Reading Material

- Chapter 8, Section 3
A divide-and-conquer algorithm works as follows for solving a problem:

- A problem’s instance of size $n$ is divided into $b$ smaller instances of the same problem, ideally of about the same size $n/b$.
- Some (say $a$ of the $b$ subproblems) of the smaller instances are solved (typically recursively).
- If necessary, the solutions to the solves subproblems are combined to get a solution to the original instance.
Sorting

- Input: A list $L$ of $n$ elements that can be totally ordered.
- Output: $L$ with its elements appearing in ascending order.
A (terribly) brute-force algorithm would go through all $n!$ permutations of the list’s elements and returns a sorted one.

- Takes $O(n \cdot n!)$ time.
- Of course, we can do much better.
MergeSort

- **MergeSort** is a divide-and-conquer algorithm that
  - divides the list into two halves,
  - sorts each of them (recursively), and
  - merges the two sorted halves while making use of the fact that each is sorted.
MergeSort
MergeSort

**Input:** List L[0..n-1] of "orderable" elements

**Modifies:** List L is sorted in-place in ascending order

**Output:** None

If n>1
  
  copy L[0..[n/2]-1] to A[0..[n/2]-1];
  copy L[[n/2]..n-1] to B[0.. [n/2]-1];
  MergeSort(A[0..[n/2]-1]);
  MergeSort(B[0..[n/2]-1]);
  Merge(A, B, L);
MergeSort
MergeSort

Merge
Input: Two sorted lists A[0..p-1] and B[0..q-1], and list L
Modifies: List L contains the elements of A and B sorted in ascending order
Output: None

i←0; j←0; k←0;

While i<p and j<q
    If A[i]≤B[j]
        L[k]←A[i];
        i←i+1;
    Else
        L[k]←B[j];
        j←j+1;
        k←k+1;

If i=p
    copy B[j..q-1] to L[k..p+q-1]
Else
    copy A[i..p-1] to L[k..p+q-1]
What is the running time $T(n)$ of MergeSort?

$T(n) = 2T(n/2) + O(n)$

What is a solution to this recurrence?
MASTER THEOREM

Let $f$ be an increasing function that satisfies the recurrence relation

$$f(n) = af(n/b) + cn^d$$

whenever $n = b^k$, where $k$ is a positive integer, $a \geq 1$, $b$ is an integer greater than 1, and $c$ and $d$ are real numbers with $c$ positive and $d$ nonnegative. Then

$$f(n) \text{ is } \begin{cases} 
O(n^d) & \text{if } a < b^d, \\
O(n^d \log n) & \text{if } a = b^d, \\
O(n^{\log_b a}) & \text{if } a > b^d.
\end{cases}$$
What is the running time $T(n)$ of MergeSort?

- $T(n)=2T(n/2)+O(n)$

What is a solution to this recurrence?

- Answer: $O(n \log n)$
Binary Search

- **Input:** Sorted list $L$ and element $x$.
- **Output:** True if $x$ is in $L$, and False otherwise.
Binary Search

- A divide-and-conquer algorithm:
A divide-and-conquer algorithm:

**BinarySearch**

**Input:** Ordered list \(L[0..n-1]\), and element \(x\)

**Output:** True if \(x\) is in \(L\), and False otherwise

- If \(|L|=1\)
  - Return \((x=L[0])\);
- Else If \((x = L[\lfloor n/2 \rfloor - 1])\)
  - Return True;
- Else If \((x < L[\lfloor n/2 \rfloor - 1])\)
  - **BinarySearch**\((L[0..\lfloor n/2 \rfloor - 1]);\)
- Else
  - **BinarySearch**\((L[\lfloor n/2 \rfloor ..n-1]);\)
What is the recurrence $T(n)$ for the running time of BinarySearch?

What is a solution to this recurrence?
Questions?