Algorithm Analysis and Design

Divide and Conquer strategy

(Maximum Sub-array Problem)

Lecture -15

Overview

- Learn the technique of "divide and conquer" in the context of the maximum sub-array with analysis.

The Maximum subarray Problem
(A Divide and Conquer Approach)

- **Divide** the problem into a number of sub problems.
- **Conquer** the sub problems by solving them recursively.
  - ***Base case:*** If the sub problems are small enough, just solve them by brute force.
- **Combine** the sub problem solutions to give a solution to the original problem.

The Maximum subarray problem

- **Problem:** In a share market you can buy a unit of stock, only one time, then sell it at a later date
  - Buy/sell at end of day
- **Strategy:** buy low, sell high
  - The lowest price may appear after the highest price
  - Assume you know future prices
- **Objective:** Can you maximize profit by buying at lowest price and selling at highest price?

The Maximum subarray problem

- Example 1:

Day	0	1	2	3	4
Price	10	11	7	10	6

Concept: Buy lowest sell highest
Objective : Maximize the profit

The Maximum subarray problem

- **Transformation of Example 1**
  - Find sequence of days so that:
    - the net change from last to first is maximized
  - Look at the daily change in price
    - $Change\ on\ day\ i\ =\ price\ on\ day(i)\ -\ price\ day\ (i-1)$
    - We now have an array of changes (numbers),

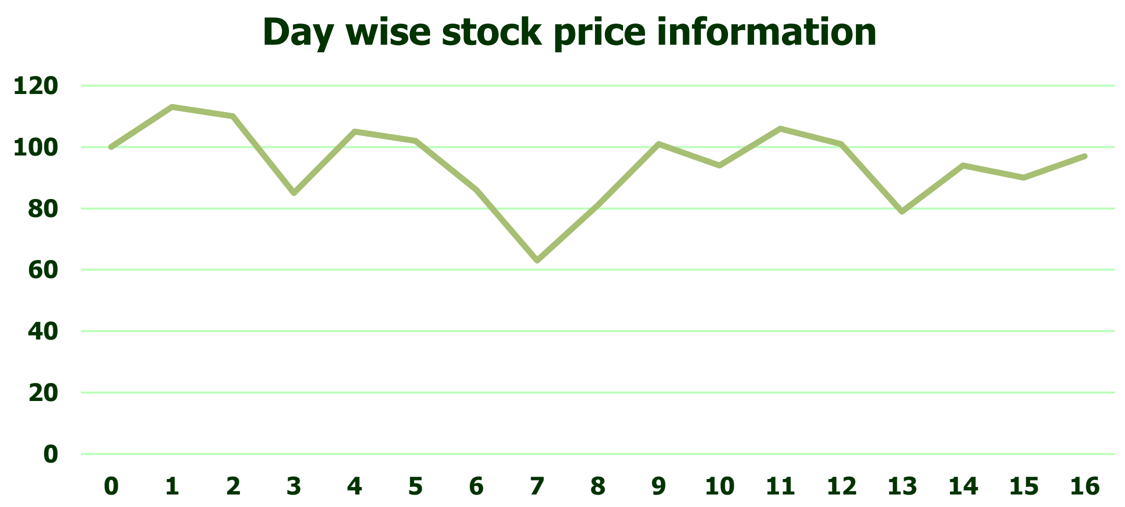
Day	0	1	2	3	4
Price	10	11	7	10	6
Changes		1	-4	3	-4

    - Hence the changes are : -1, -4, 3, -4
    - Find contiguous subarray with largest sum
    - maximum subarray–E.g.: buy after day 2, sell after day 3

The Maximum subarray problem

- Example 2:

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



Concept: Buy lowest sell highest

Objective : Maximize the profit

The Maximum subarray problem

- **Transformation of Example 2:**
  - Find sequence of days so that:
    - the net change from last to first is maximized
  - Look at the daily change in price
    - $Change\ on\ day\ i\ =\ price\ on\ day(i)\ -\ price\ day\ (i-1)$
    - We now have an array of changes (numbers),

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
Changes		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

    - Hence the changes are : 13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, and 7
    - Find contiguous subarray with largest sum
    - maximum subarray–E.g.: buy after day 7, sell after day 11

The Maximum subarray problem

- **Brute force Approach**
  - How many buy/sell pairs are possible over ‘n’ days?

    (i.e. search every possible pair of buy and sell dates in which the buy date precedes the sell date)
  - Evaluate each pair and keep track of maximum.

The Maximum subarray problem

- **Brute force Approach**

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
Changes(A)		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7
S[1,1]		13															
S[1,2]			10														
S[1,3]				-15													
S[1,4]					5												
S[1,5]						2											
S[1,6]							-14										
S[1,7]								-37									
S[1,8]									-19								
S[1,9]										1							
S[1,10]											-6						
S[1,11]												6					
S[1,12]													1				
S[1,13]														-21			
S[1,14]															-6		
S[1,15]																-10	
S[1,16]																	-3

The Maximum subarray problem

➢ **Brute force Approach**

A[1..16]		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7
		SUB STRING ARRAY (i.e. S Array)															
		[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]
	[1]	13	10	-15	5	2	-14	-37	-19	1	-6	6	1	-21	-6	-10	-3
	[2]		-3	-28	-8	-11	-27	-50	-32	-12	-19	-7	-12	-34	-19	-23	-16
	[3]			-25	-5	-8	-24	-47	-29	-9	-16	-4	-9	-31	-16	-20	-13
	[4]				20	17	1	-22	-4	16	9	21	16	-6	9	5	12
	[5]					-3	-19	-42	-24	-4	-11	1	-4	-26	-11	-15	-8
	[6]						-16	-39	-21	-1	-8	4	-1	-23	-8	-12	-5
	[7]							-23	-5	15	8	20	15	-7	8	4	11
	[8]								18	38	31	43	38	16	31	27	34
	[9]									20	13	25	20	-2	13	9	16
	[10]										-7	5	0	-22	-7	-11	-4
	[11]											12	7	-15	0	-4	3
	[12]												-5	-27	-12	-16	-9
	[13]													-22	-7	-11	-4
	[14]														15	11	18
	[15]															-4	3
	[16]																7

Hence, maximum subarray–E.g.: buy after day 7, sell after day 11

The Maximum subarray problem

- **Brute force Approach**
  - The total number of pairs are $\binom{n}{2}$. Hence the complexity is $\Theta(n^2)$
  - Can we do better?

The Maximum subarray problem

The maximum sum subarray problem is the task to find a contiguous subarray with the largest sum of a given one-dimensional array $Arr[1..n]$ of numbers. The task is to find indices $'\boldsymbol{i}'$ and $'\boldsymbol{j}'$ with the condition $\mathbf{1} \le \boldsymbol{i} \le \boldsymbol{j} \le \boldsymbol{n},$ such that:

$$\sum_{\boldsymbol{x}=\boldsymbol{i}}^{\boldsymbol{j}} \boldsymbol{Arr}[\boldsymbol{x}]$$

Is as large as possible.

(Note: The number of the input array may be positive, negative or zero)

The Maximum sum subarray problem

- **Input:** an array $A[1..n]$ of n numbers
  - Assume that some of the numbers are negative, because this problem is trivial when all numbers are nonnegative
- **Output:** a nonempty subarray $A[i..j]$ having the largest sum

$$S[i, j] = A_i + A_{i+1} + \ldots + A_j$$

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
Changes		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

$$\underbrace{18 \quad 20 \quad -7 \quad 12}_{\text{maximum subarray}}$$

The Maximum subarray problem

➢ Divide and Conquer Approach

- **Subproblem:** Find a maximum subarray of A[low .. high]

  In initial call, low =1 and high= n.

- **Divide:** the subarray into two subarrays of as equal size as possible. Find the midpoint mid of the subarrays, and consider the subarrays A[low ..mid] and A[mid+1 .. high] .
- **Conquer:** by finding the maximum subarrays of A[low .. mid] and A[mid+1..high] .
- **Combine:** by finding a maximum subarray that crosses the midpoint, and using the best solution out of the three (the subarray crossing the midpoint and the two solutions found in the conquer step).

The Maximum subarray problem

- **Divide and Conquer Approach**

  Possible locations of a maximum subarray $A[i..j]$ of $A[low..high]$, where $mid = \lfloor (low+high)/2 \rfloor$

  - entirely in $A[low..mid]$ ($low \le i \le j \le mid$)
  - entirely in $A[mid+1..high]$ ($mid < i \le j \le high$)
  - crossing the midpoint ($low \le i \le mid < j \le high$)

The Maximum subarray problem

- **Divide and Conquer Approach**

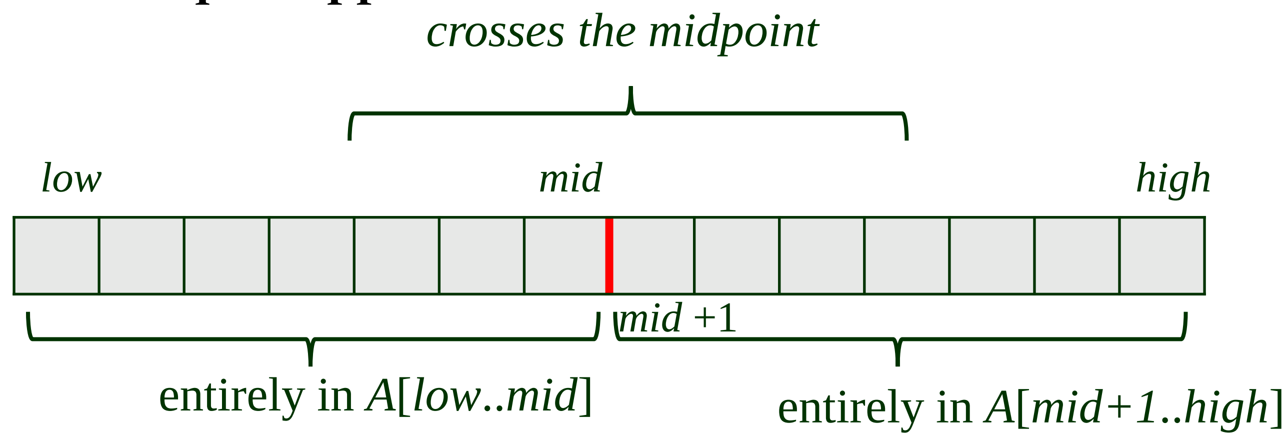


Fig (a): Possible locations of subarrays of A[low..high]

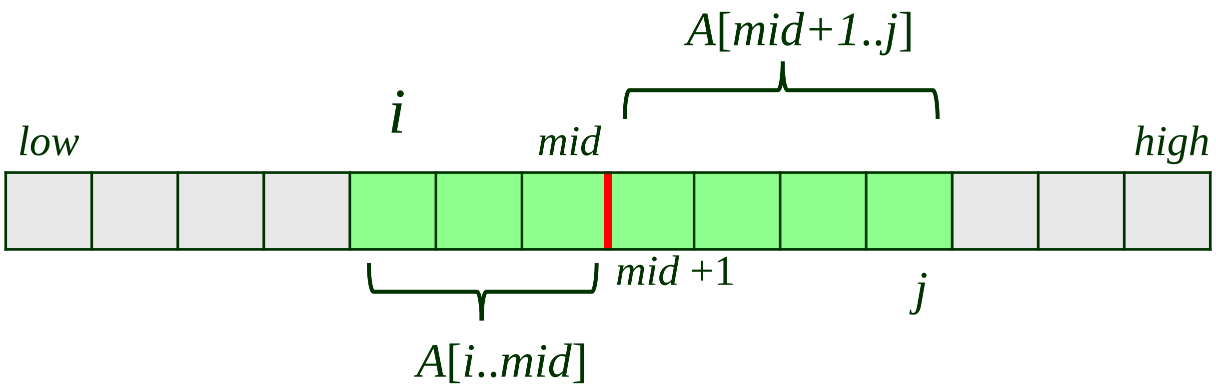


Fig (b): A[i..j] comprises two subarrays A[i..mid] and A[mid+1..j]

The Maximum subarray problem

Changes(A)		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7		
indices		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		
S[8..8]			$max - left \Rightarrow$						18	20									S[9..9]
S[7..8]								-5			13								S[9..10]
S[6..8]							-21					25		$\Leftarrow max - right$					S[9..11]
S[5..8]						-24							20						S[9..12]
S[4..8]					-4									-2					S[9..13]
S[3..8]				-29											13				S[9..14]
S[2..8]			-32													9			S[9..15]
S[1..8]		-19															16		S[9..16]

$\Longrightarrow maximum\ subarray\ crossing\ mid\ is\ S[8..11]\ =\ 18 + 25 = 43$

The Maximum subarray problem

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
  // Find a maximum subarray of the form A[i..mid].
  left-sum = -∞
  sum = 0
  for i = mid downto low
      sum = sum + A[i]
      if sum > left-sum
          left-sum = sum
          max-left = i
  // Find a maximum subarray of the form A[mid+1..j].
  right-sum = -∞
  sum = 0
  for j = mid + 1 to high
      sum = sum + A[j]
      if sum > right-sum
          right-sum = sum
          max-right = j
  // Return the indices and the sum of the two subarrays.
  return (max-left, max-right, left-sum + right-sum)
```

Changes(A)		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7		
indices		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		
S[8..8]					$max-left \Rightarrow$				18	20									S[9..9]
S[7..8]								-5			13								S[9..10]
S[6..8]							-21					25			$\Leftarrow max-right$				S[9..11]
S[5..8]						-24							20						S[9..12]
S[4..8]					-4									-2					S[9..13]
S[3..8]				-29											13				S[9..14]
S[2..8]			-32													9			S[9..15]
S[1..8]		-19															16		S[9..16]

The Maximum subarray problem

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
  if high == low
      return (low, high, A[low])          // base case: only one element
  else mid = ⌊(low + high)/2⌋
      (left-low, left-high, left-sum) =
          FIND-MAXIMUM-SUBARRAY(A, low, mid)
      (right-low, right-high, right-sum) =
          FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
      (cross-low, cross-high, cross-sum) =
          FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
      if left-sum ≥ right-sum and left-sum ≥ cross-sum
          return (left-low, left-high, left-sum)
      elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
          return (right-low, right-high, right-sum)
      else return (cross-low, cross-high, cross-sum)
```

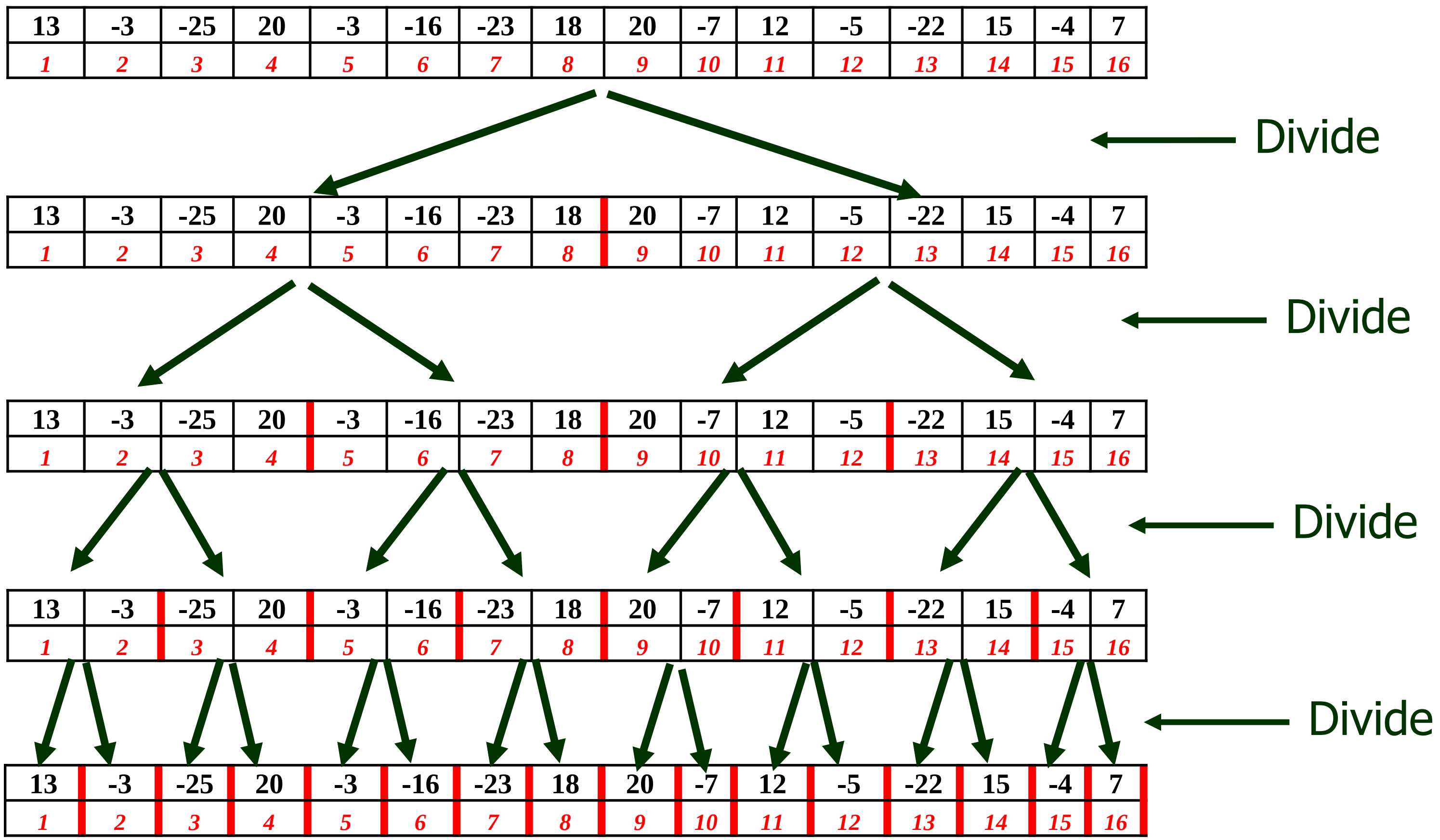
The Maximum subarray problem

Initial call: FIND-MAXIMUM-SUBARRAY(A,1,n)

- Divide by computing *mid*.
- Conquer by the two recursive calls to FIND-MAXIMUM-SUBARRAY.
- Combine by calling FIND-MAX-CROSSING-SUBARRAY and then determining
- which of the three results gives the maximum sum.
- Base case is when the subarray has only 1 element.

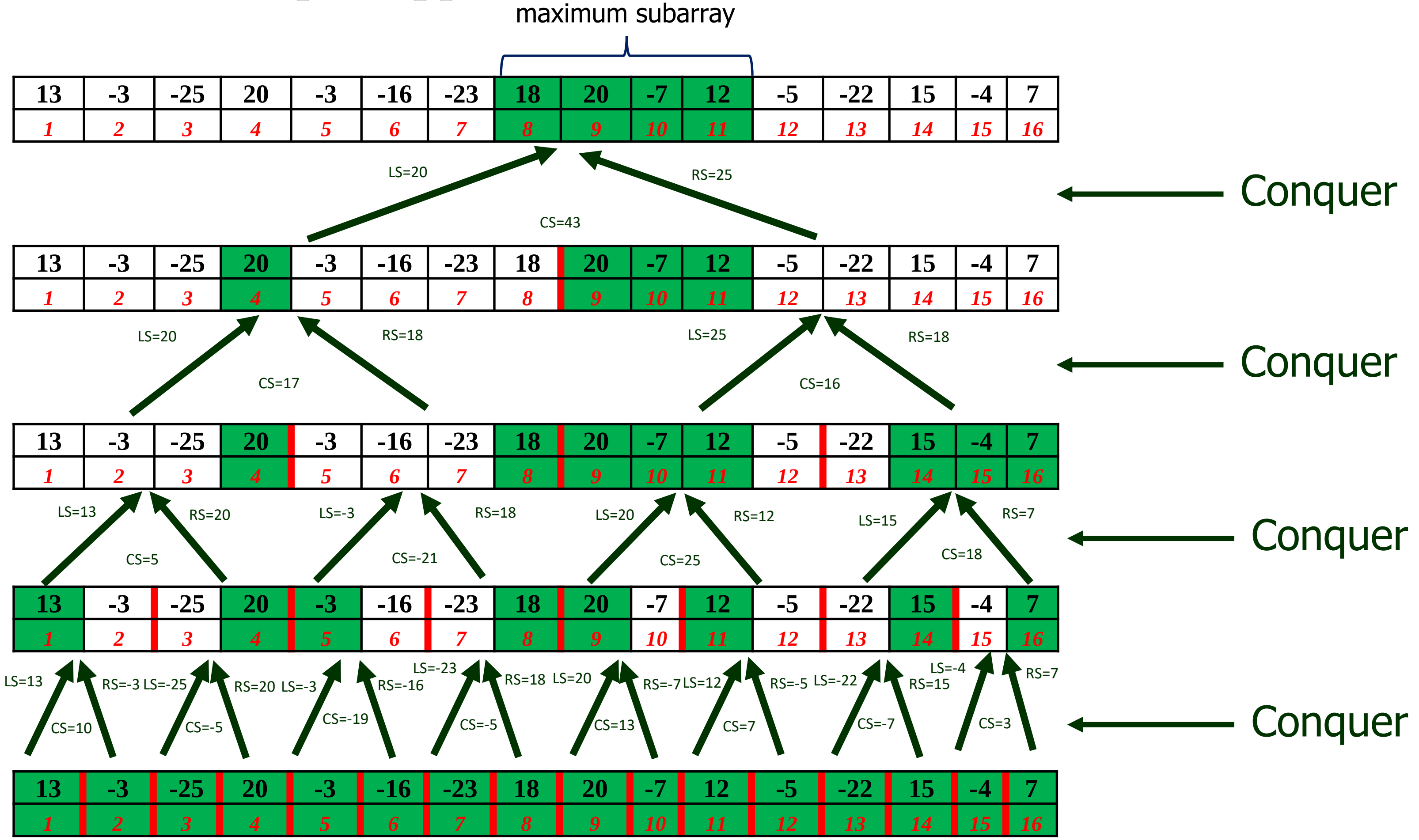
The Maximum subarray problem

➢ **Divide and Conquer Approach**



The Maximum subarray problem

➢ **Divide and Conquer Approach**



[Note: Where LS (Left Sum), RS (Right Sum) and CS (Cross Sum)]

Analysing Maximum subarray problem

Simplifying assumption: Original problem size is a power of 2, so that all subproblem sizes are integer. *[We made the same simplifying assumption when we analyzed merge sort.]*

Let $T(n)$ denote the running time of FIND-MAXIMUM-SUBARRAY on a subarray of n elements.

Base case: Occurs when *high* equals *low*, so that $n = 1$. The procedure just returns $\Rightarrow T(n) = \Theta(1)$.

Recursive case: Occurs when $n > 1$.

- Dividing takes $\Theta(1)$ time.
- Conquering solves two subproblems, each on a subarray of $n/2$ elements. Takes $T(n/2)$ time for each subproblem $\Rightarrow 2T(n/2)$ time for conquering.
- Combining consists of calling FIND-MAX-CROSSING-SUBARRAY, which takes $\Theta(n)$ time, and a constant number of constant-time tests $\Rightarrow \Theta(n) + \Theta(1)$ time for combining.

Analysing Maximum subarray problem

Recurrence for recursive case becomes

$$
\begin{aligned}
T(n) &= \Theta(1) + 2T(n/2) + \Theta(n) + \Theta(1) \\
&= 2T(n/2) + \Theta(n) \qquad \text{(absorb } \Theta(1) \text{ terms into } \Theta(n)\text{)}\ .
\end{aligned}
$$

The recurrence for all cases:

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

Same recurrence as for merge sort. Can use the master method to show that it has solution $T(n) = \Theta(n \lg n)$.

Thus, with divide-and-conquer, we have developed a $\Theta(n \lg n)$-time solution. Better than the $\Theta(n^2)$-time brute-force solution.

Home Assignment

- Solve the Maximum Subarray problem in $\Theta(n)$ time.

Thank u