CS4102 Algorithms
Spring 2019

Warm up

Show that finding the minimum of an unordered list requires $\Omega(n)$ comparisons

1

Find Min, Lower Bound Proof

Show that finding the minimum of an unordered list requires $\Omega(n)$ comparisons

Suppose (toward contradiction) that there is an algorithm for Find Min that does fewer than $\frac{n}{2} = \Omega(n)$ comparisons.

This means there is at least one "uncompared" element
We can't know that this element wasn't the min!

2	8	19	20		3	9	-4
0	1	2	3	4	5	6	7

2

Announcements

- HW4 due Monday 3/4 at 11pm
  - Sorting
  - Written (use LaTeX!)
- No Instructor Office Hours this week
  - I'll be at SIGCSE
  - Available on Piazza and Email!
- HW1 solutions in-class on Wednesday
- Midterm next Wednesday
  - Covers material through today
  - Review session M or Tu evening

3

Today's Keywords

- Sorting
- Linear time Sorting
- Counting Sort
- Radix Sort
- Maximum Sum Continuous Subarray

4

CLRS Readings

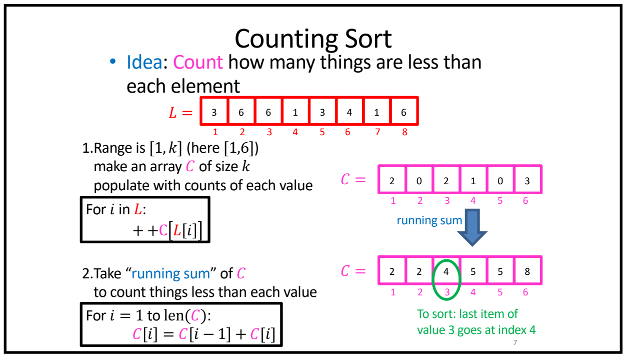
- Chapter 8

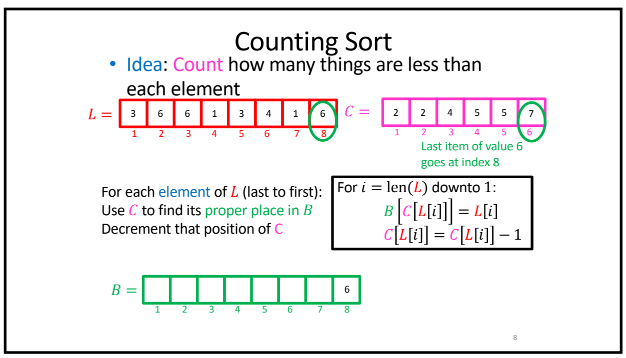
5

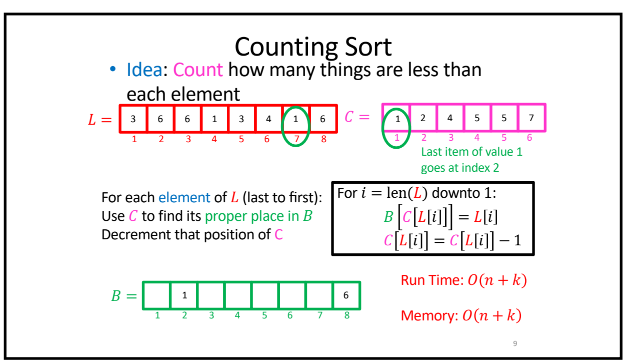
Sorting in Linear Time

- Cannot be comparison-based
- Need to make some sort of assumption about the contents of the list
  - Small number of unique values
  - Small range of values
  - Etc.

6


Counting Sort
• Idea: Count how many things are less than each element
L = 3 6 6 1 3 4 1 6
1 2 3 4 5 6 7 8
1.Range is [1, k] (here [1,6])
make an array C of size k
populate with counts of each value
For i in L:
++C[L[i]]
C = 2 0 2 1 0 3
1 2 3 4 5 6
running sum
C = 2 2 4 5 5 8
1 2 3 4 5 6
2.Take "running sum" of C
to count things less than each value
For i = 1 to len(C):
C[i] = C[i − 1] + C[i]
To sort: last item of value 3 goes at index 4


Counting Sort
• Idea: Count how many things are less than each element
L = 3 6 6 1 3 4 1 6
1 2 3 4 5 6 7 8
C = 2 2 4 5 5 7
1 2 3 4 5 6
Last item of value 6 goes at index 8
For each element of L (last to first):
Use C to find its proper place in B
Decrement that position of C
For i = len(L) downto 1:
B[C[L[i]]] = L[i]
C[L[i]] = C[L[i]] − 1
B = 6
1 2 3 4 5 6 7 8


Counting Sort
• Idea: Count how many things are less than each element
L = 3 6 6 1 3 4 1 6
1 2 3 4 5 6 7 8
C = 1 2 4 5 5 7
1 2 3 4 5 6
Last item of value 1 goes at index 2
For each element of L (last to first):
Use C to find its proper place in B
Decrement that position of C
For i = len(L) downto 1:
B[C[L[i]]] = L[i]
C[L[i]] = C[L[i]] − 1
B = 1 6
1 2 3 4 5 6 7 8
Run Time: O(n + k)
Memory: O(n + k)

Counting Sort

- Why not always use counting sort?
- For 64-bit numbers, requires an array of length $2^{64} > 10^{19}$
  - 5 GHz CPU will require > 116 years to initialize the array
  - 18 Exabytes of data
    - Total amount of data that Google has

10

12 Exabytes

11

Radix Sort

- Idea: Stable sort on each digit, from least significant to most significant

103	801	401	323	255	823	999	101	113	901	555	512	245	800	018	121
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Place each element into a "bucket" according to its 1's place

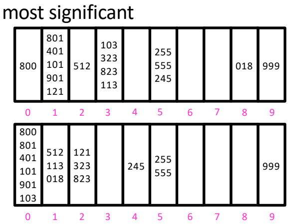
800	801 401 101 901 121	512	103 323 823 113		255 555 245			018	999
0	1	2	3	4	5	6	7	8	9

12

Radix Sort

- Idea: Stable sort on each digit, from least significant to most significant

Place each element into a "bucket" according to its 10's place

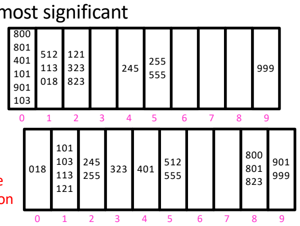


Radix Sort

- Idea: Stable sort on each digit, from least significant to most significant

Place each element into a "bucket" according to its 100's place

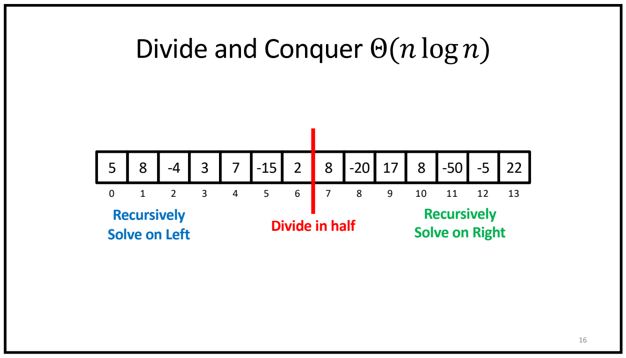
Run Time: $O(d(n+b))$
$d =$ digits in largest value
$b =$ base of representation

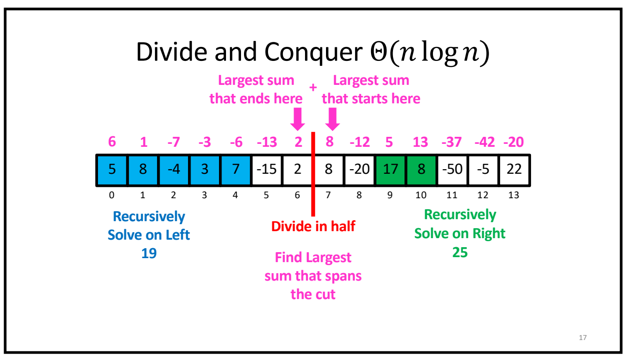


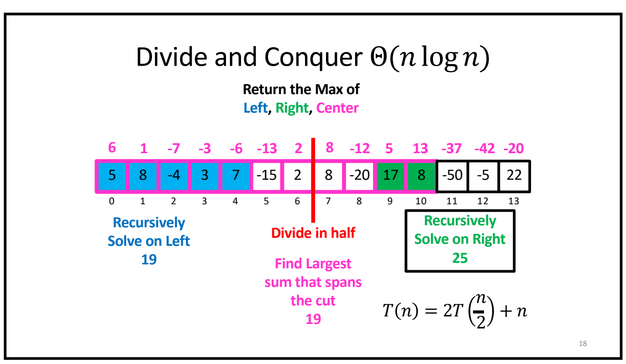
Maximum Sum Continuous Subarray Problem

The maximum-sum subarray of a given array of integers A is the interval $[a, b]$ such that the sum of all values in the array between a and b inclusive is maximal.

Given an array of n integers (may include both positive and negative values), give a $O(n \log n)$ algorithm for finding the maximum-sum subarray.


Divide and Conquer $\Theta(n \log n)$
5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Recursively Solve on Left
Divide in half
Recursively Solve on Right
16


Divide and Conquer $\Theta(n \log n)$
Largest sum that ends here + Largest sum that starts here
6 1 -7 -3 -6 -13 2 8 -12 5 13 -37 -42 -20
5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Recursively Solve on Left 19
Divide in half
Recursively Solve on Right 25
Find Largest sum that spans the cut
17


Divide and Conquer $\Theta(n \log n)$
Return the Max of Left, Right, Center
6 1 -7 -3 -6 -13 2 8 -12 5 13 -37 -42 -20
5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Recursively Solve on Left 19
Divide in half
Recursively Solve on Right 25
Find Largest sum that spans the cut 19
$T(n) = 2T\left(\frac{n}{2}\right) + n$
18

Divide and Conquer Summary

Typically multiple subproblems.
Typically all roughly the same size.

- Divide
  - Break the list in half
- Conquer
  - Find the best subarrays on the left and right
- Combine
  - Find the best subarray that "spans the divide"
  - I.e. the best subarray that ends at the divide concatenated with the best that starts at the divide

Generic Divide and Conquer Solution

```
def myDCalgo(problem):
    if baseCase(problem):
        solution = solve(problem) #brute force if necessary
        return solution
    subproblems = Divide(problem)
    for sub in subproblems:
        subsolutions.append(myDCalgo(sub))
    solution = Combine(subsolutions)
    return solution
```

20

MSCS Divide and Conquer $\Theta(n \log n)$

```
def MSCS(list):
    if list.length < 2:
        return list[0]    #list of size 1 the sum is maximal
    {listL, listR} = Divide (list)
    for list in {listL, listR}:
        subSolutions.append(MSCS(list))
    solution = max(solnL, solnR, span(listL, listR))
    return solution
```

21

Types of "Divide and Conquer"

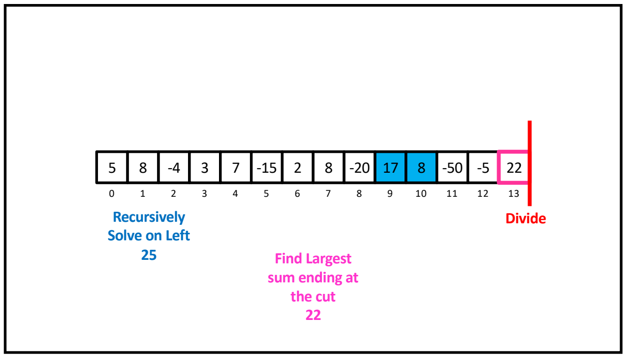
- Divide and Conquer
  - Break the problem up into several subproblems of roughly equal size, recursively solve
  - E.g. Karatsuba, Closest Pair of Points, Mergesort…
- Decrease and Conquer
  - Break the problem into a single smaller subproblem, recursively solve
  - E.g. Gotham City Police, Quickselect, Binary Search

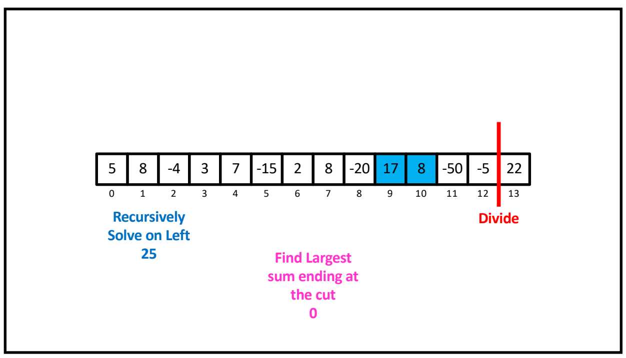
Pattern So Far

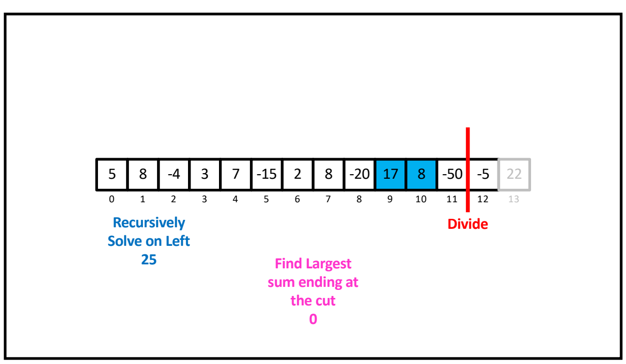
- Typically looking to divide the problem by some fraction (½, ¼ the size)
- Not necessarily always the best!
  - Sometimes, we can write faster algorithms by finding unbalanced divides.

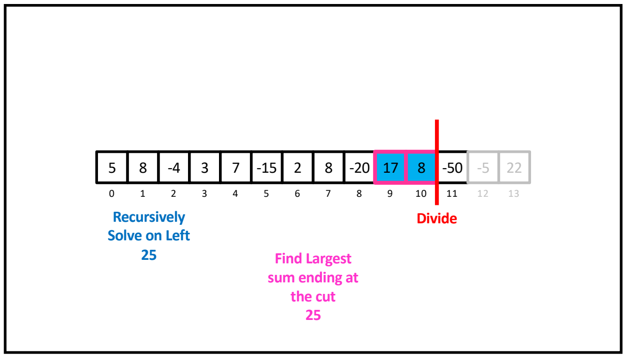
Unbalanced Divide and Conquer

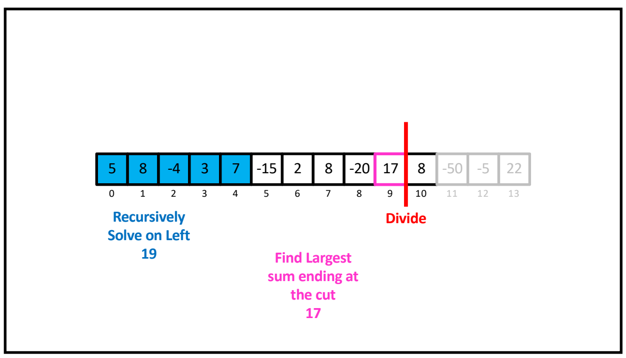
- Divide
  - Make a subproblem of all but the last element
- Conquer
  - Find best subarray on the left ($BSL(n-1)$)
  - Find the best subarray ending at the divide ($BED(n-1)$)
- Combine
  - New Best Ending at the Divide:
    - $BED(n) = \max(BED(n-1) + arr[n],\ 0)$
  - New best on the left:
    - $BSL(n) = \max\left(BSL(n-1),\ BED(n)\right)$

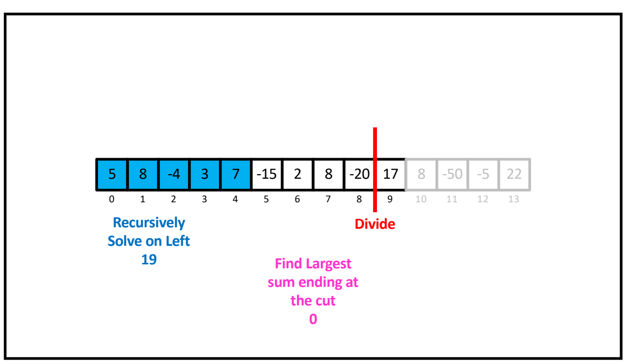

5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Divide
Recursively Solve on Left 25
Find Largest sum ending at the cut 22

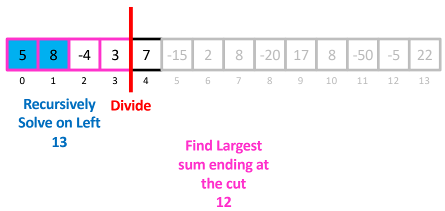

5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Divide
Recursively Solve on Left 25
Find Largest sum ending at the cut 0


5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Divide
Recursively Solve on Left 25
Find Largest sum ending at the cut 0


5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Recursively Solve on Left 25
Divide
Find Largest sum ending at the cut 25


5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Recursively Solve on Left 19
Divide
Find Largest sum ending at the cut 17


5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Recursively Solve on Left 19
Divide
Find Largest sum ending at the cut 0



Unbalanced Divide and Conquer

- Divide
  - Make a subproblem of all but the last element
- Conquer
  - Find best subarray on the left ($BSL(n-1)$)
  - Find the best subarray ending at the divide ($BED(n-1)$)
- Combine
  - New Best Ending at the Divide:
    - $BED(n) = \max(BED(n-1) + arr[n],\ 0)$
  - New best on the left:
    - $BSL(n) = \max\left(BSL(n-1),\ BED(n)\right)$

Was unbalanced better? YES

- Old:
  - We divided in Half
  - We solved 2 different problems:
    - Find the best overall on BOTH the left/right
    - Find the best which end/start on BOTH the left/right respectively
  - Linear time combine

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

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

- New:
  - We divide by 1, n-1
  - We solve 2 different problems:
    - Find the best overall on the left ONLY
    - Find the best which ends on the left ONLY
  - Constant time combine

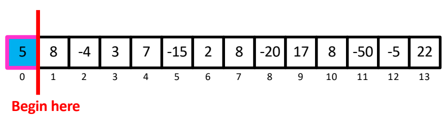
$$T(n) = 1T(n-1) + 1$$

$$T(n) = \Theta(n)$$

Maximum Sum Continuous Subarray Problem Redux

- Solve in $O(n)$ by increasing the problem size by 1 each time.
- Idea: Only include negative values if the positives on both sides of it are "worth it"

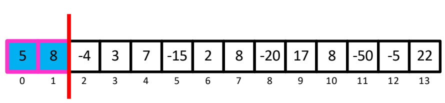
$\Theta(n)$ Solution



Remember two values: Best So Far 5 Best ending here 5

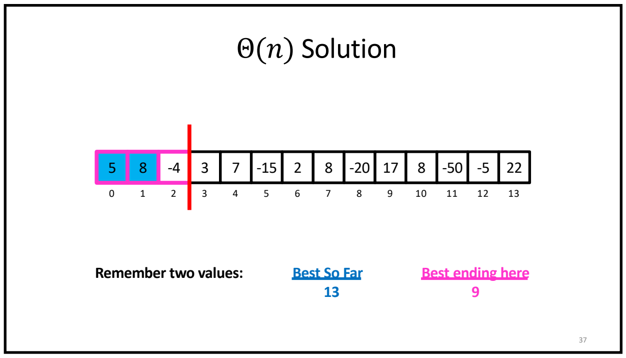
35

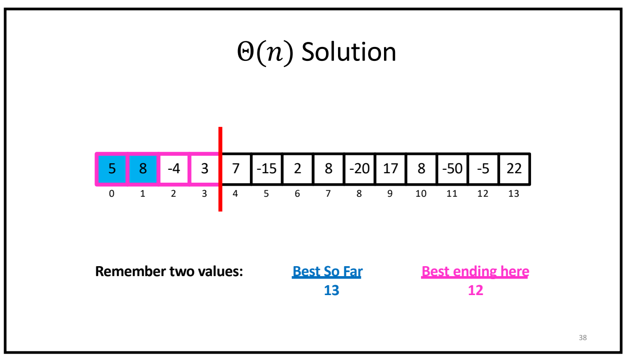
$\Theta(n)$ Solution

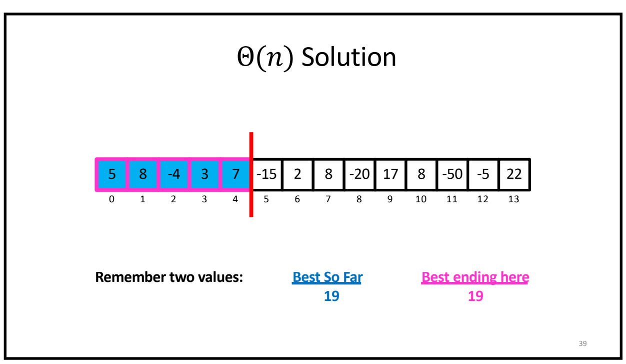


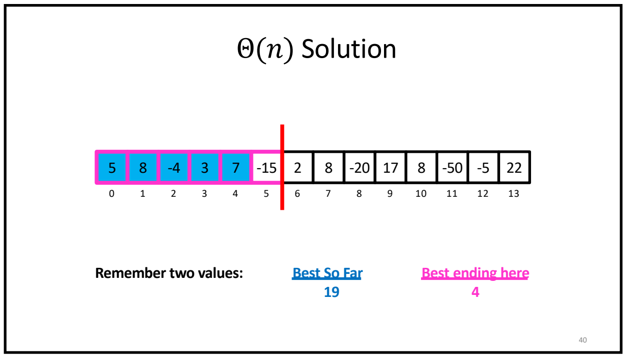
Remember two values: Best So Far 13 Best ending here 13

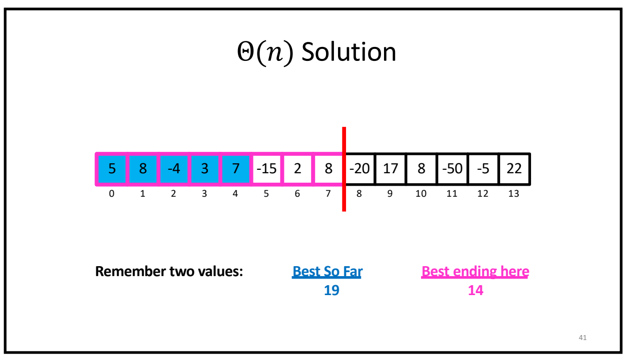
36

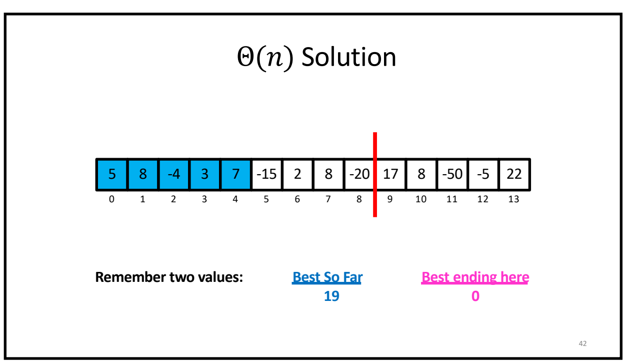

Θ(n) Solution
5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Remember two values:
Best So Far
13
Best ending here
9
37


Θ(n) Solution
5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Remember two values:
Best So Far
13
Best ending here
12
38


Θ(n) Solution
5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Remember two values:
Best So Far
19
Best ending here
19
39


Θ(n) Solution
5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Remember two values:
Best So Far
19
Best ending here
4
40


Θ(n) Solution
5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Remember two values:
Best So Far
19
Best ending here
14
41


Θ(n) Solution
5 8 -4 3 7 -15 2 8 -20 17 8 -50 -5 22
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Remember two values:
Best So Far
19
Best ending here
0
42

$\Theta(n)$ Solution

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Remember two values: Best So Far **19** Best ending here **17**

43

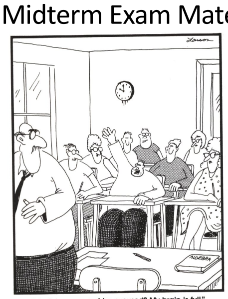
$\Theta(n)$ Solution

5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Remember two values: Best So Far **25** Best ending here **25**

44

End of Midterm Exam Materials!



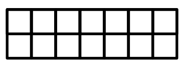
"Mr. Osborne, may I be excused? My brain is full."

45

Mid-Class Stretch

How many ways are there to tile a $2 \times n$ board with dominoes?

How many ways to tile this:



With these?