



DIVIDE AND CONQUER

THE MAXIMUM SUBARRAY

Tanjina Helaly

DIVIDE AND CONQUER

- Divide:
 - **Divide** the problem into number of sub-problems that are smaller instances of the same problem.
- Conquer
 - **Conquer** the sub-problems by solving them recursively. If the sub-problem sizes are small enough, solve them in a straight forwards manner.
- Combine
 - **Combine** the solutions to the sub-problems into the solution for the original problem.



THE MAXIMUM SUBARRAY

- The **maximum sub-array problem** is the task of finding the contiguous sub-array within a one-dimensional array of numbers which has the largest sum.
 - Input:
 - an array $a[1..n]$ of (positive/negative) numbers.
 - Output:
 - Indices i and j such that $A[i..j]$ has the greatest sum of any nonempty, contiguous subarray of a , along with the sum of the values in $a[i..j]$.
 - Note:
 - Maximum subarray might not be unique, though its value is, so we speak of a maximum subarray, rather than the maximum subarray.



THE MAXIMUM SUBARRAY - EXAMPLE

- Example 1

Day	0	1	2	3	4
Price	10	11	7	10	6
Change a[...]		1	-4	3	-4

Max Sub Array : $a[3] = 3$

- Example 2

Day	0	1	2	3	4	5	6	7
Price	10	11	7	10	9	12	15	13
Change a[...]		1	-4	3	-1	3	3	-2

Max Sub Array : $a[3...6] = 8$



MAXIMUM SUBARRAY -BRUTE FORCE ALGORITHM

- Calculate the sum of subarray of size 1, then 2, then 3 and so on. Find the subarray that gives the max sum.
- Algorithm

```
for each subarray{
    sum = sum of all elements of that subarray.
    if(sum>ans)
        ans = sum
}
```



MAXIMUM SUBARRAY -BRUTE FORCE ALGORITHM

- Algorithm

subArray(a)

For i = subArraySize = 1 to length

sum = 0;

For start_index = 1 to length

if(start_index +subArraySize> length)

break;

For j = start_index to (start_index +subArraySize)

sum += a[i+subArraySize-1];

if(sum> max)

max = sum;

return max

- Time Complexity - $\Theta(n^3)$



MAXIMUM SUBARRAY –IMPROVISED BRUTE FORCE ALGORITHM

- Algorithm

subArray(a)

For i = 1 to length

sum = 0;

For subArraySize = 1 to length

if(i+subArraySize> length)

break;

sum += a[i+subArraySize-1];

if(sum> max)

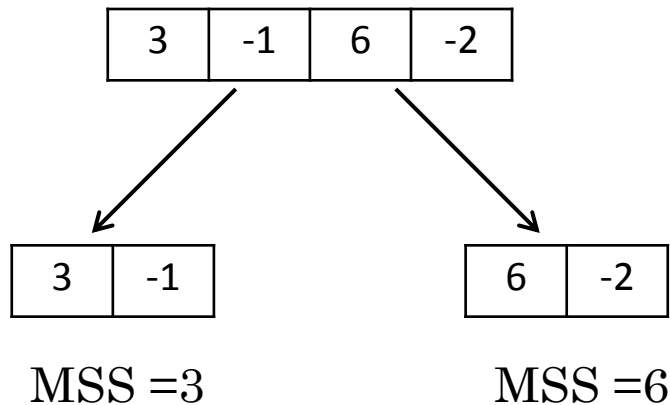
max = sum;

return max

- Time Complexity - $\Theta(n^2)$



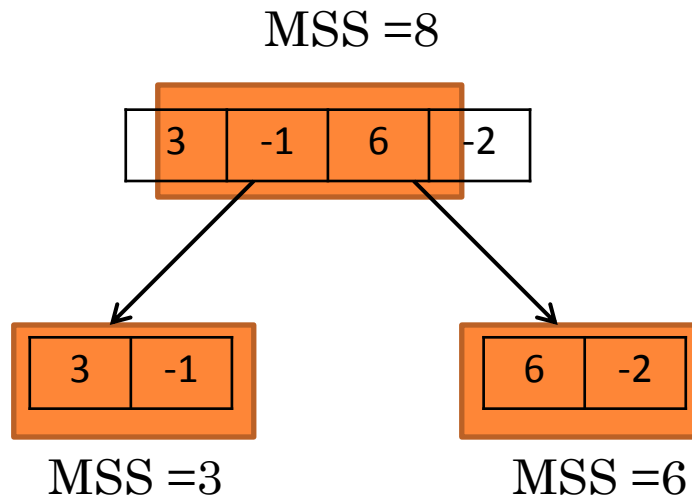
MAXIMUM SUBARRAY—DIVIDE & CONQUER



- Divide the array to 2 subarrays.
- Find the MSS of 2 subarrays.
 - Also need the find the MSS that crosses the 2 subarrays.
- Combine the Solution.



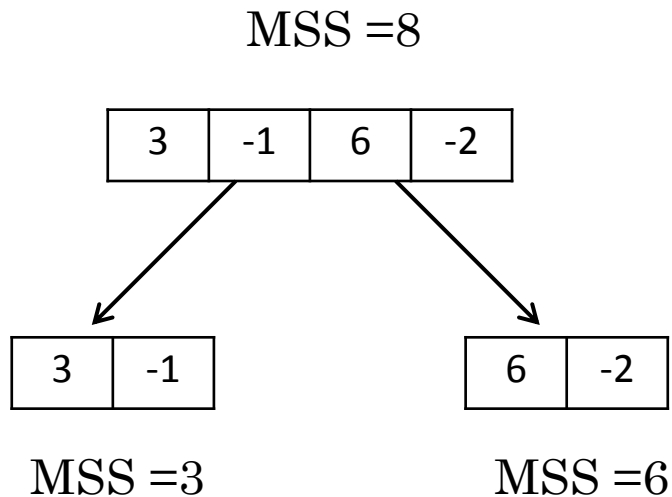
MAXIMUM SUBARRAY—DIVIDE & CONQUER



- Combine the solution
- MSS of the whole array could be
 - Entirely in left subarray, or
 - Entirely in right subarray, or
 - crossing the midpoint, some part lie in left subarray and the remaining in right



MAXIMUM SUBARRAY—DIVIDE & CONQUER




- Combine the Solution.
 - MSS of the whole array would be the biggest MSS among three(left, right and cross)



MAXIMUM SUBARRAY—DIVIDE & CONQUER

```
FindMaxSubArray(int[] A, low, high){  
    if(high == low)  
        return (low, high, A[low]);  
    else{  
        mid = (low+high)/2;  
        (leftlow, lefthigh, leftsum) = FindMaxSubArray(A, low, mid);  
        (rightlow, righthigh, rightsum) = FindMaxSubArray(A, mid+1,  
            high);  
        (crosslow, crosshigh, crosssum) = FindCrossMaxSubArray(A,  
            low, mid, high);  
        if(leftsum>rightsum && leftsum>crosssum)  
            return (leftlow, lefthigh, leftsum);  
        else if(rightsum>leftsum && rightsum>crosssum)  
            return (rightlow, righthigh, rightsum);  
    }  
    return (crosslow, crosshigh, crosssum);  
}
```



CROSSING SUBARRAY

FindCrossMaxSubArray(A, low, mid, high)

leftSum = - α

sum=0;

For i= mid down to low

sum += A[i];

if(sum>leftSum){

leftSum = sum;

maxLeft = i;

rightSum = - α

sum=0;

For i= mid+1 to high

sum += a[i];

if(sum>rightSum){

rightSum = sum;

maxRight = i;

return (maxLeft, maxRight, leftSum+rightSum);



TIME COMPLEXITY

$$T(n) = \begin{cases} \Theta(1), \dots \text{if } n = 1 \\ 2T(n/2) + \Theta(n) + \Theta(1) \dots \text{if } n > 1 \end{cases}$$

- $T(n/2)$ for each subarray of size $n/2$
 - $\Theta(n)$ for the cross array MSS
 - $\Theta(1)$ for the comparing the 3 MSS(left, right and cross)
-
- This is similar to merge sort
 - $T(n) = \theta(n \log n)$



KADANE'S ALGORITHM – ORIGINAL VERSION - $\Theta(N)$

- Scan/iterate through the list once.
 - Time Complexity: $\Theta(n)$
- Drawback: Original version only works if the array has at least 1 positive element.
- Algorithm

```
findMaxSubArraySum(int[] a)
    max_so_far = 0, max_ending_here = 0;
    For i = 1 to length
        max_ending_here += a[i];

        if(max_ending_here < 0)
            max_ending_here = 0;

        if(max_ending_here > max_so_far)
            max_so_far = max_ending_here;

    return max_so_far;
```



KADANE'S ALGORITHM – MODIFIED VERSION $\Theta(N)$

- Works for any array- for both positive and/or negative elements.
- Algorithm

```
findMaxSubArraySum(int[] a)  
    max_so_far = a[1], max_ending_here = a[1];  
    // To find start and end index, need the following variables.  
    // If you want you can also return these value  
    s_index=0,e_index=0, s_index_ending_here =0;  
    For i = 2 to length  
        max_ending_here += a[i];  
        if(max_ending_here<a[i])  
            max_ending_here = a[i];  
            // start index after getting a negative sum  
            s_index_ending_here = i;  
  
        if(max_ending_here>max_so_far)  
            max_so_far = max_ending_here;  
            e_index = i;  
            s_index = s_index_ending_here;  
    return max_so_far;
```



APPLICATION/EXAMPLE

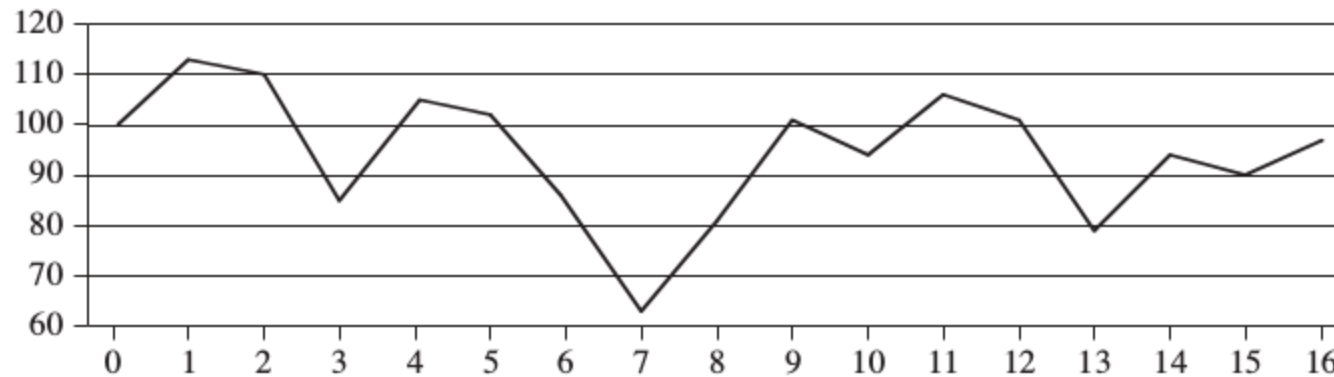


VOLATILE CHEMICAL CORPORATION – A STOCK COMPANY.

- Suppose that you been offered the opportunity to invest in the Volatile Chemical Corporation.
- You are allowed to buy one unit of stock only one time and then sell it at a later date, buying and selling after the close of trading for the day.
- To compensate for this restriction, you are allowed to learn what the price of the stock will be in the future.



VOLATILE CHEMICAL CORPORATION – A STOCK COMPANY.



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

Figure 4.1 Information about the price of stock in the Volatile Chemical Corporation after the close of trading over a period of 17 days. The horizontal axis of the chart indicates the day, and the vertical axis shows the price. The bottom row of the table gives the change in price from the previous day.



THE STOCK PROBLEM

- Your target is to maximize profit
- Find the lowest point/price?
- Choose the highest price?
 - The difference should give you maximum profit.
- But what if the highest comes before the lowest price.



TRY MAXIMUM SUBARRAY

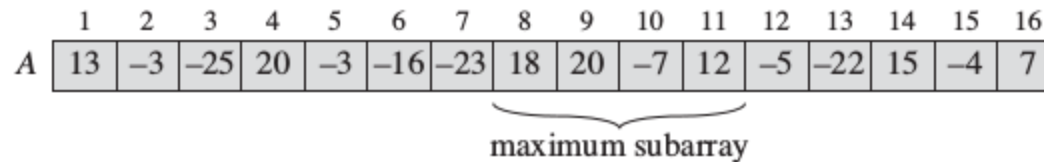
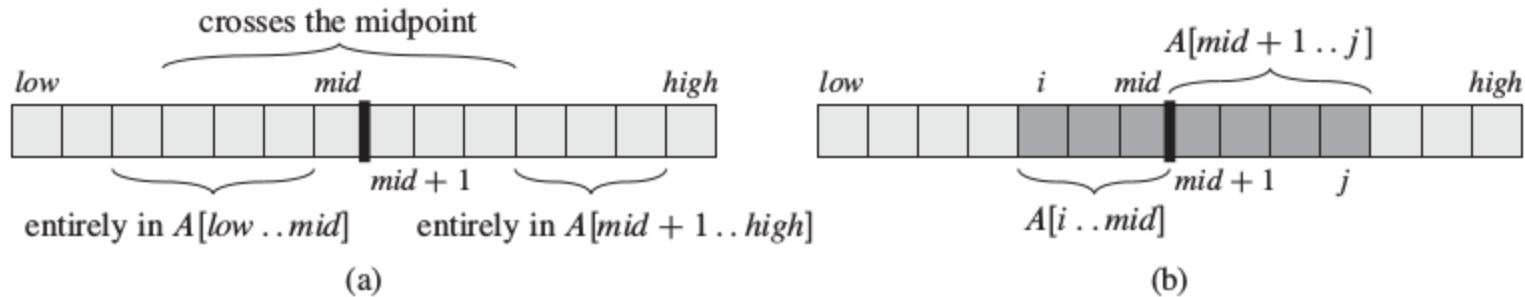


Figure 4.3 The change in stock prices as a maximum-subarray problem. Here, the subarray $A[8 \dots 11]$, with sum 43, has the greatest sum of any contiguous subarray of array A .

- Brute force
 - Try all pair
 - will give you $\theta(n^3)$ or $\theta(n^2)$ time complexity



TRY MAXIMUM SUBARRAY – DIVIDE & CONQUER



- We want to find a maximum subarray of $A[low : high]$
- Any contiguous subarray $A[i : j]$ must lie:
 - Entirely in the subarray $A[low : mid]$, so that $low \leq i \leq j \leq mid$
 - entirely in the subarray $A[mid + 1 : high]$, so that $mid \leq i \leq j \leq high$, or
 - crossing the midpoint, so that $low \leq i \leq mid \leq j \leq high$



TRY THESE

- What does this function return if all numbers are positive?
- What does this function return if all numbers are negative?



REFERENCE

- Chapter 4.1 (Cormen)

