Greedy Method:- Technique to solve Problem like divide and Conquer Approach. (Exhaustive)

* Solve optimization $\boxed{\text{Problem}}$ ⟨ Max / Min

→ Many soln exist. (...n) ↳ Constraint / Condition

↓

Soln. satisfying the condition ⇒ feasible soln. (Satisfy the cond)

↓ {min // max}

optimal soln. Only @ 1.

- A problem must be solved in stages
- If soln. is feasible for an input include it in soln.
  * Input is picked one after other.

Knapsack ⟨ fractional → Greedy / 0/1 → Not Greedy.

Activity Selection:- $S = \{a_1, a_2 \ldots a_n\}$. n activities wish for a resource - that can serve one activity at a time.

a_i - start time. s_i
- finish time f_i

$0 \le s_i \le f_i < \infty$

Selection of Activity:
half open interval.

$[s_i, f_i)$, $[s_j, f_j)$ > donot overlap
↳ Activities are compatible.

$s_j > f_i$. "mutually compatible activities"

"$f_1 \le f_2 \le f_3 \ldots \le f_{n-1} \le f_n$" "activities are sorted." in increasing order of finish time.

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

Select with earliest finish time. (First one to finish room for others).
Sorted finish time → select first (other option ⇒ possible max combination available)

Scanned with CamScanner

Activity Selection

Choice $\longrightarrow$ **Greedy**

Select the activity with earliest finish time | then select any satisfy the constraints

Sorted w.r.t finish time — pick 1st

Sol – $i = 1$ ✓

$\{a_1\}$, a_2 ? $s_2 < f_1$ ✗

a_3 ? $s_3 < f_1$ ✗

a_4 ? $s_4 \nless f_1$ ✓

$\{a_1, a_4\}$

a_5 ? $s_5 < f_4$ ✗

a_6 ? $s_6 < f_4$ ✗

a_7 ? $s_7 < f_4$ ✗

a_8 ? $s_8 \nless f_4$ ✓

$\{a_1, a_4, a_8\}$

a_9 ? $s_9 < f_8$ ✗

a_{10} ? $s_{10} < f_8$ ✗

a_{11} ? $s_{11} \nless f_8$ ✓

$\{a_1, a_4, a_8, a_{11}\} \Rightarrow$ optimal Soln

other soln.

$\{a_2, a_4, a_9, a_{11}\}$

Knapsack fractional v_i/w_i → Selection criteria

$O(n \lg n)$

↳ Select with max value per pound

↳ Keep all till (item exhausted)

repeat.

↳ within Cap. of Sack.

# of items = 3 {item 1 10 pound \$ 60

W = 50 pound. item 2 20 pound \$ 100

item 3 30 pound (\$ 120}

i	1	2	3
v_i	60	100	120
w_i	10	20	30
v_i/w_i	6	5	4
	{1	1	2/3}

take $i = 1$ (greater value)

$i = 2$

$i = 3$

$w = 50 - 10 = 40$

$w = 40 - 20 = 20$

$w = 20$

0/1

⇒ item 2, item 3

optimal. 20 + 30

100 + 120

item 1 included in set ⇒ sub optimal.

$V = 60$

$V = 60 + 100$

$V = 160 + 80$

$= 240$.

Huffman Codes (data Compression Technique)

prefix

⇒ 100 th.

300,000 bits

	a	b	c	d	e	f.
freq. in (1000)	45	13	12	16	9	5
fixed length code words	000	001	010	011	100	101
variable length code words.	0	101	100	111	1101	1100

\# bits = 3

Can do better

Code : Huffman (C)

$n = |C|$

$Q = C$

for $i = 1$ to $n-1$

allocate a new node z

z.left = x = Extract-Min(Q)

z.right = y = Extract_Min(Q)

z.freq = x.freq + y.freq.

Insert (Q, z)

return Extract Min (Q) //root of tree

C → is a set of n char. and $c \in C$

c.freq ⇒ freq of character.

Algo builds Tree (bottom-up)

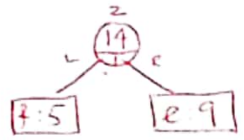
$|C|$ ⇒ leaves $|C| - 1$ merge op.

Q "min priority Queue."

merge two objects to form new obj whose freq is sum of two merged objects.

Q: f:5 e:9 c:12 b:13 d:16 a:45

Code:



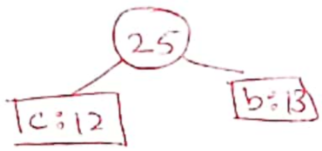
Q:

c:12 b:13 d:16 a:45

Code:

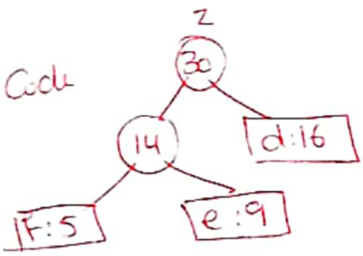
Q:

d:16

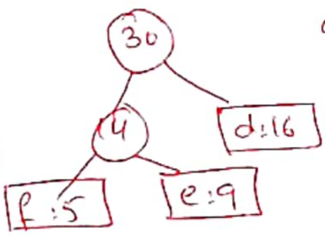


a:45.

Code

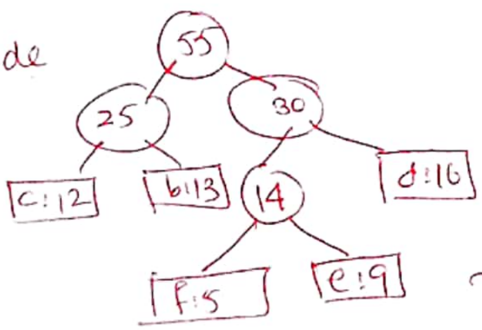


Q:

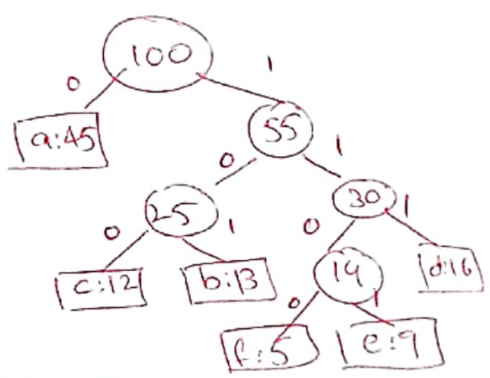


a:45

code



$\Longrightarrow$



Q: a:45

a:0	d:111
b:101	f:1100
c:100	e:1101

Codes.

"Analysis"

Why not Dynamic Programming → Recursive (repeated calls)
→ Memoization.
→ Tabular.

Cormen — Comment // "Generate an optimal soln. w/o solving the sub problem!"

Greedy Choice: Select the activity with earliest finish time and select the other compatible with the selected one!

"Work Top down with greedy approach"
↳ make a choice
↳ work on sub. prob

Code:

```
Rec-Act-Selector(s, f, k, n)
    m = k+1;
    while m ≤ n and s[m] < f[k]     // each activity is examined exactly once in loop
        m = m+1                     *Loop break when activity is found.*
    if m ≤ n
        return {am} ∪ Rec-Act-Selector (s, f, m, n)
    else return ∅
```

Code:

```
Gre-Act-Selector(s, f)
    n = s.length
    A = {a1}
    k = 1
    for m = 2 to n
        if s[m] ≥ f[k]
            A = A ∪ {am}
            k = m
    return A
```

executed n times → runtime $\Theta(n)$

pre Condition: already sorted activities in terms of finish time.

* if sorting required.
Run time $= O(n \lg n)$.

Huffman Analysis

'Q' ⟹ binary Min-heap
Implemented as

'C' set of 'n' characters → Q is initialized and build in $O(n)$ time using procedure Build-Min-Heap

For loop execution $n-1$ times.
heap operation in loop (Extract Min) $O(\lg n)$
" " " " (Extract Min) $O(\lg n)$

$\boxed{O(n \log n)}$ run-time of Huffman Code Algo.

*Can be reduced to $O(n \lg \lg n)$ using Van Emde Boas tree. (Cormen book Ch # 20)

KnapSack fractional Run time $O(n \lg n)$ if not sorted

Frac-knapSack (W, v, w).
while $w > 0$ and items remaing. → $O(n)$ if sorted
pick item with max v_i / w_i.
$x_i \leftarrow \min(1, w/w_i)$
remove i item from list.
$w \leftarrow w - x_i w_i$.