

Q1

ANSWER:

Minimum scalar multiplications = 43

Optimal parenthesization = $[(M_1 M_2) M_3] M_4$

WORKING:

Input:

$$\begin{array}{cccc}
 M_1 & M_2 & M_3 & M_4 \\
 \begin{matrix} 1 \times 5 \\ m_0 \end{matrix} & \begin{matrix} 5 \times 2 \\ m_1 \end{matrix} & \begin{matrix} 2 \times 3 \\ m_2 \end{matrix} & \begin{matrix} 3 \times 9 \\ m_3 \end{matrix} \\
 & m_1 & m_2 & m_3 & m_4
 \end{array}$$

$$M_i = m_{i-1} \times m_i$$

m_0	m_1	m_2	m_3	m_4
1	5	2	3	9

Cases: $c(i,i) = 0$

$$c(i,j) = \min \left\{ c(i,k) + c(k+1,j) + (m_{i-1} \times m_k \times m_j) \right\} \quad \begin{matrix} \forall k \text{ s.t} \\ i \leq k \leq j \end{matrix}$$

Run-through: $c(1,1) = c(2,2) = c(3,3) = c(4,4) = 0$

(done diagonally) $\forall c(i,j) \text{ s.t } i > j = X$

$$c(1,2) = \min \left\{ \underset{0}{c(1,1)} + \underset{0}{c(2,2)} + \underset{5 \times 2}{(m_0 \times m_1 \times m_2)} \right\} = 10$$

$$c(2,3) = \min \left\{ \underset{0}{c(2,2)} + \underset{0}{c(3,3)} + \underset{5 \times 2 \times 3}{(m_1 \times m_2 \times m_3)} = 30 \right\}$$

$$c(3,4) = \min \left\{ \underset{0}{c(3,3)} + \underset{0}{c(4,4)} + \underset{2 \times 3 \times 9}{(m_2 \times m_3 \times m_4)} = 54 \right\}$$

	1	2	3	4
1	0			
2	X	0		
3	X	X	0	
4	X	X	X	0

$$C(1,3) = \min \left\{ \begin{array}{l} C(1,1) + C(2,3) + (m_0 \times m_1 \times m_3) = 45 \\ 0 \quad \quad \quad 30 \quad \quad 1 \quad 5 \quad 3 \end{array} \right.$$

$$C(1,2) + C(3,3) + (m_0 \times m_2 \times m_3) = 16 \checkmark \quad \begin{array}{cccc} \times & 0 & 30 & 54 \end{array}$$

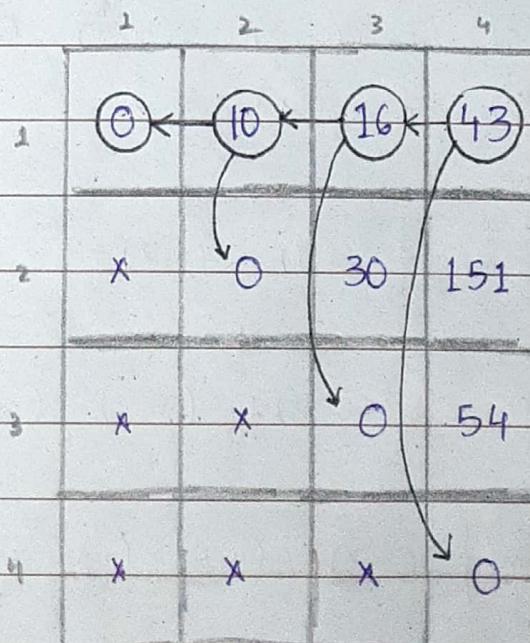
$$C(2,4) = \min \left\{ \begin{array}{l} C(1,2) + C(3,4) + (m_1 \times m_2 \times m_4) = 154 \\ 10 \quad \quad \quad 54 \quad \quad 5 \quad 2 \quad 9 \end{array} \right.$$

$$C(1,3) + C(4,4) + (m_1 \times m_3 \times m_4) = 151 \checkmark \quad \begin{array}{cccc} \times & \times & 0 & 54 \end{array}$$

$$C(1,4) = \min \left\{ \begin{array}{l} C(1,1) + C(2,4) + (m_0 \times m_1 \times m_4) = 196 \\ 0 \quad \quad \quad 151 \quad \quad 1 \quad 5 \quad 9 \end{array} \right.$$

$$C(1,2) + C(3,4) + (m_0 \times m_2 \times m_4) = 82 \quad \begin{array}{cccc} \times & \times & 0 & 54 \end{array}$$

$$C(1,3) + C(4,4) + (m_0 \times m_3 \times m_4) = 43 \checkmark$$



Backtracking:

ROUGH WORK & PROOF OF CONCEPT : (Shown on next pages) ↗

$$M_1 \cdot M_2 \cdot M_3 \cdot M_4$$

$$1 \times 5 \quad 5 \times 2 \quad 2 \times 3 \quad 3 \times 9$$

$$m_0 \quad m_1 \quad m_2 \quad m_3 \quad m_4$$

Rules:

$$c(i,i) = 0$$

$$c(i,j) = \min_{\substack{\forall k \text{ s.t} \\ i \leq k < j}} \left\{ c(i,k) + c(k+1,j) + (m_{i-1} \times m_k \times m_j) \right\}$$

dimensions

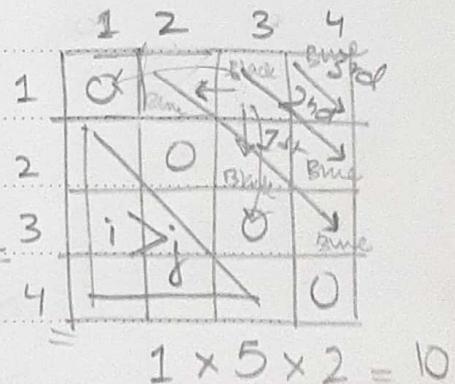
$$M_i = m_{i-1} \times m_i$$

$$m_0 \quad m_1 \quad m_2 \quad m_3 \quad m_4$$

$$1 \quad 5 \quad 2 \quad 3 \quad 9$$

First diagonal:

$$\frac{i}{(1,2)} \quad \frac{k}{1} \quad \frac{k+1}{2} \quad \frac{c(i,k)}{m_{i-1}} \quad \frac{c(k+1,j)}{m_k} \quad m_j$$



$$(2,3) \quad 2 \quad 3 \quad (2,2) \quad (3,3) \quad m_1 \quad m_2 \quad m_3$$

$$(3,4) \quad 3 \quad 4 \quad (3,3) \quad (4,4) \quad m_2 \quad m_3 \quad m_4$$

$$(1,3) \quad 1 \quad 2 \quad (1,1) \quad (2,3) \quad m_0 \times m_1 \times m_3$$

$$(1,3) \quad 2 \quad 3 \quad (1,2) \quad (3,3) \quad m_0 \times m_2 \times m_3$$

AU
 $i \leq k < j \quad i < k \neq j$

$m_0 \quad m_1 \quad m_2 \quad m_3 \quad n_1$
1 5 2 3 9

i	j	$i-1$	k	$k+1$	$\frac{m_{i-1} \times m_k \times m_j}{m_0 \times m_1 \times m_2}$	$c(i,k)$	$c(k+1,j) \times$
1	2	0	1	2	$\frac{1 \times 5 \times 2}{1 \times 5 \times 2} = 10$	$c(1,1) = 0$	$c(2,2) = 10$

2	3	1	2	3	$m_1 \times m_2 \times m_3 = 5 \times 2 \times 3 = 30$	$c(2,2) = 0$	$c(3,3) = 0$
3	4	2	3	4	$m_2 \times m_3 \times m_4 = 2 \times 3 \times 9 = 54$	$c(3,3) = 0$	$c(4,4) = 54$

1	3	0	1	2	$m_0 \times m_1 \times m_3 = 1 \times 5 \times 3 = 15$	$c(1,1) = 0$	$c(2,3) = 30$
					$= 15 + 30 = 45$		16

1	3	0	2	3	$m_0 \times m_2 \times m_3 = 1 \times 2 \times 3 = 6$	$c(1,2) = 10$	$c(3,3) = 0$
					$- 6 + 10 = 16$	✓	

2	4	1	2	3	$m_1 \times m_2 \times m_4 = 5 \times 2 \times 9 = 90$	$c(1,2) = 10$	$c(3,4) = 54$
					$= 90 + 10 + 54 = 154$		151

2	4	1	3	4	$m_1 \times m_3 \times m_4 = 5 \times 3 \times 9 = 135$	$c(1,3) = 16$	$c(4,4) = 0$
					$= 135 + 16 = 151$	✓	

<u>i</u>	<u>j</u>	<u>i-1</u>	<u>k</u>	<u>k+1</u>	<u>$m_{i-1} \times m_k \times m_j$</u>	<u>$c(i,k)$</u>	<u>$c(k+1,j)$</u>	<u>Minimum</u>
1	4	0	1	2	$m_0 \times m_1 \times m_4$	$c(1,1)$	$c(2,4)$	
					$1 \times 5 \times 9$	0	151	
					$= 45 + 151$			
1	4	0	2	3	$m_0 \times m_2 \times m_4$	$c(1,2)$	$c(3,4)$	43
					$1 \times 2 \times 9$	10	54	
					$= 18 + 10 + 54 =$			
1	4	0	3	4	$m_0 \times m_3 \times m_4$	$c(1,3)$	$c(4,4)$	
					$1 \times 3 \times 9$	16	0	
					$= 27 + 16 = 43$	✓		

	1	2	3	4
1	0	10	16	43
2	x	0	30	151
3	x	x	0	54
4	x	x	x	0

Answer is 10

Q2

ANSWER: 5

WORKING:

Input:

$s_1 = \text{INTENTION} = a$, $s_2 = \text{EXECUTION} = b$

$a_1 \dots a_i \dots a_m$

$b_1 \dots b_j \dots b_n$

Cases:

~~Match~~

~~Mismatch~~

$ED(0, j) = j$ $|a| = 0$ and only b has letters so add all b letters
delete

$ED(i, 0) = i$ $|b| = 0$ and only a has letters so add all a letters
delete

$ED(i, j) =$ match ($a_i = b_j$) — $ED(i-1, j-1)$

mismatch ($a_i \neq b_j$) — $\min \left\{ \begin{array}{l} 1 + ED(i, j-1) \\ 1 + ED(i-1, j) \\ 1 + ED(i-1, j-1) \end{array} \right\}$

O E X E C U T I O N

0 0 1 2 3 4 5 6 7 8 9

1 0 1 1 2 3 4 5 6 6 7 8

N 2 2 2 3 4 5 6 7 7 7

T 3 3 3 3 4 5 5 5 6 7 8

E 4 3 4 4 5 6 7 8 9

N 5 4 4 4 4 6 7 8 8

T 6 5 5 5 5 5 6 7 8

I 7 6 6 6 6 6 6 6 6

O 8 7 7 7 7 7 7 6 6

N 9 8 8 8 8 8 7 6 5

I N T E N - T I O N

E X - E C U T I O N

ROUGH WORK SHOWN FOR PROOF OF CONCEPT : (on next page ↗)

$$ED(0, j) = j$$

$$a_1 \dots a_i \dots a_m$$

$$ED(i, 0) = i$$

$$b_1 \dots b_j \dots b_n$$

$$ED(i, j) = \boxed{ED(i-1, j-1) \quad a_i == b_j}$$

$$ED(i, j) = \min \left\{ \begin{array}{l} 1 + ED(i, j-1) \text{ insert } b_j \text{ into } A \\ 1 + ED(i-1, j) \text{ remove } a_i \text{ from } A \\ 1 + ED(i-1, j-1) \text{ replace } a_i \text{ with } b_j \end{array} \right.$$

a = INTENTION

b = EXECUTION

O	O	E	X	E	C	U	T	I	O	N
O	(0)	1	2	3	4	5	6	7	8	9
I	1	1	2	3	4	5	6	7	8	9
N	2	2	2	3	4	5	6	7	7	7
T	3	3	3	3	4	5	5	6	7	8
E	4	3	3	3	4	4	5	6	7	8
N	5	5	4	4	4	5	5	6	7	8
T	6	5	5	5	5	5	5	6	7	8
I	7	6	6	6	6	6	6	5	6	6
O	8	7	7	7	7	7	7	7	6	6
N	9	8	8	8	8	8	8	7	6	5

I	N	T	E	A	-	T	I	O	N
E	X	-	E	C	U	T	I	O	N



Q4

Input: $x = \langle x_1, \dots, x_m \rangle$

$y = \langle y_1, \dots, y_n \rangle$

Cases:

$\text{SCS}(0, j) = j$ { Nothing in x so SCS is all elements of y)

$\text{SCS}(i, 0) = i$ { Nothing in y so SCS is all elements of x)

$\text{SCS}(i, j) = \boxed{\text{if } x_i == y_j \text{ then } 1 + \text{SCS}(i-1, j-1)}$

$\boxed{\text{else if } x_i != y_j \text{ then min} \{ \text{SCS}(i-1, j), \text{SCS}(i, j-1) \}}$

Remove x_i
check $x_{i-1} = y_j ?$

Remove y_j
check $x_i = y_{j-1} ?$

Traverse in row-wise fashion:

	i-1, j-1	i-1, j
i, j-1		i, j

~~Algorithm: SCSFinding (int[][] SCS, int m, int n) {~~

~~SCS = int[m][n] ;~~
~~for i = 0 to m~~

{ for j = 0 to n

{

 if (i == 0) then $\text{SCS}[i][j] = j$;

 else if (j == 0) then $\text{SCS}[i][j] = i$;

 else if ($x[i] == y[j]$) then $\text{SCS}[i][j] = 1 + \text{SCS}[i-1][j-1]$;

 else $\text{SCS}[i][j] = 1 + \min \{ \text{SCS}[i-1][j], \text{SCS}[i][j-1] \}$;

}

}



E

EI

return SCS [m] [n];

{

Time complexity = $O(m \times n)$

↳ Two for-loops nested; one runs till m , the other till n where m is length of x and n is length of y .

60
98
100
191
2990
10

Space complexity: $O(m \times n)$

↳ We are maintaining a $(M+1) \times (N+1)$ matrix for traversal.

NT

WORKING (AS PROOF OF CONCEPT): shown on next pages ↗(th)
(th)(h)
(h)

$$x = \langle x_1, \dots, x_m \rangle$$

$$y = \langle y_1, \dots, y_n \rangle$$

Time complexity =

Space complexity = $O(m \times n)$

$$x = \langle 1, 2, 2, 3 \rangle$$

$$y = \langle 2, 3, 2 \rangle$$

$$z = SCS = \langle 1, 2, 2, 3, 2 \rangle$$

Cases: $\text{scs}(0, j) = j$ ($a_i = \text{zero length}$) ($b_j = j \text{ length}$)

$$SCS(i, 0) = i \quad (b_j = zero\ length) \\ (a_i = i\ length)$$

$$= 1 + \text{scs}(i-1, j-1) \quad a_i == b_j$$

$$SCS(i, j) = \begin{cases} SCS(i-1, j) & \text{Remove } a_i \\ & \text{check } a_{i-1} = b_j \\ 1 + \min \left\{ \begin{array}{l} SCS(i, j-1) \\ \text{Remove } b_j \\ \text{check } a_i = b_{j-1} \end{array} \right\} & \text{priority} \end{cases}$$

$i-1, j-1$	$i-1, j$
$i, j-1$	i, j

A	B	B	C
---	---	---	---

a

B	E	B
---	---	---

i	j	a_i	b_j	$c(i, j-1)$	$c(i+1, j)$	$\frac{c(i, j)}{c(i+1, j)}$	1 + whatever
1	1	A	B	$c(1, 0)$	$c(0, 1)$	$\frac{c(1, 0)}{c(0, 1)}$	2
1	2	A	C	$c(1, 1)$	$c(0, 2)$	$c(0, 1)$	2
1	3	A	B	$c(1, 2)$	$c(0, 3)$	$c(0, 2)$	2
2	1	B	B			$c(1, 0)$	2
2	2	B	C	$c(2, 1)$	$c(1, 2)$	$c(1, 1)$	3
2	3	B	B	$c(2, 2)$	$c(1, 3)$	$c(1, 2)$	3
3	1	B	B	$c(3, 0)$	$c(2, 1)$	$c(2, 0)$	3
3	2	B	C	$c(3, 1)$	$c(2, 2)$	$c(2, 1)$	4
3	3	B	B	$c(3, 2)$	$c(2, 3)$	$c(2, 2)$	4
4	1	C	B	$c(4, 0)$	$c(3, 1)$		4
4	2	C	C	$c(4, 1)$	$c(3, 2)$	$c(3, 1)$	4
4	3	C	B	$c(4, 2)$	$c(3, 3)$		5

saying.....
of being.....

	B	C	B	B
0	0	number	number	number
A	1	2	2	2
B	2	2	3	3
B	3	3	4	4
C	4	4	4	5

Q6: Input: $S = \langle s_1, \dots, s_n \rangle$

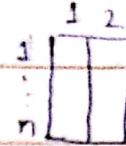
Cases:

- j s.t. $1 \leq j < i$ so $s_1 \dots s_j \dots s_i \dots s_n$
- $\text{LIS}[i] = 1$ for $\forall i$ and $|\text{LIS}| = n$
- $\text{LIS}[1] = s_1$ (if only one element makes up sequence, then it is SCS)
- $\text{LIS}[i] = \forall j$ s.t. $s_i \leq s_j$ — $\text{LIS}[i]$.
 $1 \leq j < i$

$$s_i > s_j \rightarrow \max \left\{ \begin{array}{l} \text{LIS}[i] \\ 1 + \text{LIS}[j]. \end{array} \right.$$

Algorithm :

```
function LIS ( s[1...n] ) {  
    LIS = int[n][2]; // array of nx2  
    for i=1 to n // initialize all to 1.  
    { LIS[i][1] = 1; }  
    LIS[1][1] = s1;  
  
    for i=2 to n  
    {  
        for j=1 to i  
        {  
            if ( si > sj )  
            {  
                if ( LIS[i][1] > 1 + LIS[j][1] )  
                { LIS[i][1] = LIS[j][1];  
                    LIS[i][2] = j; }  
                }  
            else  
            { LIS[i][1] = LIS[j][1] + 1;  
                LIS[i][2] = j; }  
            }  
        }  
    }  
    LIS[i][1] = LIS[i][1];  
    LIS[i][2] = LIS[1][1];  
    }  
}
```



```

int n = LIS[0];
int i = n;
while (i > 0) // Backtracking to get subsequence printed
{
    LIS[i][1].print();
    i = LIS[i][2];
}
// end

```

Time complexity = $O(n \times n)$

Two for loops with one running to n and one running to i . Since max value of i is n so sum of form $\sum_{i=1}^n (n+1)$ which is $O(n^2)$.

Space complexity: $O(n)$

LIS array recorded for traversal which is size $n \times 2$ which is $O(n)$.

Working to show proof of concept:



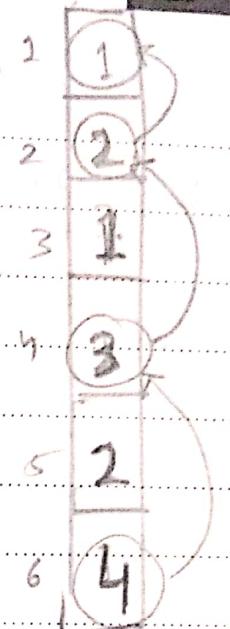
1	2	3	4	5	6	7	8
30	22	9	33	21	50	41	60

i j $j < i$ $i = 0$ then 0
 $i = 1$ then $a[1]$
 $j =$ then 1.

$\max(LIS[i], LIS[i+1]) \text{ or } LIS[i]$

<u>i</u>	<u>j</u>	<u>$LIS[i]$</u>	<u>a_i</u>	<u>a_j</u>	<u>$LIS[j] + 1$</u>	
2	1	$LIS[2]$	22	10	$LIS[1] + 1$	2
3	1	$LIS[3]$	9	10		1
3	2	$LIS[3]$	9	22		1
4	1	$LIS[4]$	33	10	$LIS[1] + 1$	2
4	2	$LIS[4]$	33	22	$LIS[2] + 1$ $2+1=3$ ✓	3
4	3	$LIS[4]$	33	9	$LIS[3] + 1$ $1+1=2$ ✓	2
5	1	$LIS[5]$	21	10	$LIS[1] + 1$ $1+1=2$	2
5	2	"	21	22		1
5	3	"	21	9	$LIS[3] + 1$ $1+1=2$	2
5	4	"	21	33	L	1
6	1	$LIS[6]$	50	10	$LIS[1] + 1$ $1+1$	2
6	2	"	50	22	$LIS[2] + 1$ $2+1$	3
6	3	"	50	9	$LIS[3] + 1$ $1+1$	2

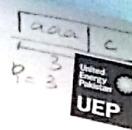
	10	22	9	33	21	50	41	60	80	8
①	1									
②	2									
③	1	1								
④	33	2	3	2						
⑤	21	2	1	2	1					
⑥	50	2	3	2	3	3				
⑦	41									
⑧	60									
⑨	80									
i										



$\frac{i}{6}$	$\frac{j}{4}$	$LIS[i]$	a_i	a_j	$LIS[j] + 1$	max or $LIS[i]$
6	4	8	50	33	$LIS[4] + 1$ 3+1	4
6	5	1	50	21	$LIS[5] + 1$ 2+1	3

Time = $O(n^2)$
Space = $O(n)$.

Q3 : Input : $s = \langle s_1, \dots, s_i, \dots, s_j, \dots, s_n \rangle$
 $i \leq j$



Cases :

$n.length = 0$ so string is null so no palindrome.

$n.length = 1$ so only one element in sequence so palindrome of size 1.

P (2, 6)
 $\begin{matrix} 2 & 3 & 4 & 2 & 1 \end{matrix}$

$$PS(i, j) = \boxed{j=i-1}$$

$$s_i == s_j \quad \boxed{2} \quad j = i+1.$$

match

$$2 + PS(i+1, j-1)$$

include and remove.

$$PS(i, j-1)$$

remove s_j

$$s_i \neq s_j - \text{max}$$

not match

$$PS(i, j)$$

remove s_i

Traverse as diagonal like in
Chain Matrix Multiplication.

$i, j-1$	i, j
$i+1, j-1$	i, j

```

FindLPS ( s[1...n] ) {
    if ( n == 0 || n == 1 ) // string is empty or has only one
                           // element
        return n;
    PS = mat[n][n]; // matrix of size nxn.
    for i = 1 to n
    {   PS[i][i] = 1; } // all initialized to 1 where j=i
    int incrementor = 1;
    for while ( incrementor < n )
    {
        for i = 1 to n
        {
            j = i + incrementor;
            if ( a[i] == a[j] )
            {
                if ( j == i+1 ) then PS[i][j] = 2;
                else PS[i][j] = 2 + PS[i+1][j-1];
            }
            else
                PS[i][j] = max( PS[i][j-1], PS[i-1][j] );
        }
        incrementor++;
    }
    return PS[1][n];
} //end.

```

Time complexity = $O(n^2)$

↳ Two loops iterating over $n \times n$ matrix total so $O(n^2)$

Space complexity = $O(n^2)$ → cause $n \times n$ matrix made to keep track.

WORKING (as proof of concept, shown on next pages ↗):

Page # 1/2

Marks _____ Signature of Instructor _____

Page

LONGEST PALINDROMIC SUBSEQUENCE

Brute force = $O(2^n)$

No. of subproblems = $O(n^2)$

Space complexity = $O(n^2)$

Input = $a_1 \dots a_i \dots a_j \dots a_n$

$i \leq j$

Cases:

$n = 0$

a length is 0 so no palindrome

$n = 1$

so return 1. [only one element
so it is palindrome]

$$PS(i, i) = 1$$

$$PS(i, j) = \begin{cases} 2 & a_i == a_j \\ 2 + PS(i+1, j-1) & \text{same} \\ \max(PS(i, j-1), PS(i+1, j)) & a_i \neq a_j \\ \text{not same} \end{cases}$$



UEP

Bottom-up:

if ($n.length == 0$) return 0;

if ($n.length == 1$) return 1;

for $i = 1$ to n

{ $PS[i, i] = 1$; }

$i, j-1$	i, j
$i+1, j-1$	$i-1, j$

1, 2

2, 3

3, 4

4, 5

5, 6

int incrementor = 1;

while (incrementor < n)

{

for $i = 1$ to n

{ $j = i + incrementor$;

if ($a_i == a_j$)

{ if ($j = i+1$) $PS[i][j] = 2$;

else $PS[i][j] = PS[i+1][j-1] + 2$;

}

else

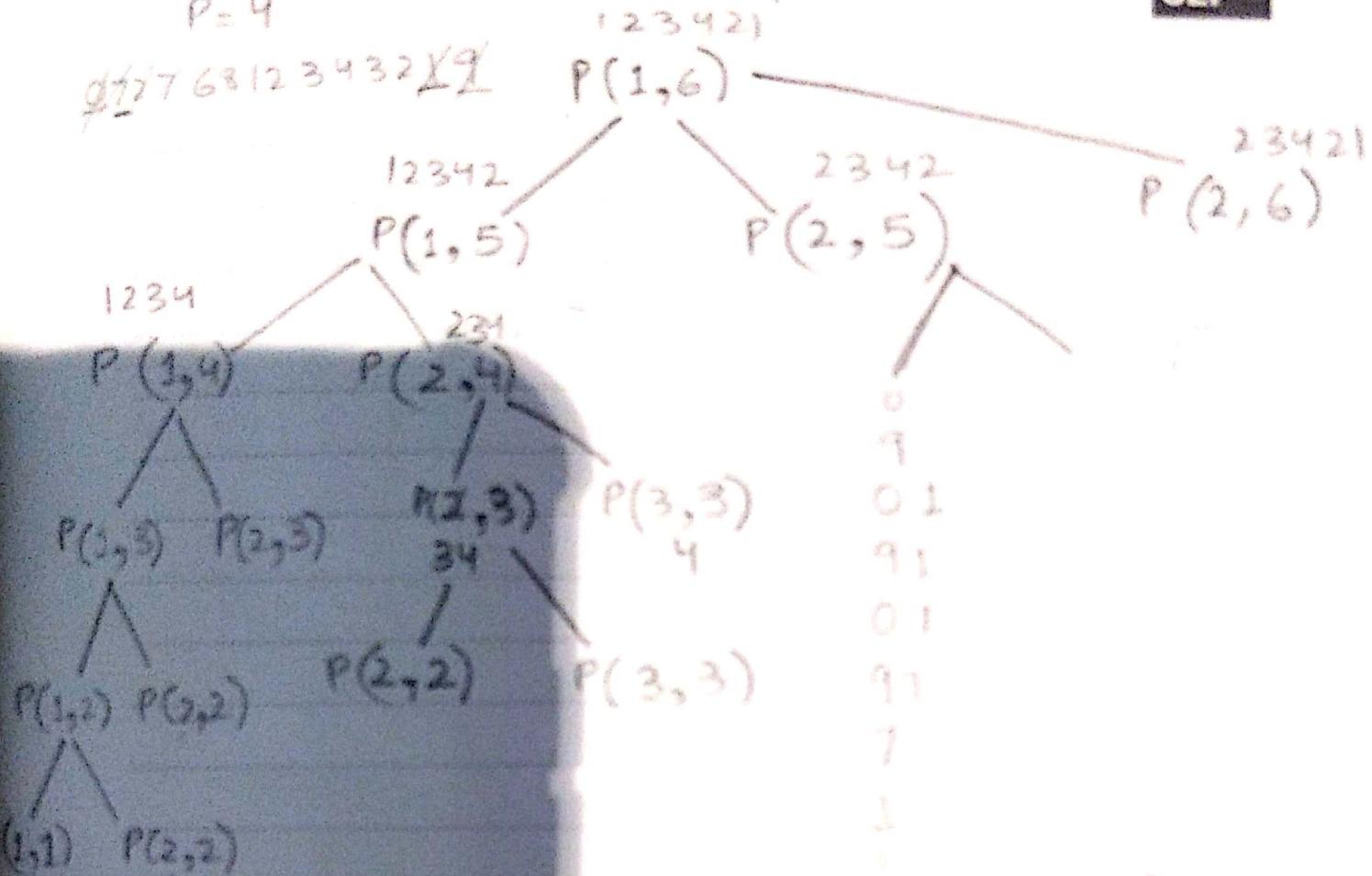
{ $PS[i][j] = \max(PS[i, j-1], PS[i-1, j])$;

aaa a
3

$$\begin{array}{|c|c|} \hline \text{aaa} & b \\ \hline \underbrace{}_3 & \\ \hline \end{array}$$

$P = 3$

A rectangular label with a barcode at the top and the letters "UEP" in large bold letters at the bottom.



- ① Part of longer sequence
 - ② Part of current longest seq.
 - ④ Not part of sequence

(Q8): Input : $A = [a_1, \dots, a_i, \dots, a_n]$.

Cases :

Current element a_i forms part of sum $\Rightarrow M_i + a_i$

Current " " does not form " " " \Rightarrow previous M_i remains max or a_i is negative so we add 0.

So we get :

Max Sum until index i = $MSS_i - 0$

Max Sum overall out of all MSS_i 's calculated so far = $MSS = 0$;

$$MSS_i = \begin{cases} MSS_i + a_i \\ 0 & \text{if } a_i + MSS_i < 0 \text{ (negative)} \end{cases}$$

$$MSS = \max \left\{ \begin{array}{l} MSS_i \\ MSS \text{ (current value)} \end{array} \right\}$$

Algorithm :

Find $MSS (a_1, \dots, a_n) \{$

int $MSS, MSS_i = 0;$

for $i = 1$ to n

{ if ($MSS_i + a_i \geq 0$) then $MSS_i = MSS_i + a_i$;

else $MSS_i = 0$;

$MSS = \max (MSS, MSS_i);$

}

return MSS ; // question says to just give sum.

}

Time complexity = $O(n)$

↳ only one for-loop running from 1 to n.

Space complexity = $O(1)$

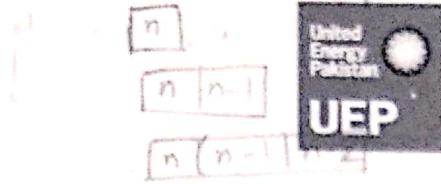
↳ Only 2 constant variables required; MSS & MSS; so constant space complexity.

Working shown as proof of concept: (Shown on next pages)

Input: $a_1 \dots a_k \dots a_n = a[]$

Time & space
complexities

Brute force: $O(n^2)$



$a_1 \dots a_j \dots a_i \dots a_n$

$\text{MSS}[1] = a_1$

$\text{MSS}[i] = a_{i+1} \dots a_n$ where $1 \leq i \leq n-1$

$\rightarrow \text{MSS}[i] < 0 = a_i$

A	-2	-3	4	-1	-2	1	5	-3
	2	3	4	5	6	7	8	

MSS	-2	-3	1	-1	-2	1	(6)	2
	1	2	3	4	5	6	7	8

$$\text{MSS}[i] - [(a[i] + \text{MSS}[i-1]) > 0 \Rightarrow a[i] + \text{MSS}[i-1]]$$

$a[i]$.

max end	0	1	2	3	4	5	6	7
max start	0	0	4	3	1	2	7	7

$$M_j = \begin{cases} M_j + a_{ij} & \\ 0 & \text{if } M_j + a_{ij} < 0 \end{cases}$$

$$MSS = \max \left[\begin{matrix} MSS \\ M_j \end{matrix} \right]$$

$$\{ M_j = 0, MSS = 0$$

for $i = 1$ to n

{ if $(M_j + a_{ij}) > 0$
 $M_j = M_j + a_{ij};$

else

$M_j = 0;$

$MSS = \max (MSS, M_j);$

}

return $MSS;$

}

$O(n) = \text{Time}$ $O(1) = \text{space}$

(Q7)

Input : $m \times n$ grid matrix.

Idea : Calculate column aggregate sum from column i to j

$\forall j \quad i \leq j \leq n$. Get maximum subarray sum of each aggregate combination and choose max of these to get max aggregate of entire matrix i.e. max subgrid sum. (Note; question asks only for max sum, not subgrid coordinates).

Find MSG ($A [m \times n]$) {

Column array = int [1][m]; // already initialized to zeros.

int MSG = 0;

for $i = 1$ to n // current column

{ for $j = i$ to n // column to be added to this column.

{ for $r = 1$ to m // rows

{ Column Array [r] += M [r][j]; }

MSG = max (MSG, MaxSubArraySum (Column Array)); using Kadane's algorithm Each row takes O(m)

for $k = 1$ to m Column Array [1 ... m] = [0, 0, ..., 0] // reinitialized to zeros for loop.

}

return MSG;

Full Name: _____
Student ERP I/D No: _____
Subject: _____
Quiz #: _____ Date : _____
Class/Section: _____
Teacher's Name: _____

Time complexity: $\max(O(n \times n \times m), O(m \times m \times m))$

```

graph TD
    A["for i = 1 to n  
(no. of columns)"] --> B["for j = 1 to n  
(max. no. of rows)"]
    B --> C["for t = 1 to m  
or Kadane  
(both sum)"]
    C --> D["same  
as previous"]
    D --> E["same  
as previously  
stated."]
    E --> F["Terminating  
column by column to  
0"]

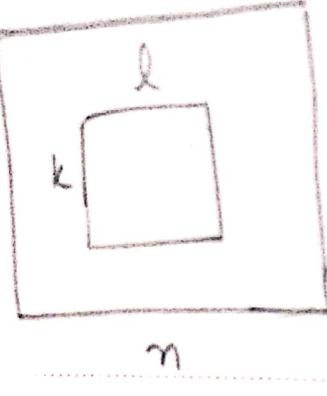
```

The flowchart illustrates the iterative process of finding the maximum sum submatrix. It starts with an outer loop for columns (i), followed by an inner loop for rows (j). Inside this, there is a choice between a standard summation approach (t=1 to m) or Kadane's algorithm for finding the maximum sum subarray within a row. The state of the maximum sum variable is tracked as 'previous' and updated as 'same as previous'. Finally, the process terminates when the column index reaches zero.

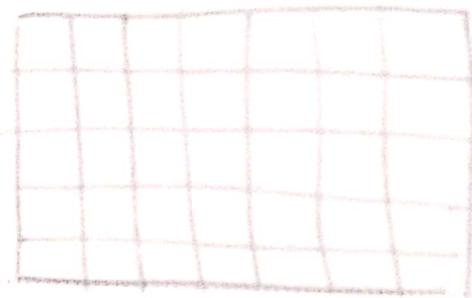
Space complexity: $O(m)$

↳ column Array has size m and MSG has constant size.

Working: (as proof of concept; shown on next pages ↗)



Brute force : $O(N^6)$

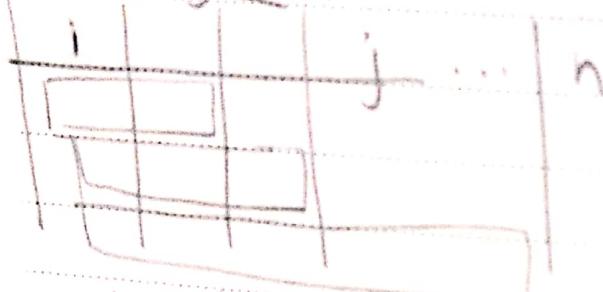


Time = $O(N^3)$

Space = $O(1)$

$$l \leq i \leq j \leq n$$

$$l = i \dots k$$



Array of column sums $[1 \dots m] = [0, 0, \dots, 0]$

for $i = 1$ to n
 { initialize array of columns with column i values.
 { for $j = i$ to n

for $r = 1$ to m

{ Array of column sums $[r] += M[r][j]$ }

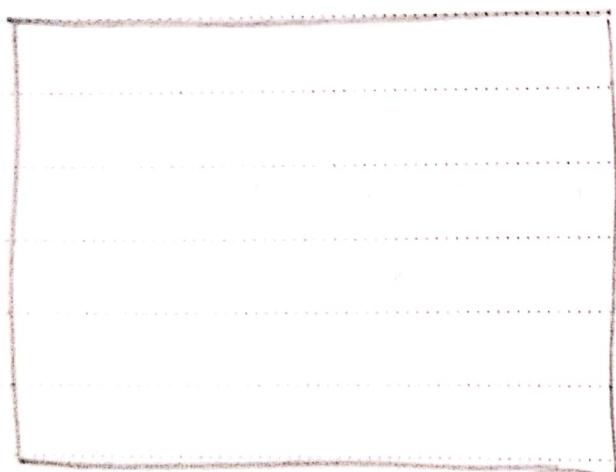
// use Kadane's now

MSG = Kadane^{max(m, n)}(Array of column sums)) ; $\rightarrow O(n)$

0

9	7	8	3	7	34
-6	-2	-1	2	-4	-11
-7	5	5	2	-6	-1
3	2	9	-5	1	20
-1	<u>12</u>	21	12	-2	

$$\begin{array}{ccccc}
 -1 & 11 & 32 & 44 & 42 = 44 \\
 12 & 33 & 45 & 43 & = 45 \\
 21 & 33 & 31 & & = 45 \\
 12 & 10 & & & = 45 \\
 & -2 & & & = 45
 \end{array}$$



20 19 8 42

-1 -12 22
-11 23

34

34 23 22 42 - 42
-11 -12 8
-1 19
20

(Qq)

Input: $\langle s_1, s_2, \dots, s_n \rangle$, t

Cases : $n.length == 0$ so no subset exists

n. `length == 1` : then $s_n = t$ so return True

else $s_n \neq t$ so return False.

current valid

Each s_i either belongs to \mathcal{L} subset and $t = t - s_i$

or does not belong to subset and $t = t$.

or it belongs to another possible valid subset so

This gives:

both cases
possible.

$$\text{TSS}(i, t) = \begin{cases} \text{True } t = 0 \\ \text{TSS}(i-1, t) & s_i > t \\ \text{TSS}(i-1, \cancel{t - s_i}) & \text{keeping } s_i \\ \underline{\text{OR}} & \text{TSS}(i-1, t) & \text{not keeping } s_i \end{cases}$$

Algorithm :

Find TSS ($s_1 \dots s_n$, t) {

 if ~~n~~ $= 0$ return print (No subset possible);

 else if $t = 0$ return True and print empty subset;

 else {

 TSS = ~~bool~~ [n^+] [t^+] ;

 for $i = 0$ to n

 TSS [i] [0] = True ;

 for $s = 0$ to t

 TSS [0] [s] = ~~false~~ ;

 for $i = 1$ to n

 { for $T = 1$ to t

 { if ($s_i > T$) then $TSS[i-1][T] = TSS[i-1][T]$;

 else

$TSS[i][T] = TSS[i-1][T - s_i]$ } } $TSS[i-1][T]$;

 }

 }

// T-F table made .

// (for getting subset).

Algorithm:

```
int i = n, $ = t; // $ = new Stack;
TSS = int[n][t];
while (i ≠ 0 || $ ≠ 0)
{
    if (TSS[i][$] == T && TSS[i-1][$-i] == T)
    {
        $ .push($);
        i = i - 1;
        $ = $ - 1;
    }
    else
        i = i - 1;
}
return $;
```

} End

Time complexity: O(n × t)

→ Traversing next TSS matrix.

Space complexity:

→ making next matrix.

Working (shown on next page) → proof of concept on

Input : $a_1, \dots, a_n \rightarrow$ TargetSum. Find subset

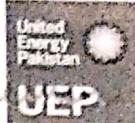
$n.length = 0$

so no subset exists.

$n.length = 1$

$a_1 == \text{Target Sum}$

True



$a[1]$

False

return no subset exists

a_1, \dots, a_n

Target sum, a_1, \dots, a_{n-1}

$a_1, \dots, a_{n-1}, \text{Target Sum} - a_{n-1}$

Target sum, a_1, \dots, a_{n-2}

Target sum, a_1, \dots, a_{n-2}

1	3	9		
2	4	7	4	9
3	5	3	5	5
T	4	2	3	TSS(i, target)

9

1 2 3 4
3, 4, 5, 2

True Target = 0

TSS(i-1, target) $a_i > \text{target}$

0 1 2 3 4

0	0	3	4	5	2
1	T	F	T	T	T
2	F				
3	F				
4	F				
5	F				
6	F				
7	F				
8	F				
9	F				

TSS(i-1, target - sum)

or TSS(i-1, target)

$i \geq a[i]$

$F \leftarrow$

5

	0	1	2	3	4	5	6	7	8	9
0	T	F	F	F	F	F	F	F	F	F
3	T	F	F	T	F	F	F	F	F	F
4	T	F	F	T	T	F	T	F	F	F
5	T	F	F	T	T	T	F	T	F	T
2	T	F	T	T	T	T	T	T	T	T

Diagonal = include

$$3^4 \cdot 2 = 9$$

Up = do not include

$$4^5 = 9$$

$O(\sum^* n)$ = time & space both.

only traverse if tree exists tree at end.

$i = n$, $s = \text{target}$

while ($i \neq 0 \wedge s \neq 0$)

{

if $TSS[i][s] == T$ & $TSS[i-1][s-i] == T$

{ stack.push(a);

$i = i-1$, $s = s-i$

}

else

{ $i = i-1$. }

}

return stack.



(Q5)

DISCLAIMER: This solution has been found after consulting a series of sources including geeksforgeeks, afteracademy.com, tutorialspoint, techiedelight.com, chatGPT, pepcodeacamedy (YouTube channel) and medium.com. However, I do understand the solution and the workings and I have not plagiarized but there will be many similarities between codes found at these sources and my approach just because it is the nature of the problem. In short, please don't think I am a cheater. Thank you.

Input : m , n , s
 \downarrow faces \downarrow dice \downarrow
target sum.

Cases :

$s=0$; if target sum is 0 then just by not throwing any other dice is how we get to the sum and that is the only way so n must be 0 too. \Rightarrow

$n=0$; if no dice are thrown, we cannot reach our sum s so there is no way (return 0) that this happens.

$s < 0$; if sum being calculated is negative value, there is no way to reach it because all dice can hold only positive values from 1 to m so return 0.

$s = s - i$; } Suppose $s > 0$ and $i \neq 0$ then
 $n = n - 1$; we can check what would be the
 results (i.e. could we reach a sum) if this
 n^{th} dice would get each of its 1 to m
 faces. If any of these faces result in it
 getting or not getting the sum, add it to the
 number of ways of reaching the sum at this
 dice. At each throw, s would be reduced
 by the value i between 1 & m , and no. of
 dice be reduced by 1.

To log these results, we will need a matrix of $(l+n) \times (S_l)$ size.

~~D[i][j]~~ $D[n][s]$ corresponds to total ways of reaching the sum s from dice n 's outcomes.

Algorithm: (Using top-down approach)

~~Matrix~~ // matrix of size $(l+n) \times (S_l+1)$ created to log results.

$D = \text{int}[n+1][s+1];$

for $i = 0$ to n

{ for $j = 0$ to S

{ $M[i][j] = -1;$ }

Now we call actual algorithm: DICE($n, m, M[], s$);

DICE (n, m, M[], S)

{

if ($S == 0 \ \& \ n == 0$) then return 1;

if ($n == 0$) then return 0;

if ($S < 0$) then return 0;

if $W[n][S] \neq -1$ then return $W[n][S]$; // $W[n][S]$ already been computed so return.
else $W[n][S] = 0$

for $i = 1$ to m

{

$W[n][S] = W[n][S] + DICE(n-1, m, M[], S-i);$

}

// One die gone and then the value of gone guy subtracted from original S to give new S.

return $W[n][S];$

}

Time complexity: $O(m \times n \times S)$

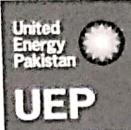
Traverse each element of $(n+1) \times (S+1)$ matrix in $O(n \times S)$ time. In each entry, there traverse a 1 to m for-loop. So it becomes $O(m \times n \times S)$.

Space complexity: $O(n \times S)$

$(n+1) \times (S+1)$ size matrix used to log results.

WORKING: (Thought process shown as proof of concept)

PERFECT SUBSET SUM



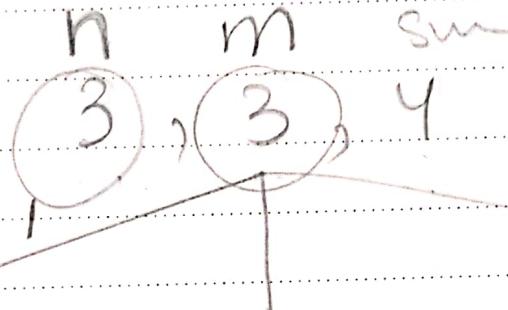
4, 3, 9
n m s

n number of dices
m number of faces
s sum

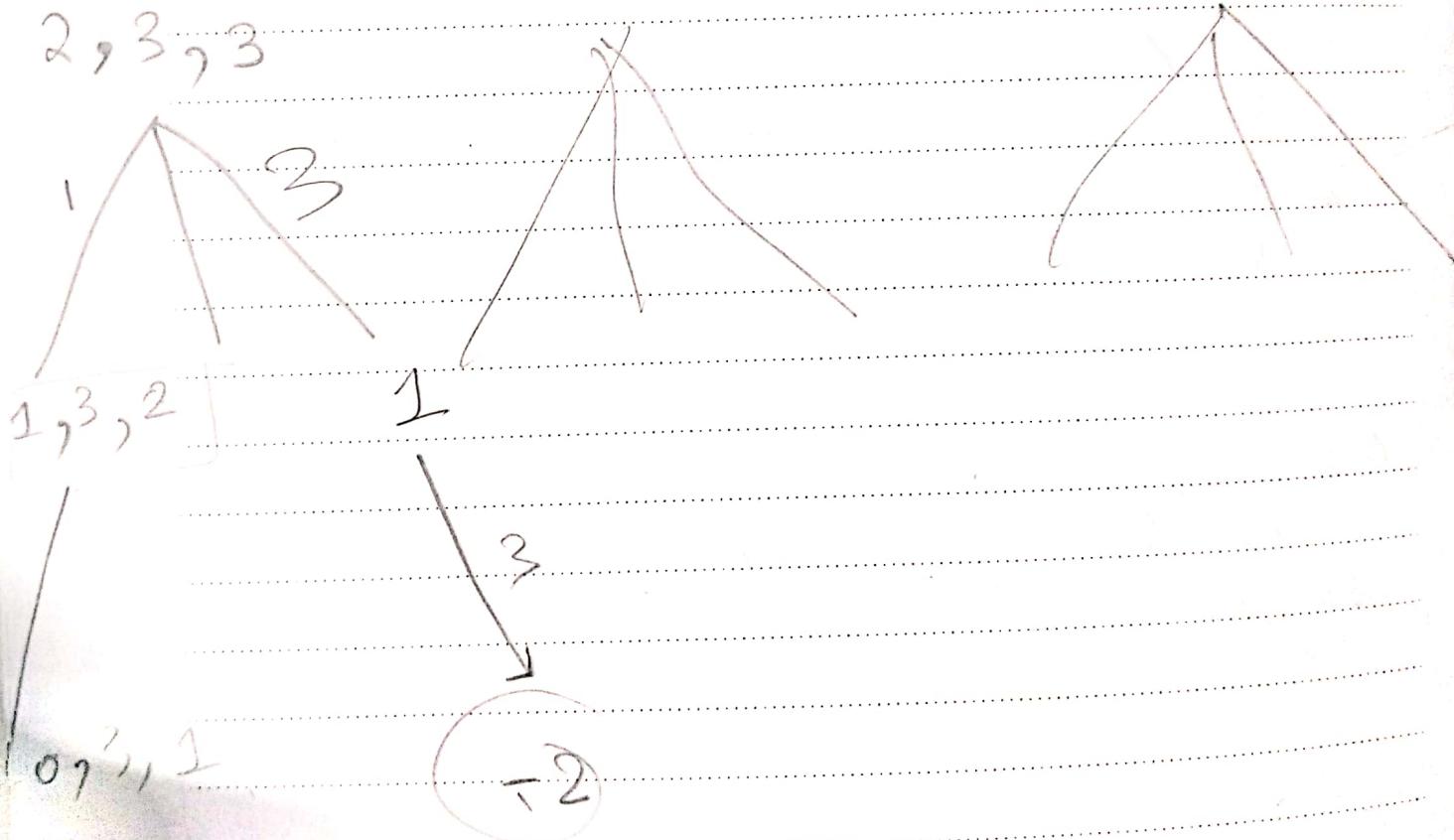
When you roll a die — $d_i < s$ $s = s - d_i, n=n+1$

$d_i > s$ $s = s, n=n+1$

$s = 0$ $n=0$ so start both



2, 3, 3



4, 2

5

3

5m

5m

2 - 7

1m



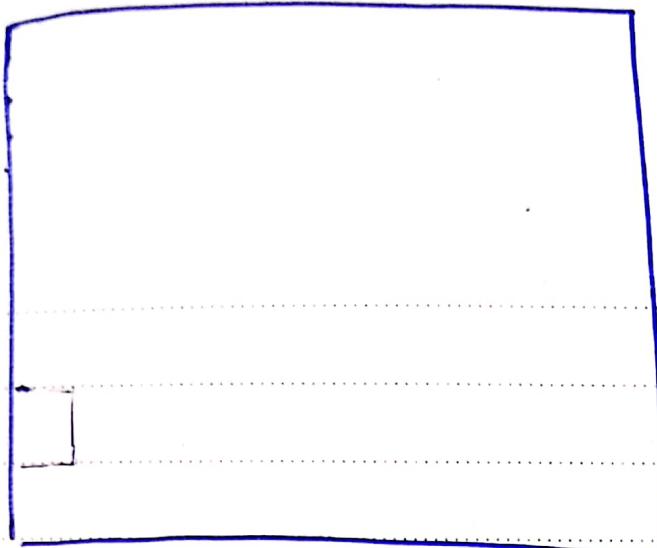
3, 3, 4



Q5:

m faces

n dice



Given sum to reach.

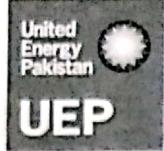
After each throw of a dice, we are left with n -throws throws and at each throw we can get 1 to m number.

Base cases:

- * Sum not reached and no throws left \rightarrow Pass
- * Sum not reached and no throws left \rightarrow Fail
- * Current sum already achieved and throws remain \rightarrow Pass
- * Traverse to find sum- i value and reduce throws by 1.

Algorithm:

Find Dice (n, m, s) {
 DICEFACESUM = $[m \times s]$ matrix
 for



$$\begin{array}{ccccccccc} & 0 & \dots & j & \dots & s \\ \hline 0 & | & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$m \times S$

n, S

if ($s == 0$ || $n == 0$) // target sum = 0 &
// no dice known,
return 1; // then you can achieve
sum in 1 way.
 $M[0][0]$

if ($s < 0$ || $n == 0$) // sum can never be negative
and its not like we can
return 0; know no coins so both
cases mean there is no way
to calculate sum.

$$W = mt[n+1][S+1]$$

for $i = 0$ to n :
for $j = 0$ to S :
 $W[i][j] = -1$

$O(nS)$