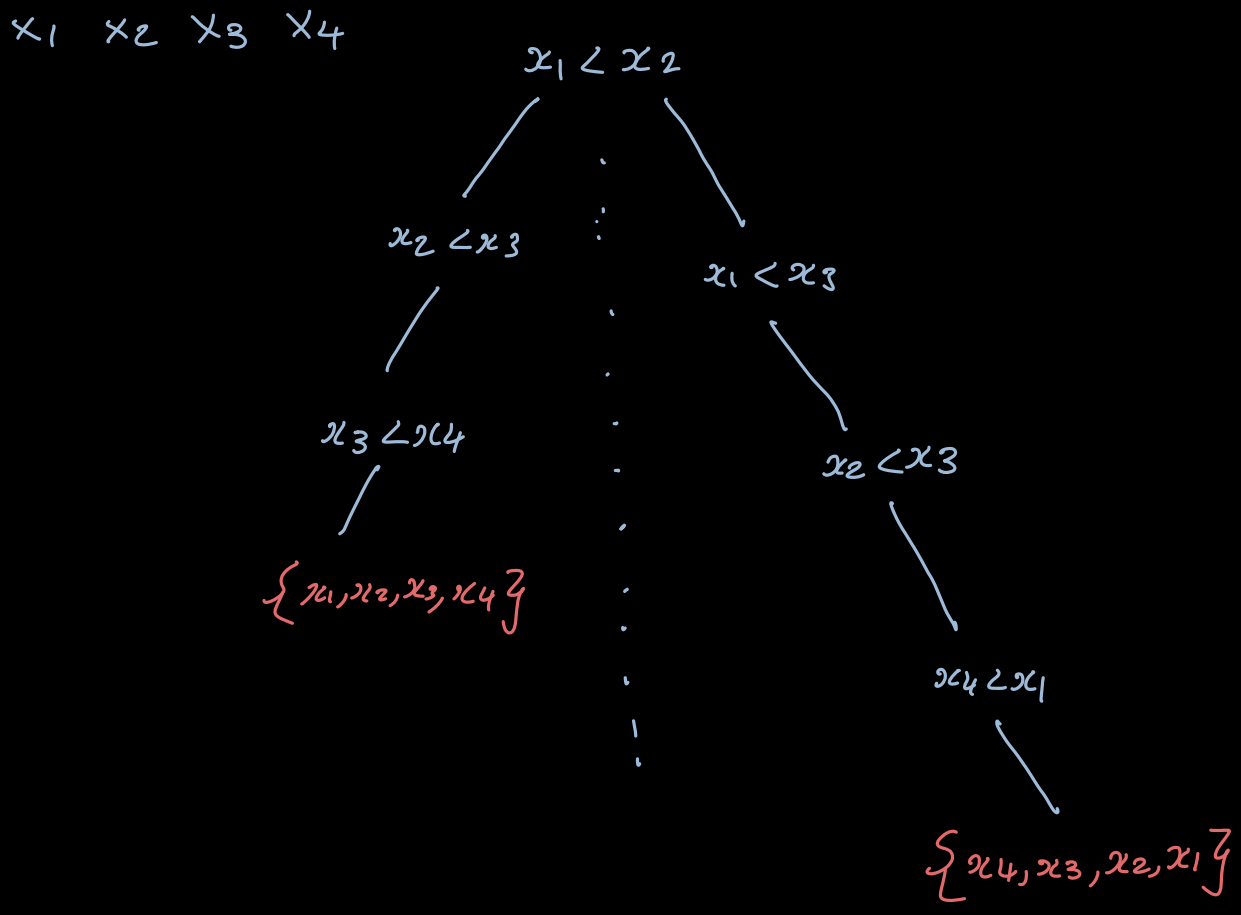
LECTURE 9

GOWTHAM RAJESHSHEKARAN
W1607812
OCT 25, 2021

COMPARISION ⇒ nlogn PROOF :



Total Solution Set = $n!$ [∵ $n!$ PERMUTATION]

Height of the tree = 2^h → longest path from root to the leaf node

"since height varies across branches"

$$n! \leq \text{leaves} \leq 2^h$$

$$n! \leq 2^h$$

$$\log(n!) \leq h$$

stirling's ↑

$$n\log n \leq h$$

$h = \Omega(n\log n)$ → it does min $n\log n$ comparisions

III[rd] we can prove for mergesort ⇒ ∴ All comparision sort algorithms does "$n\log n$" comparisions

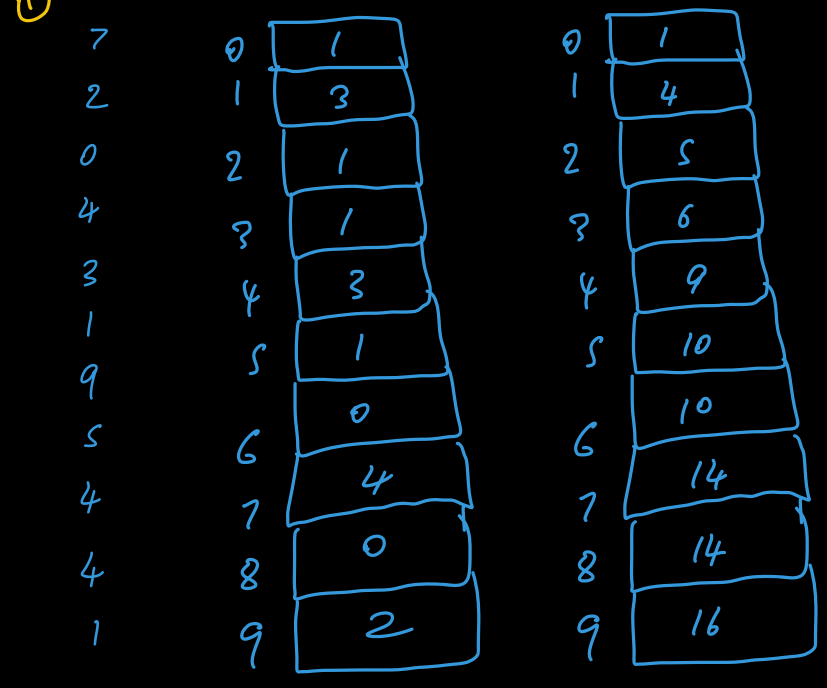
LINEAR TIME SORTING ALGORITHMS

Analyse the input and use their properties to sort

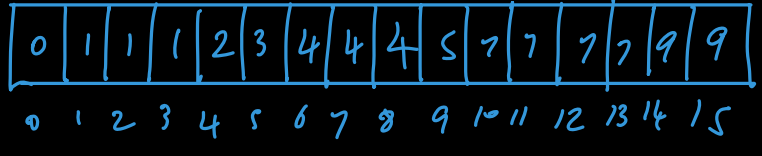
Don't do comparision

COUNTING SORT

①



GIVEN ARR → 'A'



'B' → RESULT ARR

IMPLEMENTATION

countsort (A, n)

```
for i=0 ... max(A)
    C[i] = 0                    // initialise 0s

for i=0 .... n-1
    C[A[i]] += 1                // count frequency

for i=1 ... max(A)
    C[i] = C[i-1] + C[i]        // running sum

for i=0 .... n-1
    B[C[A[i]]-1] = A[i]         // Result Array
    C[A[i]] = C[A[i]] - 1
```

TC = O(n+k) → max value of A

↳ can be written as max(n, k)

SC = O(n+k)

②

GIVEN → A:

100	9	27	15	32	14	2	1	16	29	12	32

C → FREQ ARR

[1] - 1
[2] - 1
[9] - 1
[12] - 1
[14] - 1
[15] - 1
[16] - 1
[27] - 1
[29] - 1
[32] - 2
[100] - 1

RES ARR → B

1	2	9	12	14	15	16	27	29	32	32	100

STABILITY

A sorting algorithm is stable if all the duplicates in resultant array are in the same order as original array.

COUNTING SORT - PROPERTIES

* Works only for non-negative integers.
* Can be useful when the range of values is less.
* Is used to implement RADIX SORT Algorithm.
* It is a stable sorting algorithm
* Can be used for characters as well → CONVERT TO INT → ASCII
* Application space → VERY SMALL
  Running time → VERY FAST (linear)

RADIX SORT

Sort the elements digit by digit from LSD to MSD

ALGORITHM

TC: $\theta(m(n+k))$

for $d = 1 \ldots m$

sort $(A[d])$ → should be a stable sorting algorithm (counting sort)

PROPERTIES

You can use arrays to store the digits → more space required

You can balance between m & b to reduce the runtime → take more digits

(m & b inversely proportional)

TC: $\theta(m(n+k))$

$\theta(5(n+9))$

$2(3(n+99))$

$\theta(\log_{10}{}^{k}(n+10)) \rightarrow \theta\left(\frac{\log_b{}^{k}}{2}(n+b^2)\right)$

$\rightarrow \theta\left(\frac{\log b^{k}}{n}(n+b^{n})\right)$ → make this only dependent on 'n'

$n \geq b^{n}$

$n \geq \log b^{n}$

$\rightarrow \theta\left(\frac{\log_b{}^{k}}{\log_b n}(n)\right)$

$\rightarrow \theta(n)$

MATH PROPERTY

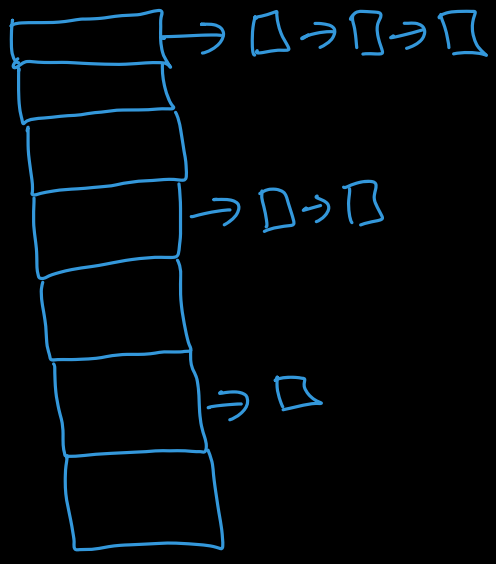
Decimal No $= x$

$\log_{10} x \rightarrow$ no. of digits

BUCKET SORT

Apply to range $[0.0, 1.0)$

APPLICATION "Uniform Distribution ⇒ normalising"



'n' buckets

RUN TIME ANAL:

$\theta(n)$

$n \rightarrow \sum_{i=0}^{n-1} \theta(n_i)^2$

TC: $\theta(n+m)$ ← buckets

if we use 'n' buckets

TC: $\theta(n+n) \Rightarrow \theta(n)$

NOTES

All above 3 algorithms run in linear time $\rightarrow \theta(n)$

We cannot go lower than this → need to check each element

DYNAMIC PROGRAMMING

Programming → represent tables

FIBONACCI

```
Fib(n)
    if n<=1
        return n

    return fib(n-1) + fib(n-2)
```

TC: $O(2^n)$

* lot of recomputations
* Use memoization (TABLE) to save values

```
Fib(n,R)
    if n<=1
        return n
    if R[n] exists
        return R[n]
    else
        R[n] = fib(n-1,R) + fib(n-2,R)
```

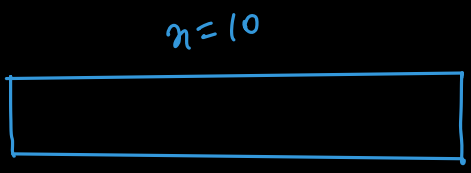
TC: $O(n)$

DP REQUIREMENTS

1) Overlapping Subproblems
2) Optimal Substructure

* Naive approach had "overlapping subproblems"
* Used DP Table (MEMOIZATION) to save values and use it.
* TWO TYPES
    1. TOP-BOTTOM (MEMOIZATION)
    2. BOTTOM UP
* Here in FIB, we assume subprob has same value but not generally the case

① ROD CUTTING



$P_{10} = \$18$

$P_8 = \$25$

if $n=100$

↳ $12 P_8 = 300$

↳ $10 P_{10} = 180$ ✓

lets say $P_6 = \$50$ available

$n=100$

↳ $18 P_6 = 900$

In reality we need to try all combinations of P_6 P_8 P_{10} to get the best value

P_1

$\vdots$

P_n

$n-1$ cut positions

k possible cuts

$$\sum_{k=1}^{n-1} \binom{n-1}{k} = 2^n \quad \text{(exp run time)}$$

ALGORITHM

We know D&C $\quad 1 \le x \le n$

Divide into 2 $\quad x \mid n-x$

$$R(n) = \max(R(x) + R(n-x))$$

Max Revenue ⇒ find all combinations of all dividers of various size cuts & find max value of all

DRY RUN

$n = 5$

5

0 cut

$4+1$	$3+1+1$	$2+1+1+1$	$1+1+1+1+1$
$3+2$	$1+3+1$	⋮	
$2+3$	$1+1+3$		
$1+4$	$2+2+1$		
1 C	$1+2+2$		
	$2+1+2$		
	2 C's	3 C's	4 C's

↳ rearrange all based on 1st values

P_1 $1+\underline{3+1}$ $R(4)$ $\quad 2+\ldots$ $\quad 3+\ldots$

$1+\ldots$ $\quad 2+\ldots$ $\quad 3+\ldots$

$1+\ldots$ $\quad 2+\ldots$

⋮ ⋮

$P_1 + R(4)$ $\quad P_2 + R_3$ $\quad P(3) + R(2)$

$$R(n) = \max(P[x] + R(n-x))$$

↳ * brought to 1 subproblem with overlapping computations

* Simple to solve

Can we guarantee the ans is the most optimal

↳ YES by proving optimal substructure

OPTIMAL SUBSTRUCTURE

Optimal solutions of big problems has optimal solutions of the sub problem.

$$R(n) = \max\left(P[x] + R(n-x)\right)$$

If $R[n]$ is optimal, then $R[n-x]$ is also optimal

PROOF BY CONTRADICTION

Assume $R(n-x)$ is not optimal $\rightarrow R'(n-x) > R(n-x)$

then, $R'[n] = P[x] + R'(n-x) > P[x] + R(n-x) = R(n)$

$R'(n) > R(n) \rightarrow$ CONTRADICTION

$\therefore R(n) = \max\left(P[x] + R(n-x)\right)$

PSEUDOCODE → TOP DOWN

```
rodcut(n, P)
    if n==0 return 0                                  ---- 1
    if R[n] exists ⇒ return R[n] → MEMO
    max = -∞                                          ---- 1
    for x = 1 ... n                                   ---- Σ_{x=1}^{n}
        r = P[x] + rodcut((n-x), P)                   ---- Σ_{x=1}^{n} T(n-x)
        if r > max                                    ---- Σ_{x=1}^{n} 1
        max = r                                       ---- Σ_{x=1}^{n} 1
        c[n] = x → cut
    R[n] = max → MEMO
    return max                                        ---- 1
```

RUNTIME ANALYSIS

$$T(n) = c + \sum_{x=1}^{n} T(n-x) + n + \underbrace{\sum_{x=1}^{n} 1}_{n}$$

$$= \sum_{x=1}^{n} T(n-x) + O(n)$$

OVERLAPPING SUBPROBS

$T(n-1) + T(n-2) + \dots T(0)$

$T(n-2) + T(n-3) + \dots T(0)$

You need to guarantee both has same val

↳ OPTIMAL SUBSTRUCTURE PROOF is enough

RUNTIME → DP

$T(n) = \theta(n^2)$ → Find optimal (looping) at each step for n steps

NOTES

To find the cut → x value
↳ save x value to know the cuts while finding maximum

DRY RUN

	1	2	3	4	5
P	11	5	12	20	9

n = 5

R	11	22	33	44	55

C	1	1	1	1	1

(x → cut value at each max)

CUTTING → 1 1 1 1 1

$R(5) = \max_{1 \le x \le 5} \left(P[x] + R(5-x) \right)$

$R(0) = 0$

$R(1) = P(1) + R(0) = 11$ //

$R(2) = \max \begin{cases} P[1] + R[1] \\ P[2] + R[0] \end{cases} \rightarrow \begin{matrix} 11+11 \\ 5+0 \end{matrix} \rightarrow 22$ //

$R(3) = \begin{matrix} 11+22 \rightarrow 33 \\ 5+11 \rightarrow 16 \\ 12+0 \rightarrow 12 \end{matrix}$ //

$R(4) = 44$ //

$R(5) = 55$ //

PSEUDOCODE → BOTTOM UP

```
def rodcut(n, P):
    R[0] = 0
    for i = 1 to n
        max = -∞
        for x = 1 to i
            q = P[x] + R(n-x)
            if q > max:
                max = q
                c[i] = x
        R[i] = max
    return R[n]
```

TC: $O(n^2)$

LONGEST COMMON SUBSEQUENCE (LCS)

NAIVE ⇒ TC: $O(m \cdot 2^n)$

A.length = m

B. length = n

$X_n = x_1 x_2 x_3 \ldots x_n$

$Y_m = y_1 \cdot y_2 \cdot y_3 \ldots y_m$

$Z_k = y_1 y_2 y_3 \ldots y_k$

$LCS(X_n, Y_m) = Z_k$

$X_n = X_{n-1}\ x_n$

$Y_m = Y_{m-1}\ y_m$

$Z_K = Z_{K-1}\ y_k$

PROOF BY CONTRADICTION

$|Z_{k+1}| > |y_k|$

↳ not possible

since Z_k is the largest possible answer

① $x_n = y_m$

$y_k = x_n$ and

$LCS(X_{n-1}, Y_{m-1}) = Z_{k-1}$

② $x_n \neq y_m$

$$\max \begin{cases} LCS(X_n, Y_{m-1}) \\ LCS(X_{n-1}, Y_m) \end{cases}$$

(PROOF BY CONTRADICTION)

IMPLEMENTATION → D&C

$x_i \quad y_j$

$$L_{ij} = \begin{cases} 0 & \text{if } i=0 \,||\, j=0 \\ 1 + L_{i-1,j-1} & \text{if } x_i = y_j \\ \max(L_{i-1,j}, L_{i,j-1}) & \text{if } x_i \neq y_j \end{cases}$$

DIV & CON - CODE

LCS(X, Y, i, j, L): → T(n,m)

if i=0 || j=0
    return 0 → if L[i,j] exists → return L[i,j]
if X[i] == Y[j]
    L[i,j] = 1 + LCS(X, Y, i-1, j-1) → T(n-1, m-1)
else
    L[i,j] = max(LCS(X, Y, i-1, j), LCS(X, Y, i, j-1))
    ↓ T(n-1, m) ↓ T(n, m-1)
return L[i,j]

• → TABULATION (TOP DOWN)

TC: O(n²)
SC: O(n)

TABULATION

i \ j →	[0]	a	@	b	@
[0]	0	0	0	0	0
b	0 →	0	0	1	1
@	0	1	1 →	1	2
@	0	1	2	2	2

→ to trace characters (select the diagonal's characters)

ANS: aa (2)

MIDTERM 2 PORTIONS

NOV 3, 21

① SUB ✓

② Master ✓

③ Linear Sort ✓

④ DP ✓

LCS - BOTTOM UP

$X_n = X_{n-1}\ x_n$

$Y_n = Y_{n-1}\ y_n$

$Z_k = Z_{k-1}\ y_k$

$L_{1,1} = x_1$ vs y_1 $\qquad L_{2,2} = x_1x_2$ vs y_1y_2

$L_{1,2} = x_1$ vs y_1y_2 $\qquad L_{2,3}$

⋮ ⋮

*) We have dependency on previous values (3 values)
- ↳ diag i-1, j-1
- ↳ left i, j-1
- ↳ top i-1, j

*) We need to find the order to solve this (KEY)

```
LCS(X, Y, n, m):
    Create L[0....n, 0.....m]
    for k = 1...n : L[k,0] = 0
    for k = 1...m : L[0,k] = 0
     L[0,0] = 0
    for i = 1...n:
       for j = 1...m:
          if x[i] == y[j]
             L[i,j] = 1 + L[i-1,j-1]
          else:
             L[i,j] = max(L[i-1,j], L[i,j-1])
    return L[n,m]
```

TC: $O(n^2)$

SC: $O(n)$

③ OPTIMAL BINARY SEARCH TREE

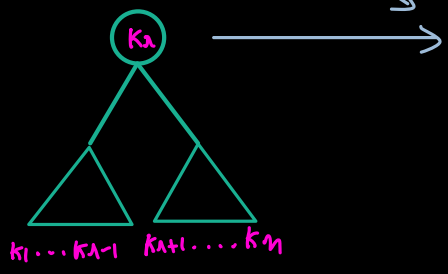
Construct a BST with smallest search cost.

Given: key and frequencies → $k_1, k_2, \ldots k_n$ (KEYS)

Number of BST = $\Omega\left(\frac{4^n}{n^{1.5}}\right)$ → TC: $O(4^n)$

STEP ①: SOLVE USING DAC

$k_1, k_2, k_3, \ldots k_{r-1}\ k_r\ k_{r+1} \ldots k_n$



→ choose k_r based on the one that gives min cost

* Still the cost is high to construct tree for all such k_r choices → exponential
* But the problem has both
  1) Overlapping sub problems
  2) Optimal Substructure
* Use DP to solve it by saving pre-computed values

$SC_i = [\text{depth } k_i + 1] * P_i$ → Probability of that key (after normalisation)

↓

+1 cos depth starts from 0

$TSC = \sum_{i=1}^{n} SC_i$ → we need to min this

$$T_{1,n} = T_{1,r-1} + k_r + T_{r+1,n}$$

$$T_{1,r-1} = \sum_{i=1}^{r-1} (depth(k_i) + 1) * p_i$$

$$T'_{1,r-1} = \sum_{i=1}^{r-1} (depth(k_i) + 1 + 1) * p_i$$ ⇒ when added to a node

[because the depth of sub tree goes down by 1, goes under that node]

$$T'_{1,r-1} = T_{1,r-1} + \sum_{i=1}^{r-1} p_i \quad -①$$

Illy,

$$T'_{r+1,n} = T_{r+1,n} + \sum_{i=r+1}^{n} p_i \quad -②$$

$$\therefore TSC = TSC_{1,r-1} + TSC_{r+1,n} + \sum_{i=1}^{n} p_i$$ → ① + ② + p_r

RECURRENCE

$$TSC_{x,y} = \begin{cases} 0 & \text{if } y = x-1 \\ \min\limits_{x \le r \le y} \left(TSC_{x,r-1} + TSC_{r+1,y} + \sum_{i=x}^{y} p_i \right) & \text{if } x \le y \end{cases}$$

PSEUDOCODE

```
OBST(p, x, y, C[1..n, 1...n]):
    if y < x : return 0
    if C[x,y] exists return C[x,y]
    for i = x...y:
        sump += p[i]
    min = ∞
    for r = x...y
        q = min(OBST(p,x,r-1) + OBST(p,r+1,y))
    if q < min:
        min = q
    C[x,y] = min
return min + sump
```

OPTIMAL SUBSTRUCTURE ↪ to ensure we can use DP

If $TSC_{x,y}$ is optimal then $TSC_{x,r-1}$ is also optimal (Doing it for one sub tree)

BY CONTRADICTION,

Assume $TSC_{x,r-1}$ is not optimal

then, there is $TSC'_{x,r-1} < TSC_{x,r-1}$

⋮

Prove that it contradicts (Book)

MATRIX CHAIN MULTIPLICATION

NOV 8, 21

A	B	C	=	D
2×3	3×1	1×4		2×4

SCALAR MULTIPLICATIONS = 2×3×4

↳ we need to paranthesise A·B·C·D

to reduce these scalar multiplications

$(A_1 \cdot A_2 \cdot A_3)(A_4 \cdot \dots \cdot A_n)$ → need to figure where to put out this division/cutting/paranthesis

$P_0 \times P_1 \; P_1 \times P_2 \; P_2 \times P_3 \; \dots \; P_{n-1} \times P_n$

GREEDY APPROACH

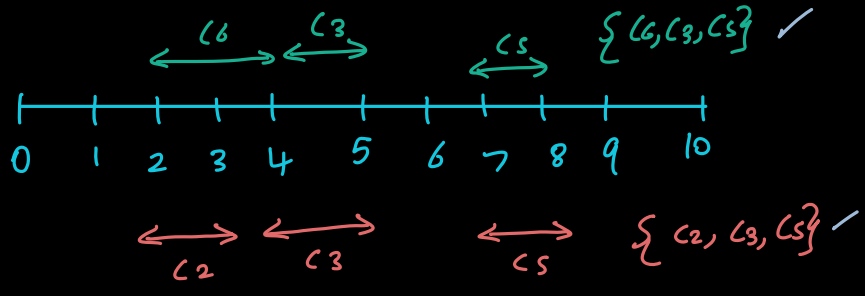
Ability to make greedy choice without solving subproblems

SCHEDULING ACTIVITIES

Given a resource → maximise utilisation

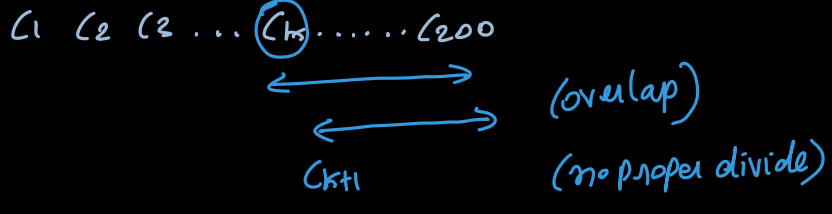
Ex: classroom → maximum activities possible

	START	END
C1	1	6
C2	2	3
C3	4	5
C4	3	9
C5	7	8
C6	2	4



2 solutions

DIVIDE AND CONQUER



$S_{ij} = \{ c_x \quad start_x \geq stop_i \text{ and } stop_x \leq stop_j \}$

$S_{ij} = S_{ik} + \{c_k\} + S_{kj}$

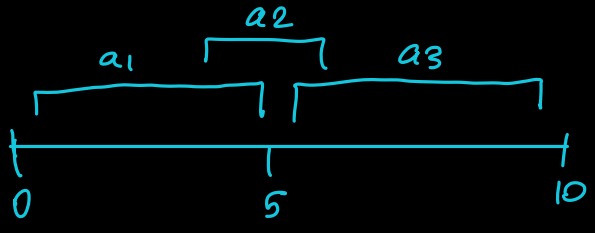
$A_{ij} = A_{ik} + \{c_k\} + A_{kj}$ → maximise this by selecting diff c_k $i \leq k \leq j$

(SOL) (SUB1 SOL) (SUB2 SOL)

CHOICES

① shortest job → sub optimal greedy choice

↙ example case



② EARLIEST ENDING TASK ✓ optimal greedy choice

<PROVE THIS>

PROOF sorted by finishing time

$S_i = \{ c_s, \cdots \}$ GIVEN

$A_i = \{ c_1, \cdots \}$ SOLUTION

TO PROVE: $c_s = c_1$ METHOD ↙

ASSUME OPTIMAL SOL & PROVE GREEDY CHOICE IS IN

if $c_1 = c_s$ → directly proved

if $c_1 \neq c_s$

$A_i - \{c_1\} + \{c_s\}$ — VALID?

A_i

A_i by squeezing greedy into optimal

HUFFMAN CODING

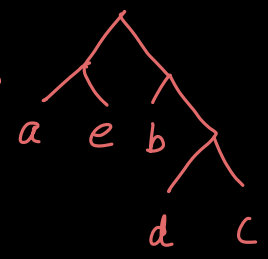
aa bb cc → 192 bits

↓

reduce it

a	00
b	10
c	111
d	110
e	01

PATTERN ⇒

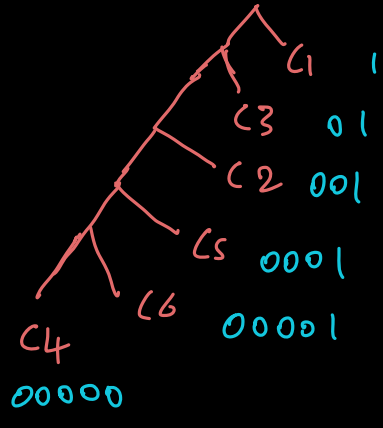


characters as leaf nodes

CHAR	FREQ
C_1	f_1
C_2	f_2
C_3	f_3
C_4	f_4
⋮	⋮
C_n	f_n

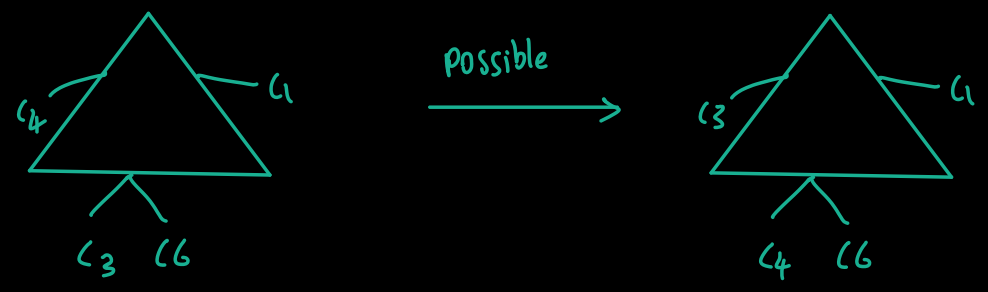
GREEDY CHOICE ⇒ less frequency characters are in lowest level with eq. freq. as siblings

CH	FREQ
C_1	0.50
C_2	0.10
C_3	0.24
C_4	0.05
C_5	0.10
C_6	0.01



No. of bits = No. of edges / Depth

PROOF → GREEDY CHOICE



$$B(T) = \sum_i depth_i \, f_i$$

TO PROVE

$B(T_2) \leq B(T_1)$

$$B(T_1) = \sum_i depth_i \, f_i + depth(c_4) \cdot f_4 + depth(c_3) \cdot f_3$$

$$B(T_2) = \sum_i depth_i \, f_i + depth(c_3) \cdot f_4 + depth(c_4) \cdot f_3$$

[Because same depth]

$$B(T_1) - B(T_2) = depth(c_4) \cdot f_4 + depth(c_3) \cdot f_3 - depth(c_3) \cdot f_4 + depth(c_4) \cdot f_3$$

≥ 0

$$= depth(c_4)[f_4 - f_3] + depth(c_3)[-f_3 + f_4]$$

$$= [depth(c_4) - depth(c_3)] \cdot [f_4 - f_3]$$

< 0 $\quad$ < 0

$\therefore B(T_2) \leq B(T_1)$

HENCE PROVED

TEST FROM HOME

NOV 10, 21

GREEDY ALGORITHM REQUIRES

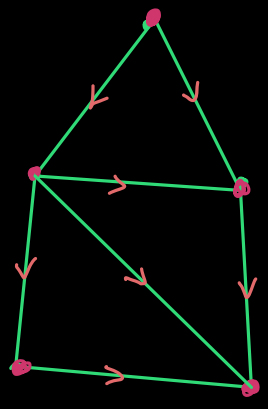
1) GREEDY CHOICE ⇒ the choice that gives the optimal solution

2) PROVE selected greedy choice is optimal

GRAPHS

DS with set of vertices 'V' and set of Edges 'E'

Graph $G = (V, E)$



Can be either directed or undirected

Can be cyclic or acyclic (no loops)

Trees are a form of graph but "acyclic" and "directed"

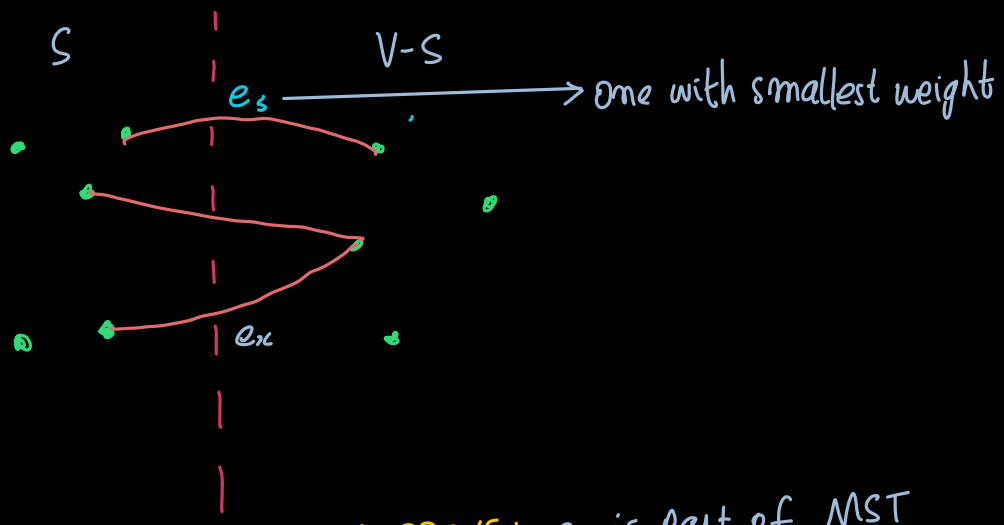
BINARY TREE ⇒ Directed Acyclic Graph

SPANNING TREE ⇒ Spans across all the vertices with least

MINIMUM SPANNING TREE ⇒ Spanning tree with minimum cost

GREEDY CHOICE ⇒ Choose the one that crosses the cut with lowest weight

PROOF



To PROVE: e_s is part of MST

$$w(MST_1) = \sum_{e} w_e$$

$$= \sum_{e \neq e_x} w_e + w_x$$

$$w(MST_2) = \sum_{e \neq e_x} w_e + w_s$$

$$= w(MST_1) - w_x + w_s$$

$$w(MST_2) - w(MST_1) = w_s - w_x \quad (w_x \rightarrow \text{bigger})$$

$$\leq 0$$

$$w(MST_2) \leq w(MST_1)$$

but MST_1 is the optimal MST

so, $w(MST_2) = w(MST_1)$

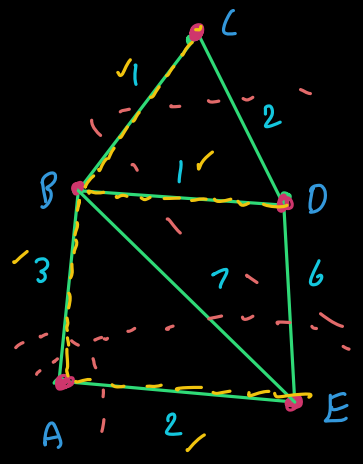
↳ it has w_s

∴ Choosing e_s gave the MST

∴ Our greedy choice is right

HENCE PROVED

PRIM'S METHOD

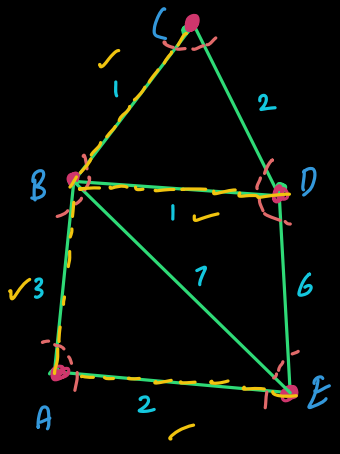


Pick the smallest one connecting the chosen vertices

$O(V \log E)$

"Dependency on vertex"

KRUSKAL'S METHOD



Pick the overall smallest weight edge that

$O(V \log E)$

"Dependency on vertex and edge"

OTHER APPLICATIONS

1) Shortest Path (DIJKSTRA'S ALGORITHM)
2) Fractional knapsack
3) Coins Change
4) Optimal Merge Pattern
5) Selection Sort (But suboptimal)

GREEDY Vs DP

① DP ⇒ exhaustive search to find optimal

GREEDY ⇒ choose one which is optimal

② DP ⇒ Solve subproblems to solve main problem

③ GREEDY ⇒ Pick greedy choice & left with one subproblem which you solve later (so we don't solve sub probs first)

NP-COMPLETE (NPC)

NP-COMPLETE

* There is a solution but very expensive
  - → More than Polynomial
  - → Exponential / Factorial

* Has a solution above polynomial but there is a chance it can get lower

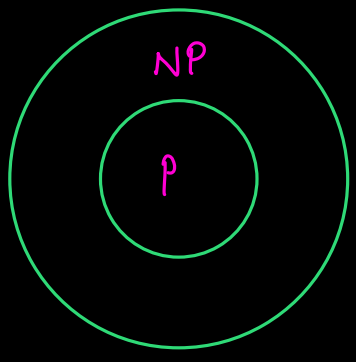
* Tower of Hanoi → factorial but not NP complete because proved cannot better.

P → class of problems can be solved in polynomial time

$x \in P$
if x can be solved in Polynomial time

NP

$x \in NP$
if x can be verified in polynomial time

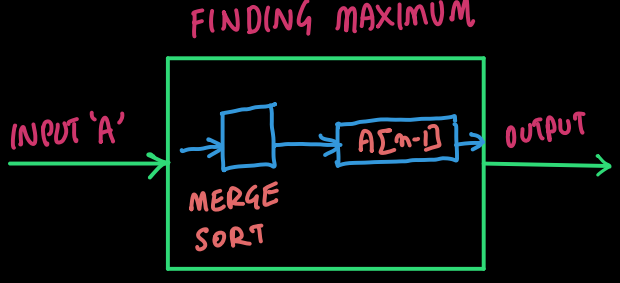


* All problems can be verified in polynomial time

* HAMILTONIAN PATH → visit every single edge only once
  ↳ TRAVELLING SALESMAN PROBLEM → A variation of it

} can validate in polynomial time

* Composing solution of one by solving another → REDUCIBILITY
  Ex: Solving maximum problem by sorting



NP-HARD

$x \in$ NP-HARD
if all problems in NP reducible to X

NP-COMPLETE = NP + NP HARD