Lecture 1b:
The Maximum Contiguous Subarray Problem

- Reference: Chapter 8 in *Programming Pearls, (2nd ed)* by Jon Bentley.
- History: 1-D version of a a 2-D pattern recognition problem.
- 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
  - A $\Theta(n)$ algorithm by revisualizing the problem
- *Cost* of algorithm will be number of primitive operations, e.g., comparisons and arithmetic operations, that it uses.

ACME CORP – PROFIT HISTORY

Year	1	2	3	4	5	6	7	8	9
Profit M$	-3	2	1	-4	5	2	-1	3	-1

Betweeen 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
$-3 + 2 + 1 - 4 + 5 + 2 - 1 + 3 - 1 = 4$ 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

Input: An array of reals $A[1 \ldots N]$.

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

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

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 **maximum** value.

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 then
            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

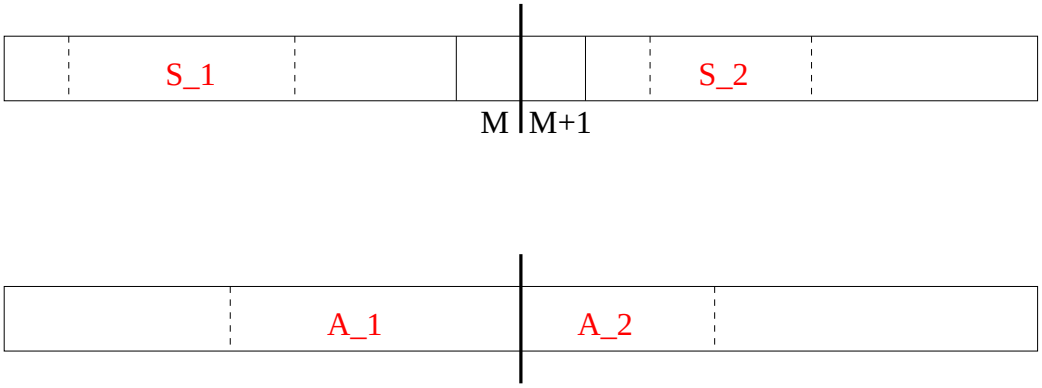
$$V(i,j) = \sum_{x=i}^{j} A[x] = V(i,j-1) + A[j].$$

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 then
            VMAX $= V;$
        }
Return(VMAX);

$\Theta(n \log n)$ solution: Divide-and-Conquer

Idea: Set $M = \lfloor (N+1)/2 \rfloor$.
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 : The MCS that *contains both* $A[M]$ and $A[M+1]$.
  Equivalently, $A = A_1 \cup A_2$.



A_1 = MCS on left containing A[M] A_2 = MCS on right containing A[M+1]

A= A_1 U A_2

Example:

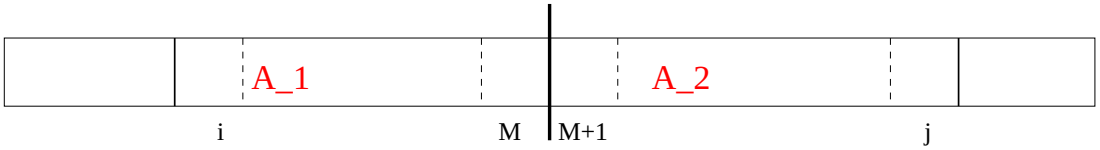
1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

1	-5	4	2	-7	3	6	-1	2	-4	7	-10	2	6	1	-3

$S_1 = [3, 6]$ and $S_2 = [2, 6, 1]$.
$A_1 = [3, 6, -1]$ 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 such sequences, so, A_1 the maximum valued such one, can be found in $O(M) = O(N)$ time.

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

$A = A_1 \cup A_2$ can therefore be found in $O(N)$ time.

The Full Divide-and-Conquer Algorithm

Input: $A[i \ldots j]$ with $i \leq j$.

$MCS(A, i, j)$

1. If $i == j$ return (i, j)
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 $A\left[\lfloor \frac{i+j}{2} \rfloor\right]$ and $A\left[\lfloor \frac{i+j}{2} \rfloor + 1\right]$;
6. Return Maximum of the three sequences found

Let $T(N)$ be time needed to run
$MSC(A, i, i + N - 1)$.
Step (1) requires $O(1)$ time.
Steps (3) and (4) each require $T(N/2)$ time.
Step (5) requires $O(N)$ time.
Step (6) requires $O(1)$ time

Then $T(1) = O(1)$ and
for $N > 1$, $T(N) = 2T(N/2) + O(N)$

$\Rightarrow T(N) = O(N \log N)$.

Review of Analysis of a D-and-C Algorithm

Note: For more details see CLRS, chapter 3.

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

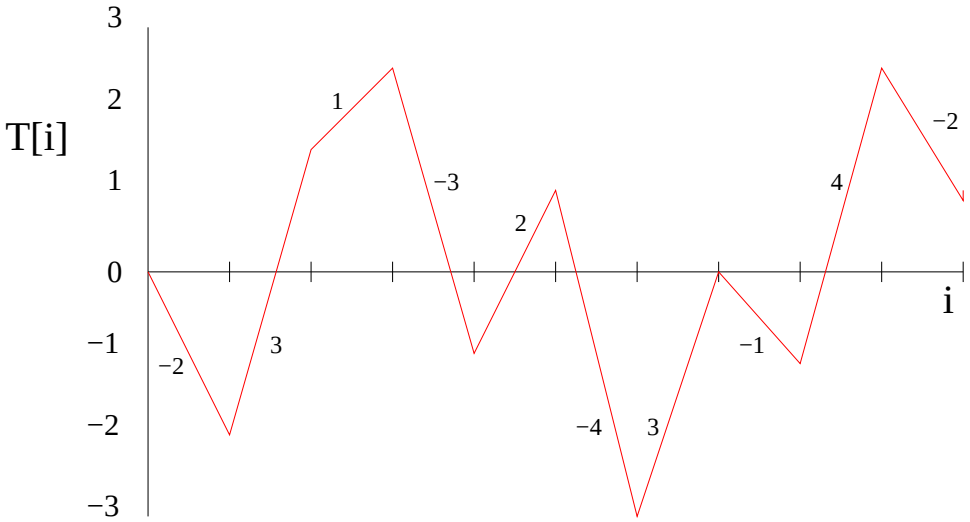
- In the DC algorithm, Steps 5 and 6 together requires $O(n)$ operations.
- Hence, $T(n) \leq 2\,T(\frac{n}{2}) + c\,n$. Repeating this recurrence gives

$$\begin{aligned} T(n) &\leq 2\,T\left(\frac{n}{2}\right) + c\,n \\ &\leq 2\left[2\,T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right] + c\,n \\ &= 2^2 T\left(\frac{n}{2^2}\right) + 2c\,n \\ &\leq 2^2\left[2\,T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right] + 2c\,n \\ &= 2^3 T\left(\frac{n}{2^3}\right) + 3c\,n \\ &\leq \ldots \\ &= 2^h T\left(\frac{n}{2^h}\right) + hc\,n \end{aligned}$$

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

$$T(n) \leq n\,T(1) + (\log_2 n)c\,n = O(n \log_2 n).$$

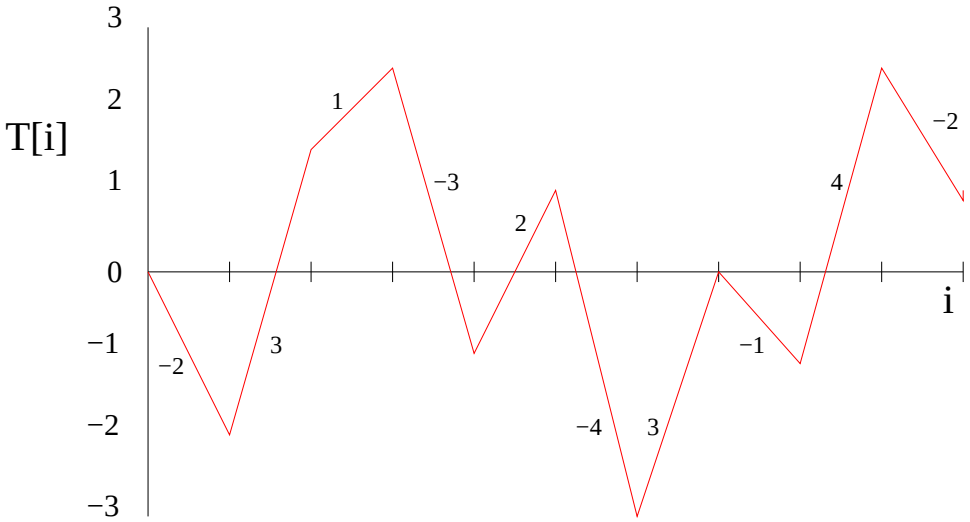
(Re)Visualizing the Problem



i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$		-2	3	1	-3	2	-4	3	-1	4	-2
$T(i)$	0	-2	1	2	-1	1	-3	0	-1	3	1

Define $T(i) = \Sigma_{x=1}^{i} A[x]$.
Then $V(i,j) = \Sigma_{x=i}^{j} A[x] = T(j) - T(i-1)$.

(Re)Visualizing the Problem



i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$		-2	3	1	-3	2	-4	3	-1	4	-2
$T(i)$	0	-2	1	2	-1	1	-3	0	-1	3	1

Define $T(i) = \Sigma_{x=1}^{i} A[x]$.
Then $V(i,j) = \Sigma_{x=i}^{j} A[x] = T(j) - T(i-1)$.

In particular, for fixed j we can find maximal $V(i,j)$ by *finding* $i \leq j$ *with minimal value of* $T(i-1)$.

Idea behind an $O(n)$ algorithm

1. Suppose we've already seen

$$A[1], A[2], \ldots A[j-1].$$

Let V be the value of MCS in $A[1, \ldots j-1]$.
Let $T_{\min}$ be the minimum $T(i)$ value so far.

2. After seeing $A[j]$ calculate

$$\begin{aligned} T(j) &= T(j-1) + A[j]. \\ V_{\text{possible}} &= T(j) - T_{\min} \\ &= \text{maximum value for a } V(i,j). \end{aligned}$$

3. Update Information:

If $T(j) < T_{\min}$ then
  $T_{\min} = T(j)$;
If $V < V_{\text{possible}}$ then
  $V = V_{\text{possible}}$

The actual $O(n)$ algorithm

$T = A[1]; V = A[1]$
$T_{\min} = \min(0, T);$

For $j = 2$ to N
    { $T = T + A[j];$ update $T(j)$
    If $T - T_{\min} > V$ If $V_{\text{possible}} > V$
        then $V = T - T_{\min};$
    If $T < T_{\min}$
        then $T_{\min} = T;$
    }
Return(V);

This algorithm implements exactly the ideas on the previous page so it returns the corect answer. Furthermore, since it does only $O(1)$ work for each $A[j]$, it runs in $O(N)$ time.

Review

In this lecture we saw 4 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
- A $\Theta(n)$ algorithm by revisualizing the problem

After completing this class you should be capable of finding the first three algorithms by yourself. Deriving those algorithms only require standard algorithmic design tools and approaches.

Tools for deriving the fourth, $O(n)$ time, algorithm can't really be taught in a short class. That derivation requires both cleverness and a mindset that can usually only be developed through practice.