Maximum Contiguous Subsequence Sum

Why do we look at this problem?

- It's an interesting problem …
  - … with one obvious (but inefficient) solution
  - and other less obvious but more efficient solutions
- Analyzing the obvious solution is instructive:
  - Accounting for nested loops
  - Evaluating typical sums
  - Using an analogy that makes it easier
- We can make the program more efficient
  - A trivial improvement
  - Another improvement (but is it correct?)

A Nice Algorithm Analysis Example from the Weiss book

- Maximum Contiguous Subsequence Sum algorithms and their analysis.
- Section 5.3
- **Problem**: Given a sequence of numbers, find the maximum sum of a contiguous subsequence.
- Easy cases:
  - What if all numbers in the original sequence are positive?
  - What if they are all negative?
  - The problem is only interesting if there is a mixture of positive and negative numbers.
  - What if we left out “contiguous” from the problem statement?

Maximum Contiguous Subsequence Sum

- **Problem**: Given a sequence of numbers, find the maximum sum of a contiguous subsequence.
- Easy cases:
  - What if all numbers in the original sequence are positive?
  - What if they are all negative?
  - The problem is only interesting if there is a mixture of positive and negative numbers.
  - What if we left out “contiguous” from the problem statement?

Problem definition: details

Problem definition: Given a sequence of n integers $A_1, A_2, \ldots, A_n$, ($n \geq 0$, some numbers may be negative). Find the maximum sum of a consecutive subsequence $S_{i,j} = \sum_{k=i}^{j} A_k$. If i>j or if all of the numbers $A_i, \ldots, A_j$ are negative, we define the sum to be zero. We use the abbreviation $A_{i,j}$ to stand for $A_i, \ldots, A_j$. Note that $S_{i,j}$ is the sum of the numbers in $A_{i,j}$.

Examples (from the DS book):

What is $S_{2,4}$?

{-2, **11**, **-4**, **13**, -5, 2} (maximum subsequence sum is 20),
{1, -3, **4**, **-2**, **-1**, **6**} (maximum subsequence sum is 7).

Can a max-sum subsequence begin with a negative number?

When we store the sequence in a Java array, the subscripts begin with 0.

When we refer to the problem in the abstract here, the subscripts will begin with 1. (because the Weiss DS book does it this way, as do most mathematicians)

A quick-and-dirty algorithm

- Design one right now.
  - Efficiency doesn't matter.
  - It has to be easy to understand.
  - 3 minutes
- For your reference:
  - {5, 6, -3, 2, 8, 4, -12, 7, 2}

First Algorithm

Find the sums of *all* subsequences

```
public final class MaxSubTest {
    private static int seqStart = 0;
    private static int seqEnd = 0;

    /* First maximum contiguous subsequence sum algorithm.
     * seqStart and seqEnd represent the actual best sequence.
     */
    public static int maxSubSum1( int [ ] a ) {
        int maxSum = 0;
          //In the analysis we use "n" as a shorthand for "a.length"
        for( int i = 0; i < a.length; i++ )
            for( int j = i; j < a.length; j++ ) {
                int thisSum = 0;

                for( int k = i; k <= j; k++ )
                    thisSum += a[ k ];

                if( thisSum > maxSum ) {
                    maxSum   = thisSum;
                    seqStart = i;
                    seqEnd   = j;
                }
            }
        return maxSum;
    }
```

i: beginning of subsequence

j: end of subsequence

k: steps through each element of subsequence

Where will this algorithm spend the most time?

How many times (exactly, as a function of N = a.length) will that statement execute?

Analysis of this Algorithm

We need to count the number of ordered triples (i, j, k) with $1 \le i \le k \le j \le n$

Outer numbers could be 0 and n – 1, and we'd still get the same answer.

```
      //In the analysis we use "n" as a shorthand for "a.length "
for( int i = 0; i < a.length; i++ )
    for( int j = i; j < a.length; j++ ) {
        int thisSum = 0;

        for( int k = i; k <= j; k++ )
            thisSum += a[ k ];
```

Counting is hard!

We need to count the number of ordered triples (i, j, k) with $1 \le i \le k \le j \le n$

- **Can we write this as a summation?**

$$\sum_{i=1}^{n}\left(\sum_{j=i}^{n}\left(\sum_{k=i}^{j}1\right)\right)$$

Now let's simplify

Simplify the sum

$$\sum_{i=1}^{n}\left(\sum_{j=i}^{n}\left(\sum_{k=i}^{j}1\right)\right)$$

- **First we do it by hand, for practice with summation manipulation.**

One part of the process will be

$$\sum_{j=1}^{n} j = \sum_{j=1}^{i-1} j + \sum_{j=i}^{n} j$$

We have seen this idea before

Then we can solve for the last term to get a formula that we need on the next slide:

$$\sum_{j=i}^{n} j = \sum_{j=1}^{n} j - \sum_{j=1}^{i-1} j = \frac{n(n+1)}{2} - \frac{(i-1)i}{2}$$

Simplify the sum

$$\sum_{i=1}^{n}\left(\sum_{j=i}^{n}\left(\sum_{k=i}^{j} 1\right)\right)$$

- **Do the first few steps**

For your reference later

$$\sum_{i=1}^{n}\left(\sum_{j=i}^{n}\left(\sum_{k=i}^{j}1\right)\right)=\sum_{i=1}^{n}\left(\sum_{j=i}^{n}(j-i+1)\right)=\sum_{i=1}^{n}\left(\sum_{j=i}^{n}j-\sum_{j=i}^{n}i+\sum_{j=i}^{n}1\right)$$

$$=\sum_{i=1}^{n}\left(\frac{n(n+1)}{2}-\frac{(i-1)i}{2}-i(n-i+1)+(n-i+1)\right)$$

$$=\sum_{i=1}^{n}\left(\frac{n(n+1)}{2}+n+1-i(n+\tfrac{3}{2})+\tfrac{1}{2}i^2\right)=\left(\frac{n(n+1)}{2}+n+1\right)\sum_{i=1}^{n}1-(n+\tfrac{3}{2})\sum_{i=1}^{n}i+\tfrac{1}{2}\sum_{i=1}^{n}i^2$$

$$=\left(\frac{n^2+3n+2}{2}\right)n-(n+\tfrac{3}{2})\frac{n(n+1)}{2}+\tfrac{1}{2}\frac{n(n+1)(2n+1)}{6}$$

Simplify the sum

$$\sum_{i=1}^{n}\left(\sum_{j=i}^{n}\left(\sum_{k=i}^{j}1\right)\right)$$

- **When it gets down to "just Algebra", we rely on an old friend.**

Help from Maple, part 1

Simplifying the last step of the monster sum

```
> simplify((n^2+3*n+2)/2*n
  -(n+3/2)*n*(n+1)/2+1/2*n*(n+1)*(2*n+1)/6);
```

$$\frac{1}{6}n^3+\frac{1}{2}n^2+\frac{1}{3}n$$

```
> factor(%);
```

$$\frac{1}{6}(n+2)\,n\,(n+1)$$

Help from Maple, part 2

Letting Maple do the whole thing for us:

```
sum(sum(sum(1, k=i..j), j=i..n), i=1..n);
```

$$\frac{1}{2}(n+1)\,n^2+2\,(n+1)\,n+\frac{1}{3}n+\frac{5}{6}-\frac{1}{2}n\,(n+1)^2-(n+1)^2$$
$$+\frac{1}{6}(n+1)^3-\frac{1}{2}n^2$$

```
> factor(simplify(%));
```

$$\frac{1}{6}(n+2)\,n\,(n+1)$$

We get same answer if we sum from 0 to n-1, instead of 1 to n

```
factor(simplify(sum(sum(sum(1,k=i..j), j=i..n),
i=1..n)));
```

$$\frac{n(n+2)(n+1)}{6}$$

```
factor(simplify(sum(sum(sum(1,k=i..j),j=i..n-1),
i=0..n-1)));
```

$$\frac{n(n+2)(n+1)}{6}$$

Interlude

- If the automobile had followed the same development cycle as the computer, a Rolls-Royce would today cost $100, get a million miles per gallon, and explode once a year, killing everyone inside.
  - Robert X. Cringely

Tangent: Another (clever) way to count it

- Count what?
  - **the number of ordered triples (i, k, j) with $1 \le i \le k \le j \le n$**
- Sometimes we can count a hard-to-count set by finding a one-to-one correspondence between that set and an easier-to-count set.
  - **What's a one-to-one correspondence?**

The "equivalent count" set

- **We want to count: the number of ordered triples (i, k, j) with $1 \le i \le k \le j \le n$.**
- Suppose we have an urn (why is it always an urn in this kind of math problem?) containing n+2 balls. Choose 3 of them.
  - n white balls contain the numbers 1, 2, ..., n.
  - There is one red ball and one blue ball.
  - If red drawn, = min of other 2
  - If blue drawn, = max of other 2
- Choosing 3 from (n+2) is ${}_{(n+2)}C_3$
- Do they match?

The correspondence

There is a one-to-one correspondence between
triples of balls that we can draw out of the urn
and
triples that satisfy the above inequality.

triple of balls	Corresponding triple of numbers
(i, k, j)	`(i, k, j)`
(red, i, j)	`(i, i, j)`
(blue i, j)	`(i, j, j)`
(red, blue, i)	`(i, i, i)`

Can you see that it is a 1:1 correspondence?

What is the main source of the simple algorithm's inefficiency?

```
//In the analysis we use "n" as a shorthand for "a.length "
for( int i = 0; i < a.length; i++ )
    for( int j = i; j < a.length; j++ ) {
        int thisSum = 0;

        for( int k = i; k <= j; k++ )
            thisSum += a[ k ];
```

Once we see that the performance is bad, we look for ways to improve it.

Eliminate the most obvious inefficiency, get Θ(??)

```
for( int i = 0; i < a.length; i++ ) {
    int thisSum = 0;
    for( int j = i; j < a.length; j++ ) {
        thisSum += a[ j ];

        if( thisSum > maxSum ) {
            maxSum = thisSum;
            seqStart = i;
            seqEnd   = j;
        }
    }
}
```

Can we do even better?