Week 2.2, Wednesday, August 28

Homework 1 available on course web page
(Due: September 3 at 11:59PM on Gradescope)

Instructor Office Hours: (Monday 2:30-3:30PM, Wed 5:30-6:30PM)
Suppose that

\[ f(n) = 2000n + n^2 \]
\[ g(n) = 10n \log n \]
\[ h(n) = \frac{n^2}{1000} \]

Which of the following claims are true?

1. \( g(n) = O(f(n)) \)
2. \( h(n) = O(f(n)) \)
3. \( h(n) = \Omega(f(n)) \)
4. \( h(n) = O(g(n)) \)

A. Claim 4 only
B. Claim 1 & 2 only
C. Claims 1, 2 and 3
D. All claims are true
E. None of them
The divide-and-conquer algorithm design paradigm

1. *Divide* the problem (instance) into subproblems.

2. *Conquer* the subproblems by solving them recursively.

3. *Combine* subproblem solutions.
Divide-and-Conquer

- Divide-and-conquer.
  - Break up problem into several parts.
  - Solve each part recursively.
  - Combine solutions to sub-problems into overall solution.

- Most common usage.
  - Break up problem of size $n$ into two equal parts of size $\frac{n}{2}$.
  - Solve two parts recursively.
  - Combine two solutions into overall solution in linear time.

- Consequence.
  - Brute force: $n^2$.
  - Divide-and-conquer: $n \log n$.

Divide et impera.
Veni, vidi, vici.
- Julius Caesar
Running Time (Recurrences):

- Let $T(n)$ be the time to solve problem of size $n$ (worst-case).
- Suppose we split input $X$ into 3 equal size parts $A$, $B$ and $C$ recursively solve smaller problems $A$, $B$ and $C$ and then merge the solutions.

$$T(n) \leq 3T\left(\frac{n}{3}\right) + \#\text{Steps(Merge)}$$

Correctness?

- Induction!
- Prove that algorithm is correct on small inputs (e.g., $n \leq 2$)
- Prove that merge algorithm is correct (QED)
What you should learn?

- Solve Recurrences
- Identify recurrence associated with divide and conquer algorithm
- Prove that a divide and conquer algorithm is correct
- **Creative:** Design efficient divide and conquer algorithms
  - Build intuition about when the divide and conquer approach will work.
Mergesort

- Mergesort.
  - Divide array into two halves.
  - Recursively sort each half.
  - Merge two halves to make sorted whole.

\[
T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + O(n)
\]

Jon von Neumann (1945)
Merging

- Merging. Combine two pre-sorted lists into a sorted whole.

- How to merge efficiently?
  - Linear number of comparisons.
  - Use temporary array.

- Challenge for the bored. In-place merge. [Kronrud, 1969]

  using only a constant amount of extra storage
Pre-condition. [Merge] A and B are sorted.
Post-condition. [Sort] L is sorted.

Sort(L) {
    if list L has one element
        return 0 and the list L

    Divide the list into two (equal) halves A and B
    A ← Sort(A)
    B ← Sort(B)
    L ← Merge(A, B)

    return L
}

Mergesort
Mergesort Correctness

- $P(n) = \text{“Mergesort correctly sorts all lists } L \text{ of length } |L| = n\text{”}$
- Base Case: $|L| = 1$ (check)

```plaintext
Sort(L) {
    if list L has one element
        return 0 and the list L

    Divide the list into two (equal) halves A and B
    A ← Sort(A)
    B ← Sort(B)
    L ← Merge(A, B)

    return L
}
```
P(n) = “Mergesort correctly sorts all lists L of length |L| = n”

Base Case: |L| = 1

Strong Inductive Hypothesis: P(k) holds for all k < n i.e., correct on any list of length < n

Inductive Step:
- Algorithm splits input L into A and B
  \[ A \leftarrow \text{Sort}(A), \ B \leftarrow \text{Sort}(B) \]
- IH \( \rightarrow \) both A and B both sorted correctly
- Therefore, algorithm is correct (as long as merge step is implemented correctly)

QED
Def. \( T(n) = \) number of comparisons to mergesort an input of size \( n \).

Mergesort recurrence.

Solution. \( T(n) = O(n \log_2 n) \).

Assorted proofs. We describe several ways to prove this recurrence. Initially we assume \( n \) is a power of 2 and replace \( \leq \) with \( = \).
Claim. If \( T(n) \) satisfies this recurrence, then \( T(n) = n \log_2 n \).

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + n & \text{otherwise}
\end{cases}
\]

assumes \( n \) is a power of 2

Pf. (by induction on \( n \))

- Base case: \( n = 1 \).
- Inductive hypothesis: \( T(n) = n \log_2 n \).
- Goal: show that \( T(2n) = 2n \log_2 (2n) \).

\[
T(2n) = 2T(n) + 2n \\
= 2n \log_2 n + 2n \\
= 2n(\log_2 (2n) - 1) + 2n \\
= 2n \log_2 (2n)
\]
Claim. If \( T(n) \) satisfies the following recurrence, then \( T(n) \leq n \lceil \lg n \rceil \).

\[
T(n) \leq \begin{cases} 
0 & \text{if } n = 1 \\
T\left(\left\lfloor n/2 \right\rfloor \right) + T\left(\left\lceil n/2 \right\rceil \right) + n & \text{otherwise}
\end{cases}
\]

Pf. (by induction on \( n \))
- Base case: \( n = 1 \).
- Define \( n_1 = \left\lfloor n / 2 \right\rfloor \), \( n_2 = \left\lceil n / 2 \right\rceil \).
- Induction step: assume true for \( 1, 2, \ldots, n-1 \).
Claim. If T(n) satisfies the following recurrence, then T(n) ≤ n \lfloor \lg n \rfloor.

Pf. (by induction on n)
- Base case: n = 1.
- Define n_1 = \lfloor n / 2 \rfloor, n_2 = \lceil n / 2 \rceil.
- Induction step: assume true for 1, 2, ..., n-1.

\[
T(n) \leq T(n_1) + T(n_2) + n \\
\leq n_1 \lfloor \lg n_1 \rfloor + n_2 \lceil \lg n_2 \rceil + n \\
\leq n_1 \lfloor \lg n_2 \rfloor + n_2 \lceil \lg n_2 \rceil + n \\
= n \lfloor \lg n_2 \rfloor + n \\
\leq n(\lceil \lg n \rceil - 1) + n \\
= n \lfloor \lg n \rfloor
\]

\[
n_2 = \lceil n / 2 \rceil \\
\leq \lceil 2 \lfloor \lg n \rfloor / 2 \rceil \\
= 2 \lfloor \lg n \rfloor / 2 = 2 \lfloor \lg n \rfloor - 1 \\
\Rightarrow \lg n_2 \leq \lfloor \lg n \rfloor - 1
\]
Problem: Compute $a^n$, $n > 0$. Minimize number of multiplications.

**Naive algorithm:** $\Theta(n)$ multiplications

```
Exp(a,n) {
  if n=0 return 1
  else if n=1 return a
  else if n is even
    b ← Exp(a,n/2)
    return b x b
  else // n>1 is odd
    b ← Exp(a,(n-1)/2)
    return b x b x a
}
```
Which of the following recurrences describes the number of multiplications in the above algorithm?

A. \( T(n) \leq T(n - 1) + 1 \)  
B. \( T(n) \leq T(n/2) + 2 \)  
C. \( T(n) \leq 4T(n/2) + n \)  
D. \( T(n) \leq 3T(n/3) + 1 \)  
E. \( T(n) \leq T\left(\frac{n}{2}\right) + n/2 \)
Which of the following recurrences describes the number of multiplications in the above algorithm?

A. \( T(n) \leq T(n - 1) + 1 \)  
B. \( T(n) \leq T(n/2) + 2 \)  
C. \( T(n) \leq 4T(n/2) + n \)  
D. \( T(n) \leq 3T(n/3) + 1 \)  
E. \( T(n) \leq T\left(\frac{n}{2}\right) + n/2 \)