CS 381 – FALL 2019

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)

Review: Asymptotic Notation

Suppose that

$$f(n) = 2000n + n^2$$
$$g(n) = 10\, n \log n$$
$$h(n) = n^2/1000$$

Which of the following claims are true?

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

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

Analysis: Divide and Conquer

- 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.

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

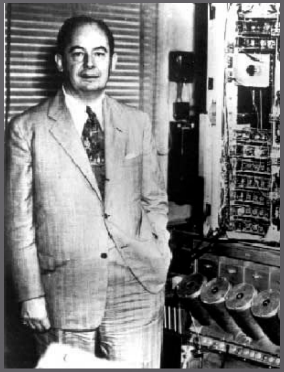
- Correctness?
  - Induction!
  - Prove that algorithm is correct on small inputs (e.g., $n \leq 2$)
  - Prove that merge algorithm is correct (QED)

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.



Jon von Neumann (1945)

A	L	G	O	R	I	T	H	M	S

A	L	G	O	R		I	T	H	M	S

Divide O(1)

A	G	L	O	R		H	I	M	S	T

Sort 2T(n/2)

A	G	H	I	L	M	O	R	S	T

Merge O(n)

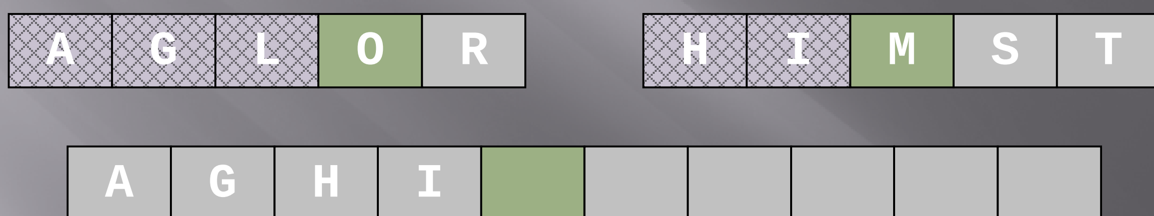
$$T(n) \leq 2 \cdot T\left(\frac{n}{2}\right) + O(n)$$

Merging

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

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

05demo-merge.ppt



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

using only a constant amount of extra storage

Mergesort

- 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) = "Mergesort correctly sorts all lists L of length |L| = n"
- Base Case: |L| = 1 (check)

```
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) = "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 ← Sort(A), B ← Sort(B)
  - **IH → both A and B both sorted correctly**
  - **Therefore, algorithm is correct (as long as merge step is implemented correctly) QED**

A Useful Recurrence Relation

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

- Mergesort recurrence.

$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

- 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 ≤ with =.

Proof by Induction

- Claim. If T(n) satisfies this recurrence, then T(n) = n $\log_2$ n.

assumes n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

- 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).

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2n\log_2 n + 2n \\ &= 2n\left(\log_2(2n) - 1\right) + 2n \\ &= 2n\log_2(2n) \end{aligned}$$

Analysis of Mergesort Recurrence

- Claim. If T(n) satisfies the following recurrence, then $T(n) \le n \lceil \lg n \rceil$.

$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

- 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.

Analysis of Mergesort Recurrence

- Claim. If T(n) satisfies the following recurrence, then $T(n) \le n \lceil \lg n \rceil$.
- 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, ... , n–1.

$$\begin{aligned}
T(n) &\le T(n_1) + T(n_2) + n \\
&\le n_1 \lceil \lg n_1 \rceil + n_2 \lceil \lg n_2 \rceil + n \\
&\le n_1 \lceil \lg n_2 \rceil + n_2 \lceil \lg n_2 \rceil + n \\
&= n \lceil \lg n_2 \rceil + n \\
&\le n(\lceil \lg n \rceil - 1) + n \\
&= n \lceil \lg n \rceil
\end{aligned}$$

$$\begin{aligned}
n_2 &= \lceil n/2 \rceil \\
&\le \left\lceil 2^{\lceil \lg n \rceil} / 2 \right\rceil \\
&= 2^{\lceil \lg n \rceil} / 2 = 2^{\lceil \lg n \rceil - 1}
\end{aligned}$$

$$\Rightarrow \lg n_2 \le \lceil \lg n \rceil - 1$$

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
}
```

Clicker Question

```
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. $\mathrm{T(n)} \le \mathrm{T}(n-1)+1$

B. $\mathrm{T(n)} \le \mathrm{T}(n/2)+2$

C. $\mathrm{T(n)} \le 4\mathrm{T}(n/2)+\mathrm{n}$

D. $\mathrm{T(n)} \le 3\mathrm{T}(n/3)+1$

E. $\mathrm{T(n)} \le \mathrm{T}\left(\frac{n}{2}\right)+\mathrm{n}/2$

Clicker Question

```
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. $\mathrm{T(n)} \leq \mathrm{T}(n-1)+1$

B. $\mathrm{T(n)} \leq \mathrm{T}(n/2)+2$

C. $\mathrm{T(n)} \leq 4\mathrm{T}(n/2)+\mathrm{n}$

D. $\mathrm{T(n)} \leq 3\mathrm{T}(n/3)+1$

E. $\mathrm{T(n)} \leq \mathrm{T}\left(\frac{n}{2}\right)+\mathrm{n}/2$