Maximum Segment Sum

Input: Sequence S of n numbers

Output: Maximum sum in any contiguous subsequence of S.

Example: $S = [2, -3, \underbrace{5, 1, -4, 3, 7}_{12}, -6, -2, 7]$

- If all elements are positive, include everything.
- If all elements are negative, include nothing. (Sum of an empty subsequence is 0.)

Algorithm MaxSeqSum1(S):

```
maxSum ← 0 ..................... O(1)
for a ← 1 to n do ..... ≤n x
  for b ← a to n do ..... ≤n x
    sum ← 0 ..... O(1)
    for i ← a to b do ..... ≤n x
      sum ← sum + S[i] ..... O(1)
    maxSum ← max(maxSum, sum) ... O(1)
```

O(n) for the inner block; O(n³) overall.

- Correct? Y
- Terminates? Y
- Efficient? $O(n^3)$ Can we do better?

Yes, by avoiding recomputation of sums:

$$\sum_{i=a}^{b+1} S[i] = \sum_{i=a}^{b} S[i] + S[b+1]$$

Algorithm MaxSeqSum2(S):

```
maxSum ← 0 ..... O(1)
for a ← 1 to n do ..... ≤ n ×
    sum ← 0 ..... O(1)
    for b ← a to n do ..... ≤ n ×
        sum ← sum + S[b] ..... O(1)
        maxSum ← max(maxSum, sum) ..... O(1)
```

(the inner loop body: $O(n)$; the outer loop body: $O(n^2)$)

- Correct? Yes, by argument above.
- Terminates? Y
- Efficient? $O(n^2)$

Can we do better?

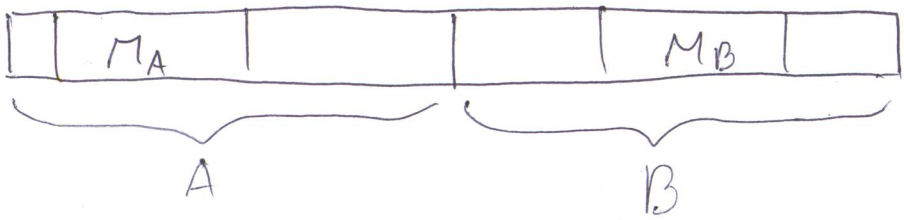
Not by inspecting every contiguous subsequence: there are $\frac{n \times (n-1)}{2} = O(n^2)$ of them.

~~How~~ Let's try divide-and-conquer:

- Divide? $\underbrace{S[1..n/2]}_{A}$, $\underbrace{S[n/2+1, n]}_{B}$

- Merge?

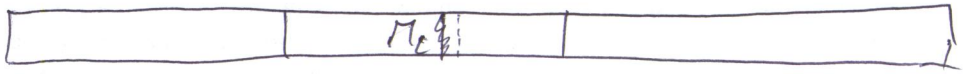
- Merge?



Let M_A be the maximum segment sum in A and M_B the same in B.

Can we just return $\max(M_A, M_B)$?

No! We would miss segments that span the middle. Let M_C be the maximum sum of such a segment:



We return $\max(M_A, M_B, M_C)$.

How do we compute M_C?
It is the combination of the best "left side", starting with $S[n/2]$ and extending to the left, and the best "right side".

Algorithm MaxSegSum3(S):
if $|S|=0$ then return 0
else if $|S|=1$ then return $S[1]$
else
$M_A \leftarrow$ MaxSegSum3($S[1..n/2]$)
$M_B \leftarrow$ MaxSegSum3($S[n/2+1..n]$)
$M_C \leftarrow$ MaxCrossingSegSum(S)
return $\max(M_A, M_B, M_C)$

Algorithm MaxCrossingSeqSum(S):

```
sum ← 0; max_L ← 0
for i ← n/2 downto 1 do
    sum ← sum + S[i]
    max_L ← max(max_L, sum)
sum ← 0; max_R ← 0
for i ← n/2 + 1 to n do
    sum ← sum + S[i]
    max_R ← max(max_R, sum)
return max_L + max_R
```

- Correct? (The segment with maximal sum must be in one of the halves, or cross the middle. Thus, the max of the best of each type is indeed the right answer.) Y
- Terminates? Yes - recursive calls get strictly smaller.
- Efficient?

$$T(n) = \begin{cases} O(1) & \text{if } n \le 1 \\ 2T(n/2) + O(n) & \text{o.w.} \end{cases} \Rightarrow T(n) \text{ is } O(n \log n)$$

(This is the same recurrence as MergeSort)

Can we do better? Yes - $O(n)$ is possible!

(Hint: incrementally compute the max of segments ending at i).

Let E(i) be the maximum value of any segment ending at index i.

The maximum segment must end somewhere so the max of all E(i)'s is the answer.

How can we compute E(i) quickly?
If we know E(i-1): E(i) = max(E(i-1)+S[i], 0)
- either extend the previous best segment
- or start a new, empty one

Algorithm MaxSegmentSum4(S)
```
maxSum ← 0
maxEndingHere ← 0
for i ← 1 to n do
    maxEndingHere ← max(maxEndingHere + S[i], 0)
    maxSum ← max(maxSum, maxEndingHere)
```

- Correct? Y
- Terminates? Y
- Efficient? O(n)

Can we do better?

No! We have to inspect every element. If we miss S[i], that could be the maximum segment all by itself.