Mid-term Review

CSCI 2300

Asymptotic Complexity

Big-O

- f(n) = O(g(n)) means that there exists a constant c > 0 and n_0 > 0 such that f(n) ≤ cg(n), for all n≥n_0

Big-Omega

- f(n) = Ω(g(n)) means that there exists a constant c > 0 and n_0 > 0 such that cg(n) ≤ f(n), for all n≥n_0

Big-Theta: Big-O AND Big-Omega

Among the functions below, which one is the smallest asymptotically?

A. $N \lg(N)$
B. N^2
C. $N+\lg(N)$
D. N
E. $N^{\lg(N)}$

Among the functions below, which one is the largest asymptotically?

A. $N \lg(N)$
B. N^2
C. $N+\lg(N)$
D. N
E. $N^{\lg(N)}$

Comparison based sorting

Insertion sort(A, N) // think "picking up cards in sorted order"

```
for i = 2 to N
  Insert A[i] into A[1]..A[i-1] in a sorted order
end for
```

What is the worst case run time complexity of Insertion sort?

A. $N\ lg(N)$
B. N^2
C. $N+lg(N)$
D. N
E. $N^{lg(N)}$

Comparison based sorting

Merge-Sort (A, P, R)
if (p < r)

```
Q = ⌊(P + R)/2⌋
Merge-Sort (A, Q, P)
Merge-Sort(A, Q+1, R)
Merge(A, P, Q, R)
```

What is the correct recurrence relation characterizing Merge-Sort

A. $T(N) = T(N/2)$
B. $T(N) = 2T(N/2)$
C. $T(N) = 2T(N/2) + N$
D. $T(N) = 3N$
E. $T(N) = 2T(N-1) + N$

Recurrence Relations

Substitution Method

Master Theorem

Recurrence Trees

Master theorem

Given a recurrence: $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

Compare f(n) to $n^{log_b a}$

If f(n) is	Then T(n) is
$O(n^{log_b a - \varepsilon})$	$\Theta(n^{log_b a})$
$\Theta(n^{log_b a})$	$\Theta(n^{log_b a} * \lg(n))$
$\Omega(n^{log_b a + \varepsilon})$	$\Theta(f(n))$

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

If f(n) is	Then T(n) is
$O(n^{\log_b a - \varepsilon})$	$\Theta(n^{\log_b a})$
$\Theta(n^{\log_b a})$	$\Theta(n^{\log_b a} * \lg(n))$
$\Omega(n^{\log_b a + \varepsilon})$	$\Theta(f(n))$

A. $\Theta(n^2)$

B. $\Theta(n^3)$

C. $\Theta(n^2 * \lg(n))$

D. $\Theta(n)$

E. $\Theta(n^{\log_9 3} * \lg(n))$

$$T(n) = 3T\left(\frac{n}{9}\right) + n$$

If f(n) is	Then T(n) is
$O(n^{\log_b a - \varepsilon})$	$\Theta(n^{\log_b a})$
$\Theta(n^{\log_b a})$	$\Theta(n^{\log_b a} * \lg(n))$
$\Omega(n^{\log_b a + \varepsilon})$	$\Theta(f(n))$

A. $\Theta(n^2)$

B. $\Theta(n^3)$

C. $\Theta(n^2 * \lg(n))$

D. $\Theta(n)$

E. $\Theta(n^{\log_9 3} * \lg(n))$

$$T(n) = T\left(\frac{3n}{5}\right) + 1$$

If f(n) is	Then T(n) is
$O(n^{\log_b a - \varepsilon})$	$\Theta(n^{\log_b a})$
$\Theta(n^{\log_b a})$	$\Theta(n^{\log_b a} * \lg(n))$
$\Omega(n^{\log_b a + \varepsilon})$	$\Theta(f(n))$

A. $\Theta(n^2)$

B. $\Theta(n^3)$

C. $\Theta(\lg(n))$

D. $\Theta(n \lg(n))$

E. $\Theta(n^{\log_5 3} * \lg(n))$

Dynamic Programming

Used to solve optimization problems
- Minimize
- Maximize

Examples covered in class:
- Longest Common Subsequence
- Subset Sum Problem
- Rod Cutting Problem

Key elements:
- Recurrence relation
- Overlapping sub-problems

Solution includes memorization step

Greedy Algorithms

Like dynamic programming, used to solve optimization problems.

Problems exhibit optimal substructure (like DP).

Problems also exhibit the **greedy-choice** property.

- When we have a choice to make, make the one that looks best *right now*.
- Make a **locally optimal choice** in hope of getting a **globally optimal solution**.

Greedy strategy – the choice that looks best at the moment

- Prove that the greedy choice leads to optimal solution
- Proof strategy:
  - Suppose another optimal solution exists
  - Make changes to that other optimal solution until it is the same as greedy solution
  - Show that changes preserve optimality

Greedy Algorithms

Activity Selection Problem

- Weighted Activity Selection – Greedy Choice did not lead to optimal

Activity Partitioning

Minimize Schedule Delay

Elementary Graph Algorithms

Adjacency List and Adjacency Matrix

Breadth First Search

Depth First Search