not good

FFT → Divide & Conquer, skip for now

DP

Rod

Matrix

LCS Edith

DTW

0/1 Knapsack

Algo depend on sack capacity, can't skip - bad
not input size of sack.

Quasi-Polynomial - Hard but close to soln

Quicksort

Complete analysis

Randomized Quicksort - Randomized pivot

Instead of last element, switch last with another element, then new last element as pivot

Randomized pivot - 7.1

Max - best

Assumption

Indicator random variable - spike

Uniform probability of pivot.

left & Right side is not overlapped.

Quicksort - Reasonably well, Stable

O(nlogn)

Partition

Ch-9

Median order statistics (9.1)

Median in linear time.

Exaggerated assumption

9.1~9.2

CH-2	7
DNC. quick sort - 6,7	2
CH 9	9
	15

Text, Diagram

Explain