SEDA KANIK

CSE 321 Homework 5

1) There is 1 function for this problem. This function is optimal_plan and takes two costs array of 2 locations, NY and SF and moving cost M.

This function calculates 2 option and selects one of them at the end.

for first way : lets begin with cost of Month 1 for NY.

for second way : Lets begin with cost of Month 1 for SF.

and among the total months, sums with the current cost and selects minimum cost of previous way or the other way with cost M.

So, one of them is selected, first way or second way, at the end.

This optimal_plan function includes 1 for loop from 0 to length NY, SF at the same time. If we say that length is n, then worst case running time is $\underline{O(n)}$.

Month Locat.	1	2	3	4	5
NY	2	7	3	5	10
SF	9	5	7	3	2

Here an example,

first = 2, second = 9

i = 1

second = 5 + min (9, 2+10) = 14

first = 7 + min (2, 9 + 10) = 9

old_first = 9

old_second = 14

i = 2

second = 7 + min(14, 9+10) = 21

first = 3 + min (9, 14+10) = 12

old_first = 12

old_second = 21

i = 3

second = 3 + min (21, 12+10) = 24

fist = 5 + min (12+21+10) = 17

old_first = 17

old_second = 24

i = 4

second = 2 + min (24, 17+10) = 26

first = 10 + min (17, 24+10) = 27

old_first = 26

old_second = 27

The cost of optimal plan is <u>26</u>.

[SF, SF, SF, SF, SF]

Month Locat.	1	2	3	4
NY	1	3	20	30
SF	50	20	2	4

Here another example;

first = 1 , second = 50

i = 1

first = 3 + min (1, 50+10) = 4

second = 20 + min (50, 1+10) = 31

old_first = 4

old_second = 31

i = 2

first = 20 + min (4, 31+10) = 24

second = 2 + min (31, 4+10) = 16

old_first = 24

old_second = 16

i = 3

first = 30 + min (24, 16+10) = 54

second = 4 + min (16, 24+10) = 20

old_first = 54

old_second = 20

The cost of optimal plan is <u>20</u>.

[NY, NY, SF, SF]

2) There are 2 functions to find the optimal schedule. We want to join as much session as possible.

1th function is called by driver; findOptimalschedule. This function takes start times, finish times as array and finds optimum start times and finish times, keeps into the start-jobsSelected and finish-jobsSelected arrays. This function takes filled array and sort to order respect to their finish time. Firstly, the first times selected, and it will use for comparisons. This algorithm compares start time and finish time of previous sessions. If start time is late, then add the session to the schedule, with start time and finish time of it, and update previous session with this selected session. Over the times (length of the array start or finish times), this process will continue.

We said there were 2 functions at the beginning. The 2nd function is sort-finish-time. This function takes start time and finish time, sorts respect to finish time. Because finish times determine which season you can join.

Start Time	17	16	9	10	16	14	12	16
Finish Time	20	18	11	11	17	15	17	19

* start times and finish times should be from 0 to 24.

sorted by finish time;

Start Time	9	10	14	12	16	16	16	17
Finish Time	11	11	15	17	17	18	19	20

Selected sessions are not colored.
9-11, 14-15, 16-17, 17-20

findOptimalSchedule → includes 1 for loop from 1 to length of times, if we say that times n, O(n).
→ calls sort-finishtime

sort-finishtime → includes 2 for loop nested from 0 to length of finish time, we assumed that as n, $O(n^2)$

TOTAL worst-case complexity is $O(n^2)$.

3) There are 2 functions to find a subset with the total sum of elements equal zero.

The first function is findSubarrays, this function takes an array to find its subarrays and res to keep all the subarrays, sub to keep small elements and odd into res and index. This function works recursively, increases index and pets together elements to more a subarray.

The second function is findSumofSub, this function takes an array which holds subarrays, checks all the elements one by one, if any sum of subarray is equal to zero, then returns this subarray.

Array	-2	4	-7	-5	22	11	1

Array(R)	-7	-5	11	1

→ This array is result. $\underline{-7-5+11+1=0}$

findSubarrays → recursive function, finds all the subarrays → $O(2^n)$

findSumofSub → includes 1 for loop from 0 to length of subarrays (2^n)
$O(2^n)$

Total worst-case complexity is $O(2^n)$

4) There is 1 function to get minimum penalty. The name of this function is getMinimumPenalty. This function takes two string to compare. By the way, the values of mistmach-score and gap-score are assigned with positive values, but multiplied by (-1) to make negative. In this function, hold some values in a matrix array, which 0th row and 0th column has integer values from 0 to sum of lengths of two string. In other lines, compares two string respect all the letters, and assigns the other array respect to indices of letters of two strings. After all these processes, compared two string respect created array, we discuss above. Mis match count and gap count is helped us to create new array which includes gaps. Mis match, match and gap counts are determined by created array. Gaps is determined by '_'.
As we mentioned above, after these, penalties multiplied by (-1), and calculated sum of match count and penalties, returned.

Example 1) Sequence A: ALIGNMENT
Sequence B: SLIME

A	A	L	I	G	N	M	E	N	T
B	S	L	I	-	-	M	E	_	_
	x	✓	✓			✓	✓		

match = 4
mismatch = 1
gap = 4

$(4*2)+(1*-2)+(4*-1) = 2$

Example 2) Sequence A: ABDULLAH
Sequence B: SADULLAH

A	_	A	B	D	U	L	L	A	H
B	S	A	_	D	U	L	L	A	H
		✓		✓	✓	✓	✓	✓	✓

match = 7
mismatch = 0
gap = 2

$(7*2)+(2*-1) = 12$

we assume the length of sequence A is m, the length of sequence B is n.

getMinimumPenalty → includes 1 for loop from 0 to m+n,
2 for loops nested from 1 to m, 1 to n,
while loop from 1 to min length of m or n,
the other loops are smaller than these.

most complex loop is 2 for loops → m*n

The worst case complexity is $\underline{O(m*n)}$

5) There is one function to calculate the sum of the array with the minimum number of operations. We should start by adding the smallest values in array, to be able to perform the addition with minimum number of operation.

For example ;

Assume that, the elements of array are ,

$3 \quad 4 \quad 2 \quad 1 \quad 5$

we should sort in increasing order

$\underbrace{1 \quad 2} \quad 3 \quad 4 \quad 5$

First step $1+2=3$ the sum and operation count is 3

new array :

$\underbrace{3 \quad 3} \quad 4 \quad 5$

Second step $3+3=6$ the sum and operation count is 6

new array :

$\underbrace{4 \quad 5} \quad 6$

third step $4+5=9$ the sum and operation count is 9

new array

$\underbrace{6 \quad 9}$

fourth step $6+9=15$ the sum and operation count is 15

new array

$\underline{15}$ → 15 is sum of elements with $\underbrace{(3+6+9+15)}_{33}$ numbers of operation .

The count of operations hold in an array.

If we made addition in decreasing order :

$\underbrace{5 \quad 4} \quad 3 \quad 2 \quad 1$

First step $5+4=9$ the sum and operation count is 9

$\underbrace{9 \quad 3} \quad 2 \quad 1$

second step $9+3=12$ the sum and operation count is 12

$\underbrace{12 \quad 2} \quad 1$

third step $12+2=14$ the sum and operation count is 14

$\underbrace{14 \quad 1}$

fourth step $14+1=15$ the sum and operation count is 15

$\underline{15}$ → 15 is sum of elements with $\underbrace{(9+12+14+15)}_{50}$ numbers of operations.

As we see above, we must start the smallest numbers in addition to make minimum number of operation. To calculate the number of operation, all the results of addition stored in another array.

sumofarray → recursive function, in every step uses sort() function which has $O(n.\log n)$ complexity. In each step decreases by 2 and increased by 1 the number of elements, in total decreases the number of elements by 1, so complexity is $O(n)$.

Total worst-case complexity is $\underline{O(n^2.\log n)}$