Lecture 2: Maximum Contiguous Subarray Problem

**Overview**

- Reference: Chapter 8 in *Programming Pearls, (2nd ed)* by Jon Bentley.

- Clean way to illustrate basic algorithm design
  - A $\Theta(n^3)$ brute force algorithm
  - A $\Theta(n^2)$ algorithm that reuses data.
  - A $\Theta(n \log n)$ divide-and-conquer algorithm

- *Cost* of algorithm will be number of primitive operations, e.g., comparisons and arithmetic operations, that it uses.
Between years 5 and 8 ACME earned
\[5 + 2 - 1 + 3 = 9\] Million Dollars

This is the **MAXIMUM** amount that ACME earned in *any* contiguous span of years.

Examples:
Between years 1 and 9 ACME earned
\[\text{\( -3 + 2 + 1 - 4 + 5 + 2 - 1 + 3 - 1 \)}} = 4\] M$
and between years 2 and 6
\[2 + 1 - 4 + 5 + 2 = 6\] M$.

The **Maximum Contiguous Subarray Problem** is to find the span of years in which ACME earned the most, e.g., \((5, 8)\).
Formal Definition


The value of subarray $A[i \ldots j]$ is

$$V(i, j) = \sum_{x=i}^{j} A(x).$$

The Maximum Contiguous subarray problem is to find $i \leq j$ such that

$$\forall (i', j'), \quad V(i', j') \leq V(i, j).$$

Output: $V(i, j)$ s.t. $\forall (i', j'), \quad V(i', j') \leq V(i, j)$.

Note: Can modify the problem so it returns indices $(i, j)$. 
\(\Theta(n^3)\) Solution: Brute Force

Idea: Calculate the value of \(V(i, j)\) for each pair \(i \leq j\) and return the \textbf{maximum} value.

```plaintext
VMAX=A[1];
for (i=1 to N) {
    for (j=i to N) {
        // calculate \(V(i, j)\)
        V=0;
        for (x= i to j)
            V=V+A[x];
        if (V > VMAX)
            VMAX=V;
    }
}
return VMAX;
```
$\Theta(n^2)$ solution: Reuse data

Idea: We don’t need to calculate each $V(i, j)$ from “scratch” but can exploit the fact that

$$V(i, j) = \sum_{x=i}^{j} A[x] = V(i, j - 1) + A[j].$$

```c
VMAX=A[1];
for (i=1 to N) {
    V=0;
    for (j=i to N) {
        // calculate V(i, j)
        V=V+A[j];
        if (V > VMAX)
            VMAX=V;
    }
}
return VMAX;
```
Θ(n \log n) solution: Divide-and-Conquer

Idea: Set \( M = \lfloor (N + 1)/2 \rfloor \).
Let \( A_1 \) and \( A_2 \) be the MCS that MUST contain \( A[M] \) and \( A[M + 1] \) respectively. Note that the MCS must be one of

- \( S_1 \): The MCS in \( A[1 \ldots M] \),

- \( S_2 \): The MCS in \( A[M + 1 \ldots N] \),

- \( A \): Where \( A = A_1 \cup A_2 \).

\[ A_1 = \text{MCS on left containing } A[M] \quad A_2 = \text{MCS on right containing } A[M+1] \]

\( A = A_1 \cup A_2 \)
Example

\[
\begin{array}{cccccccccccc}
1 & -5 & 4 & 2 & -7 & 3 & 6 & -1 & 2 & -4 & 7 & -10 & 2 & 6 & 1 & -3 \\
1 & -5 & 4 & 2 & -7 & 3 & 6 & -1 & 2 & -4 & 7 & -10 & 2 & 6 & 1 & -3
\end{array}
\]

\[
S_1 = [3, 6] \text{ and } S_2 = [2, 6, 1].
A_1 = [3, 6, -1] \text{ and } A_2 = [2, -4, 7];
A = A_1 \cup A_2 = [3, 6, -1, 2, -4, 7]
\]

Since \( Value(S_1) = 9 \), \( Value(S_2) = 9 \) and \( Value(A) = 13 \)，
the solution to the problem is \( A \).
Finding $A$ : The conquer stage

$A_1$ is in the form $A[i \ldots M]$ : there are only $M - i + 1$ such sequences, so, $A_1$ can be found in $O(M - i)$ time.

```
MAX=A[M];
SUM=A[M];
for (k=M-1 down to i)
{
    SUM+=A[k];
    if (SUM > MAX) MAX=SUM;
}
A_1=MAX;
```

Similarly, $A_2$ is in the form $A[M+1 \ldots j]$ : there are only $j - M$ such sequences, so, $A_2$ the maximum valued such one, can be found in $O(j - M)$ time.

```
A = A_1 \cup A_2 \text{ can therefore be found in } O(j - i) \text{ time, which is linear to the input size.}
```
The Full Divide-and-Conquer Algorithm

// Input: A[i...j] with $i \leq j$
// Output: the MCS of $A[i...j]$

$MCS(A, i, j)$
1. If $i == j$ return $A[i]$
2. Else
3. Find $MCS(A, i, \lfloor \frac{i+j}{2} \rfloor)$;
4. Find $MCS(A, \lfloor \frac{i+j}{2} \rfloor + 1, j)$;
5. Find MCS that contains both $A\left\lfloor \frac{i+j}{2} \right\rfloor$ and $A\left\lfloor \frac{i+j}{2} \right\rfloor + 1$;
6. Return Maximum of the three sequences found
A full example

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{example.png}
\end{figure}
Analysis of the DC Algorithm

Let $T(m)$ (where $m$ is the problem size) be time needed to run

$MSC(A, i, j), (j - i + 1 = m)$

Step (1) requires $O(1)$ time.

Steps (3) and (4) each require $T(m/2)$ time.

Step (5) requires $O(m)$ time.

Step (6) requires $O(1)$ time.

Then $T(1) = O(1)$ and

for $n > 1$, $T(n) = 2T(n/2) + O(n)$
Analysis of the DC Algorithm

To simplify the analysis, we assume that \( n \) is a power of 2.

\[
T(n) \leq 2 \ T\left(\frac{n}{2}\right) + c \ n.
\]
Repeating this recurrence gives

\[
T(n) \leq 2 \ T\left(\frac{n}{2}\right) + c \ n
\]
\[
\leq 2 \left[ 2 \ T\left(\frac{n}{2^2}\right) + c \frac{n}{2} \right] + c \ n
\]
\[
= 2^2 T\left(\frac{n}{2^2}\right) + 2 c \ n
\]
\[
\leq 2^2 \left[ 2 \ T\left(\frac{n}{2^3}\right) + c \frac{n}{2^2} \right] + 2 c \ n
\]
\[
= 2^3 T\left(\frac{n}{2^3}\right) + 3 c \ n
\]
\[
\leq \ ...
\]
\[
= 2^h T\left(\frac{n}{2^h}\right) + h c \ n
\]

Set \( h = \log_2 n \), so that \( 2^h = n \). With this substitution, we have

\[
T(n) \leq n \ T(1) + (\log_2 n) c \ n = O(n \log_2 n).
\]
Review

In this lecture we saw 3 different algorithms for solving the maximum contiguous subarray problem. They were

- A $\Theta(n^3)$ brute force algorithm
- A $\Theta(n^2)$ algorithm that reuses data.
- A $\Theta(n \log n)$ divide-and-conquer algorithm