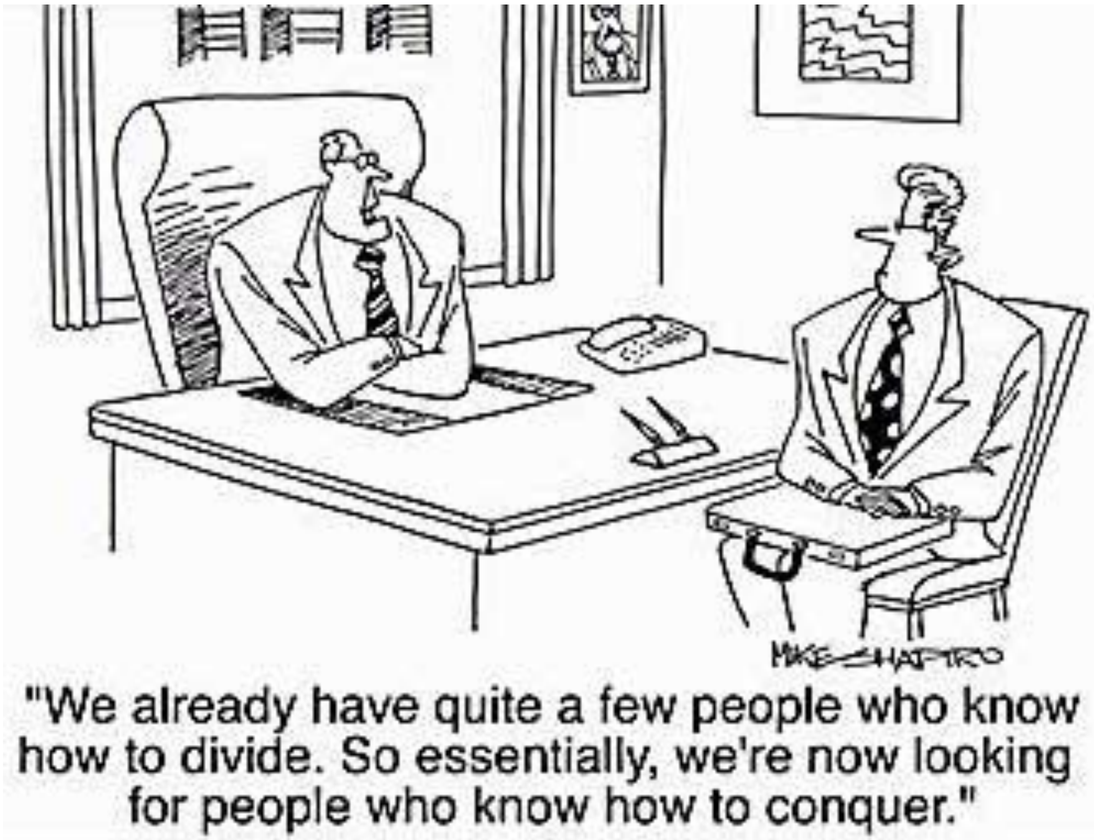
Data Structures and Algorithm Analysis (CSC317)



Divide and conquer

From previous class

Go over proofs for growth of functions (on the board)

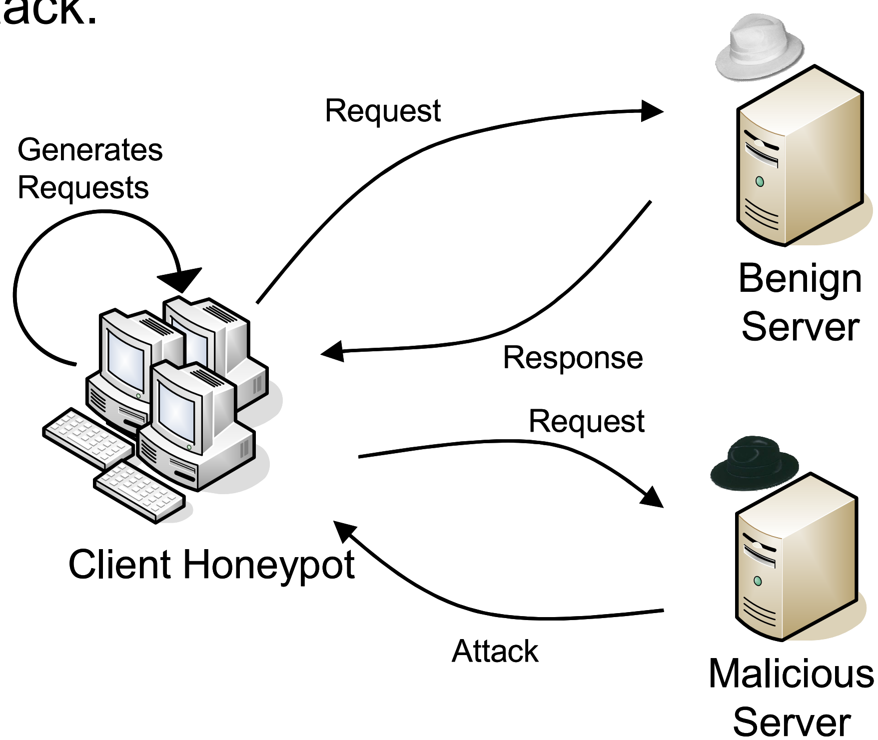
Goals

What kind of recurrences arise in algorithms and how do we solve more generally (than what we saw for merge sort)?

- More recurrence examples
- Run time not always intuitive, so need tools

Usefulness in recent applications

Application of divide-and-conquer algorithm paradigm to improve the detection speed of high interaction client honeypots. Seifert et al. 2008. "…one needs to be able to find malicious servers on a network… Client honeypots are the new emerging technology that can perform such searches… they are faced with crawling the Internet with its millions of servers. Finding a malicious server might be similar to finding a needle in a haystack."



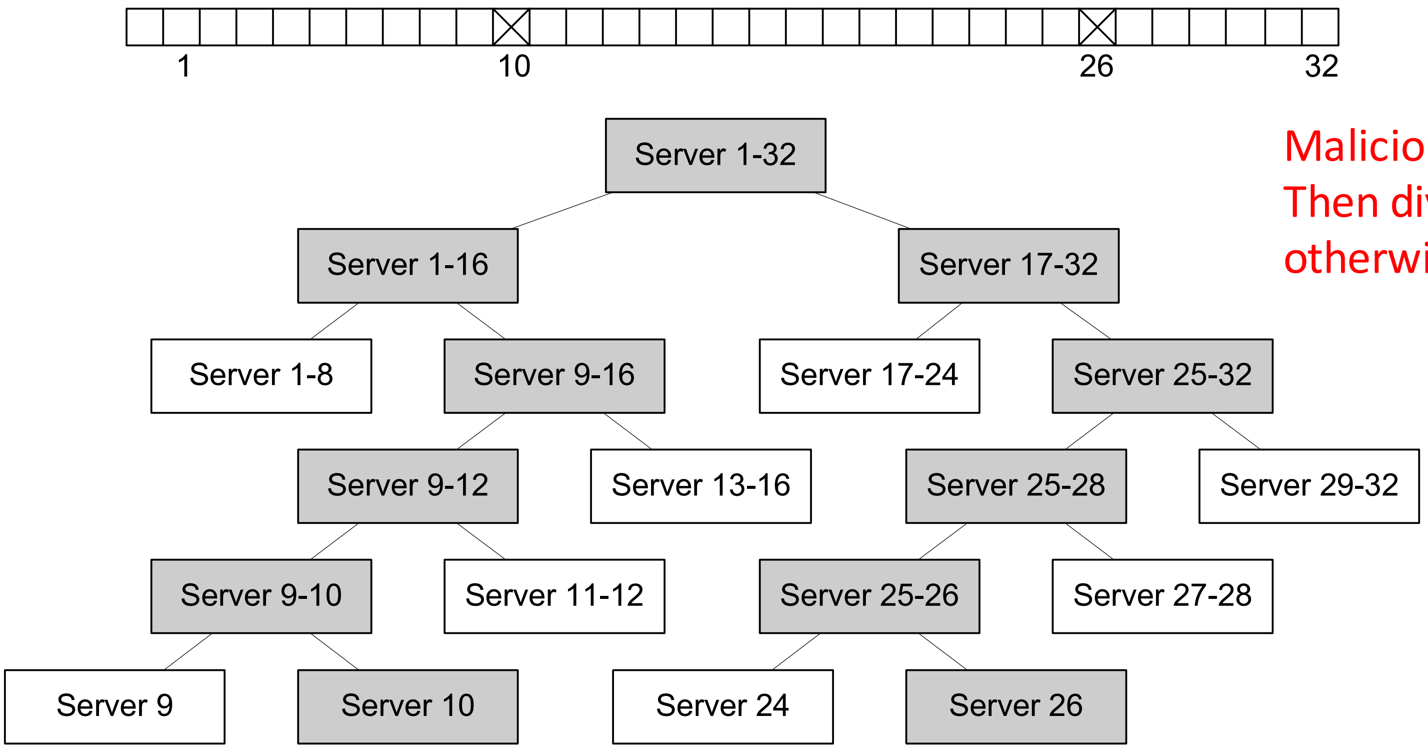
Usefulness in recent applications

Application of divide-and-conquer algorithm paradigm to improve the detection speed of high interaction client honeypots. Seifert et al. 2008.

- Sequential: Make server request one by one to a large set of servers, and detect malicious
- Detect malicious by making server requests in parallel for set of servers in a buffer. Each time mark a given set as malicious or not, but can't determine server identity without manual check
- Use the divide and conquer approach

Usefulness in recent applications

Application of divide-and-conquer algorithm paradigm to improve the detection speed of high interaction client honeypots. Seifert et al. 2008.



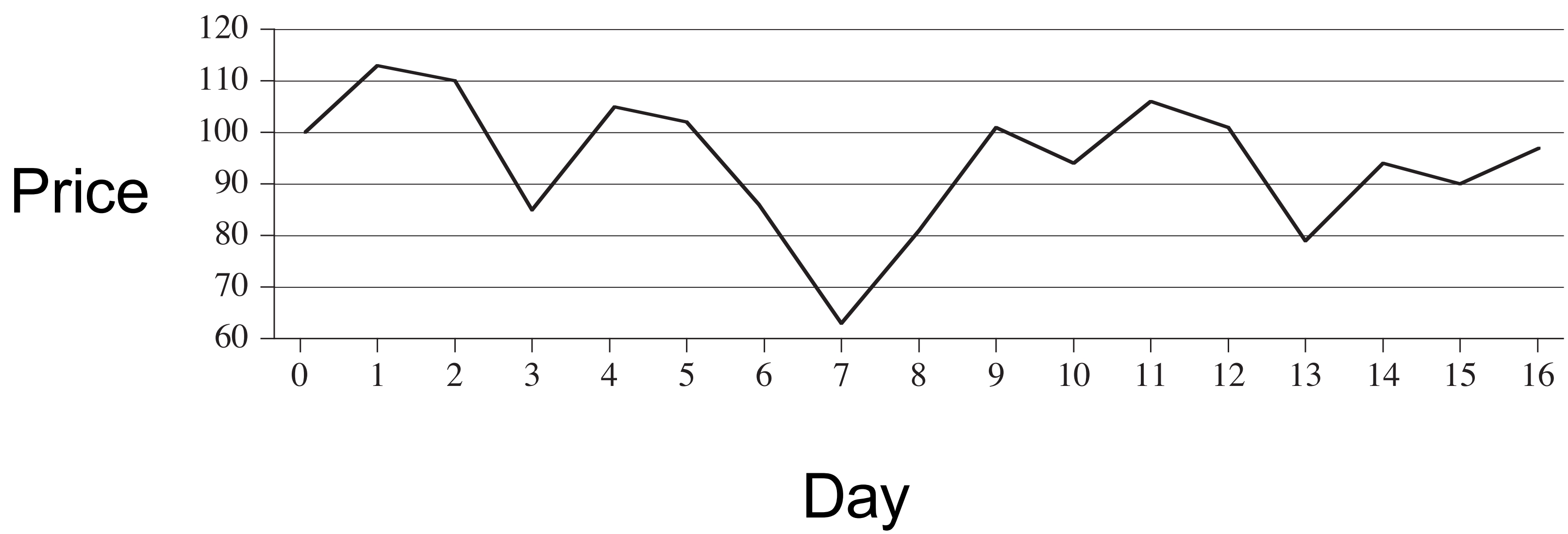
Malicious detected
Then divide in half;
otherwise stop

Figure 7: Divide-and-conquer Algorithm Example

More Detailed Algorithm Examples

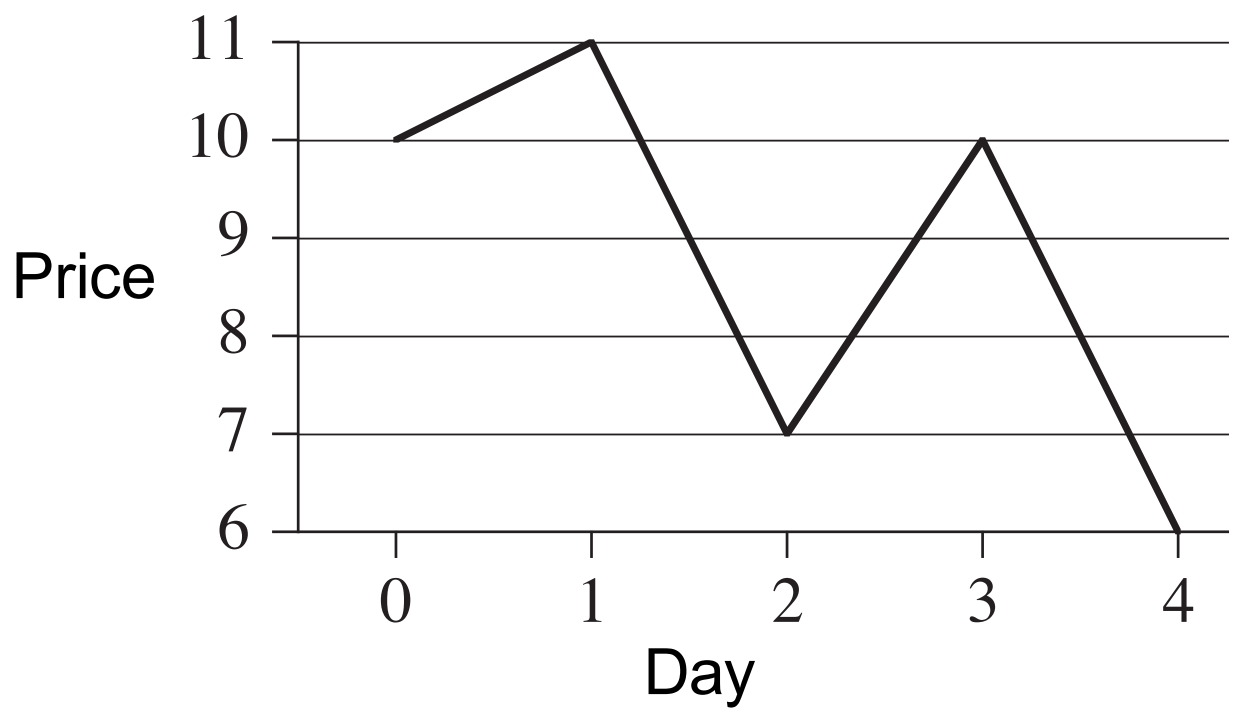
Max Subarray Problem

Problem: Can buy stock once, sell stock once. Want to maximize profit; allowed to look into the future



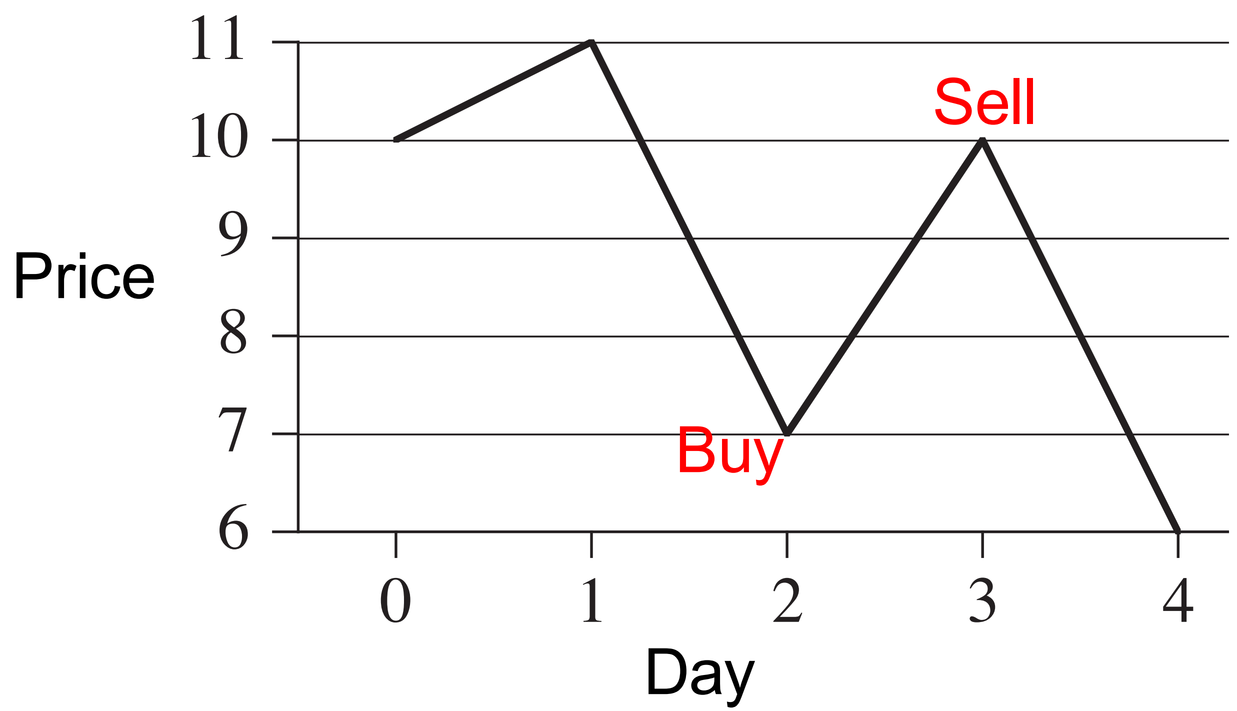
Max Subarray Problem

Problem: Can buy stock once, sell stock once. Want to maximize profit; allowed to look into the future



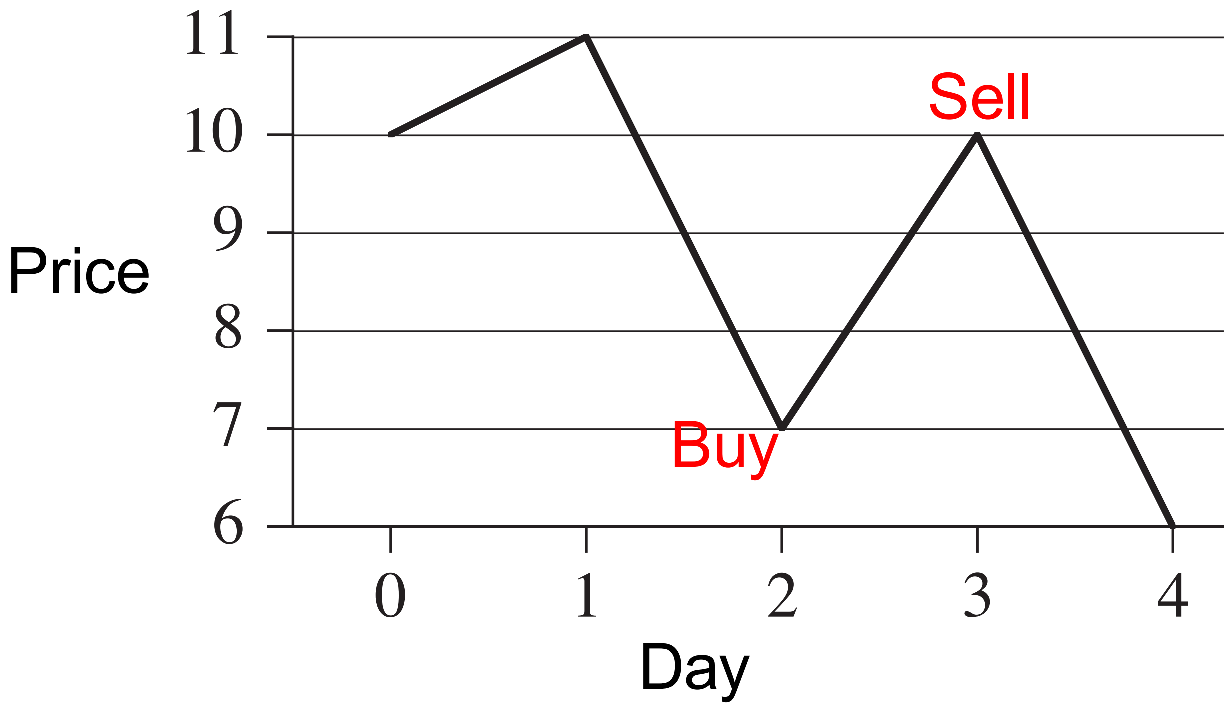
Max Subarray Problem

Problem: Can buy stock once, sell stock once. Want to maximize profit; allowed to look into the future



Max Subarray Problem

Problem: Can buy stock once, sell stock once. Want to maximize profit; allowed to look into the future. Complexity?



Max Subarray Problem

Brute force: Try every possible pair of buy and sell dates:

$$\binom{n}{2} = \frac{n!}{(n-2)!2!} = \frac{n(n-1)(n-2)!}{(n-2)!2!} = \frac{n(n-1)}{2} = \boxed{\Theta(n^2)}$$

Max Subarray Problem

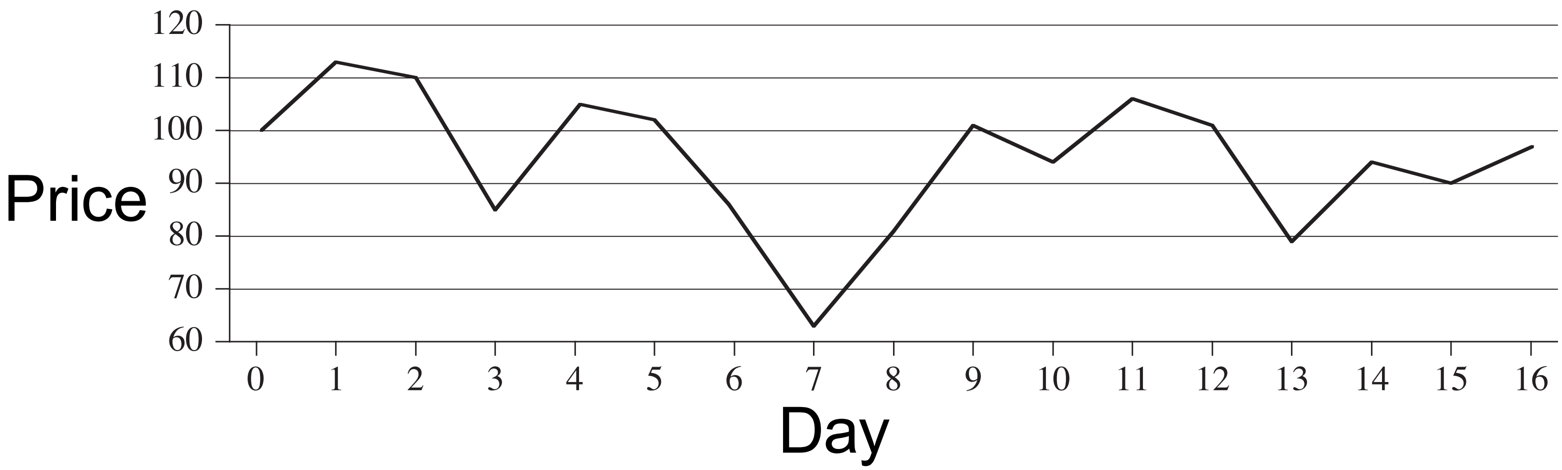
Brute force: Try every possible pair of buy and sell dates:

$$\binom{n}{2} = \frac{n!}{(n-2)!2!} = \frac{n(n-1)(n-2)!}{(n-2)!2!} = \frac{n(n-1)}{2} = \boxed{\Theta(n^2)}$$

Can we do better?

Max Subarray Problem

Brute force: Can we do better? Try to reframe as greatest sum of any contiguous array

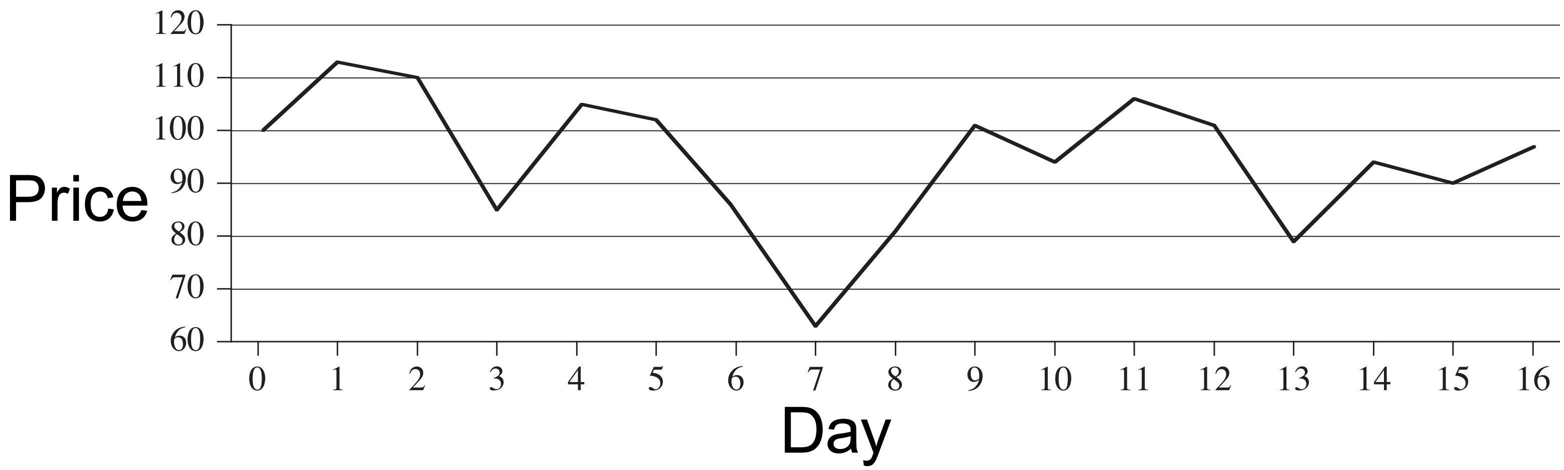


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

best contiguous sum representing gain from buy to sell!

Max Subarray Problem

Brute force: Can we do better? Try to reframe as greatest sum of any contiguous array

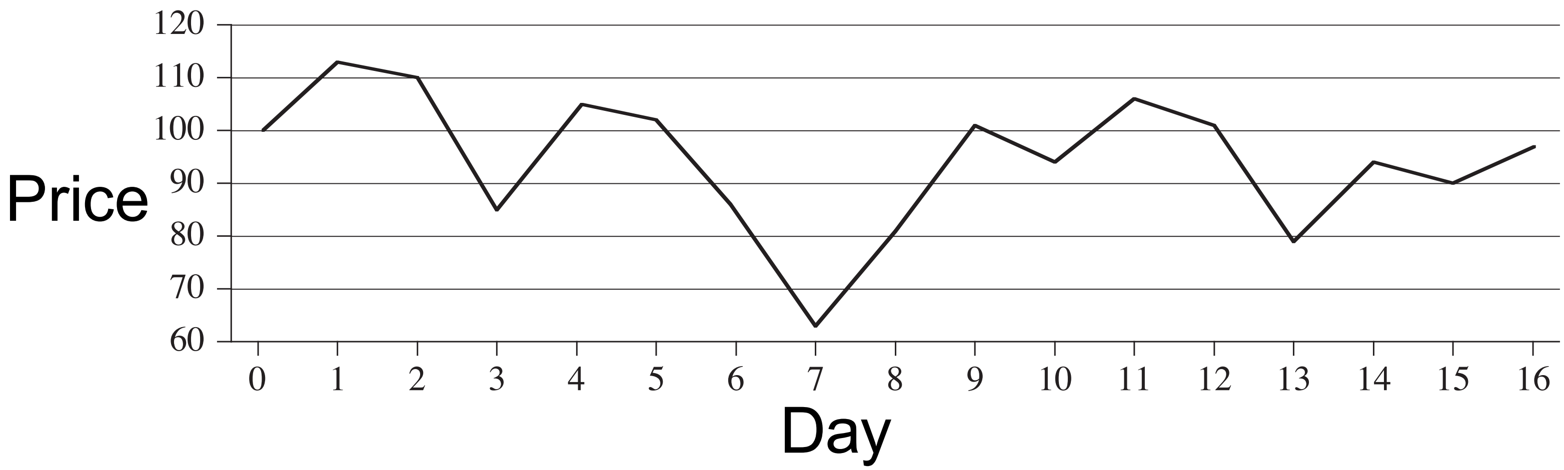


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

maximum subarray

Max Subarray Problem

Try to reframe as greatest sum of any contiguous Array. **Efficiency?**

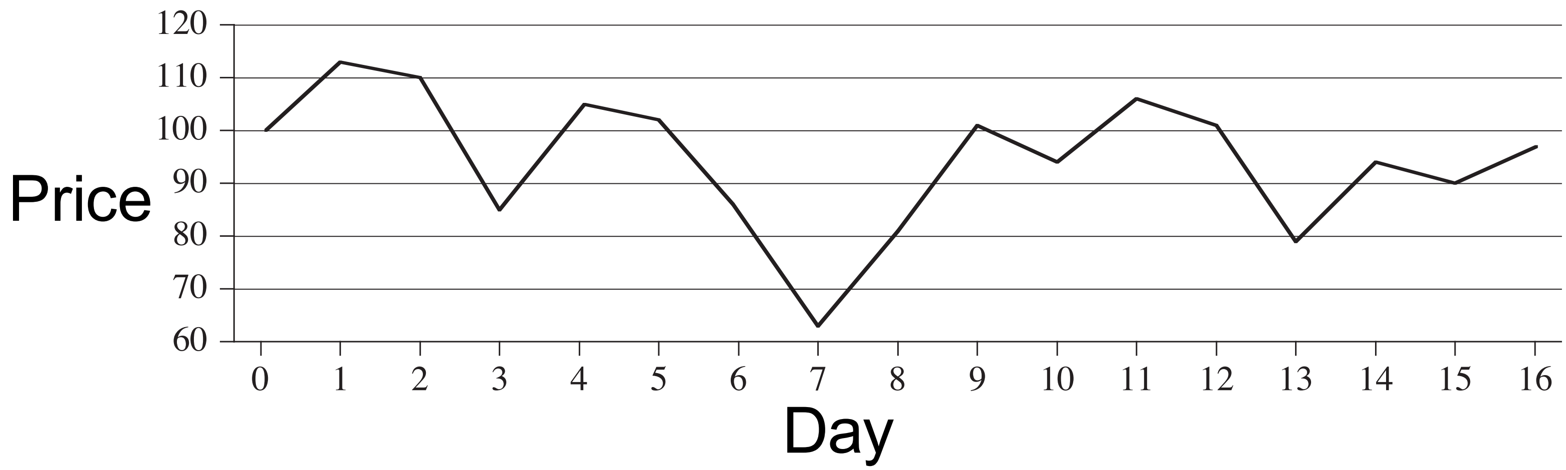


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

maximum subarray

Max Subarray Problem

Try to reframe as greatest sum of any contiguous array. **Efficiency? It’s still brute force**



	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

maximum subarray

Max Subarray Problem

Try to reframe as greatest sum of any contiguous array. **If all the array values were positive?**

Max Subarray Problem

Try to reframe as greatest sum of any contiguous array. **If all the array values were positive?**

A = [1 10 12 13 23 33 2]

Max Subarray Problem

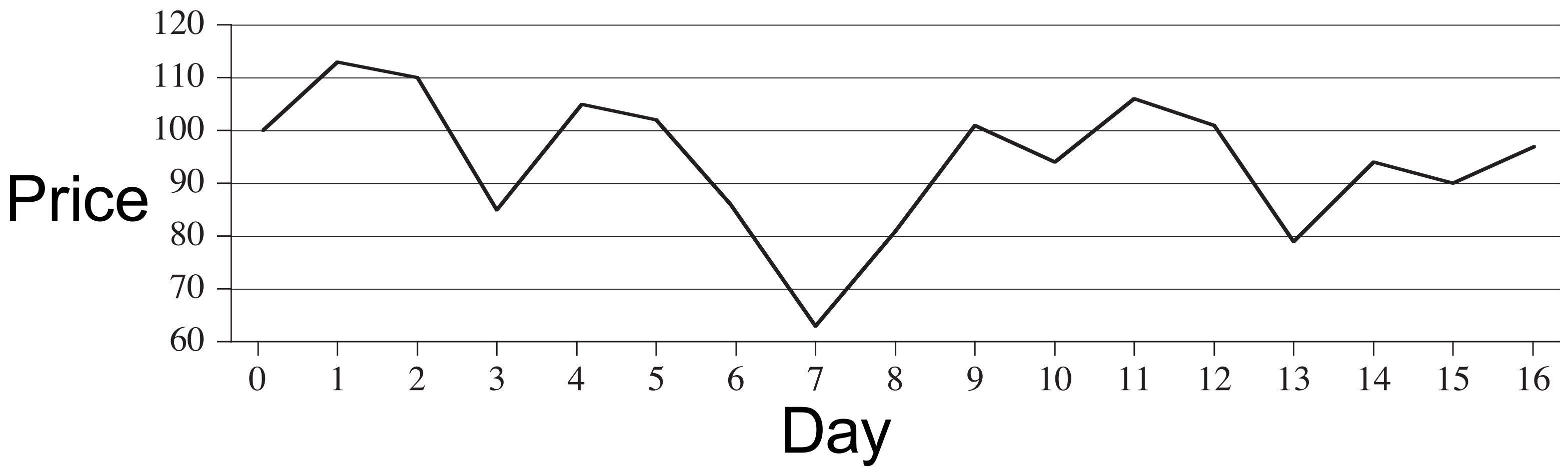
Try to reframe as greatest sum of any contiguous array. **If all the array values were positive?**

A = [1 10 12 13 23 33 2]

Not interesting – summing all array values gives the max…

Max Subarray Problem

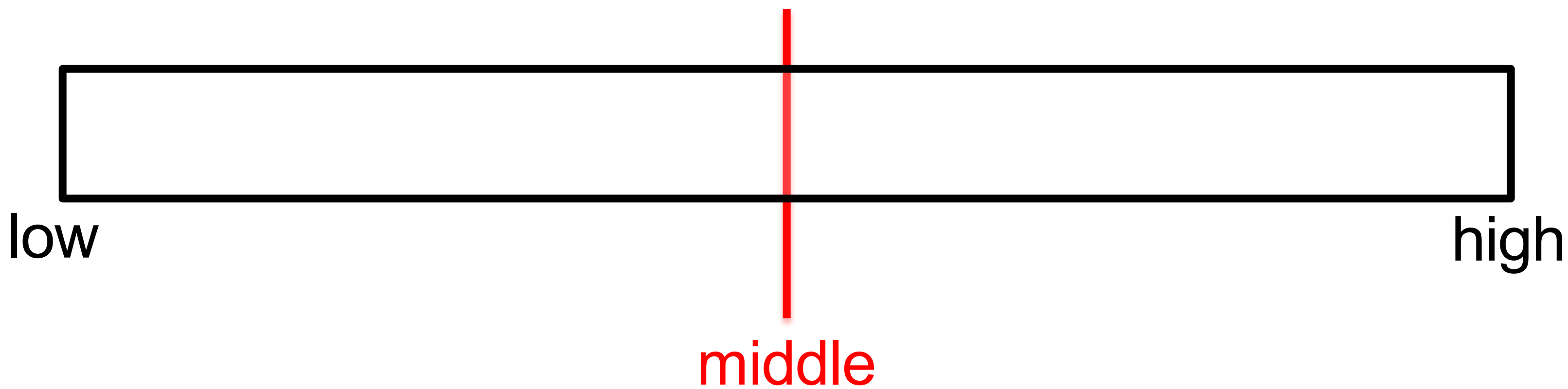
For positive and negative values, it's **still brute force.**
Divide and Conquer?



	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

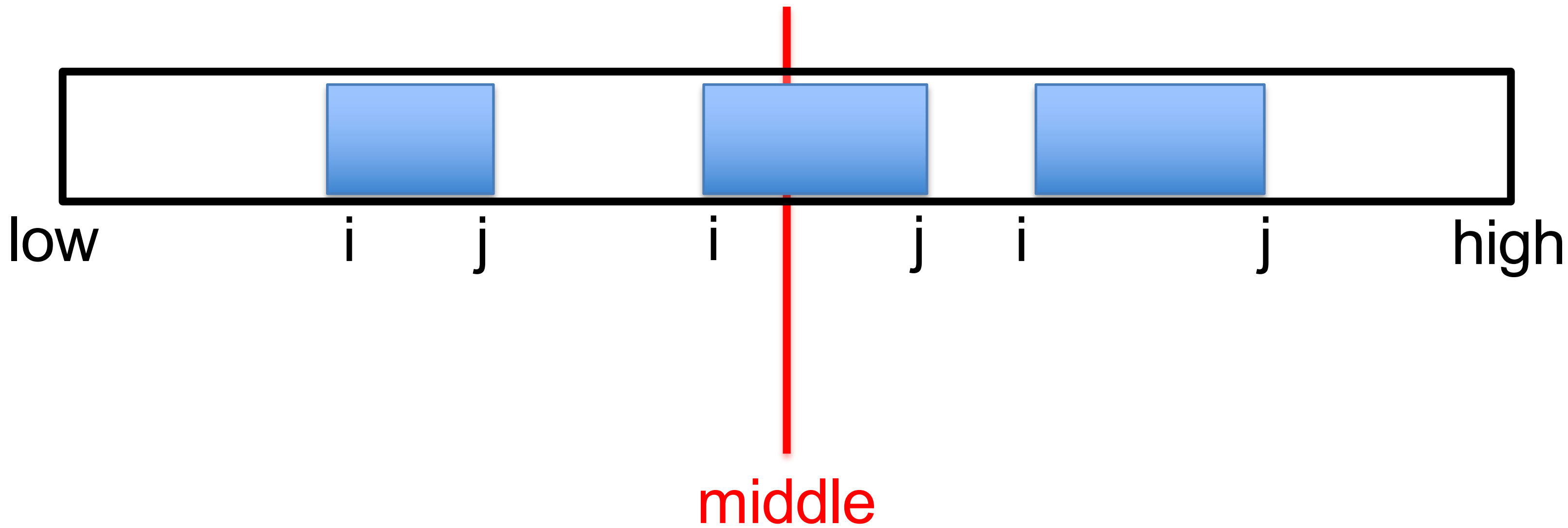
Max Subarray Problem

Divide and conquer approach
Where can max subarray be?



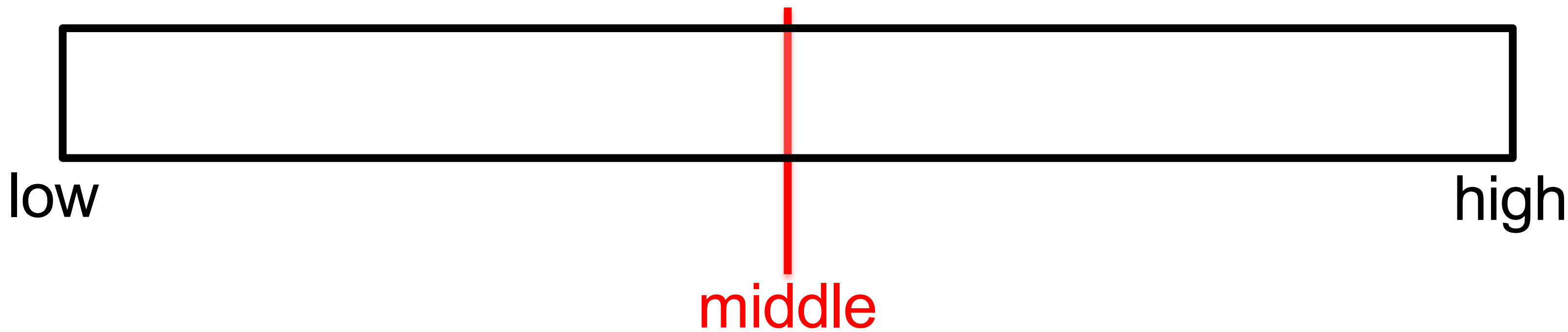
Max Subarray Problem

Divide and conquer approach
Where can max subarray be?



Max Subarray Problem

Divide and conquer approach



1. **Divide** subarray into two equal size subarrays, A[low..mid] and A[mid+1..high]
2. **Conquer**, finding max of subarrays A[low..mid] and A[mid+1..high]
3. **Combine**, finding best solution of:
   a. the two solutions found in conquer step
   b. solution of subarray crossing the midpoint

Max Subarray Problem

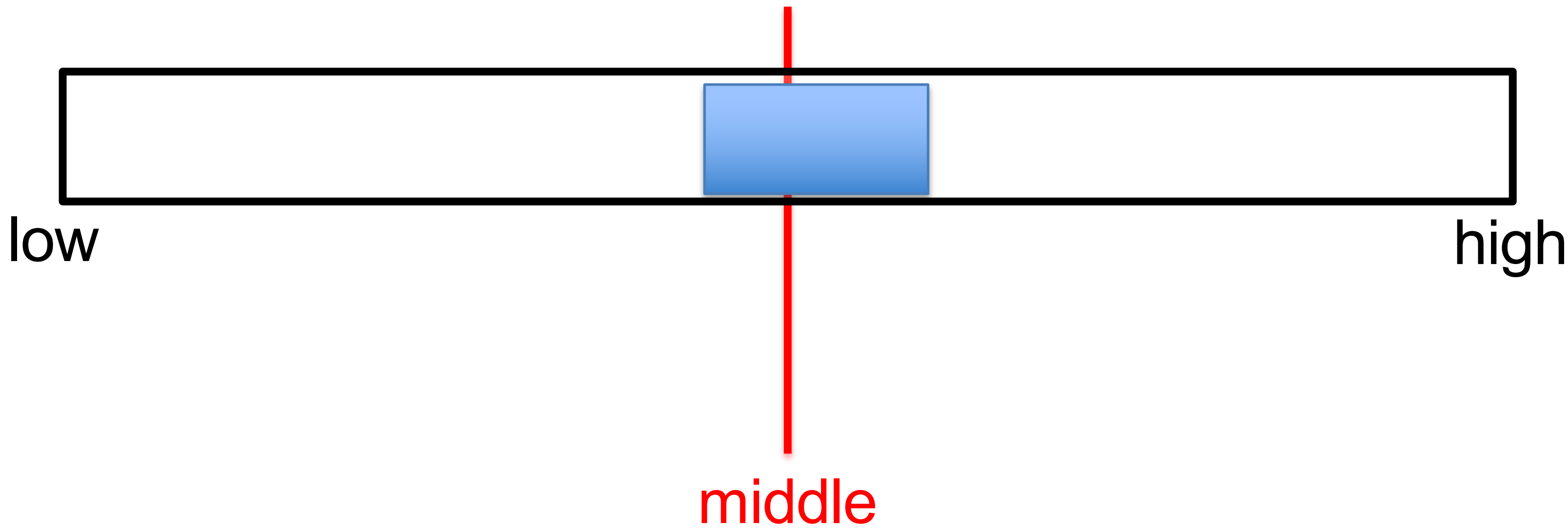
Divide and conquer approach

Keep recursing until low=high (one element left)-

1. Divide subarray into two equal size subarrays, A[low..mid] and A[mid+1..high]
2. Conquer, finding max of subarrays A[low..mid] and A[mid+1..high]
3. Combine, finding best solution of:
   a. the two solutions found in conquer step
   b. **solution of subarray crossing the midpoint**

Max Subarray Problem

Algorithm for max subarray crossing midpoint?



Max Subarray Problem

Subarray crossing midpoint

[-16 -23 18 20 | -7 12 -5 -22]

- **Start from middle**
- **Traverse to left until get maximum sum (?)**
- **Traverse to right until get maximum sum (?)**
- **Return total left and right sum (?)**

Complexity?

Max Subarray Problem

Subarray crossing midpoint

[-16 -23 18 20 -7 12 -5 -22]

- **Start from middle**
- **Traverse to left until get maximum sum (?)**
- **Traverse to right until get maximum sum (?)**
- **Return total left and right sum (?)**

Complexity? $\Theta(n)$

Max Subarray Problem

Subarray crossing midpoint

[-16 -23 18 20 | -7 12 -5 -22]

FIND-MAX-CROSSING-SUBARRAY$(A, low, mid, high)$

```
left-sum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
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 (max-left, max-right, left-sum + right-sum)
```

$\Theta(n)$

Max Subarray Problem

Divide and conquer approach: full example:

[-16 -23 18 20 -7 12 -5 -22]

On the board…

Max Subarray Problem

Divide and conquer approach: example:

[-16 -23 18 20 -7 12 -5 -22]

On the board…

Max Subarray Problem

Costs:

1. **Divide:** $\Theta(1)$

2. **Conquer:** $2T(\frac{n}{2})$

3. **Combine:** $\Theta(n) + \Theta(1)$

Subarray crossing

Comparisons

Max Subarray Problem

Costs:

1. **Divide:** $\Theta(1)$

2. **Conquer:** $2T(\frac{n}{2})$

3. **Combine:** $\Theta(n) + \Theta(1)$

Subarray crossing

Comparisons

Total: $T(n) = 2T(\frac{n}{2}) + \Theta(n) = ?$

Max Subarray Problem

Costs:

1. **Divide:** $\Theta(1)$

2. **Conquer:** $2T(\frac{n}{2})$

3. **Combine:** $\Theta(n) + \Theta(1)$

Subarray crossing

Comparisons

Total: $T(n) = 2T(\frac{n}{2}) + \Theta(n) = \Theta(n \log n)$

Like merge sort….

Classical example: matrix multiplication

$$\begin{bmatrix} a_{11} a_{12} a_{13} a_{14} \\ a_{21} a_{22} a_{23} a_{24} \\ a_{31} a_{32} a_{33} a_{34} \\ a_{41} a_{42} a_{43} a_{44} \end{bmatrix} \begin{bmatrix} b_{11} b_{12} b_{13} b_{14} \\ b_{21} b_{22} b_{23} b_{24} \\ b_{31} b_{32} b_{33} b_{34} \\ b_{41} b_{42} b_{43} b_{44} \end{bmatrix} = \begin{bmatrix} c_{11} c_{12} c_{13} c_{14} \\ c_{21} c_{22} c_{23} c_{24} \\ c_{31} c_{32} c_{33} c_{34} \\ c_{41} c_{42} c_{43} c_{44} \end{bmatrix}$$

Classical example: matrix multiplication

$$\begin{bmatrix} a_{11}a_{12}a_{13}a_{14} \\ a_{21}a_{22}a_{23}a_{24} \\ a_{31}a_{32}a_{33}a_{34} \\ a_{41}a_{42}a_{43}a_{44} \end{bmatrix} \begin{bmatrix} b_{11}b_{12}b_{13}b_{14} \\ b_{21}b_{22}b_{23}b_{24} \\ b_{31}b_{32}b_{33}b_{34} \\ b_{41}b_{42}b_{43}b_{44} \end{bmatrix} = \begin{bmatrix} c_{11}c_{12}c_{13}c_{14} \\ c_{21}c_{22}c_{23}c_{24} \\ c_{31}c_{32}c_{33}c_{34} \\ c_{41}c_{42}c_{43}c_{44} \end{bmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41};$$

$$\ldots$$

Classical example: matrix multiplication

$$\begin{bmatrix} a_{11}a_{12}a_{13}a_{14} \\ a_{21}a_{22}a_{23}a_{24} \\ a_{31}a_{32}a_{33}a_{34} \\ a_{41}a_{42}a_{43}a_{44} \end{bmatrix}\begin{bmatrix} b_{11}b_{12}b_{13}b_{14} \\ b_{21}b_{22}b_{23}b_{24} \\ b_{31}b_{32}b_{33}b_{34} \\ b_{41}b_{42}b_{43}b_{44} \end{bmatrix} = \begin{bmatrix} c_{11}c_{12}c_{13}c_{14} \\ c_{21}c_{22}c_{23}c_{24} \\ c_{31}c_{32}c_{33}c_{34} \\ c_{41}c_{42}c_{43}c_{44} \end{bmatrix}$$

n by n matrix

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

- Run time?

Classical example: matrix multiplication

$$\begin{bmatrix} a_{11}a_{12}a_{13}a_{14} \\ a_{21}a_{22}a_{23}a_{24} \\ a_{31}a_{32}a_{33}a_{34} \\ a_{41}a_{42}a_{43}a_{44} \end{bmatrix} \begin{bmatrix} b_{11}b_{12}b_{13}b_{14} \\ b_{21}b_{22}b_{23}b_{24} \\ b_{31}b_{32}b_{33}b_{34} \\ b_{41}b_{42}b_{43}b_{44} \end{bmatrix} = \begin{bmatrix} c_{11}c_{12}c_{13}c_{14} \\ c_{21}c_{22}c_{23}c_{24} \\ c_{31}c_{32}c_{33}c_{34} \\ c_{41}c_{42}c_{43}c_{44} \end{bmatrix}$$

n by n matrix

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

- Run time? $O(n^3)$

Answer: Naïve implementation

Classical example: matrix multiplication

Square-Matrix-Multiply(A,B)

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

1. n = A.rows
2. Let C be a new n by n matrix
3. **For** i=1 to n
4.   **For** j=1 to n
5.   cij = 0
6.   **For** k=1 to n
7.   cij = cij+ aik bkj
8. **Return** C

$O(n^3)$

Classical example: matrix multiplication

- Run time?

Answer: Naïve implementation $O(n^3)$

Can we do better? (next class; divide and conquer approaches)