Week 2.1, 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)
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.

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

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

```python
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 Correctness

- $P(n) = \text{“Mergesort correctly sorts all lists } L \text{ of length } |L| = n\text{”}$
- 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 $B$
    - $A \leftarrow \text{Sort}(A), B \leftarrow \text{Sort}(B)$
  - $\text{IH} \Rightarrow$ both $A$ and $B$ both sorted correctly
  - Therefore, algorithm is correct (as long as merge step is implemented correctly)
- QED
Def. $T(n) = \text{number of comparisons to mergesort an input of size } n$.

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 $T(n) = n \log_2 n$.

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

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 $T(n) \leq n \lceil \lg n \rceil$.

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 $1, 2, \ldots, n-1$. 

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

solve left half \quad solve right half \quad merging
Claim. If $T(n)$ satisfies the following recurrence, then $T(n) \leq n \lceil \lg n \rceil$.

Pf. (by induction on $n$)

- Base case: $n = 1$.
- Define $n_1 = \lfloor n / 2 \rceil$, $n_2 = \lceil n / 2 \rceil$.
- Induction step: assume true for 1, 2, ..., $n-1$.

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

\[
T(n) \leq n \lceil \lg n \rceil
\]

\[
T(n) \leq n \lceil \lg n \rceil
\]

\[
n_2 = \lceil n / 2 \rceil \\
\leq 2 \lceil \lg n \rceil / 2 = 2 \lceil \lg n \rceil - 1
\]

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

Naive algorithm: $\Theta(n)$ multiplications

```c
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. $T(n) \leq 2T(n/2) + 2$
C. $T(n) \leq 4T(n/2) + n$
D. $T(n) \leq 3T(n/3) + 1$
E. $T(n) \leq T\left(\frac{n}{2}\right) + n/2$
Running time of a divide and conquer algorithm can be captured by a recurrence relation.
How does one determine the running time?

**General method**

(1) “Guess” the solution.
(in closed exact form or in asymptotic form)
(2) Prove it correct by induction.

If the assumed solution is incorrect, the induction will fall apart somewhere.
Assume the basis is $T(1) = \Theta(1)$

- $T(n) = T(n/2) + c$
- $T(n) = T(n/2) + cn$
- $T(n) = 2T(n/2) + cn$
- $T(n) = 2T(n-1) + 1$
- $T(n) = 4T(n/2) + n$
- $T(n) = T(n/4) + T(n/2) + n^2$
- $T(n) = T(2n/3) + n$
- $T(n) = T(\sqrt{n}) + c$
Recurrences from divide and conquer algorithms

Assume the basis is $T(1) = \Theta(1)$

- $T(n) = T(n/2) + c \quad \Theta(\log n)$
- $T(n) = T(n/2) + cn \quad \Theta(n)$
- $T(n) = 2T(n/2) + cn \quad \Theta(n \log n)$
- $T(n) = 2T(n-1) + 1 \quad \Theta(2^n)$
- $T(n) = 4T(n/2) + n \quad \Theta(n^2)$
- $T(n) = T(n/4) + T(n/2) + n^2 \quad \Theta(n^2)$
- $T(n) = T(2n/3) + n \quad \Theta(n)$
- $T(n) = T(\sqrt{n}) + c \quad \Theta(\log \log n)$
Example: $T(n) = 4T(n/2) + n$ ($n$ is a power of 2)

Claim: $T(n) = O(n^3)$

Induction Hypothesis:
Assume $T(k) \leq ck^3$ for all $k < n$, for some constant $c$

$T(n) = 4T(n/2) + n$
\[ \leq 4 \times c \times (n/2)^3 + n \]
\[ = cn^3/2 + n \]
\[ = cn^3 - (cn^3/2 - n) \]
\[ \leq cn^3 \]

Need to show $cn^3/2 - n \geq 0$. True for $c \geq 2$

But $O(n^3)$ is not a tight bound! $\Theta(n^3)$ does not hold.
Show that $T(n) = O(n^2)$

Claim: $T(n) \leq cn^2$

$$T(n) = 4T(n/2) + n \leq cn^2$$

$$\leq 4c(n/2)^2 + n$$

$$= cn^2 + n$$

$$\leq cn^2 \quad \text{NO!}$$

Wrong argument: we just made the constant $c$ larger

To show that it is $O(n^2)$, we need to subtract lower order terms!
Claim: $T(n) \leq c_1 n^2 - c_2 n$, for some constants $c_1$ and $c_2$

Inductive hypothesis: $T(k) \leq c_1 k^2 - c_2 k$ for all $k < n$

\[
T(n) = 4T(n/2) + n \\
\leq 4 \left( c_1 (n/2)^2 - c_2 (n/2) \right) + n \\
= c_1 n^2 - 2c_2 n + n \\
= c_1 n^2 - c_2 n - c_2 n + n \\
\leq c_1 n^2 - c_2 n
\]

Need $-c_2 n + n \leq 0$: true if $c_2 \geq 1$
Recursion Tree Method

Use the recursion tree to find the solution to a recurrence

• Tree represents a model of the cost of the recursive algorithm
• Getting a closed form can be messy
• Insight obtained from tree can give a good initial guess to be used in an induction
\[ T(n) = 4T(n/2) + n, \quad n=2^k \]

Work done at each level

- Level 0: \( n \)
- Level 1: \( 2n \) (4 instances of size \( n/2 \) each)
- Level 2: \( 4n \) (\( 4^2 = 16 \) instances of size \( n/4 \) each)
- Clicker Question: How much total work is done at Level 3?

A. 6n  B. 8n  C. 9n  D. 16n  E. 32n
\[ T(n) = 4T(n/2) + n, \quad n=2^k \]

Work done at each level

- Level 0: \( n \)
- Level 1: \( 2n \) (4 instances of size \( n/2 \) each)
- Level 2: \( 4n \) (\( 4^2=16 \) instances of size \( n/4 \) each)
- Level 3: \( 8n \) (\( 4^3 \) instances of size \( n/2^3 \) each)
- At level \( i \), there are \( 4^i \) instances of size \( n/2^i \) results in \( 2^i n \) total work for level \( i \)
Total work done at all levels of the recursion tree

at level i of the tree
• there are $4^i$ nodes, each doing work of size $n/2^i$
• results in $n2^i$ total work for level i

$$\sum_{i=0}^{k} n \cdot 2^i = n(2^{k+1} - 1) = n(2n - 1) = 2n^2 - n$$

This gives $T(n) = \Theta(n^2)$
Prove this correct by induction as an exercise.
Divide and conquer algorithms

- Mergesort
- Quicksort
- Binary Search

- Skyline Problem
- Maximum Subarray
- Counting inversions
- Linear-time selection