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
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.

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

- 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 $\leq$ with $=.$
Proof by Induction

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

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

  sorting both halves, merging

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

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

Pf. (by induction on $n$)

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

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, ..., \( n-1 \).

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

\[
\begin{align*}
n_2 &= \lceil n / 2 \rceil \\
&\leq 2^{\lfloor \lg n \rfloor / 2} \\
&= 2^{\lfloor \lg n \rfloor - 1}
\end{align*}
\]

\( \Rightarrow \lg n_2 \leq \lfloor \lg n \rfloor - 1 \)
Problem: Compute $a^n$, $n>0$. Minimize number of multiplications.

Naive algorithm: $\Theta(n)$ multiplications

```cpp
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. $T(n) \leq T(n - 1) + 1$
B. $T(n) \leq T\left(\frac{n}{2}\right) + 2$
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. $T(n) \leq T\left(\frac{n}{2}\right) + 2$
C. $T(n) \leq 4T(n/2) + n$
D. $T(n) \leq 3T(n/3) + 1$
E. $T(n) \leq T\left(\frac{n}{2}\right) + n/2$
Clicker Question

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

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

\[ T(n) = T(n/2) + \Theta(1) \implies T(n) = \Theta(\log n) \]
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: $a_1, a_2, ..., a_n$.
- Songs $i$ and $j$ inverted if $i < j$, but $a_i > a_j$.

### Counting Inversions

#### Songs

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Me</strong></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td><strong>You</strong></td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

**Inversions**

3-2, 4-2

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

- 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

Conquer: \( 2T(n/2) \)
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: \( O(1) \).

<table>
<thead>
<tr>
<th>1 5 4 8 10 2</th>
<th>6 9 12 11 3 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 blue-blue inversions</td>
<td>8 green-green inversions</td>
</tr>
</tbody>
</table>

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

**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.

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

**Merge:** $O(n)$

$$T(n) \leq T\left(\left\lfloor n/2 \right\rfloor\right) + T\left(\left\lceil n/2 \right\rceil\right) + O(n) \Rightarrow T(n) = O(n \log n)$$
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
    \( (r_A, A) \leftarrow \text{Sort-and-Count}(A) \)
    \( (r_B, B) \leftarrow \text{Sort-and-Count}(B) \)
    \( (r, L) \leftarrow \text{Merge-and-Count}(A, B) \)

    return \( r = r_A + r_B + r \) and the sorted list L
}
Pre-condition. [Merge-and-Count] A and B are sorted.
Post-condition. [Sort-and-Count] L is sorted.

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.

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

\[(r_A, A) \leftarrow \text{Sort-and-Count}(A) \quad \text{// Correct by IH}\]
\[(r_B, B) \leftarrow \text{Sort-and-Count}(B) \quad \text{// Correct by IH}\]
\[(r, L) \leftarrow \text{Merge-and-Count}(A, B) \quad \text{// } r \text{ counts } A,B \text{ inv}\]
\[\Rightarrow r_A + r_B + r \text{ is correct total count + L is sorted}\]
Clicker Question

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

- $T(n) = T(n/2) + c \quad \Theta(\log n)$
- $T(n) = T(n/2) + cn \quad \Theta(n)$
- $T(n) = 2T(n/2) + cn \quad \Theta(n \log n)$
- $T(n) = 2T(n-1) + 1 \quad \Theta(2^n)$
- $T(n) = 4T(n/2) + n \quad \Theta(n^2)$
- $T(n) = T(n/4) + T(n/2) + n^2 \quad \Theta(n^2)$
- $T(n) = T(2n/3) + n \quad \Theta(n)$
- $T(n) = T(\sqrt{n}) + c \quad \Theta(\log \log n)$
Divide and conquer algorithms

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