CS4102 Algorithms

Nate Brunelle

Fall 2017

Warm up (from practice exam)

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.

Cool Down

Give a $O(n)$ 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

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

Find Largest sum that spans the cut

Recursively Solve on Right
25

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

Find Largest sum that spans the cut
19

Recursively Solve on Right
25

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

Today’s Keywords

- Dynamic Programming
- Gerrymandering
- World Domination

CLRS Readings

- Chapter 15

Homeworks

- Hw5 released Thursday Oct. 19
  - Dynamic Programming
  - Programming assignment (use Python!)

Midterm

- Thursday October 19 in class
  - Covers all content through sorting
  - Take-home programming problem included
    - Submit by 11pm Tuesday Oct. 24

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  1. Identify recursive structure of the problem
  2. Select a good order for solving subproblems
     - Usually smallest problem first

DP Algorithms so far

- $2 \times n$ domino tiling (fibonacci)
- Log cutting
- Matrix Chaining
- Longest Common Subsequence
- Seam Carving

Domino Tiling

Tile(n):

Initialize Memory M

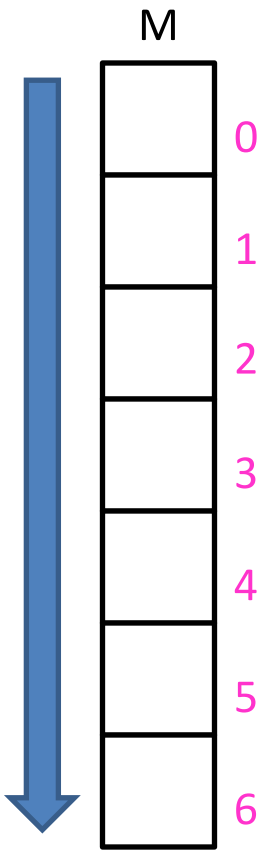
M[0] = 0

M[1] = 0

for i = 0 to n:

M[i] = M[i-1] + M[i-2]

return M[n]



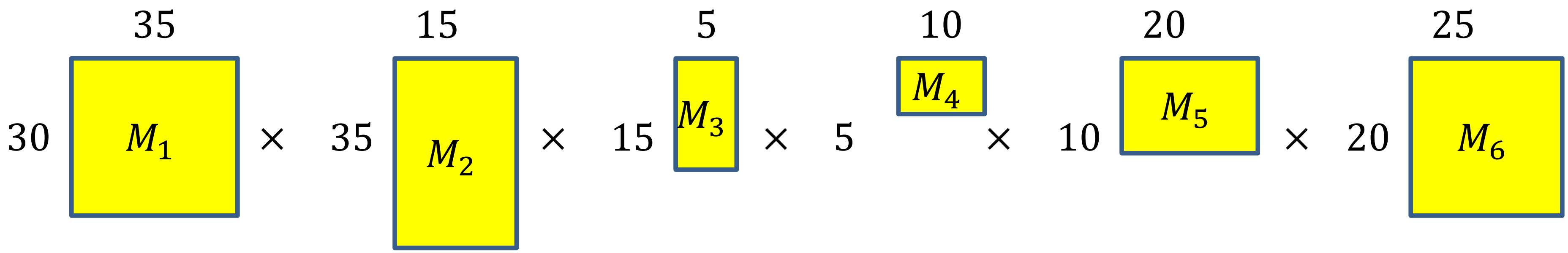
Log Cutting

Solve Smallest subproblem first

$$Cut(4) = \max \begin{cases} Cut(3) + P[1] \\ Cut(2) + P[2] \\ Cut(1) + P[3] \\ Cut(0) + P[4] \end{cases}$$

$Cut(i)$:	0										
Length:	0	1	2	3	4	5	6	7	8	9	10

Matrix Chaining



$$Best(i,j) = \min_{k=1}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

$j =$ 1	2	3	4	5	6	$i =$
0	15750	7875	9375	11875	15125	1
	0	2625	4375	7125	10500	2
		0	750	2500	5375	3
			0	1000	3500	4
				0	5000	5
					0	6

$$Best(1,6) = \min \begin{cases} Best(1,1) + Best(2,6) + r_1 r_2 c_6 \\ Best(1,2) + Best(3,6) + r_1 r_3 c_6 \\ Best(1,3) + Best(4,6) + r_1 r_4 c_6 \\ Best(1,4) + Best(5,6) + r_1 r_5 c_6 \\ Best(1,5) + Best(6,6) + r_1 r_6 c_6 \end{cases}$$

Logest Common Subsequence

$$LCS(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\ \max(LCS(i, j-1), LCS(i-1, j)) & \text{otherwise} \end{cases}$$

$Y =$ \ $X =$	0	A 1	T 2	C 3	T 4	G 5	A 6	T 7
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
T 1	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**
G 2	**0**	**0**	**1**	**1**	**1**	**2**	**2**	**2**
C 3	**0**	**0**	**1**	**2**	**2**	**2**	**2**	**2**
A 4	**0**	**1**	**1**	**2**	**2**	**2**	**3**	**3**
T 5	**0**	**1**	**2**	**2**	**3**	**3**	**3**	**4**
A 6	**0**	**1**	**2**	**2**	**3**	**3**	**4**	**4**

To fill in cell (i,j) we need cells $(i-1,j-1), (i-1,j), (i,j-1)$

Fill from Top->Bottom, Left->Right (with any preference)

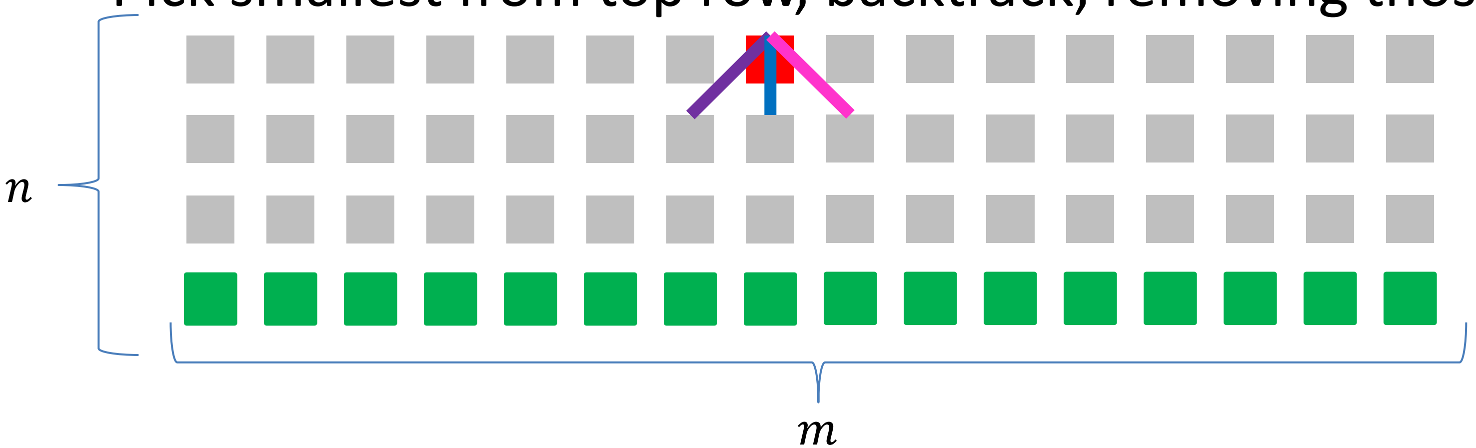
Seam Carving

Start from bottom of image (row 1), solve up to top

Initialize $S(1, k) = e(p_{1,k})$ for each pixel $p_{1,k}$

For $i > 2$ find $S(i, k) = \begin{cases} S(n-1, k-1) + e(p_{n,k}) \\ S(n-1, k) + e(p_{n,k}) \\ S(n-1, k+1) + e(p_{n,k}) \end{cases}$

Pick smallest from top row, backtrack, removing those pixels



Energy of the seam initialized to the energy of that pixel

Supreme Court Associate Justice Anthony Kennedy gave no sign that he has abandoned his view that extreme partisan gerrymandering might violate the Constitution. | Eric Thayer/Getty Images

Supreme Court eyes partisan gerrymandering

Anthony Kennedy is seen as the swing vote that could blunt GOP's map-drawing successes.

Gerrymandering

- Manipulating electoral district boundaries to favor one political party over others
- Coined in an 1812 Political cartoon

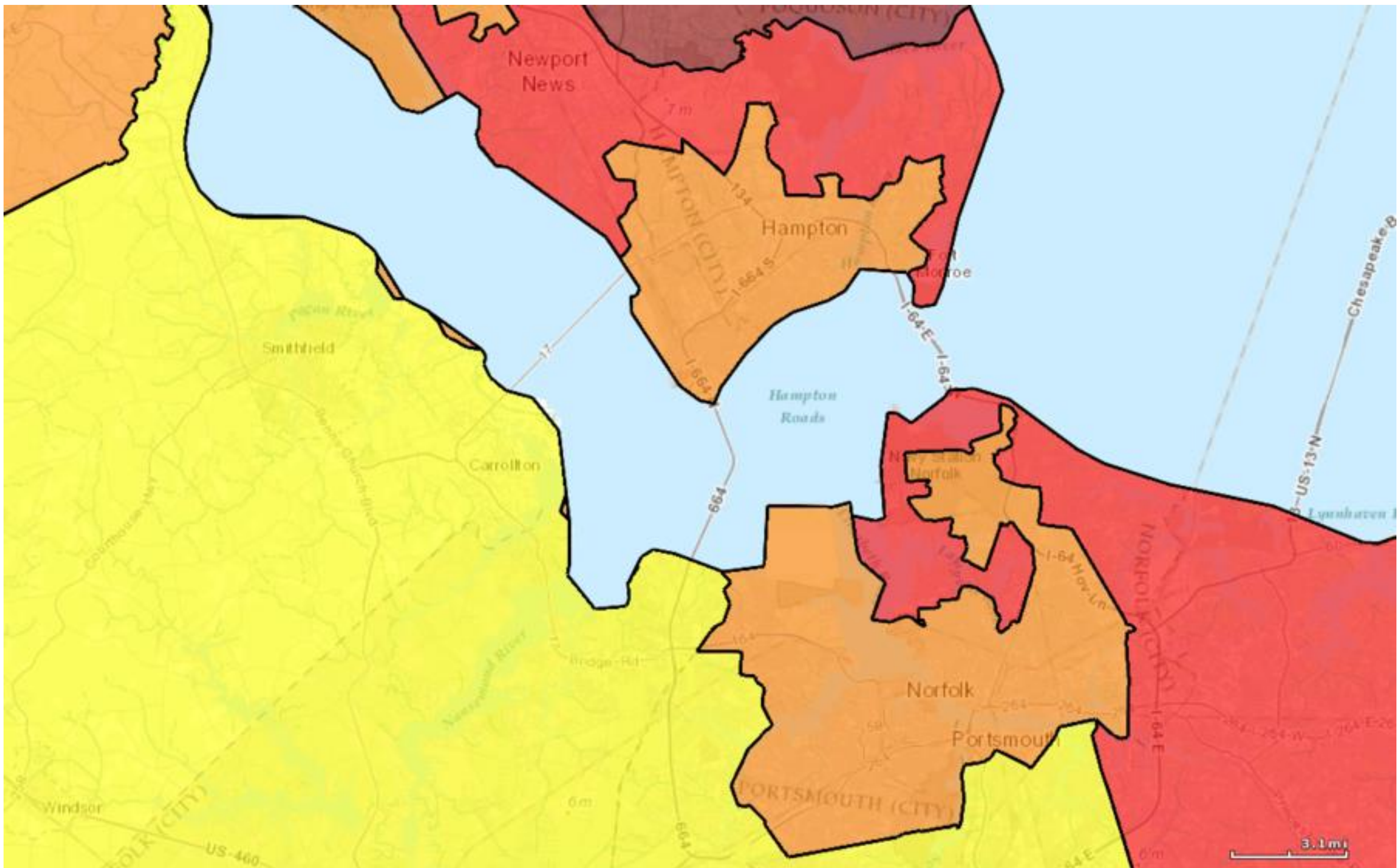
The Gerrymander

According to the Supreme Court

- Gerrymandering cannot be used to:
  - Disadvantage racial/ethnic/religious groups
- It can be used to:
  - Disadvantage political parties

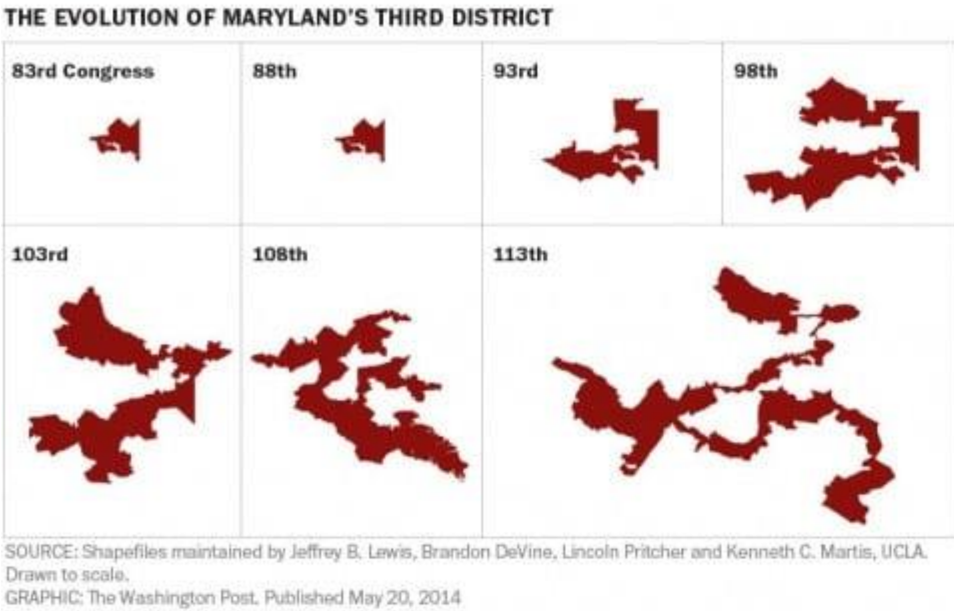
Gerrymandering Today

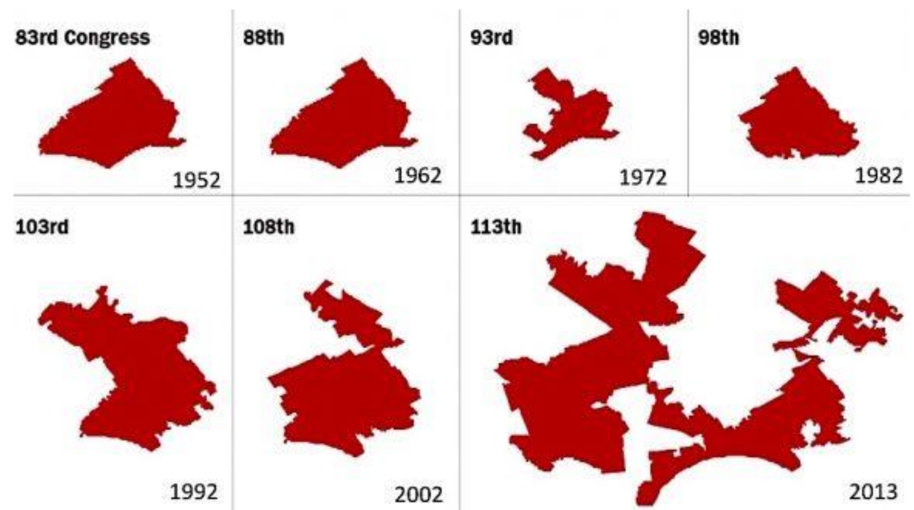
- Computers make it really effective



Gerrymandering Today

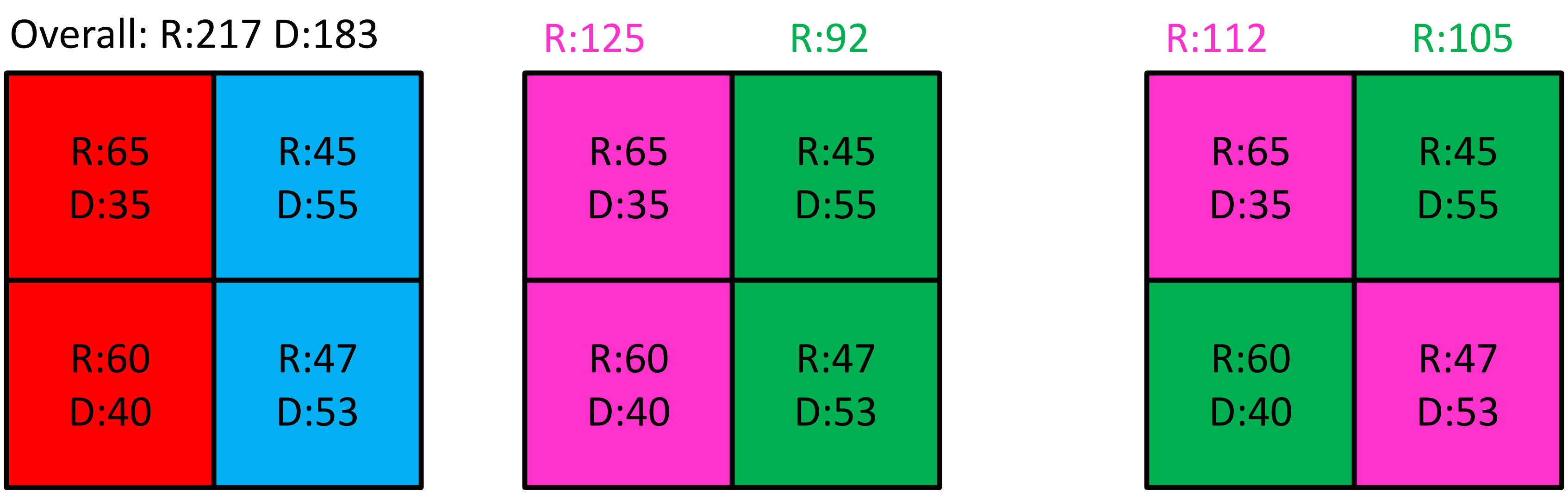
- Computers make it really effective





How does it work?

- States are broken into precincts
- All precincts have the same size
- We know voting preferences of each precinct
- Group precincts into districts to maximize the number of districts won by my party



Gerrymandering Problem Statement

- Given:
  - A list of precincts: $p_1, p_2, \ldots, p_n$
  - Each containing m voters
- Output:
  - Districts $D_1, D_2 \subset \{p_1, p_2, \ldots, p_n\}$
  - Where $|D_1| = |D_2|$
  - $R(D_1), R(D_2) > \frac{mn}{4}$
    - $R(D_i)$ gives number of "Regular Party" voters in D_1
    - $R(D_i) > \frac{\mathrm{mn}}{4}$ means D_i is majority "Regular Party"
  - "failure" if no such solution is possible

Consider the last precinct

D_1
k precincts
x voters for R

If we assign p_n to D_1

D_1
$k + 1$ precincts
$x + R(p_n)$ voters for R

Valid solution if:
$k + 1 = \frac{n}{2}$,
$x + R(p_n), y > \frac{mn}{4}$

After assigning the first $n - 1$ precincts

p_n

D_2
$n - k - 1$ precincts
y voters for R

If we assign p_n to D_2

D_2
$n - k$ precincts
$y + R(p_n)$ voters for R

Valid solution if:
$\mathrm{n} - k = \frac{n}{2}$,
$x, y + R(p_n) > \frac{mn}{4}$

Define Recursive Structure

$S(j, k, x, y) =$ True if among the first j precincts
k are assigned to D_1
x vote for R in D_1
y vote for R in D_2

4D Dynamic Programming!!!

Two ways to satisfy $S(j,k,x,y)$:

$S(j,k,x,y)$:
among the first j precincts
k are assigned to D_1
x vote for R in D_1
y vote for R in D_2

D_1
$k-1$ precincts
$x-R(p_j)$ voters for R

D_2
$j-k$ precincts
y voters for R

p_j

Then assign p_j to D_1

D_1
k precincts
x voters for R

D_2
$j-k$ precincts
y voters for R

OR

D_1
k precincts
$x$voters for R

D_2
$j-1-k$ precincts
$y-R(p_j)$ voters for R

p_j

Then assign p_j to D_2

$$S(j,k,x,y) = S(j-1,k-1,x-R(p_j),y) \vee S\left(j-1,k,x,y-R(p_j)\right)$$

Final Algorithm

$$S(j,k,x,y) = S\left(j-1,k-1,x-R(p_j),y\right) \vee S\left(j-1,k,x,y-R(p_j)\right)$$

Initialize $S(0,0,0,0) = \text{True}$
for $j = 1, \dots, n$:
  for $k = 1, \dots, \min(j, \frac{n}{2})$:
    for $x = 0, \dots, jm$:
      for $y = 0, \dots, jm$:
        $S(j,k,x,y) =$
          $S\left(j-1,k-1,x-R(p_j),y\right)$
          $\vee\, S\left(j-1,k,x,y-R(p_j)\right)$

Search for True entry at $S(n, \frac{n}{2}, > \frac{mn}{4}, > \frac{mn}{4})$

Run Time

$S(j,k,x,y) = S\left(j-1, k-1, x-R\left(p_j\right), y\right) \vee S\left(j-1, k, x, y-R\left(p_j\right)\right)$

Initialize $S(0,0,0,0) = \text{True}$

n for $j = 1, \ldots, n$:

$\frac{n}{2}$ for $k = 1, \ldots, \min(j, \frac{n}{2})$:

nm for $x = 0, \ldots, jm$:

nm for $y = 0, \ldots, jm$:

$$S(j,k,x,y) =$$
$$S\left(j-1, k-1, x-R\left(p_j\right), y\right)$$
$$\vee\, S\left(j-1, k, x, y-R\left(p_j\right)\right)$$

$\Theta(n^4 m^2)$

Search for True entry at $S(n, \frac{n}{2}, > \frac{mn}{4}, > \frac{mn}{4})$

$$\Theta(n^4 m^2)$$

- Runtime depends on the *value* of m, not *size* of m
- Run time is exponential in *size* of input
- Note: Gerrymandering is NP-Complete

$\Theta(n)$ 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

Begin here

Remember two values:

Best So Far

5

Best ending here

5

$\Theta(n)$ 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

13

Best ending here

13

$\Theta(n)$ 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** **13** **Best ending here** **9**

$\Theta(n)$ 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
13

Best ending here
12

$\Theta(n)$ Solution

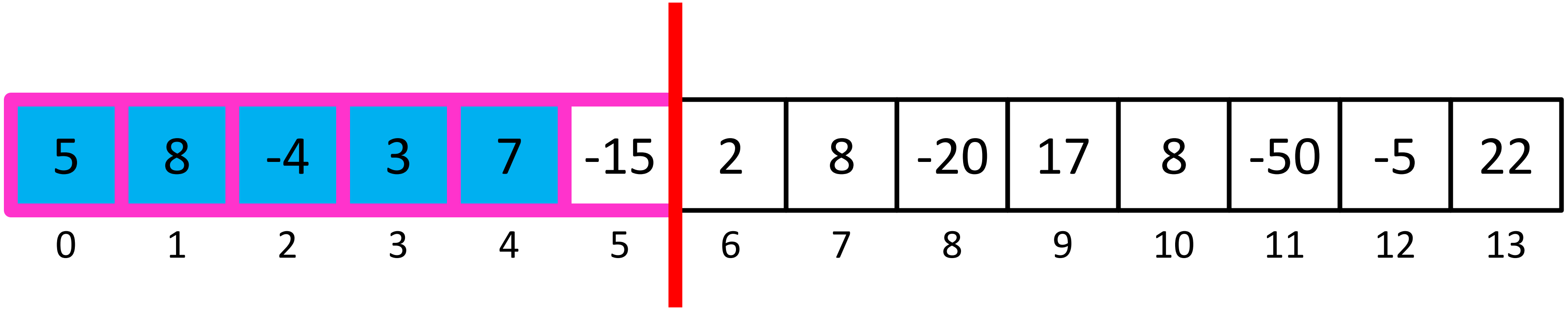
5	8	-4	3	7	-15	2	8	-20	17	8	-50	-5	22
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Remember two values:

Best So Far
19

Best ending here
19

$\Theta(n)$ Solution



Remember two values:

Best So Far
19

Best ending here
14

$\Theta(n)$ 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

14

$\Theta(n)$ 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
0

$\Theta(n)$ 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

$\Theta(n)$ 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