Algorithms for the Max Sum Problem

Maximum Sum problem

Given an array (list) of numbers, we can look at sums of segments (subarrays).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
6	-1	2	-4	5	3	-1	2	-6	0	8	12	-4	6	8	4

In the segment above, the sum is

$$(-4) + 5 + 3 + (-1) + 2 + (-6) + 0 + 8 = 7$$

Maximum Sum problem

Given an array (list) of numbers, we can look at sums of segments (subarrays).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
6	-1	2	-4	5	3	-1	2	-6	0	8	12	-4	6	8	4

In the segment above, the sum is

$$(-4) + 5 + 3 + (-1) + 2 + (-6) + 0 + 8 = 7$$

Question: Which segment has the largest sum?

Maximum Sum problem

Given an array (list) of numbers, we can look at sums of segments (subarrays).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
6	-1	2	-4	5	3	-1	2	-6	0	8	12	-4	6	8	4

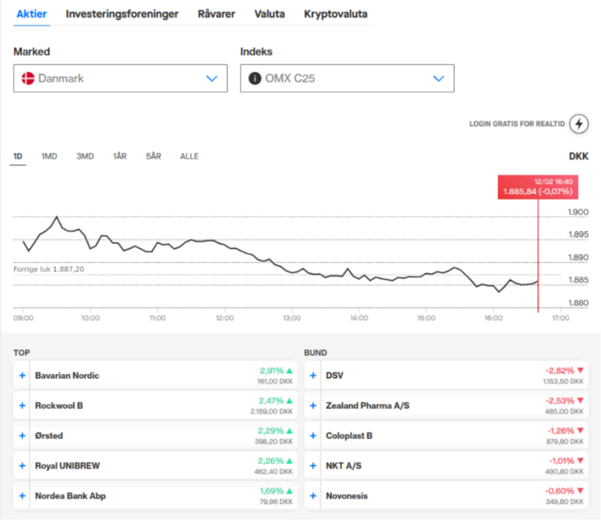
In the segment above, the sum is

$$(-4) + 5 + 3 + (-1) + 2 + (-6) + 0 + 8 = 7$$

Question: Which segment has the largest sum?

A simple and fundamental problem

More motivation for the MaxSum problem: stock analysis



(From www.euroinvestor.dk)

Stock analysis

We have data of the following type:

Stock for Company X:

2023.02.20	2023.02.21	2023.02.22	2023.02.23	2023.02.24	2023.02.25	2023.02.26
$+2\%$	-3%	$+8\%$	-1%	-3%	$+3\%$	$+11\%$

Question: In which period has it been best to own the stock?

Percentage calculation

If 1000 kr. increases by 3% it becomes

Percentage calculation

If 1000 kr. increases by 3% it becomes $1000.1.03 = 1030$ kr.

Percentage calculation

If 1000 kr. increases by 3% it becomes $1000.1.03 = 1030$ kr.

If 1000 kr. falls with 2% it becomes

Percentage calculation

If 1000 kr. increases by 3% it becomes $1000.1.03 = 1030$ kr.

If 1000 kr. falls with 2% it becomes $1000.\ 0.98 = 980$ kr.

Percentage calculation

If 1000 kr. increases by 3% it becomes $1000.1.03 = 1030$ kr.

If 1000 kr. falls with 2% it becomes $1000.\ 0.98 = 980$ kr.

If 1000 kr. first increases by 3% and then falls with 2% will it be

Percentage calculation

If 1000 kr. increases by 3% it becomes 1000.1.03 = 1030 kr.

If 1000 kr. falls with 2% it becomes 1000. 0.98 = 980 kr.

If 1000 kr. first increases by 3% and then falls with 2% will it be

1000.1.03 .0.98 (= 1009.40) kr.

Percentage calculation

If 1000 kr. increases by 3% it becomes 1000.1.03 = 1030 kr.

If 1000 kr. falls with 2% it becomes 1000. 0.98 = 980 kr.

If 1000 kr. first increases by 3% and then falls with 2% will it be

1000.1.03 .0.98 (= 1009.40) kr.

2023.02.20	2023.02.21	2023.02.22	2023.02.23	2023.02.24	2023.02.25	2023.02.26
$+2\%$	-3%	$+8\%$	-1%	-3%	$+3\%$	$+11\%$

During the period above, the stock has changed by a factor of

0.97. 1.08 .0.99 . 0.97 .1.03

Stock analysis

2023.02.20	2023.02.21	2023.02.22	2023.02.23	2023.02.24	2023.02.25	2023.02.26
+2%	−3%	+8%	−1%	−3%	+3%	+11%

Question: In which period has it been best to own the stock?

$\Downarrow$

1.18

1.02	0.97	1.08	0.99	0.97	1.03	1.11

Question: Which segment has the largest product?

From maximum product to maximum sum

Logarithms are growing functions. So

$$0.94.\ 1.05\ .\ 0.99 \leq 0.96\ .\ 1.03\ .\ 1.01$$

if and only if

$$\log\,(0.94.\ 1.05\ .\ 0.99) \leq \log\,(0.96\ .\ 1.03\ .\ 1.01)$$

From maximum product to maximum sum

as input gets larger, output gets larger

Logarithms are growing functions. So

$$0.94 . 1.05 . 0.99 \leq 0.96 . 1.03 . 1.01$$

if and only if

because log is a grown function <= holds. fx 1/x, would flip <=.

$$\log (0.94 . 1.05 . 0.99) \leq \log (0.96 . 1.03 . 1.01)$$

transforms products into sums

As $\log(x . y) = \log(x) + \log(y)$, the above applies if and only if

$$\log(0.94) + \log(1.05) + \log(0.99) \leq \log(0.96) + \log(1.03) + \log(1.01)$$

From maximum product to maximum sum

biggest product

Then the segment that has the largest product in this array:

1.02	0.97	1.08	0.99	0.97	1.03	1.11

yes

is the same as the segment with the largest sum in this array:

log(a * b * c) = log(a) + log(b) + log(c)

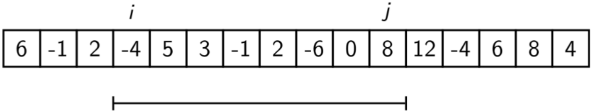
log 1.02	log 0.97	log 1.08	log 0.99	log 0.97	log 1.03	log 1.11

biggest sum

0.0286	-0.0439	0.1110	-0.0145	-0.0439	0.0426	0.1506

Algorithms for MaxSum

We need to find the sum for all segments:



Algorithms for MaxSum

We need to find the sum for all segments:

			i							*j*					
6	-1	2	-4	5	3	-1	2	-6	0	8	12	-4	6	8	4

check all subarrays. Natural algorithm = bruteforce

Natural algorithm based on the definition:

i starting index, j ending index

For all i and for all j ≥i , compute the sum from i to j (inclusive).

			i				*k*			*j*					
6	-1	2	-4	5	3	-1	2	-6	0	8	12	-4	6	8	4

First algorithm for MaxSum

natural algorithm / bruteforce

```
MaxSum1(n)
    maxSoFar = 0
    for i = 0 to n − 1          Θ(n)
        for j = i to n − 1       Θ(n
            sum = 0
            for k = i to j       Θ(n
                sum += A[k]
            maxSoFar = max(maxSoFar, sum);
    return maxSoFar
```

3 nested for loops, so n*n*n = Θ(n^3)

			i				*k*			*j*					
6	-1	2	-4	5	3	-1	2	-6	0	8	12	-4	6	8	4

Correct? Follows from the problem definition.

Running time? $\Theta(n^3)$ using the same argument as Algorithm 3 from the asymptotic analysis examples (they have the exact same structure).

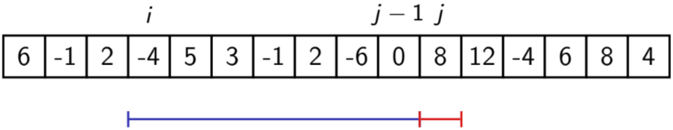
Observation

$$(-4) + 5 + 3 + (-1) + 2 + (-6) + 0 = (-1)$$

$$\Downarrow$$

you dont need to re calculate the sum, when add an element, you can use the sum from before

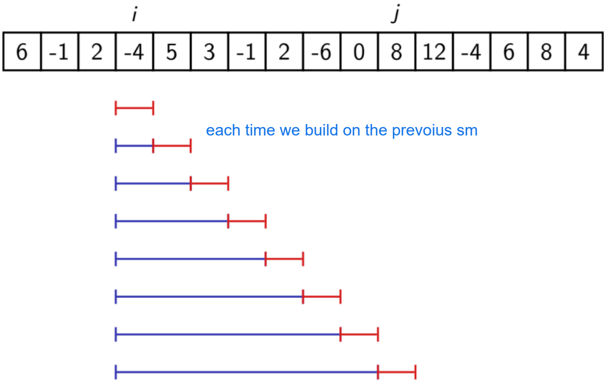
$$(-4) + 5 + 3 + (-1) + 2 + (-6) + 0 + 8 = (-1) + 8 = 7$$



sums of overlapping subarrays can be calculated incrementally by reusing previous sums. This avoids redundant calculations and paves the way for developing more efficient algorithms for the Maximum Subarray Sum problem.

Idea for improved algorithm

Algorithm: For each i, calculate sums for increasing j with one new addition per sum.



Second algorithm for MaxSum

$\textsc{MaxSum2}(n)$

```
maxSoFar = 0
for i = 0 to n − 1            Θ(n)
    sum = 0
    for j = i to n − 1        Θ(n)
        sum += A[j]           This is the part that build on top of the prevoius sum
        maxSoFar = max(maxSoFar, sum);
return maxSoFar
```

two nested for loops that run every time, so n*n = Θ(n^2) (remeber for loops that does not run every time, will not nessecary raise the run time by * n)

Correct? Follows from the definition of the problem and the observation above.

Running time?Θ(n2), with approximately the same argument as for Algorithm 2 from the asymptotic analysis examples.

New observation

$$x_1 \le x_2$$

$$\Updownarrow$$

$$x_1 + 2 \le x_2 + 2$$

New observation

$$x_1 \leq x_2$$

$$\Updownarrow$$

$$x_1 + 2 \leq x_2 + 2$$

This results in:

fx max{2+2,4+2,6+2,8+2} = 8+2 = max{2,4,6,8} = 8+2

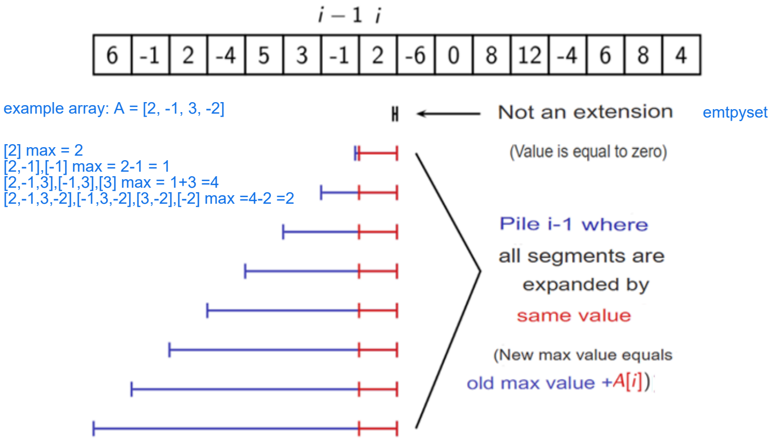
$$\max\{x_1+2\ ,\ x_2+2,\ \ldots,\ x_i+2\} = \max\{x_1, x_2, \ldots, x_i\} + 2$$

it does not matter, when we add the value, before or after finding the maximum value

Idea: Can we look at segments in piles, so that the new pile is the same as the old pile with all segments expanded by the same value?

Idea for improved algorithm

Let pile in be all segments that end at the right edge of A[i].Then the pile **i** is the same as pile **i −1** with all segments extended by the same value, plus the empty segment:



Third algorithm for MaxSum

Kadane's Algorithm

$\textsc{MaxSum3}(n)$

```
MaxSum3(n)
    maxSoFar = 0
    maxEndingHere = 0
    for i = 0 to n − 1
        maxEndingHere = max(maxEndingHere + A[i], 0)
        maxSoFar = max(maxSoFar, maxEndingHere);
    return maxSoFar
```

max for the given subarray

compare max at the subarray with max of the prevoius saved subarray.
And only compare the biggest subarray with the new subarray

Correct? Follows from the definition of the problem and the new observation above, which ensures that we take the maximum over all segments (since every segment is included in a group, namely the group for the i where the segment ends).

Driving time? There are n iterations, each taking Θ(1) time.This gives a total of Θ(n).