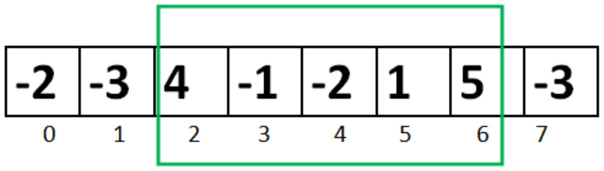
PROBLEM A: MAXIMUM CONTIGUOUS SUBSEQUENCE

TECHNIQUE : ACCUMULATION



4 + (-1) + (-2) + 1 + 5 = 7

Maximum Contiguous Array Sum is 7

INPUT

A sequence of n integers, n ≤ 100000.

OUTPUT

The sum of the maximum subsequence (the contiguous subsequence within the given sequence of numbers which has the largest sum).

ADDITIONAL MATERIAL

- The test-case files, archived as maxsum.zip (downloadable from course's web page)

PRELIMINARY

Write a Python code that reads a sequence of numbers and store them as a list of integers.

STRAIGHTFORWARD SOLUTION

Utilize the following function, which computes the sum of elements in list x from x[i] to x[j], i ≤ j.

```
def Sum(x, i, j):
    s = 0
    for k in range(i,j+1):
        s += x[k]
    return s
```

1) Write a straightforward algorithmic solution (as a Python program) that utilizes the function Sum above to find the sum of the maximum subsequence from a sequence of integers.

   Hint Use index i and j, i <= j, that both run through the sequence. Compute every sum from i to j, and compare each sum against the current maximum value.

2) Insert the follow code over the algorithmic part in your program (excluding the input and output steps). The code will capture the running time of your program between the “start” and “finish” recording.

```
import time

start = time.process_time()

# your code

finish = time.process_time()
print("running time =", finish-start)
```

3) Test your program with the accompanied test cases. Try to analyze how the running time grows as the input size grows. You will see that only small-sized test cases will finish within 1 second.

   Your analysis should turn out that the straightforward solution should run in O(n^3). Why so?

ACCUMULATION TECHNIQUE

4) Transform the input list such that the number at index i of the accumulated list is the accumulation of all numbers in the original list from index 0 up to index i.

   IMPORTANT Accumulation can be that completed in *one loop* of scanning the list

   For example -2 -3 4 -1 -2 1 5 -3

   becomes -2 -5 -1 -2 -4 -3 2 -1

   -5 is -2 + (-3)

   -1 is -2 + (-3) + 4

   -2 is -2 + (-3) + 4 + (-1)

   so on …

5) The accumulation technique will allow computation of Sum in *constant* running time. How? Modify the code of function Sum accordingly.

   Hint The sum from i to j no longer needs to be computed in a loop. It can be calculated directly by utilizing the values in the accumulated list.

6) Rewrite a new solution for this problem, by making use of the accumulated list. This should result in a O(n^2) algorithm. Test the running time’s growth rate in the same way as step 3.

KADANE'S ALGORITHM

7) The following example demonstrate Kadane's algorithm.

	Sum	Max sum
	0	0
−2	max(0+(-2), 0) = 0	0
−2, 1	max(0+1, 0) = 1	1
−2, 1, −3	max(1+(-3), 0) = 0	1
−2, 1, −3, 4	max(0+4, 0) = 4	4
−2, 1, −3, 4, −1	max(4+(-1), 0) = 3	4
−2, 1, −3, 4, −1, 2	max(3+2, 0) = 5	5
−2, 1, −3, 4, −1, 2, 1	max(5+1, 0) = 6	6
−2, 1, −3, 4, −1, 2, 1, −5	max(6+(-5), 0) = 1	6
−2, 1, −3, 4, −1, 2, 1, −5, 4	max(1+4, 0) = 5	6

So the sum of the maximum subsequence is 6.

If you need a good lecture on Kadane's algorithm, here it is :
https://youtu.be/86CQq3pKSUw?t=3m15s

Write the solution in Python that implements the Kadane's algorithm.

8) Test the running time's growth rate of the Kadane's algorithm in the same way as step 3 and 5. With Kadane's algorithm, what do you summarize as the growth of running time ?

9) If you haven't, create an account with acm.timus.ru website. The Judge ID and password will be sent to your email, but occasionally it goes to spam/junk mailbox, so be sure to check there.

10) Attempt this problem: http://acm.timus.ru/problem.aspx?space=1&num=1296
The correct submission will result in an "Accepted" result (in green).