CS 457, Fall 2016

Drexel University, Department of Computer Science

Lecture 5

What we have learned so far…

- Asymptotic notation
- Algorithm analysis (correctness and running time)
  - Loop invariants and induction
  - Running time as a function of n
  - Insertion Sort, Merge Sort, Quicksort,…
- Divide-and-Conquer
  - Running time as a recurrence equation
  - Three methods for solving recurrence equations

Running Time and Recurrence Equations

- Recurrence equation for divide and conquer algorithms:
  - $T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) & \text{otherwise} \end{cases}$

- Three methods for approaching such recurrence equations
  - Substitution
  - Recursion-tree
  - Master theorem

Substitution Method

1. Guess the form of the solution
2. Use mathematical induction to find the constants and show that it works

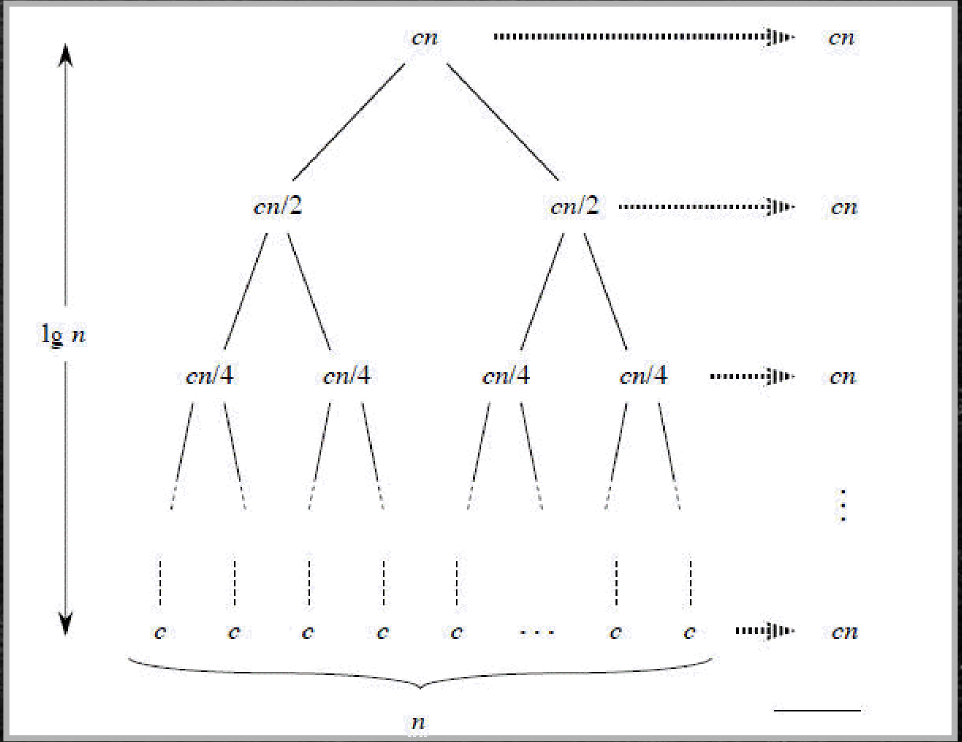
E.g., $T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(n) = 2T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$

Why wouldn't this work for $\boldsymbol{T(n) = cn}$ as well? (verify it!)

$$T(n) \leq 2[c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)] + n \;\Rightarrow$$
$$T(n) \leq cn \log(n/2) + n \;\Rightarrow$$
$$T(n) \leq cn \log n - cn \log 2 + n \;\Rightarrow$$
$$T(n) \leq cn \log n - cn + n \;\Rightarrow$$
$$T(n) \leq cn \log n \;\Rightarrow$$

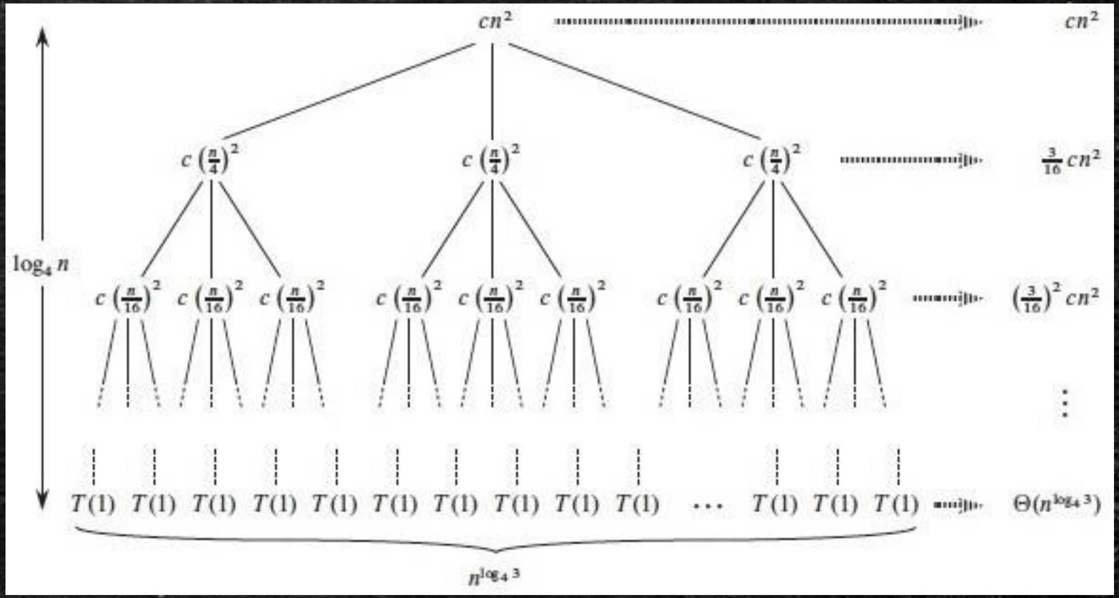
Recursion-Tree Method

- Recurrence equation for Merge Sort
  - $T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{otherwise} \end{cases}$



Recursion-Tree Method

- Recurrence equation
  - $T(n) = \begin{cases} \Theta(1) & \text{if } n \le 1 \\ 3T\left(\left\lfloor\frac{n}{4}\right\rfloor\right) + \Theta(\mathrm{n}^2) & \text{otherwise} \end{cases}$



Master Theorem

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the non-negative integers by the recurrence:

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

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a-\varepsilon})$ for some constant ε, then $T(n) = \Theta\left(n^{\log_b a}\right)$
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
3. If $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for some constant ε, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n ,then $T(n) = \Theta(f(n))$

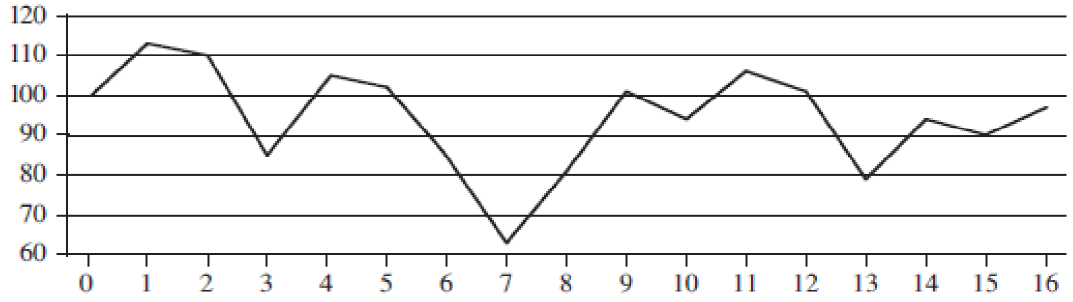
Today's Lecture

- The maximum subarray problem

- Quicksort
  - Correctness
  - Running Time

Maximum Subarray Problem

- Input: $\boldsymbol{n}$ price points
- Output: $(\boldsymbol{t_b}, \boldsymbol{t_s})$ s.t. $\boldsymbol{0 \leq t_b < ts \leq n}$, and $\boldsymbol{p(t_s) - p(t_b)}$ is maximized

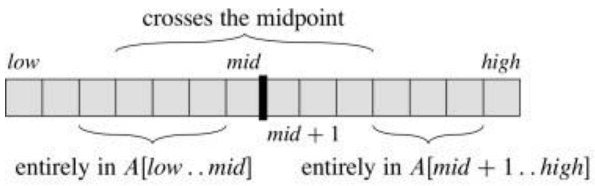


Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

1. Brute force running time?
2. Improvements?
3. Divide-and-Conquer?

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)
```



(a)

Maximum Subarray Problem

Divide-and-conquer procedure for 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)
```

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

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

Quicksort

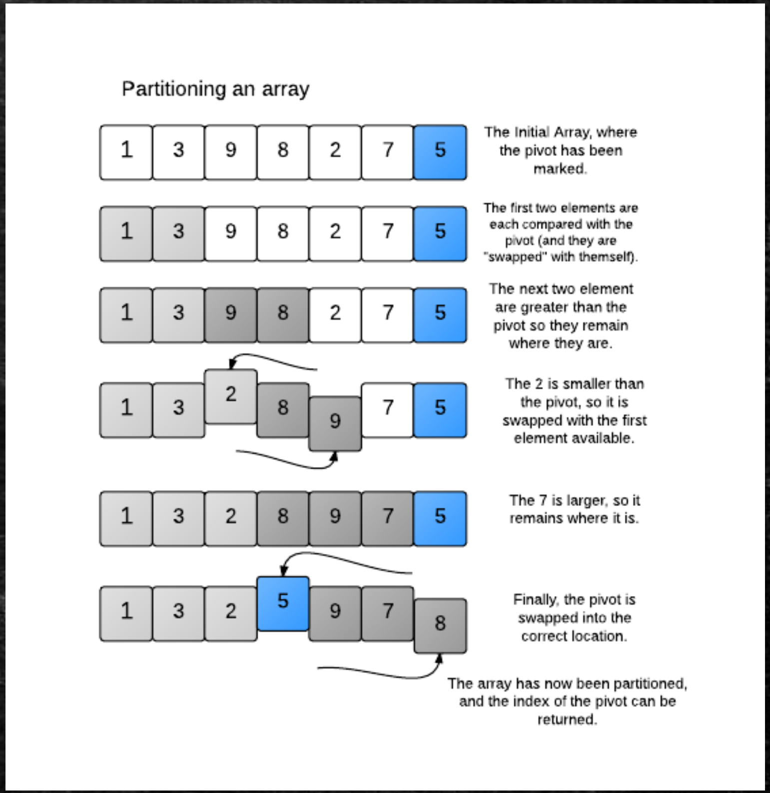
QUICKSORT (A, p, r)

```
1.   if p < r                          // Check for base case
2.        q = PARTITION(A, p, r)        // Divide step
3.        QUICKSORT (A, p, q - 1)       // Conquer step.
4.        QUICKSORT (A, q + 1, r)       // Conquer step.
```

Quicksort

PARTITION (*A*, *p*, *r*)

```
x = A[r]
i = p – 1
for  j = p  to  r – 1
        if  A[ j] <= x
                i = i + 1
                exchange A[i] with A[j]
exchange A[i+1] with A[r]
return i+1
```



Quicksort (Correctness)

PARTITION (*A*, *p*, *r*)

```
1.    x = A[r]
2.    i = p – 1
3.    for j = p to r – 1
4.        if A[ j] <= x
5.            i = i + 1
6.            exchange A[i] with A[j]
7.    exchange A[i+1] with A[r]
8.    return i+1
```

<u>Loop Invariant:</u>

At the beginning of each iteration of the loop of lines 3-6:

1. If $k \in [p, i]$, then $A[k] \leq x$
2. If $k \in [i+1, j-1]$, then $A[k] > x$
3. If $k = r$, then $A[k] = x$

Quicksort (Running Time)

QUICKSORT (*A*, *p*, *r*)

```
if p < r                              // Check for base case
    q = PARTITION(A, p, r)            // Divide step
    QUICKSORT (A, p, q - 1)           // Conquer step.
    QUICKSORT (A, q + 1, r)           // Conquer step.
```

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(q) + T(n - q - 1) + \Theta(n) & \text{otherwise} \end{cases}$$