Algorithms: Divide-and-Conquer (Analysis and Maximum-subarray problem)

Alessandro Chiesa, Ola Svensson

School of Computer and Communication Sciences

Lecture 4, 26.02.2025

Recall Last Lecture: Sorting

	worst-case running time	in-place
Insertion Sort	$\Theta(n^2)$	YES
Merge Sort	$\Theta(n \log n)$	NO

- A sorting algorithm is in-place of the numbers are rearranged within the array (with at most a constant number outside the array at any time)
- Insertion sort is incremental: having sorted the subarray $A[1 \dots j-1]$, we inserted the single element $A[j]$ into its proper place, yielding the sorted subarray $A[1 \dots j]$.
- Merge sort is divide-and-conquer: break the problem into smaller subproblems and then combine the solutions to the subproblems

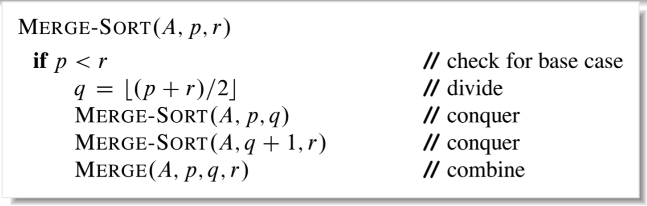
Recall: Analyzing divide-and-conquer algorithms

Use a recurrence equation to describe the running time:

- Let $T(n) =$ "running time on a problem of size n"
- If n is small enough say $n \leq c$ for some constant c then $T(n) = \Theta(1)$ (by brute force)
- Otherwise, suppose we divide into a sub problems each of size n/b.
- Let $D(n)$ be the time to divide and let $C(n)$ the time to combine solutions.
- We get the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c, \\ aT(n/b) + D(n) + C(n) & \text{otherwise.} \end{cases}$$

Recall: Analysis of Merge Sort



```
MERGE-SORT$(A, p, r)$
  if $p < r$                                  // check for base case
      $q = \lfloor (p + r)/2 \rfloor$                   // divide
      MERGE-SORT$(A, p, q)$                    // conquer
      MERGE-SORT$(A, q + 1, r)$                // conquer
      MERGE$(A, p, q, r)$                      // combine
```

Divide: takes constant time, i.e., $D(n) = \Theta(1)$

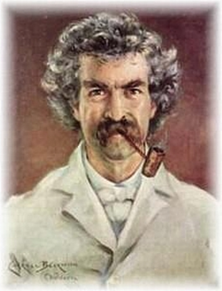
Conquer: recursively solve two subproblems, each of size $n/2 \Rightarrow 2T(n/2)$.

Combine: Merge on an n-element subarray takes $\Theta(n)$ time $\Rightarrow C(n) = \Theta(n)$.

Recurrence for merge sort running time is (if we wish to be strict)

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

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & \text{otherwise.} \end{cases}$$

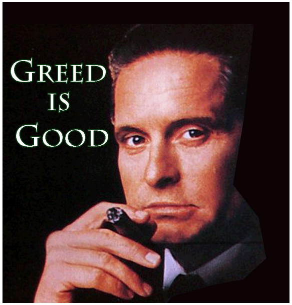


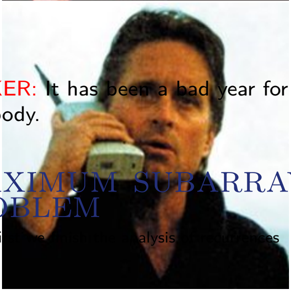
It's easier to fool people than to convince them that they have been fooled.

-Mark Twain

Does your banker fool you?

YOU: How come I just lost 20% of my fortune on the investments you recommended?





BANKER: It has been a bad year for everybody.

YOU: Oh ok.

~~YOU: Oh ok.~~

YOU: **Show me the money!**

MAXIMUM SUBARRAY PROBLEM

. . . but first we finish the analysis of recurrences

SOLVING RECURRENCES

INDUCTION

INDUCTION

INDUCTION

INDUCTION

INDUCTION

INDUCTION

INDUCTION

INDUCTION

INDUCTION

INDUCTION

Analysing Recurrences

As an example, we shall consider the following recurrence

$$T(n) = \begin{cases} c & \text{if } n = 1, \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c \cdot n & \text{otherwise.} \end{cases}$$

Note that this recurrence upper bounds and lower bounds the recurrence for MERGE-SORT by selecting c sufficiently large and small, respectively.

Indeed, there exists constants $c_1, c_2 \geq 0$ such that

$$\begin{cases} c_1 & \text{if } n = 1, \\ 2T(n/2) + c_1 \cdot n & \text{otherwise.} \end{cases} \leq \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{otherwise.} \end{cases} \leq \begin{cases} c_2 & \text{if } n = 1, \\ 2T(n/2) + c_2 n & \text{otherwise.} \end{cases}$$

Hence, $LHS = \Omega(n \log n)$ and $RHS = O(n \log n)$ implies that the recurrence for MERGE-SORT is $\Theta(n \log n)$

Analysing Recurrences

As an example, we shall consider the following recurrence

$$T(n) = \begin{cases} c & \text{if } n = 1, \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c \cdot n & \text{otherwise.} \end{cases}$$

Note that this recurrence upper bounds and lower bounds the recurrence for MERGE-SORT by selecting c sufficiently large and small, respectively.

We shall solve recurrences by using three techniques:

- The substitution method
- Recursion trees
- Master method

The substitution method

- Guess the form of the solution (forget about details such as floor, ceiling, etc.)
- **Use mathematical induction** to find the constants and show that the solution works.

$$T(n) = 2T(n/2) + c \cdot n$$

A qualified guess is that $T(n) = \Theta(n \log n)$

Proof of guess

Upper bound

There exists a constant $a > 0$ such that $T(n) \leq a \cdot n \log n$ for all $n \geq 2$

Proof by induction on n

Base cases: For any constant fixed constant k, $T(1), T(2), \ldots, T(k)$ are bounded by a constant value depending on k, selecting a sufficiently larger than this value will satisfy the base cases.

Proof of guess

Upper bound

There exists a constant $a > 0$ such that $T(n) \leq a \cdot n \log n$ for all $n \geq 2$

Proof by induction on n

Inductive step: Assume statement true $\forall n \in \{2, 3, \ldots, k-1\}$ where k is sufficiently large constant and prove the statement for $n = k$.

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + cn$$

We can thus select a to be a positive constant so that both the base cases and the inductive step holds. Hence, $T(n) = O(n \log n)$

Proof of guess

Lower bound

There exists a constant $b > 0$ such that $T(n) \geq b \cdot n \log n$ for all $n \geq 1$

Proof by induction on n

Base case: For any fixed constant k, $T(1), T(2), \ldots, T(k)$ is bounded by below by some constant (depending on k). Selecting b sufficiently smaller than this constant satisfies the base cases.

Proof of guess

Lower bound

There exists a constant $b > 0$ such that $T(n) \geq b \cdot n \log n$ for all $n \geq 1$

Proof by induction on n

Inductive step: Assume statement true $\forall n \in \{1, \ldots, k-1\}$ where k is a sufficiently large constant and prove the statement for $n = k$.

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + cn$$

We can thus select b to be a positive constant so that both the base cases and the inductive step holds. Hence, $T(n) = \Omega(n \log n)$

Floors and ceilings are a mess

- Floors and ceilings in a recurrence relation introduce a lot of low order terms.
- This makes calculations messy but it does not change the final asymptotic result.
- When analyzing recurrences we will simply assume for simplicity that all divisions evaluate to an integer.
- Do you see another reason why we may disregard floors and ceilings in the analysis of merge sort?

Common mistake using the substitution method

Be careful when using asymptotic notation!

The false proof for the recurrence $T(n) = 4T(n/4) + n$, that $T(n) = O(n)$:

$$\begin{aligned} T(n) &\leq 4(c(n/4)) + n \\ &\leq cn + n = O(n) \qquad \text{wrong!} \end{aligned}$$

Because we haven't proven the <u>exact form</u> of our inductive hypothesis (which is that $T(n) \leq cn$), this proof is false

Sometimes solution is to prove something stronger

Let $T(n) = T(n/4) + T(3n/4) + c$ if $n \geq 2$ and $T(2) = T(1) = c$.

Upper bound

There exists constants $b, b' > 0$ such that $T(n) \leq b \cdot n - b'$ for all $n \geq 1$

Proof by induction on n

Base cases: For any constant fixed constant k, $T(1)$, $T(2)$, $\ldots$, $T(k)$ are bounded by a constant value depending on k, selecting b and b' so that $b - b'$ is sufficiently larger than this value will satisfy the base cases.

Inductive step: Assume statement true $\forall n \in \{2, 3, \ldots, k-1\}$ and prove the statement for $n = k$.

$$T(n) = T(n/4) + T(3n/4) + c$$

We can thus select b and b' to be positive constants so that both the base cases and the inductive step holds. Hence, $T(n) = O(n)$

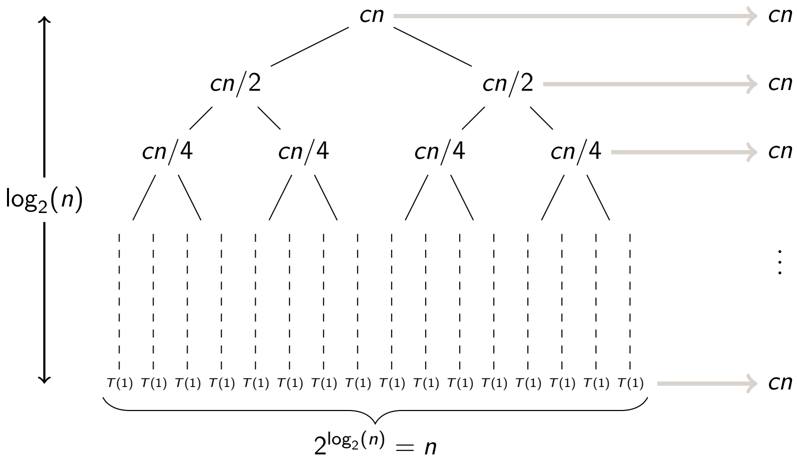
Recursion trees

Another way to generate a guess. Then verify by substitution method.

- Each node corresponds to the cost of a subproblem
- We sum the costs within each level of the tree to obtain a set of per-level costs,
- then we sum all the per-level costs to determine the total cost of all levels of the recursion.

Recursion trees

Our favorite example: $T(1) = c$ and $T(n) = 2T(n/2) + cn$

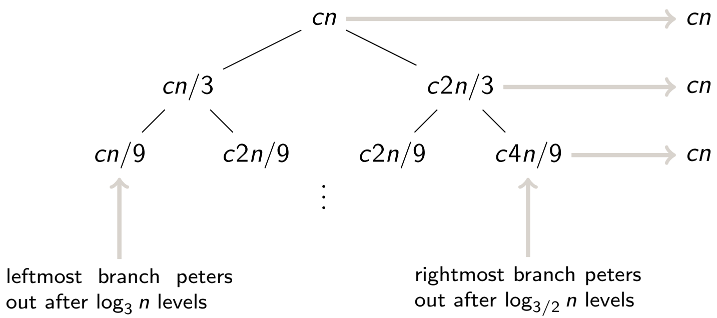


cn
$cn/2$
$cn/2$
$cn/4$
$cn/4$
$cn/4$
$cn/4$
$T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$ $T(1)$
$\log_2(n)$
$2^{\log_2(n)} = n$
cn
cn
cn
$\vdots$
cn

Qualified guess: $T(n) = cn \log_2 n = \Theta(n \log n)$

Recursion trees

Another interesting example: $T(n) = T(n/3) + T(2n/3) + cn$



cn
$cn/3$
$c2n/3$
$cn/9$
$c2n/9$
$c2n/9$
$c4n/9$
$\vdots$
cn
cn
cn
leftmost branch peters out after $\log_3 n$ levels
rightmost branch peters out after $\log_{3/2} n$ levels

- There are $\log_3 n$ full levels and after $\log_{3/2} n$ levels the problem size is down to 1.
- Each level contributes $\approx cn$

Qualified guess: exist positive constants a, b so that

$$a \cdot n \log_3(n) \le T(n) \le b \cdot n \log_{3/2} n \Rightarrow T(n) = \Theta(n \log n)$$

Master method

Used to black-box solve recurrences of the form $T(n) = aT(n/b) + f(n)$

Theorem (Master Theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $T(n)$ be defined on the nonnegative integers by the recurrence

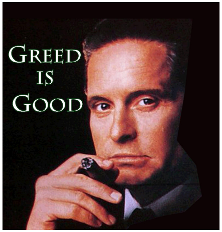
$$T(n) = aT(n/b) + f(n).$$

Then, $T(n)$ has the following asymptotic bounds

- *If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$*
- *If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$*
- *If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $a \cdot f(n/b) \leq c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$*

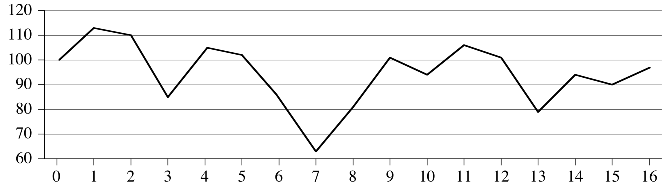
Our favorite example: $T(1) = c$ and $T(n) = 2T(n/2) + cn$

- $f(n) = \Theta(n)$ and $a = b = 2$ so $\log_b(a) = 1$ and $f(n) = \Theta(n^{\log_b(a)})$.
- By Master theorem, we have $T(n) = \Theta(n \log n)$:)



MAXIMUM-SUBARRAY PROBLEM

Scenario

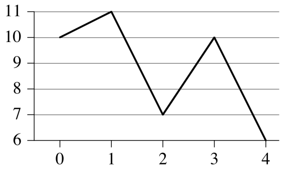


- You have the prices that a stock traded at over a period of n consecutive days
- When should you have bought the stock? When should you have sold the stock?
- Even though it's in retrospect, you can yell at your stockbroker for not recommending these buy and sell dates

Optimal Solution Structure

Why not just “buy low,sell high”?

- Lowest price might occur <u>after</u> the highest price
- But wouldn’t the optimal strategy involve buying at the lowest price or selling at the highest price?
- Not necessarily:



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

It requires us to solve the MAXIMUM-SUBARRAY PROBLEM

Maximum-subarray problem

"If we let $A[i] =$ (price after day i) $-$ (price after day $i-1$) then if the maximum subarray is $A[i \dots j]$ then we should have bought just before day i and sold just after day j."

Definition

INPUT: An array $A[1 \dots n]$ of numbers

OUTPUT: Indices i and j such that $A[i \dots j]$ has the greatest sum of any nonempty, contiguous subarray of A, along with the sum of the values in $A[i \dots j]$

Examples:

1	-4	3	-4

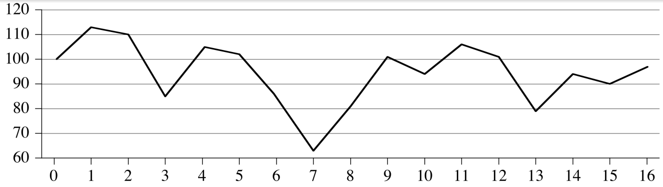
output is $i = j = 3$ and the sum 3

-2	-4	3	-1	5	7	-7	-2	4	-3	2

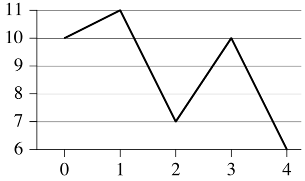
output is $i = 3$ and $j = 6$ and the sum 14

Maximum-subarray problem

More examples



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



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

FIRST ALGORITHM (brute force)

Brute Force

Simply check all possible subarrays

$\binom{n}{2} = \Theta(n^2)$ many

```
Maximum-subarray-slow($A[1 \dots n]$)
    $B.val \leftarrow -\infty$, $B.i \leftarrow 1$, $B.j \leftarrow n$
    for $i \leftarrow 1$ to $n$
        $tmp \leftarrow 0$
        for $j \leftarrow i$ to $n$
            $tmp \leftarrow tmp + A[j]$
            if $tmp > B.val$
                $B.val \leftarrow tmp$
                $B.i \leftarrow i$
                $B.j \leftarrow j$
    return $(B.i, B.j, B.val)$
```

$\boxed{\text{Current best } (B.val) = 8\ 10\ 2\ -\infty}$

$\boxed{tmp = \text{[illegible]}}$

-2	-4	3	-1	5	7	-7

and so on ...

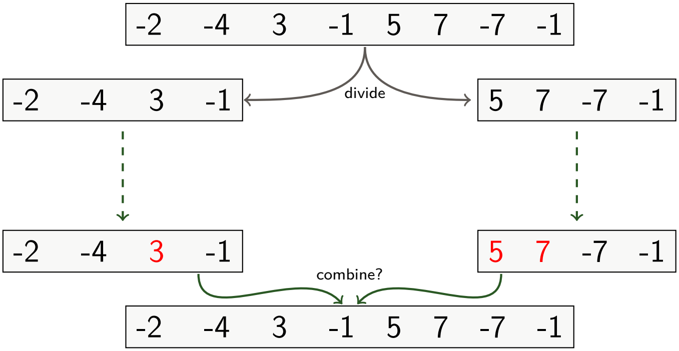
Brute Force

```
Maximum-subarray-slow($A[1 \dots n]$)
1   $B.val \leftarrow -\infty$, $B.i \leftarrow 1$, $B.j \leftarrow n$
2   for $i \leftarrow 1$ to $n$
3       $tmp \leftarrow 0$
4       for $j \leftarrow i$ to $n$
5           $tmp \leftarrow tmp + A[j]$
6           if $tmp > B.val$
7               $B.val \leftarrow tmp$
8               $B.i \leftarrow i$
9               $B.j \leftarrow j$
4   return $(B.i, B.j, B.val)$
```

What is the running time? $\Theta(n^2)$

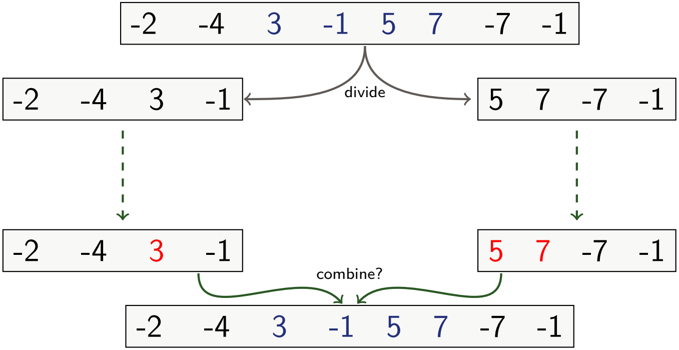
How much space do we use? $\Theta(n)$

Divide-and-Conquer



Solution

Also find the maximum subarray that crosses the midpoint!



Divide-and-Conquer approach

Divide	the subarray into two subarrays of as equal size as possible. Find the midpoint *mid* of the subarrays, and consider the subarrays $A[low \dots mid]$ and $A[mid + 1 \dots high]$.
Conquer	by finding maximum subarrays of $A[low \dots mid]$ and $A[mid + 1 \dots high]$.
Combine	by finding a maximum subarray that crosses the midpoint, and using the best solution out of the three

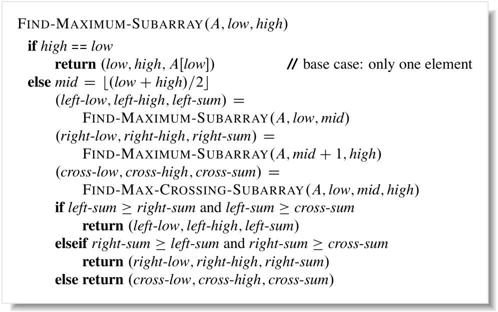
This strategy works because any subarray must either lie entirely on one side of the midpoint or cross the midpoint

Divide-and-Conquer approach

Divide the subarray into two subarrays of as equal size as possible. Find the midpoint *mid* of the subarrays, and consider the subarrays $A[low \dots mid]$ and $A[mid + 1 \dots high]$.

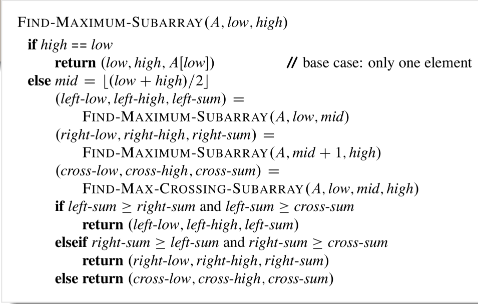
Conquer by finding maximum subarrays of $A[low \dots mid]$ and $A[mid + 1 \dots high]$.

Combine by finding a maximum subarray that crosses the midpoint, and using the best solution out of the three



```
FIND-MAXIMUM-SUBARRAY($A, low, high$)
 if $high == low$
     return ($low, high, A[low]$)            // base case: only one element
 else $mid = \lfloor (low + high)/2 \rfloor$
     ($left\text{-}low, left\text{-}high, left\text{-}sum$) =
         FIND-MAXIMUM-SUBARRAY($A, low, mid$)
     ($right\text{-}low, right\text{-}high, right\text{-}sum$) =
         FIND-MAXIMUM-SUBARRAY($A, mid + 1, high$)
     ($cross\text{-}low, cross\text{-}high, cross\text{-}sum$) =
         FIND-MAX-CROSSING-SUBARRAY($A, low, mid, high$)
     if $left\text{-}sum \geq right\text{-}sum$ and $left\text{-}sum \geq cross\text{-}sum$
         return ($left\text{-}low, left\text{-}high, left\text{-}sum$)
     elseif $right\text{-}sum \geq left\text{-}sum$ and $right\text{-}sum \geq cross\text{-}sum$
         return ($right\text{-}low, right\text{-}high, right\text{-}sum$)
     else return ($cross\text{-}low, cross\text{-}high, cross\text{-}sum$)
```

Analysis



```
FIND-MAXIMUM-SUBARRAY($A, low, high$)
  if $high == low$
      return ($low, high, A[low]$)          // base case: only one element
  else $mid = \lfloor (low + high)/2 \rfloor$
      ($left\text{-}low, left\text{-}high, left\text{-}sum$) =
          FIND-MAXIMUM-SUBARRAY($A, low, mid$)
      ($right\text{-}low, right\text{-}high, right\text{-}sum$) =
          FIND-MAXIMUM-SUBARRAY($A, mid + 1, high$)
      ($cross\text{-}low, cross\text{-}high, cross\text{-}sum$) =
          FIND-MAX-CROSSING-SUBARRAY($A, low, mid, high$)
      if $left\text{-}sum \geq right\text{-}sum$ and $left\text{-}sum \geq cross\text{-}sum$
          return ($left\text{-}low, left\text{-}high, left\text{-}sum$)
      elseif $right\text{-}sum \geq left\text{-}sum$ and $right\text{-}sum \geq cross\text{-}sum$
          return ($right\text{-}low, right\text{-}high, right\text{-}sum$)
      else return ($cross\text{-}low, cross\text{-}high, cross\text{-}sum$)
```

Assume that we can find max-crossing-subarray in time $\Theta(n)$

Divide takes constant time,i.e., $\Theta(1)$

Conquer recursively solve two subproblems, each of size $n/2 \Rightarrow T(n/2)$

Merge time dominated by FIND-MAX-CROSSING-SUBARRAY $\Rightarrow \Theta(n)$

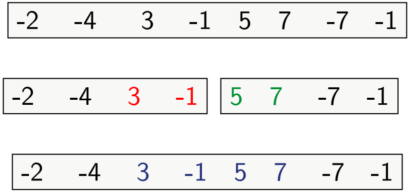
Recursion for the running time is

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

Hence, $T(n) = \Theta(n \log n)$

Finding maximum subarray crossing midpoint

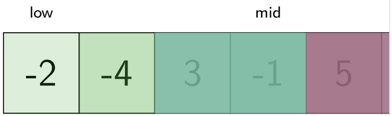
- Any subarray crossing the midpoint $A[mid]$ is made of two subarrays $A[i \dots mid]$ and $A[mid+1, \dots, j]$ where $low \leq i \leq mid$ and $mid < j \leq high$
- Find maximum subarrays of the form $A[i \dots mid]$ and $A[mid+1 \dots j]$ and then combine them.



Crossing subarray

Running time? $\Theta(n)$

Space? $\Theta(n)$



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

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

Summary

- Divide-and-conquer simple but powerful algorithmic paradigm
- Merge-sort and maximum subarray both run in time $\Theta(n \log n)$
- This is much faster than $\Theta(n^2)$ for large instances
- Remember techniques for solving recurrences
- Solving recurrences fun but delicate