

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~~

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

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

$\text{ED}(i, j) =$  match ( $a_i = b_j$ ) —  $\text{ED}(i-1, j-1)$

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

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 5 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	1	(2)	3	4	5	6	7	8	9
T	3	2	1	(3)	4	5	6	7	8	9
E	4	3	2	1	(3)	4	5	6	7	8
N	5	4	3	2	1	(4)	5	6	7	8
T	6	5	4	3	2	1	(5)	6	7	8
I	7	6	5	4	3	2	1	(5)	6	7
O	8	7	6	5	4	3	2	1	(5)	6
N	9	8	7	6	5	4	3	2	1	(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  $x$ )

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

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

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  $y$ .

 $, j) =$  $\begin{bmatrix} \end{bmatrix}$ 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 ↗

60  
98  
100  
191  
2990  
10

(th)  
(th)(th)  
(th)

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

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

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

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

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

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

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

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

B	C	B
---	---	---

a

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	<del>number</del>	<del>number</del>	<del>number</del>
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  $s_i \leq s_j — \text{LIS}[i]$ .  
 $1 \leq j < i$

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

### 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  $\sum_{i=1}^n (n+1)$  exists 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:

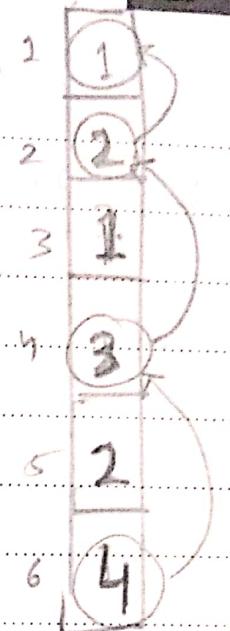


1	2	3	4	5	6	7	8
30	22	9	33	21	50	41	60

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

<u><math>i</math></u>	<u><math>j</math></u>	<u><math>a[i]</math></u>	<u><math>a[j]</math></u>	<u><math>LIS[j] + 1</math></u>	<u><math>\max(LIS[i], LIS[i+1]) \text{ or } LIS[i]</math></u>	
2	1	LIS[2] 1	22	10	LIS[1] + 1 $\checkmark 1 + 1 = 2$	2
3	1	LIS[3] 1	9	10		1
3	2	LIS[3] 1	9	22		1
4	1	LIS[4] 1	33	$>$ 10	LIS[1] + 1 $\checkmark 1 + 1 = 2$	2
4	2	LIS[4] 1	33	$>$ 22	LIS[2] + 1 $2 + 1 = 3 \checkmark$	3
4	3	LIS[4] 1	33	$>$ 9	LIS[3] + 1 $1 + 1 = 2 \checkmark$	2
5	1	LIS[5] 1	21	$>$ 10	LIS[1] + 1 $1 + 1 = 2$	2
5	2	" 1	21	22		1
5	3	" 1	21	$>$ 9	LIS[3] + 1 $1 + 1 = 2$	2
5	4	" 1	21	$<$ 33	L	1
6	1	LIS[6] 1	50	$>$ 10	LIS[1] + 1 $1 + 1$	2
6	2	" 1	50	$>$ 22	LIS[2] + 1 $2 + 1$	3
6	3	" 1	50	$>$ 9	LIS[3] + 1 $1 + 1$	2

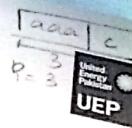
	10	22	9	33	21	50	41	60	80	8
①	1									
②	2									
③	1	1								
④	33	2	3	2						
⑤	21	2	1	2	1					
⑥	50	2	3	2	3	3				
⑦	41									
⑧	60									
⑨	80									
i										



$\frac{i}{6}$	$\frac{j}{4}$	$LIS[i]$	$a_i$	$a_j$	$LIS[j] + 1$	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.

$P(2, 6)$

$$PS(i, j) = \begin{cases} j=i-1 \\ \dots \end{cases}$$

$$s_i == s_j \quad \begin{cases} 2 & j = i+1 \\ \dots \end{cases}$$

match

$$2 + PS(i+1, j-1) \quad \begin{matrix} \text{include and} \\ \text{remove.} \end{matrix}$$

$$PS(i, j-1) \quad \begin{matrix} \text{remove } s_j \\ \dots \end{matrix}$$

$$s_i \neq s_j \quad \dots$$

not match

$$PS(i+1, j) \quad \begin{matrix} \text{remove } s_i \\ \dots \end{matrix}$$

Traverse as diagonal like in  
Chain Matrix Multiplication.

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

```

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

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

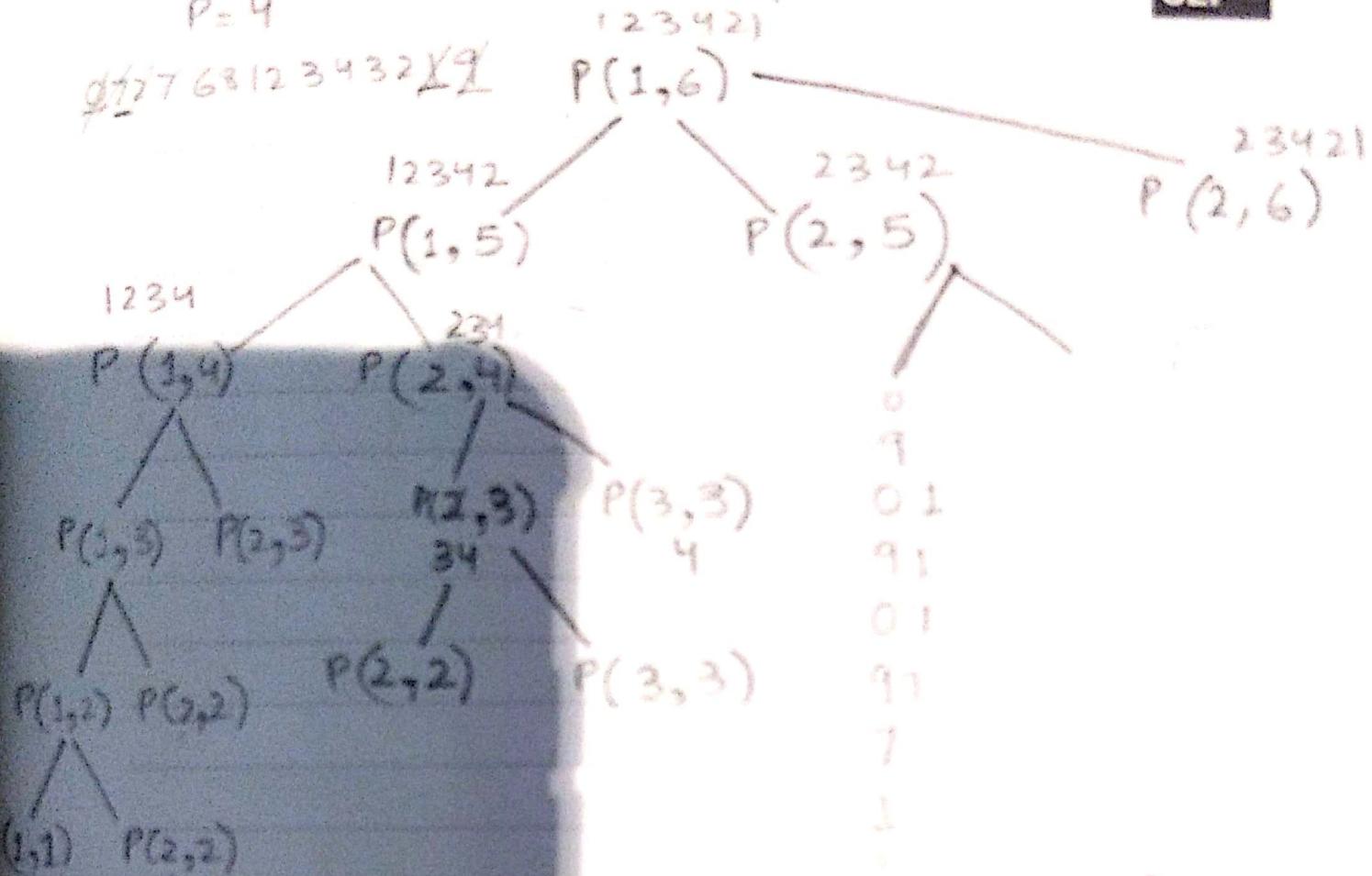


aaa a  
3

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

$P = 3$

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



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

	0	1	2	4	1	3
0	0	1	1	1	1	5
1	1	1	1	3	5	5
2	1	1	1	3	3	1
3	1	1	1	1	1	1
4	1	1	1	1	1	1
5	1	1	1	1	1	1
6	1	1	1	1	1	1

(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 =  $O(1)$

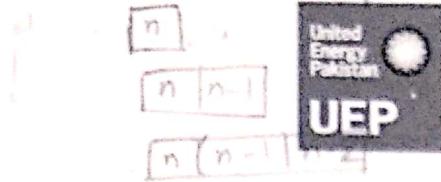
↳ Only 2 constant variables required; MSS & MSS; so constant space complexity.

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  
(min max of n  
times)"]
    B --> C["for t = 1 to m  
or Kadane  
(both sum  
time)"]
    C --> D["same  
as  
previous"]
    D --> E["same  
as previously  
stated."]
    E --> F["Terminating  
columnmax to  
0"]

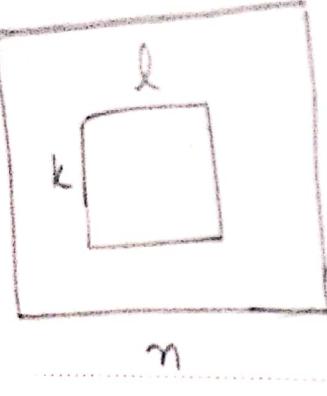
```

The flowchart illustrates the iterative process of finding the column-wise maximum of a 2D matrix. It starts with an outer loop for columns (i from 1 to n). For each column, it performs either Kadane's algorithm or a simple summation (both sum time). The result of the previous column's maximum is used as the starting point for the current column. Finally, the columnmax array is terminated at 0.

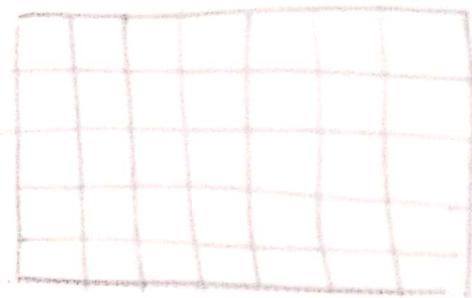
Space complexity:  $O(m)$

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

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



Brute force :  $O(N^6)$

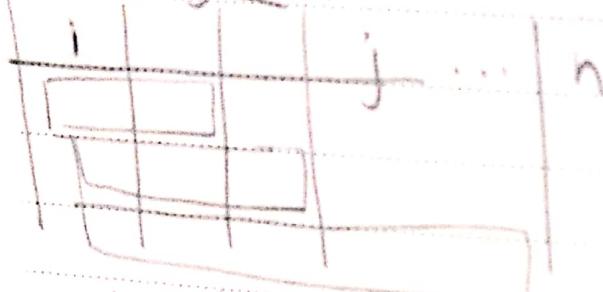


Time =  $O(N^3)$

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<sup>max(m, n)</sup>(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

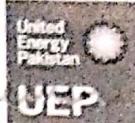
$n.length = 0$

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

{

if  $\text{TSS}[i][s] == T \wedge \text{TSS}[i-1][s-i] == T$

{ stack.push(a);

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

}

else

{  $i = i - 1.$  }

}

return stack.