THE MAXIMUM CONTIGUOUS SUBSEQUENCE SUM PROBLEM

- Problem Statement
  - Given (possibly negative) integers A_1, A_2, …, A_N, find (and identify the sequence corresponding to the maximum value of $S_{i,j} = \sum_{k=i}^{j} A_k$.
  - The maximum contiguous subsequence sum is zero if all the integers are negative.
- Examples
  - { 1, 3, -5, **7, 4**, -2} => 11
  - { -2, **11, -4, 13**, -5, 2} => 20
  - { -2, -5, -20, -5, -7} => 0

THE MAXIMUM CONTIGUOUS SUBSEQUENCE SUM PROBLEM

- Solution 1: Brute-force Solution

```
/**
 * Cubic maximum contiguous subsequence sum algorithm.
 * seqStart and seqEnd represent the actual best sequence.
 */
public static int maxSubsequenceSum( int [ ] a )
{
    int maxSum = 0;

    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;
}
```

figure 5.4

A cubic maximum contiguous subsequence sum algorithm

The Maximum Contiguous Subsequence Sum Problem

- Analysis of the Algorithm
  - The number of integer-ordered triplets (i, j, k) that satisfy $1 \le i \le k \le j \le N$ is $N(N+1)(N+2)/6$
    - Proof omitted
  - The above result gives that algorithm is $O(N^3)$
- Observation
  - In the inner most loop we are doing redundant calculations
  - i.e. $\sum_{k=i}^{j} A_k = A_j + \sum_{k=i}^{j-1} A_k$.

The Maximum Contiguous Subsequence Sum Problem

- Solution 2: Improved Version

figure 5.5

A quadratic maximum contiguous subsequence sum algorithm

```java
/**
 * Quadratic maximum contiguous subsequence sum algorithm.
 * seqStart and seqEnd represent the actual best sequence.
 */
public static int maxSubsequenceSum( int [ ] a )
{
    int maxSum = 0;

    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;
            }
        }
    }

    return maxSum;
}
```

THE MAXIMUM CONTIGUOUS SUBSEQUENCE SUM PROBLEM

- Analysis of the Algorithm
  - Complexity is $O(N^2)$ (Similar to Insertion Sort)
- Theorem 5.2:
  - Let $A_{i,j}$ be any subsequence with $S_{i,j} < 0$. If $q > j$, then $A_{i,q}$ is not a maximum contiguous subsequence.

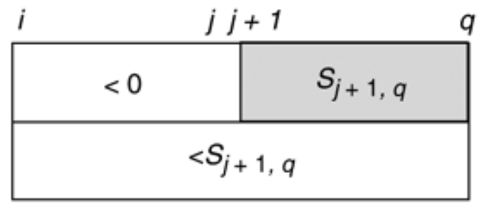

figure 5.6

The subsequences used in Theorem 5.2

THE MAXIMUM CONTIGUOUS SUBSEQUENCE SUM PROBLEM

- Theorem 5.3:
  - For any i, let $A_{i,j}$ be the first sequence with $S_{i,j} < 0$. Then for any $i \le p \le j$ and $p \le q$, $A_{p,q}$ either is not a maximum contiguous subsequence or is equal to an already seen maximum contiguous subsequence.

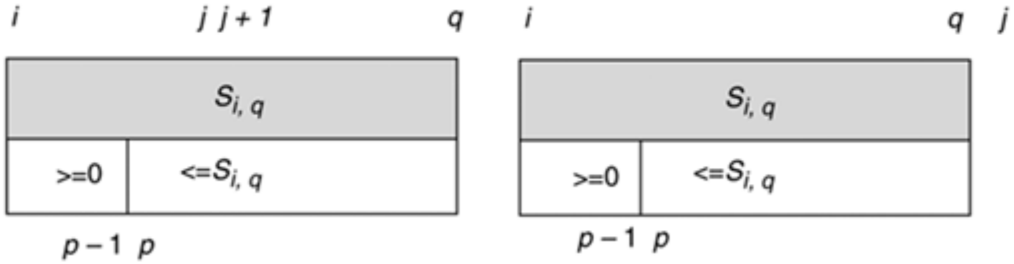

figure 5.7

The subsequences used in Theorem 5.3. The sequence from p to q has a sum that is, at most, that of the subsequence from i to q. On the left-hand side, the sequence from i to q is itself not the maximum (by Theorem 5.2). On the right-hand side, the sequence from i to q has already been seen.

The Maximum Contiguous Subsequence Sum Problem

- Solution 3: Linear Solution
  - Theorem 5.2 tells that if at any point in the inner loop of the second algorithm, ***thisSum*** is less than zero, we can break from the loop. Thus avoid exploring some solutions.
  - Theorem 5.3 tells us that we can advance the starting point of our search to end of that negative subsequence.
  - Combining these two we can derive the linear time algorithm.

THE MAXIMUM CONTIGUOUS SUBSEQUENCE SUM PROBLEM

- Solution 3: Linear Solution

figure 5.8

A linear maximum contiguous subsequence sum algorithm

```java
/**
 * Linear maximum contiguous subsequence sum algorithm.
 * seqStart and seqEnd represent the actual best sequence.
 */
public static int maximumSubsequenceSum( int [ ] a )
{
    int maxSum = 0;
    int thisSum = 0;

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

        if( thisSum > maxSum )
        {
            maxSum = thisSum;
            seqStart = i;
            seqEnd   = j;
        }
        else if( thisSum < 0 )
        {
            i = j + 1;
            thisSum = 0;
        }
    }

    return maxSum;
}
```

The Maximum Contiguous Subsequence Sum Problem

- Lab Tasks:
  - Implement three solutions
  - Test the time takes by three solutions for different input sizes (at least for 10, 100, 1000, 10000).
    - For each input size at least generate two testcases
    - For each testcase, run the program for at least three times and collect the time value and take the average of all the runs.
    - Plot the results (average execution time against the input size) for three solutions in same graph.