NAME : ASHUTOSH CHAUHAN
UIN : 232009024

CSCE : 629 ANALYSIS OF ALGORITHMS
(Homework - 1.)

Q) Problem : Maximum - Sum Contiguous Subsequence Problem.

Input : A sequence of numbers $S = (S_1, S_2, \dots S_n)$
Output : A contiguous subsequence of S whose sum is maximized.

1) Main idea of Algorithm.

We will break the complete problem into smaller subsequence problem and then solve it. Once the smaller problem is solved we will store it and use it for calulating the bigger problem in steps. This will give us the desired result in $O(n)$ time complexity. In this given problem the idea is to maintain ~~a~~ ~~[illegible]~~ a maximum sum of contiguous subsequence untill a particular index and store it there along with the starting index of the subsequence. Then we will use the solution of this smaller problem (i.e max sum at current index) to calulate.

maxsum of next index. This will give us following expression:-

$$\boxed{DP[idx] = \max(DP[idx-1] + S[idx],\ S[idx])}$$

where $DP[idx]$ = maximum sum of continuous subsequence till index idx

$DP[0] = S[0]$ i.e first element of array.

$S[idx]$ = number of array at index idx.

At each step, we can either start a new ~~st~~ subsequence from current element and store starting index as current element or add it to the existing subsequence. The sum of current subsequence can be calculated as the maximum of the previous sum and the current sum/ ~~as~~ us current element. If the current sum plus the current element is greater than the previous sum, we can start a new subsequence from the current element.

$DP[0] = S[0]$

$DP[1] = max(DP[0] + S[1], S[1])$

$DP[2] = max(DP[1] + S[2], S[2])$

$DP[3] = max(DP[2] + S[3], S[3])$

$\vdots \qquad\qquad \vdots$

$DP[n] = max(DP[n-1] + S[n], S[n])$.

2) **Pseudo Code :-**

```
⇒ recursion (index):
      if index in DP:
            return DP [index] [0]
      if index == 0:
            DP[0] = (S[index], si)
            return DP[0] [0].
      if recursion(index-1) + S [index] < S[inde
            si = index

      DP [index] = (max (recursion(index-1) + S [index
                        , S[index]) , si)
      return DP[index][0]
```

```
main():
    DP = { }
    si = 0
    recursion(len(nums)-1)
    maxsum = -math.inf
    subsequence_idx = (0,0)
    for idx in DP:
        if DP[idx][0] > maxsum:
            maxsum = DP[idx][0]
            subsequence_idx = (DP[idx][1], idx)

    return S[subsequen_idx[0] : subseque_idx[1]+1]
        , maxsum.
```

4.) <u>Time complexity</u>:

$$DP[0] = S[0] \qquad - O(1)$$

$$DP[1] = max(DP[0] + S[1], S[1]) \qquad - O(1).$$

$$DP[2] = max[max(DP[0]+S[1], S[1]) + S[2], S[2]]$$

n

$$\vdots$$

$$DP[n] = max(\qquad) + S(n), S(n) \qquad - O(1)$$

Time complexity of recursion $= O(1) + O(1)$ - - - - n times

$= O(n)$

Also, we iterate over n indexs to calculate maximum sum of conti subsequen and indexes of desired sub-sequence which will take $O(n)$.

$\therefore$ Total time complexity $= O(n) + O(n)$

$= O(n)$.