

COMP 3711 Design and Analysis of Algorithms

Maximum Subarray and Related Problems

The Maximum Subarray Problem

Input: Profit history of a company. Money earned/lost each year.

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

Problem: Find the span of years in which the company earned the most

Answer: Year 5-8 , 9 M\$

Formal definition:

Input: An array of numbers $A[1 \dots n]$, both positive and negative

Output: Find the maximum $V(i, j)$, where $V(i, j) = \sum_{k=i}^j A[k]$

A brute-force algorithm

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

```
 $V_{max} \leftarrow A[1]$   
for  $i \leftarrow 1$  to  $n$  do  
    for  $j \leftarrow i$  to  $n$  do  
        // calculate  $V(i, j)$   
         $V \leftarrow 0$   
        for  $k \leftarrow i$  to  $j$  do  
             $V \leftarrow V + A[k]$   
        if  $V > V_{max}$  then  $V_{max} \leftarrow V$   
return  $V_{max}$ 
```

To measure the running time, we will count the number of **additions**.

Running time: $\Theta(n^3)$

Intuition: Calculating value of $\Theta(n^2)$ arrays, each one, on average, $\Theta(n/2)$ long.

A data-reuse algorithm

Idea:

- Don't need to calculate each $V(i, j)$ from scratch.
- Exploit the fact: $V(i, j) = V(i, j - 1) + A[j]$

```
 $V_{max} \leftarrow A[1]$   
for  $i \leftarrow 1$  to  $n$  do  
     $V \leftarrow 0$   
    for  $j \leftarrow i$  to  $n$  do  
        // calculate  $V(i, j)$   
         $V \leftarrow V + A[j]$   
        if  $V > V_{max}$  then  $V_{max} \leftarrow V$ ;  
return  $V_{max}$ 
```

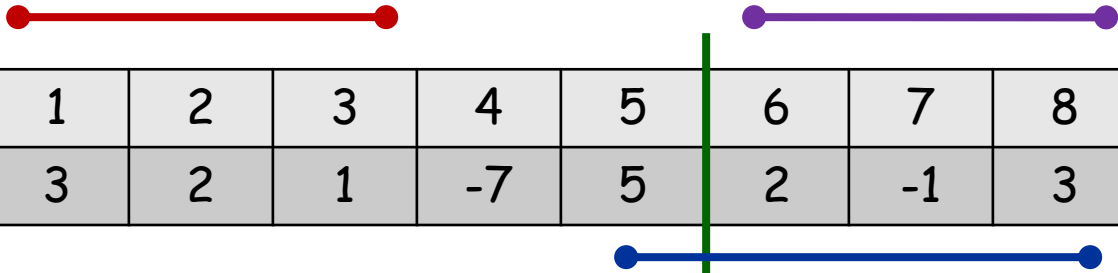
Running time: $\Theta(n^2)$

Intuition: Fix starting point i .

Calculating $V(i, j)$ from $V(i, j - 1)$ requires only $\Theta(1)$ time.

$\Rightarrow \Theta(n^2)$ in total.

A divide-and-conquer algorithm



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

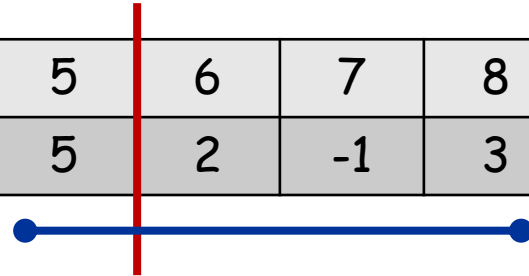
Idea:

- Cut the array into **two halves**
- All subarrays can be classified into three cases:
 - **Case 1: entirely in the first half**
 - **Case 2: entirely in the second half**
 - **Case 3: crosses the cut**
- Largest of three cases is final solution
- The optimal solutions for case 1 and 2 can be found recursively.
- Only need to consider case 3.
- Compare with merge sort:
If we can solve case 3 in linear ($O(n)$) time,
=> whole algorithm will run in $\Theta(n \log n)$ time.

$$T(n) = 2T(n/2) + n \Rightarrow T(n) = \Theta(n \log n)$$

Solving case 3

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



Idea:

- To solve problem in subarray $A[p..r]$; Let $q = \lfloor (p + r)/2 \rfloor$
- Any case 3 subarray must have $p \leq q < r$
- Such a subarray can be divided into two parts
 $A[i..q]$ and $A[q + 1..j]$, for some i and j
- The optimal solution that crosses the cut must contain $A[q]$ and $A[q+1]$. Otherwise, it would fall under cases 1 or 2.
- Just need to maximize each of them separately

To maximize $A[i..q]$ and $A[q + 1, j]$:

- Let $i = i'$, $j = j'$ be the indices that
maximize the values $A[i..q]$ and $A[q + 1, j]$;
- i', j' can be found by using separate **linear** scans to left and right of q

$\Rightarrow A[i'..j']$ has largest value of all subarrays that cross q

The complete divide-and-conquer algorithm

MaxSubarray (A, p, r) :

if $p = r$ then return $A[p]$

$q \leftarrow \lfloor (p + r) / 2 \rfloor$

$M_1 \leftarrow \text{MaxSubarray}(A, p, q)$

% MAX Left Half

$M_2 \leftarrow \text{MaxSubarray}(A, q + 1, r)$

% MAX Right Half

$L_m \leftarrow -\infty, R_m \leftarrow -\infty$

$V \leftarrow 0$

for $i \leftarrow q$ downto p

% MAX of Left

$V \leftarrow V + A[i]$

% starting at q

if $V > L_m$ then $L_m \leftarrow V$

$V \leftarrow 0$

for $i \leftarrow q + 1$ to r

% MAX of Right

$V \leftarrow V + A[i]$

% starting at q

if $V > R_m$ then $R_m \leftarrow V$

return $\max\{M_1, M_2, L_m + R_m\}$

First call: $\text{MaxSubarray}(A, 1, n)$

Analysis:

- Recurrence:

$$T(n) = 2T(n/2) + n$$

- $\Rightarrow T(n) = \Theta(n \log n)$

Linear Algorithm for Maximum Subarray

Kadane's algorithm: based on the principles of **Dynamic Programming** (powerful framework, similar to D&C, to be covered later in the semester).

Let $V[i]$ be the (local) maximum sub-array that ends at $A[i]$:

- $V[1] = A[1]$
- $V[i] = \max(A[i], A[i] + V[i - 1])$ (the max is $A[i]$ iff $V[i - 1] < 0$)

V_{max} is the maximum continuous subarray found so far

```
 $V_{max} \leftarrow -\infty, V[0] \leftarrow 0,$   
for  $i \leftarrow 1$  to  $n$  do  
     $V[i] \leftarrow V[i - 1] + A[i]$   
    if  $V[i] < A[i]$  then  
         $V[i] \leftarrow A[i]$   
    if  $V[i] > V_{max}$  then  
         $V_{max} \leftarrow V[i]$   
return  $V_{max}$ 
```

A	1	2	3	4	5	6	7	8	9
	-2	1	-3	4	-1	2	1	-5	4
$V[i]$	-2	1	-2	4	3	5	6	1	5
V_{max}	-2	1	1	4	4	5	6	6	6

Kadane's algorithm with indexes for the start and end of max subarray

```
 $V_{max} \leftarrow -\infty; V \leftarrow 0; \text{start} \leftarrow 1; \text{end} \leftarrow 1; \text{temp} \leftarrow 1$   
for  $i \leftarrow 1$  to  $n$  do  
     $V \leftarrow V + A[i];$   
    if  $V < A[i]$  then // this implies that  $V[i-1]$  is negative  
         $V \leftarrow A[i]; \text{temp} \leftarrow i$  // restart from current position  
    if  $V > V_{max}$  then // found a max sum  
         $V_{max} \leftarrow V; \text{start} \leftarrow \text{temp}; \text{end} \leftarrow i$  // update start and end  
return  $V_{max}$ 
```

Observations:

- We do not need a table for $V[i]$, since we only maintain the V value (local maximum) for the current $A[i]$.
 - We do not need to remember the previous values $V[1..i-1]$
- When we find $A[i]$ such that $V + A[i] < A[i]$, we may restart ($\text{temp} \leftarrow i$)
 - $V + A[i] < A[i]$ implies that $V[i-1]$ is negative
 - Because any subsequent max subarray will not contain the negative part before $A[i]$.

Kadane's algorithm: Example

```
 $V_{max} \leftarrow -\infty; V \leftarrow 0; \text{start} \leftarrow 1; \text{end} \leftarrow 1; \text{temp} \leftarrow 1$   
for  $i \leftarrow 1$  to  $n$  do  
     $V \leftarrow V + A[i];$   
    if  $V < A[i]$  then  $V \leftarrow A[i]; \text{temp} \leftarrow i$   
    if  $V > V_{max}$  then  $V_{max} \leftarrow V; \text{start} \leftarrow \text{temp}; \text{end} \leftarrow i$   
return  $V_{max}$ 
```

Array A: -2, 1, -3, 4, -1, 2, 1, -5, 4, $V_{max} \leftarrow -\infty, V = 0, \text{start} = 1, \text{end} = 1$

for $i = 1, A[1] = -2$:	$V \leftarrow V - 2 = -2, V_{max} = -2, \text{start} = 1, \text{end} = 1$
for $i = 2, A[2] = 1$:	$V \leftarrow V + 1 = -1, V \leftarrow 1, V_{max} = 1, \text{start} = 2, \text{end} = 2$
for $i = 3, A[3] = -3$:	$V \leftarrow V - 3 = -2, V_{max} = 1, \text{start} = 2, \text{end} = 2$
for $i = 4, A[4] = 4$:	$V \leftarrow V + 4 = 2, V \leftarrow 4, V_{max} = 4, \text{start} = 4, \text{end} = 4$
for $i = 5, A[5] = -1$:	$V \leftarrow V - 1 = 3, V_{max} = 4, \text{start} = 4, \text{end} = 4$
for $i = 6, A[6] = 2$:	$V \leftarrow V + 2 = 5, V_{max} = 5, \text{start} = 4, \text{end} = 6$
for $i = 7, A[7] = 1$:	$V \leftarrow V + 1 = 6, V_{max} = 6, \text{start} = 4, \text{end} = 7$
for $i = 8, A[8] = -5$:	$V \leftarrow V - 5 = 1, V_{max} = 6, \text{start} = 4, \text{end} = 7$
for $i = 9, A[9] = 4$:	$V \leftarrow V + 4 = 5, V_{max} = 6, \text{start} = 4, \text{end} = 7$

Maximum Sub-Array Algorithm Design

- $\Theta(n^3)$ Algorithm
 - Directly from problem definition
- $\Theta(n^2)$ Algorithm
 - Simple reuse of information
 - Trivial observation
- $\Theta(n \log n)$ Algorithm
 - Application of algorithm design principles
 - Divide-and-conquer
 - Expected of you after taking this class
- $\Theta(n)$ Algorithm
 - In the beginning, people thought that $\Theta(n \log n)$ was best possible
 - Required ah-ha moment through re-visualization of problem
 - More art than science (although knowing the science led to the art)

Maximizing Stock Profits

You have found a newspaper from the future that tells you the price of a stock over a period of n days next year. This is presented to you as an array $p[1 \dots n]$ where $p[i]$ is the price of the stock on day i .

Design an $O(n \log n)$ time divide-and-conquer algorithm that finds a strategy to make as much money as possible, i.e., it finds a pair i, j with $1 \leq i < j \leq n$ such that $p[j] - p[i]$ is maximized over all possible such pairs

Note: you are only allowed to buy the stock once and then sell it later. You can not make multiple purchases and sales over time.

If there is no way to make money, i.e., $p[j] - p[i] \leq 0$ for all pairs i, j with $1 \leq i < j \leq n$, your algorithm should return 0.

A divide-and-conquer algorithm

Day	1	2	3	4	5	6	7	8	9	10	11	12
Price	3	6	10	1	8	2	6	5	10	3	1	9

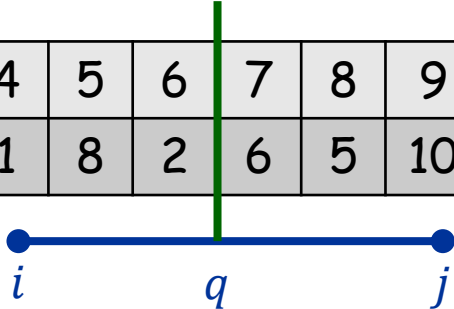
Idea:

- Cut the array into **two halves**
- All i, j solutions can be classified into three cases:
 - **Case 1:** both i, j are entirely in the first half
 - **Case 2:** both i, j are entirely in the second half
 - **Case 3:** i is in the left half and j is in the right half
- Largest profit in the three cases is final solution
- The optimal solutions for case 1 and 2 can be found recursively.
- Only need to consider case 3.
- Compare with merge sort:
If we can solve case 3 in linear ($O(n)$) time,
=> whole algorithm will run in $\Theta(n \log n)$ time.

$$T(n) = 2T(n/2) + n \Rightarrow T(n) = \Theta(n \log n)$$

Solving case 3

Day	1	2	3	4	5	6	7	8	9	10	11	12
Price	3	6	10	1	8	2	6	5	10	3	1	9



Idea:

- Let $q = \lfloor (1 + n)/2 \rfloor$
- Any case 3 solution must have $i \leq q < j$
- Maximizing a case 3 solution $p[j] - p[i]$ means finding smallest $p[i]$ in $p[1..q]$ and largest $p[j]$ in $p[q + 1..n]$

$\Rightarrow p[j] - p[i]$ has largest profit of all pairs that cross q

Such i, j can be found in $O(n)$ time;


$O(n/2)$ time to find smallest $p[i]$ in $p[a..q]$

$O(n/2)$ time to find largest $p[j]$ in $p[q + 1..b]$

Exercise on Maximizing Stock Profits

Describe a linear algorithm for Maximizing Stock Profits

Hint: You can transform the problem to max subarray and use Kadane's algorithm



Day	1	2	3	4	5	6	7	8	9	10	11	12
Price	3	6	10	1	8	2	6	5	10	3	1	9
Profit	3	4	-9	7	-6	4	-1	5	-7	-2	8	0

From the original Price (of stocks) array, I create a new array Profit that contains the price difference between consecutive days.

Specifically: $\text{Profit}[i] = \text{Price}[i + 1] - \text{Price}[i]$ indicates the profit if buy the stock on day i and sell on day $i + 1$.

I execute Kadane's algorithm on the Profit table. Suppose the algorithm outputs i, j as the **start** and **end** of the max subarray and V_{max} is its value. If $V_{max} > 0$, the maximum profit is achieved if I buy on day i and sell on day $j + 1$. If $V_{max} \leq 0$, return 0.

Exercise on Mimimum of Wave

You are given an array A of size n . A starts with $A[1] = 0$, increases to the maximum, then decreases to the minimum, and increases again and finally gets back to $A[n] = 0$. The shape of the array is then like a single wave. For example, $A = [0; 2; 5; 8; 4; 3; 1; -3; -5; -2; 0]$ is such an array but $A = [0; 3; -2; 6; -3; 0]$ is not.

Describe a D&Q algorithm to find the minimum of the array. Assume that all numbers in A are distinct; some are positive and some are negative

Let $m = \lfloor n/2 \rfloor$.

If $A[m - 1] > A[m]$ and $A[m + 1] > A[m]$ THEN RETURN $A[m]$

If $A[m] > A[m + 1]$ or $A[m] > 0$, recursively search the subarray $A[m + 1..n]$.

Otherwise, recursively search the subarray $A[1..m - 1]$.