CS 457, Fall 2019

Drexel University, Department of Computer Science

Lecture 3

First Homework

- Due: Monday 10/7 (by midnight)
  - Late days
- No use of solutions available online!
- Collaboration allowed (just for this problem set)
- Typeset your solutions (LaTeX recommended but not required)
- Dan's office hours are Tuesday 10am to noon
- Class recitation: Friday 1-3pm in room 1103 (note room change)
- Email Dan (with me cc:ed) with any homework-related questions
- Gradescope accounts

Insertion Sort (Running Time)

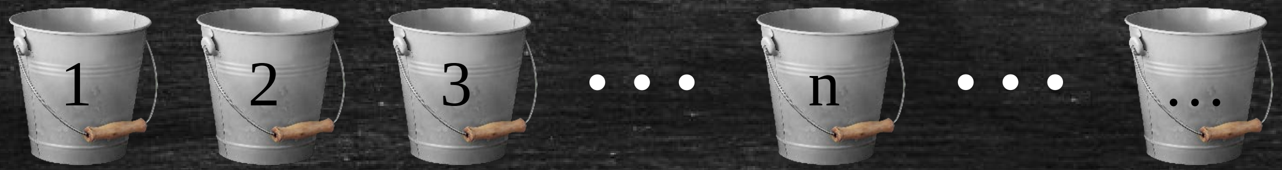
INSERTION_SORT (*A*)		COST	TIMES
for j = 2 **to** A.length	--------->	c_1	n
key = $A[j]$	--------->	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	--------->	$c_3=0$	$n-1$
$i = j - 1$	--------->	c_4	$n-1$
while $i > 0$ and $A[i] >$ key	--------->	c_5	$\sum_{j=2}^{n} t_j$
$A[i+1] = A[i]$	--------->	c_6	$\sum_{j=2}^{n}(t_j - 1)$
$i = i - 1$	--------->	c_7	$\sum_{j=2}^{n}(t_j - 1)$
$A[i+1]$ = key	--------->	c_8	$n-1$

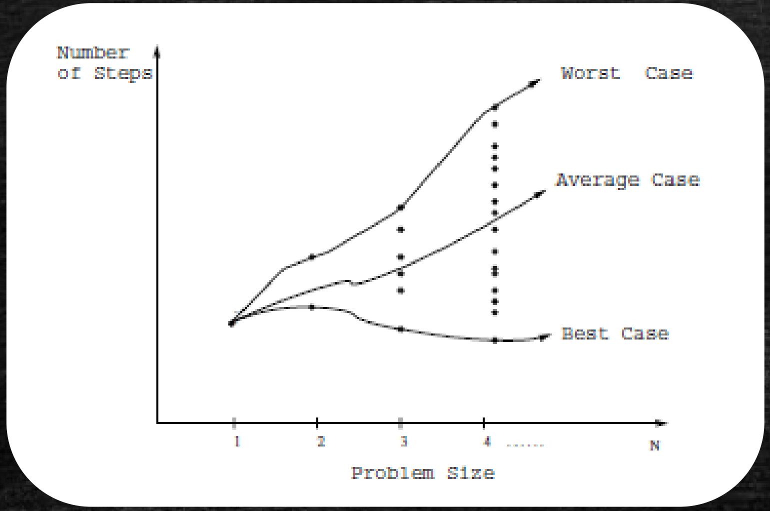
$$T(n) = c_1 n + (c_2 + c_4 + c_8)(n-1) + c_5 \sum_{j=2}^{n} t_j + (c_6 + c_7) \sum_{j=2}^{n}(t_j - 1)$$

- $t_j \geq 1$ so $\sum_{j=2}^{n} t_j \geq n - 1$ and $T(n) \geq (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$
- $t_j \leq j$ so $\sum_{j=2}^{n} t_j \leq \frac{n(n+1)}{2} - 1$ and $T(n) \leq C_1 n^2 + C_2 n + C_3$

Running time as a function of input size

- Place all instances of the same size into the same "bucket"





This is a "lossy compression"! It is missing all the details that appear in the figure regarding the performance of the algorithm in general...

- Worst case running time of an algorithm is a function of n

Asymptotic Notation

- Worst-case running time of an algorithm is a **function $f(n)$**

- How does $f(n)$ grow as a n grows larger?
  - $f(n) = n + \log n$
  - $f(n) = n + 100$
  - $f(n) = 2^n - 10n$

- Comparing algorithms for sorting:
  - Insertion sort is roughly $f(n) = c_1 n^2$. We will say that f(n) is $O(n^2)$
  - Merge sort is roughly $f(n) = c_2 n \log n$. We will say that f(n) is $O(n \log n)$
  - Would you prefer a (worst case) running time of $1000\, n$ or just $n \log n$

Asymptotic Notation

The following are all sets of functions:

$$O(g(n)) = \left\{ \begin{matrix} f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \end{matrix} \right\}$$

$$\Omega(g(n)) = \left\{ \begin{matrix} f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that} \\ 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \end{matrix} \right\}$$

$$\Theta(g(n)) = \left\{ \begin{matrix} f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that} \\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \end{matrix} \right\}$$

Asymptotic Notation (little-oh, little-omega)

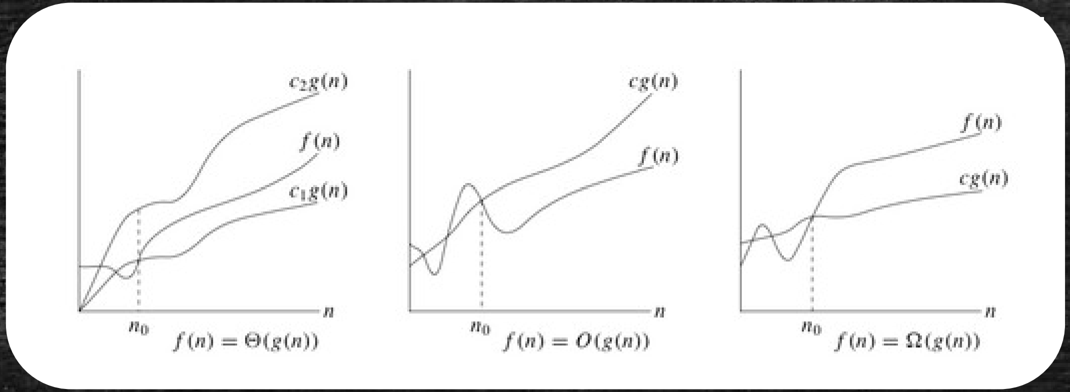
$$o(g(n)) = \left\{ \begin{array}{c} f(n) : \text{for any constant } c > 0\text{, there exists a constant } n_0 > 0 \text{ s.t.} \\ 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0 \end{array} \right\}$$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

$$\omega(g(n)) = \left\{ \begin{array}{c} f(n) : \text{for any constant } c > 0\text{, there exists a constant } n_0 > 0 \text{ s.t.} \\ 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0 \end{array} \right\}$$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

Asymptotic Notation



$f(n) = O(g(n))$ is like $a \leq b$
$f(n) = \Omega(g(n))$ is like $a \geq b$
$f(n) = \Theta(g(n))$ is like $a = b$
$f(n) = o(g(n))$ is like $a < b$
$f(n) = \omega(g(n))$ is like $a > b$

1. Find a function $g(n)$ such that $5 + \cos(n) = \Theta(g(n))$
2. Show that $10n^2 - 100n - 20 \neq \Theta(n \log n)$
3. Show that $\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$
4. Show that $\log_c n = \Theta(\lg n)$ for any constant c

Asymptotic Notation Properties

- Transitivity:
  - $f(n)=O(g(n))$ and $g(n)=O(h(n))$, then $f(n)=O(h(n))$
  - $f(n)=\Omega(g(n))$ and $g(n)=\Omega(h(n))$, then $f(n)=\Omega(h(n))$
- Reflexivity: $f(n)=\Theta(f(n))$
- Transpose Symmetry: $f(n)=O(g(n))$ if and only if $g(n)=\Omega(f(n))$
- Symmetry: $f(n)=\Theta(g(n))$ if and only if $g(n)=\Theta(f(n))$
- Trichotomy: For any two real numbers a and b:
  - $a > b$, or $a = b$, or $a < b$
  - Not satisfied by corresponding asymptotic notions:
  - E.g., $f(n) = n$ and $g(n) = (1 + \sin(n))\, n^2$

Proving these properties is a very reasonable problem for the midterm or final

Today's Lecture

- Divide and conquer algorithms
- Running time as a recurrence equation
- Analysis of recurrence equations

Maximum Subarray Problem

- You are given an array A of n numbers (both positive and negative)
  - Find a contiguous subarray with the maximum sum of numbers
  - In other words: find i, j such that $1 \leq i \leq j \leq n$ and maximize $\sum_{x=i}^{j} A[x]$
  - For example, consider the following 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

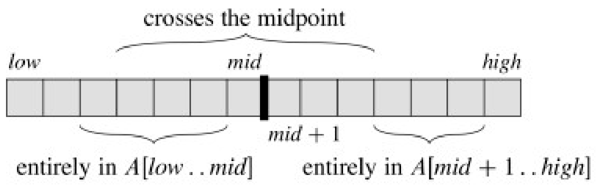
  - What is the first, simple, algorithm that comes to mind?
  - What is the running time of this algorithm?
  - Can you come up with a divide & conquer algorithm?
    - How can we analyze the (worst case) running time of such algorithms?

Divide and Conquer Algorithms

- Divide
  - *Split the problem into smaller sub-problems of the same structure*
- Conquer
  - *If sub-problem size is small enough, solve directly, o/w, solve sub-problems recursively*
- Combine
  - *Merge the solutions of sub-problems into a solution of the original problem*

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

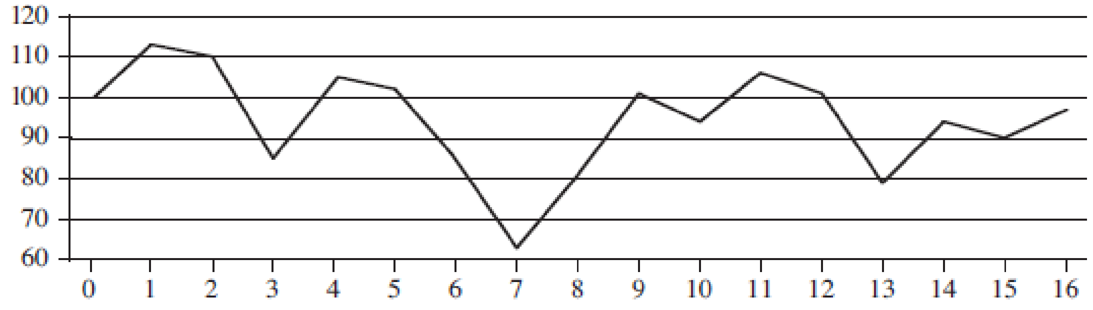
```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high == low
    return (low, high, A[low])              // base case: only one element
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY(A, 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$(A, 1, n)$

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

Profit Maximizing Stock Trade

- Input: $\boldsymbol{n}$ price points $(t_1, t_2, \ldots, t_n)$
- Output: $(\boldsymbol{t_b}, \boldsymbol{t_s})$ s.t. $\boldsymbol{0 \le t_b < ts \le n}$, 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

- How would you solve this problem?
  - This problem can be easily reduced to the maximum subarray problem

Methods for Solving Recurrences

Three methods:

1. Recursion-tree method
   - Covert into a tree and measure cost incurred at the various levels
2. Substitution method
   - Guess a bound and use mathematical induction to prove its correctness
3. Master method
   - Directly provides bounds for recurrences of the form $T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$

Recursion-Tree Method

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