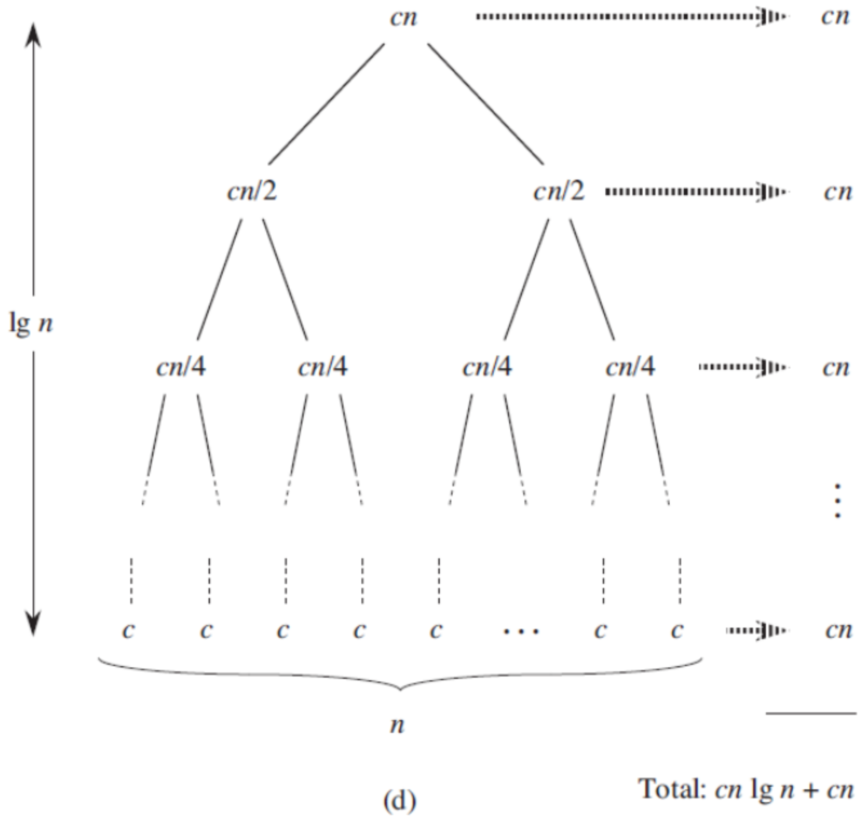
RECURSION AND RECURRENCES

CS340

Merge Sort is a recursive algorithm

1. Recursive algorithms call themselves, creating an execution stack.
2. Recursion implies levels in an execution tree.
3. Recursion levels are a lot like loops.

Complexity = $\Theta(n \lg n)$



Recursive Insertion Sort

- Base Case
  - Key is slot 2. The first item is trivially sorted
- Otherwise
  - recursively sort A[1..i-1] and then insert A[i] into the sorted array A[1..i-1]

INSERTION-SORT(A)

```
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Recursive Insertion Sort

```
private static int insertionSort(int[] A, int maxIndex) {
     if (maxIndex <= 1) {
          return maxIndex;
     }

     maxIndex = insertionSort(A, maxIndex - 1);  // recursive call

     int key = A[maxIndex];
     int i = maxIndex - 1;

     while ((i >= 0) && (A[i] > key)) {
          A[i+1] = A[i];
          i--;
     }
     A[i+1] = key;
return maxIndex + 1;
}
```

Recurrence for Recursive Insertion Sort

Since it takes $\Theta(n)$ time in the worst case to insert $A[n]$ into the sorted array $A[1\,..\,n-1]$, we get the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ T(n-1) + \Theta(n) & \text{if } n > 1\,. \end{cases}$$

$T(n) = \Theta(n^2)$

Binary Search

- **If the sequence A is sorted**, we can check the midpoint of the sequence against a desired value and eliminate half of the sequence from further consideration.
- Binary search repeats this procedure, halving the size of the remaining portion of the sequence each time.

Binary Search

```
int BINARY_SEARCH (vector<int> v, int from, int to, int val) {
   if (from>to) return -1; //val not found
   int mid = (from+to)/2;
   if (v[mid] == val)
       return mid;
   else if (val > v[mid])
      return BINARY_SEARCH(v,mid+1,to,val);
   else
      return BINARY_SEARCH(v,from,mid-1,val);
}
```

Binary Search

- Recurrence equation for binary search:
  - $T(n) = \begin{cases} \Theta(1) & if\ n = 1 \\ T\left(\frac{n}{2}\right) + \Theta(1) & if\ n > 1 \end{cases}$

- What is the **worst-case** running time of binary search?
- What is the **best-case** running time of binary search?
- What is the theta for binary search?

Basic Asymptotic Efficiency Classes

Class	Name	Comments
1	*constant*	Short of best-case efficiencies, very few reasonable examples can be given since an algorithm's running time typically goes to infinity when its input size grows infinitely large.
$\log n$	*logarithmic*	Typically, a result of cutting a problem's size by a constant factor on each iteration of the algorithm (see Section 4.4). Note that a logarithmic algorithm cannot take into account all its input or even a fixed fraction of it: any algorithm that does so will have at least linear running time.
n	*linear*	Algorithms that scan a list of size n (e.g., sequential search) belong to this class.
$n \log n$	*linearithmic*	Many divide-and-conquer algorithms (see Chapter 5), including mergesort and quicksort in the average case, fall into this category.

Basic Asymptotic Efficiency Classes

n^2	*quadratic*	Typically, characterizes efficiency of algorithms with two embedded loops (see the next section). Elementary sorting algorithms and certain operations on $n \times n$ matrices are standard examples.
n^3	*cubic*	Typically, characterizes efficiency of algorithms with three embedded loops (see the next section). Several nontrivial algorithms from linear algebra fall into this class.
2^n	*exponential*	Typical for algorithms that generate all subsets of an n-element set. Often, the term "exponential" is used in a broader sense to include this and larger orders of growth as well.
$n!$	*factorial*	Typical for algorithms that generate all permutations of an n-element set.

Recurrences

An equation that describes a function in terms of its value on smaller inputs

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1\,, \end{cases}$$

How to solve?

- Substitution Method = "guess then prove"
- Recursion Tree
- Master Method = Memorize cases of T(n) =aT(n/b) +f(n)

Practice with recursion and recurrences

```
RM1(n)
  if n = 0
      return 0
  else
      return 1 + RM1(n - 1)
```

$$T(n) = \begin{cases} \Theta(1) & if\ n = 0 \\ T(n-1) + \Theta(1) & if\ n > 1 \end{cases}$$

What does this function do?
What is its theta?

Common Recurrences

- $T(n) = T(n/2) + \Theta(1)$ binary search $\Theta(\log n)$
- $T(n) = T(n-1) + \Theta(1)$ linear search $\Theta(n)$
- $T(n) = 2T(n/2) + \Theta(1)$ tree traversal $\Theta(n)$
- $T(n) = 2T(n/2) + \Theta(n)$ merge sort $\Theta(n \log n)$
- $T(n) = T(n-1) + \Theta(n)$ selection sort $\Theta(n^2)$