1(a) (i) amount : 29

Greedy solution : 15, 11, 1, 1, 1 count = 5

Optimal " : 15, 7, 7 count = 3

(ii) Let $c(a)$ be a function which gives the minimum coin count for amount 'a'

each coin can be picked multiple times.

~~if a coin of value v_i is taken~~

~~from~~ The problem has an optimal substructure property, as if an optimal solution ~~has~~ for a includes a coin of value v_i ~~then the a for a~~ then the ~~set~~ solution gotten by excluding one coin of v_i is the optimal solution for $a - v_i$

but any coin ~~of~~ from 1, 7, 11, 15 can be taken. so,

$$c(a) = \min\left(c(a-1)+1,\ c(a-7)+1,\ c(a-11)+1,\ c(a-15)+1\right);$$

and $c(0) = 0$ ~~c(1)~~

∴ let the table be of amount 'a' and coin count 'n'

c	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
n	0	1	2	3	4	5	6	1	2	3	4	1	2	3	~~4~~ 2	1	2

c	17	18	19	20
n	3	2	3	4

∴ to make 20, one solution is 1, 1, 7, 11

so the count is 4

(b)

1. If the array size n is greater than 2, divide the array into two halves of size $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$ named L and R

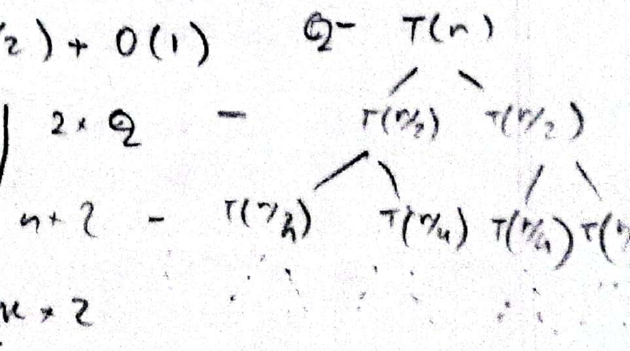
2. Find the ~~minimimum~~ minimum and maximum numbers for both arrays L and R. The minimum of the main array is the ~~minimum~~ lesser of the two minimums, and similarly the maximum is the greater of the maximums of L and R

3. To find the minimum and maximum of L and R, apply steps 1 and 2 for both of them.
So, in total, recursively keep dividing the total array into halves until the parts ~~becom~~ become smaller than 3

4. For ~~elem~~ an array or sub array of size 1, the min and max is the one element of the array. return it.
For size 2, the greater of the two numbers is the max and the other is the min

5. Corner case, if the main array is empty, there are no min or max numbers.

$$T(n) = 2T(n/2) + 2 = 2T(n/2) + O(1)$$



$k = \log_2 n$

$T(n) = 2 + 2 \cdot 2 + 4 \cdot 2 + \dots + 2^k \cdot 2 + 2^k \cdot T(1)$

$= 2(1 + 2 + 4 + 8 + \dots + 2^k) + 2^k \cdot T(1)$

$= 2 \times \frac{2^{k+1} - 1}{2 - 1} + 2^{\log n} \times O(1)$

$= 2 \times 2 \cdot 2^{\log n} - 2 + n \cdot O(1)$

$= 4 \times n - 2 + n \times O(1) = O(n) + O(n) = O(n)$

~~lets arrange the table in increasing order of weigth~~

c// For the 0/1 knapsack problem,

let an optimal solution have item i in the solution for capacity C, then by removing the item i from the solution we get the optimal solution for capacity $C - w_i$; w_i = weight of item i.

So it has an optimal substructure.

∴ let $K(C, n)$ be the solution for capacity C and items from 1 to n

$$\therefore K(C,n) = \max\left(K(C, n-1),\ K\left(\overset{(C-w_n,\, n-1)}{\cancel{C-w_i,\, n-1}}\right) + v_n \right)$$

it is the minimum of the solution of taking the nth item and not taking the nth item,

nth item can only be taken if $w_n \leq C$

∴ the dp table

n \ c	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	50	50	50	50	50	50	50
2	0	50	50	60	110	110	110	110
3	0	50	50	60	110	110	150	150
4	0	50	70	120	120	130	180	180
5	0	50	80	130	150	200	200	210

C

i	w	v
1	1	50
2	3	60
3	5	100
4	2	70
5	2	80

∴ items taken : 2, 4, 5

value " : 60 + 70 + 80 = 210

weigth " : 3 + 2 + 2 = 7 //

1(a) $T(n) = 4T(n/3) + O(\sqrt{n}\log n) + O(n)$

$T(k) = O(n)$ if $k \le 5$

(i) $T(n) = 9T(\frac{n}{3}) + 5n\sqrt{n} + 8\log n$ $a=9$ $b=3$ $c=3/2$

$= 9T(\frac{n}{3}) + O(n^{3/2})$ $\log_b a = 2 > c = 3/2$

$\therefore T(n) = O(n^{\log_b a}) = O(n^2)$

(ii) $T(n) = 3T(\frac{n}{4}) + O(n^2)$ $\log_b a = \log_4 3 < 2 = c$

$\therefore T(n) = O(n^2)$

(iii) $T(n) = 4T(\frac{n}{2}) + O(n^2)$ $\log_b a = \cancel{\ } 2 = 2 = c$

$\therefore O(n^c \log^{0+1} n) = O(n^2 \log n)$

[1,6), [6,9), [9,15)

(b) (i) ~~Bus~~ [1,6), [6,8), [9,15) $\therefore n = 3$ total 2 soln

(ii) buses 1, 3 and 4

(ii) intervals : 5, 4, 2, 6, 3, 4, 4

[2,6), [6,8), [11,15) total 1 soln

buses, 3, 6, 7

(iii) {[1,6), [6,8), [9,15)} → same but [2,6)

{[1,6), [6,8), [11,15)} → " " "

total 4 soln

~~same but [6,9)~~

~~same but [6,9)~~

C// $m[a,b]$ is the ~~a~~ total minimum operations need for the ~~the~~ chain multiplication of matrices from A_a to A_b

$m[a,a]=0$ as no multiplication is needed

chain multiplication can be obtained by the multiplication of chain product from A_a to A_i and A_{i+1} + A_b ~~for~~ where ~~i~~ $a \leq i \leq b$. so there are several possible paths. we must chose the minimum operations required.

$\therefore m[a,b] = \min(\{ m[a,i] + m[i+1,b] + A_a.r \times A_i.c \times A_b.c \} \; i: a \leq i \leq b \})$

$\therefore$ for $m[1,6] \rightarrow$

$m[1,1]+m[2,6]+1\times4\times1 = 0+95+4 = 99$

$m[1,2]+m[3,6]+1\times5\times1 = 20+75+5 = 100$

$m[1,3]+m[4,6]+1\times3\times1 = 35+60+3 = 98$

$m[1,4]+m[5,6]+1\times6\times1 = 53+42+6 = 101$

$m[1,5]+m[6,6]+1\times7\times1 = 95+0+7 = 102$

$m[1,2]=20$

$m[3,4]=90$

$m[5,6]=42$

$m[3,5] = \min(m[3,3] + m[4,5] + 5\times3\times7, \; m[3,4]+m[5,5]+5\times6\times7)$

$= 231$

$m[3,6] = \min(m[3,3]+m[4,6]+5\times3\times1, \; m[3,4]+m[5,6]+5\times6\times1, \; m[3,5]+m[6,6]+5\times7\times1)$

$= \min(75, 162, 266) = 75$

$m[1,4] = \min(m[1,1]+m[2,4]+1\times4\times6, \; m[1,2]+m[3,4]+1\times5\times6, \; m[1,3]+m[4,4]+1\times3\times6) = \min(156, 150, 53) = 53$

$\boxed{m[1,6] = 98}$

a) 1. Identify if the problem is solvable using dynamic programming.
DP ~~programming~~ is applicable for the problems which have overlapping subproblems property, and might also have optimal substructure property.

2. Decide a state for the problem. Define the relation between the current state and one or more of the smaller states.

3. Write a program (usually recursive, can be iterative) that calculates the ~~value~~ solution associated with a state using the relation.

4. Implement memoization or tabulation to reduce time complexity.

(b) For a fractional knapsack problem let $f_1, f_2, f_3, \ldots, f_i$ be the amounts of item 1 to i taken. for the optimal solution for a capacity C. if the ith item is excluded fraction then that is ~~the~~ an optimal solution for capacity $C - w_i \cdot f_i$. Because if there exists another solution which is optimal, then we could add f_i fraction of item i to get a solution for capacity C which will gain more and thus contradict our original assumption

For 0-1 knapsack problem, for items 1 to n, the optimal solution for capacity C is the maximum of the solution for capacity C using 1 to n-1 items and for capacity $C - w_n$ using 1 to n-1 items. In fact if the nth item has weight $w_n > C$ then the first case is the solution, and for $w_n \le C$, the 2nd case is the solution.

Since we can solve a subproblem to find the solution for the main problem, it has overlapping subproblems property

CamScanner

c/ 1. ~~Consi~~ Choose an arbitrary element as a pivot, 'p' without loss of generality let the pivot be the last element

2. Create two array partitions L an R. where L has all the elements less than 'p' and R contains elements greater or equal to 'p'

3. ~~First~~ sort L and R recursively using this quicksort algorithm. join L, p, R in this order to get the quick sorted array.

4. To sort L and R, perform steps 1 to 3 on each of them. Thus dividing until partitions have size 1, in which cas the partition itself is the sorted array

• Best Case :-

The pivot for each array is always the median of the elements, then the partitions are always of size $\frac{n}{2}$. merging ~~that~~ takes $O(n)$ time

$$T(n) = 2T(n/2) + O(n) \qquad \log_b a = \log_2 2 = 1 = 1 = c$$

$$\therefore T(n) = O(n \log n)$$

• Worst Case :-

The pivot is always either the minimum or max of the elements. Then the partitions are of size $n-1$ and 0.

$$T(n) = T(n-1) + T(0) + O(n) = T(n-1) + O(n)$$
$$= T(n-1) + O(n) + O$$

$$T(n) = T(n-2) + O(n-1) + O(n) = T(n-2) + O(n + (n-1))$$

$$T(n) = T(0) + O\left(\sum_{1}^{n} i\right) = O + O\left(\frac{n(n+1)}{2}\right) = O(n^2)$$

4(b) assume Q is a max priority queue of elements of type E

Pseudo code for the function.

```
ExtractMin (Q) {
    E elem = Q.ExtractMax();
    if (Q.size()== 0) {
        return elem;
    }
    E min = ExtractMin(Q);
    Q.insert (elem);
    return min;
}
```

(c) $X = \langle TAAGUATU \rangle$ $Y = \langle AUAGTUT \rangle$

$$\text{~~DP~~ } LCS(m,n) = \begin{cases} LCS(m-1,n-1)+1 & \text{if } X[m] == Y[n] \\ \max(LCS(m,n-1),\ LCS(m-1,n),\ LCS(m-1,n-1)) & \text{if } X[m] \,!\!= Y[n] \end{cases}$$

$LCS(m,0) = 0$ $LCS(0,n) = 0$

DP Table

Y \ X	_	T	A	A	G	U	A	T	U
_	0	0	0	0	0	0	0	0	0
A	0	0	1	1	1	1	1	1	1
U	0	0	1	1	1	2	2	2	2
A	0	0	1	2	2	2	3	3	3
G	0	0	1	2	3	3	3	3	3
T	0	1	1	2	3	3	3	4	4
U	0	1	1	2	3	4	4	4	5
T	0	1	1	2	3	4	4	5	5

∴ Longest common subsequences are

A A G U T

A U A T U

- A A G T U

One shortest common supersequence

- TAUAGUATUT
- TAUAGTUATU
- TAAGUAGTUT