CS 381 – FALL 2019

Week 2.3, Friday, August 30

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

Labor Day: No Class on Monday, Sept 2

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
}
```

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

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

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

Analysis of Mergesort Recurrence

- 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, ... , n–1.

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

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

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

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 × b
   else               // n>1 is odd
       b ← Exp(a,(n-1)/2)
       return b × b × 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 × b
   else            // n>1 is odd
       b ← Exp(a,(n-1)/2)
       return b × b × a
}
```

Which of the following recurrences describes the number of multiplications in the above algorithm?

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

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

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

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

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

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 × b
    else              // n>1 is odd
        b ← Exp(a,(n-1)/2)
        return b × b × 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. $\mathrm{T(n)} \le \mathrm{T}\left(\frac{n}{2}\right)+2$

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 × b
   else            // n>1 is odd
       b ← Exp(a,(n-1)/2)
       return b × b × a
}
```

$$T(n) = T(n/2) + \Theta(1) \;\Rightarrow\; T(n) = \Theta(\log n)$$

Suggested Exercise: Prove that above algorithm is correct using strong induction.

Counting Inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: 1, 2, …, n.
- Your rank: $a_1, a_2, \ldots, a_n$.
- Songs i and j inverted if $i < j$, but $a_i > a_j$.

Songs

	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5

<u>Inversions</u>

3-2, 4-2

Brute force: check all $\Theta(n^2)$ pairs i and j.

Applications

Applications.

- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's Tau distance).

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

1	5	4	8	10	2	6	9	12	11	3	7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.

1	5	4	8	10	2	6	9	12	11	3	7

Divide: $O(1)$.

1	5	4	8	10	2

6	9	12	11	3	7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.

1	5	4	8	10	2	6	9	12	11	3	7

Divide: $O(1)$.

1	5	4	8	10	2

6	9	12	11	3	7

Conquer: $2T(n / 2)$

5 blue-blue inversions

5-4, 5-2, 4-2, 8-2, 10-2

8 green-green inversions

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.
- Combine: count inversions where a_i and a_j are in different halves, and return sum of three quantities.

1	5	4	8	10	2	6	9	12	11	3	7

Divide: O(1).

1	5	4	8	10	2

6	9	12	11	3	7

5 blue-blue inversions

8 green-green inversions

Conquer: 2T(n / 2)

9 blue-green inversions
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Combine: ???

Total = 5 + 8 + 9 = 22.

Counting Inversions: Combine

play

Combine: count blue-green inversions

- Assume each half is sorted.
- Count inversions where a_i and a_j are in different halves.
- Merge two sorted halves into sorted whole.

to maintain sorted invariant

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	0

13 blue-green inversions: 6 + 3 + 2 + 2 + 0 + 0 Count: O(n)

2	3	7	10	11	14	16	17	18	19	23	25

Merge: O(n)

$$T(n) \le T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + O(n) \Rightarrow \mathrm{T}(n) = O(n \log n)$$

Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.
Post-condition. [Sort-and-Count] L is sorted.

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

   Divide the list into two halves A and B
   (rA, A) ← Sort-and-Count(A)
   (rB, B) ← Sort-and-Count(B)
   (r , L) ← Merge-and-Count(A, B)

   return r = rA + rB + r and the sorted list L
}
```

Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.

Post-condition. [Sort-and-Count] L is sorted.

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

   Divide the list into two halves A and B
   (rA, A) ← Sort-and-Count(A)
   (rB, B) ← Sort-and-Count(B)
   (r , L) ← Merge-and-Count(A, B)

   return r = rA + rB + r and the sorted list L
}
```

Proof of Correctness (Strong Induction):

Base Case: n=1 (check)

Strong Inductive Hypothesis: Sort-and-Count(L) is correct for all lists L of length |L| < n

Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.

Post-condition. [Sort-and-Count] L is sorted.

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

   Divide the list into two halves A and B
   (r_A, A) ← Sort-and-Count(A)
   (r_B, B) ← Sort-and-Count(B)
   (r, L) ← Merge-and-Count(A, B)

   return r = r_A + r_B + r and the sorted list L
}
```

Inductive Step: Let L be a list of length n with (A,B)=L

```
  (r_A, A) ← Sort-and-Count(A)        // Correct by IH
  (r_B, B) ← Sort-and-Count(B)        // Correct by IH
  (r, L) ← Merge-and-Count(A, B)   // r counts A,B inv
→ r_A + r_B + r is correct total count + L is sorted
```

Clicker Question

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

   Divide the list into two halves A and B
   (rA, A) ← Sort-and-Count(A)
   (rB, B) ← Sort-and-Count(B)
   (r, L) ← Merge-and-Count(A, B)

   return r = rA + rB + r and the sorted list L
}
```

The recurrence for the running time $T(n)$ of Sort-and-Count is

A. $T(n) = 2T(n/2) + \log n$

B. $T(n) = 2T(n-1) + 1$

C. $T(n) = 2T(n/2) + 1$

D. $T(n) = 2T(n/2) + n$

E. None of the above

Clicker Question

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

   Divide the list into two halves A and B
   (rA, A) ← Sort-and-Count(A)
   (rB, B) ← Sort-and-Count(B)
   (r, L) ← Merge-and-Count(A, B)

   return r = rA + rB + r and the sorted list L
}
```

The recurrence for the running time $T(n)$ of Sort-and-Count is

A. $T(n) = 2T(n/2)+\log n$

B. $T(n) = 2T(n-1) + 1$

C. $T(n) = 2T(n/2)+1$

D. $T(n) = 2T(n/2) + n$

E. None of the above

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.

Run-time recurrences from divide and conquer algorithms

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 $T(1) = \Theta(1)$

$T(n) = T(n/2) + c$	$\theta(\log n)$
$T(n) = T(n/2) + cn$	$\theta(n)$
$T(n) = 2T(n/2) + cn$	$\theta(n \log n)$
$T(n) = 2T(n-1) + 1$	$\theta(2^n)$
$T(n) = 4T(n/2) + n$	$\theta(n^2)$
$T(n) = T(n/4) + T(n/2) + n^2$	$\theta(n^2)$
$T(n) = T(2n/3) + n$	$\theta(n)$
$T(n) = T(\sqrt{n}) + c$	$\theta(\log \log n)$

Divide and conquer algorithms

- Mergesort
- Quicksort
- Binary Search
- *Linear-time selection*
- *Skyline Problem*
- *Maximum Subarray*
- *Counting inversions*