CSE 431 – Algorithm Analysis
Ch5. Divide and Conquer

Definitions

Divide and Conquer:

A divide-and-conquer algorithm solves a problem by dividing its given instance into several smaller instances, solving each of them recursively, and then, if necessary, combining the solutions to the smaller instances into a solution to the given instance.

Procedure:

- **Divide**: Divide instance into many smaller instances
- **Conquer**: Solve each of smaller instances recursively and separately
- **Combine**: Combine solutions to small instances to obtain a solution for the original big instance
- Write down **recurrence** for running time
- Solve recurrence using **master theorem**

Master Theorem:

In the analysis of algorithms, the master theorem provides a cookbook solution in asymptotic terms (using Big O notation) for recurrence relations of types that occur in the analysis of many divide and conquer algorithms.

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n) \quad \text{where } a \geq 1,\ b > 1$$

This recursive Relation can be successively substituted into itself and expanded to obtain expression for total amount of work done.

In the application to the analysis of a recursive algorithm, the constants and function take on the following significance:

• n is the size of the problem.

• a is the number of subproblems in the recursion.

• n/b is the size of each subproblem. (Here it is assumed that all subproblems are essentially the same size.)

• f (n) is the cost of the work done outside the recursive calls, which includes the cost of dividing the problem and
the cost of merging the solutions to the subproblems.

Merge Sort Algorithm:

ALGORITHM *Mergesort*$(A[0..n-1])$
//Sorts array $A[0..n-1]$ by recursive mergesort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
if $n > 1$
copy $A[0..\lfloor n/2 \rfloor - 1]$ to $B[0..\lfloor n/2 \rfloor - 1]$
copy $A[\lfloor n/2 \rfloor..n-1]$ to $C[0..\lceil n/2 \rceil - 1]$
Mergesort$(B[0..\lfloor n/2 \rfloor - 1])$
Mergesort$(C[0..\lceil n/2 \rceil - 1])$
Merge(B, C, A)

Merge Sort Algorithm:

ALGORITHM *Merge*$(B[0..p-1], C[0..q-1], A[0..p+q-1])$
//Merges two sorted arrays into one sorted array
//Input: Arrays $B[0..p-1]$ and $C[0..q-1]$ both sorted
//Output: Sorted array $A[0..p+q-1]$ of the elements of B and C
$i \leftarrow 0;\ j \leftarrow 0;\ k \leftarrow 0$
while $i < p$ **and** $j < q$ **do**
if $B[i] \leq C[j]$
$A[k] \leftarrow B[i];\ i \leftarrow i + 1$
else $A[k] \leftarrow C[j];\ j \leftarrow j + 1$
$k \leftarrow k + 1$
if $i = p$
copy $C[j..q-1]$ to $A[k..p+q-1]$
else copy $B[i..p-1]$ to $A[k..p+q-1]$

Problem1. (Binary Search) Here is the binary search algorithm:

```
int binarySearch(int[ ] a, int x, int low, int high) { if
   (low > high) return -1;

   int mid = (low + high)/2;  if
   (a[mid] == x) return mid;

   else if (a[mid] < x)
      return binarySearch(a, x, mid+1, high);

   else // last possibility: a[mid] > x return
      binarySearch(a, x, low, mid-1);
}
```

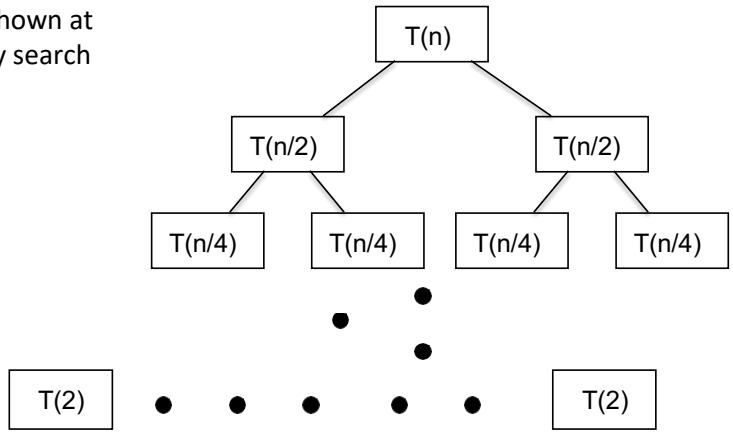
What is the cost of the base case?

If binarySearch is called with an array of size n and a recursive call is made, how big is the array passed on the recursive call?

What is the recurrence equation?

T(n) =

Problem2. The recurrence tree for merge sort is shown at the right. What does the recurrence tree for binary search look like?



How many subproblems are solved on each level of the recursion? (For merge sort there are 2 subproblems at each level.)

What is the cost at each level, excluding the cost of the recursion? (For merge sort, it is cn, due to the merge.)

How many recursion levels are in the recurrence tree? (For merge sort, it is log2n.)

What is the total cost (T(n)) of binary search, arrived at by summing the costs across all levels of the recurrence tree? (For merge sort, it is cn・log2n.)

What is the the O() of binary search? (For merge sort, it is O(n log n).)

Problem3: Master theorem. For each of the following recurrences, give an expression for the runtime T(n) if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply.

$$T(n) = 3T(n/2) + n^2$$

$$T(n) = 16T(n/4) + n$$

$$T(n) = 0.5T(n/2) + 1/n$$

$$T(n) = 3T(n/2) + n$$

Problem4: Master theorem. For the following recurrence, draw the recursion tree, give an expression for the runtime T(n)

T(n) = $\begin{Bmatrix} 3T\left(\frac{n}{4}\right) + O(n^2), & otherwise \\ O(1), & n \leq 1 \end{Bmatrix}$