

1. A recurrence relation describes runtime function recursively for a recursive algorithm. Write a recurrence relation for the Merge sort algorithm. HINT: try to count the number of executions of each statement and the cost of each [Recurrence relations for merge sort lines: \$2 = c_1, 3 = c_2, 4 = 5 = T\left\(\frac{n}{2}\right\)\$ and \$O\(n\)\$ as the runtime for merge](#)

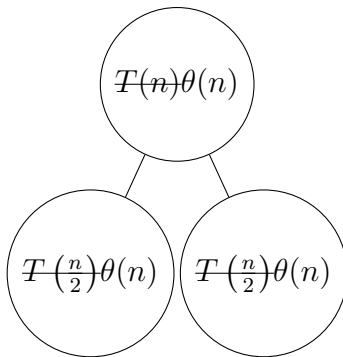
$$T(n) = c_1 + c_2 + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(n)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

Solving Recurrence Relations – Recurrence Tree Method

We solve a recurrence relation to get a function in its closed (non-recursive) form. The recurrence tree method is a visual method of repeatedly substituting in the recurrence relation for $T(n)$ on smaller and smaller n until you reach the base case, and then summing up all the nodes in the tree.

2. Draw the recurrence tree for Mergesort $T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)$, $T(1) = O(1)$



Divide and Conquer Algorithms

- Divide – divide the problem into sub-problems that can be solved independently
- Conquer – recursively solve each sub-problem
- Combine – possibly necessary, combine solutions into sub-problems

Not all problems can be solved with the divide and conquer approach. Maybe sub-problems are not independent, or solutions to sub-problems cannot be combined to find solution to main problem.

3. Write a recursive algorithm for Binary Search. Write and solve its recurrence relation. [Divide & Conquer & Combine Runtime analysis:](#)

Algorithm 1 Binary Search Algorithm

```

1: function BS( $A, \text{key}, i, j$ )                                ▷ Initial call: BS( $A$  (sorted), key, 1,  $n$ )
2:   if  $i \leq j$  then                                          ▷ Handle Base case not found
3:      $k \leftarrow \lfloor \frac{i+j}{2} \rfloor$ 
4:     if  $A[k] == \text{key}$  then
5:       return key
6:     end if
7:   end if
8: end function

```

$$T(n) = O(1) + T\left(\frac{n}{2}\right)$$

$$T(1) = O(1)$$

4. Write a recursive algorithm for Selection Sort (or insertion sort or bubble sort). Write and solve its recurrence relation.

Algorithm 2 Selection Sort Algorithm

```

1: function SELECT( $A, i, j$ )                                ▷ Not a stable sorting algorithm. Initial call: Select( $A$ , 1,  $n$ )
2:   if  $i < j$  then                                          ▷ Base case, 1 item is sorted.
3:     minSoFar =  $A[i]$ 
4:      $kk = i$ 
5:     for  $k = i+1; k \leq j; k++$  do
6:       if  $A[k] < \text{minSoFar}$  then
7:         minSoFar  $\leftarrow A[k]$ 
8:          $kk \leftarrow k$ 
9:       end if
10:    end for
11:    Swap  $A[i]$  with  $A[kk]$ 
12:    Select( $A, i+1, j$ )
13:  end if
14: end function

```

Runtime analysis: $T(n) = T(n-1) + O(n)$, $T(1) = O(1)$

5. Describe an efficient divide and conquer algorithm to count the number of times a character appears in a string of length n .

Algorithm 3 Count of char in string

```

1: function FIND_COUNT(S, from, to, char)
2:   if from < to then
3:     mid ← (from+to)/2
4:     x ← Find_Count(A, from, mid, char)
5:     y ← Find_Count(A, mid+1, to, char)
6:     return x + y
7:   else                                     ▷ 1 character left
8:     if S[from]==char then return 1
9:     elsereturn 0
10:    end if
11:  end if
12: end function

```

Inductive Proofs

(needed in next lecture to prove a solution to a recurrence relation)

6) What are the three steps in an inductive proof?

- Prove for base case
- Assume true for n , prove for larger n

7) Use an inductive proof to show the sum of the first n integers is $\frac{n(n+1)}{2}$

$$\begin{aligned}
 \sum_{k=1}^n k &= \frac{n(n+1)}{2} \\
 \sum_{k=1}^1 k &= 1 = \frac{1(1+1)}{2} \\
 1 &= \frac{1(2)}{2} \\
 1 &= 1 \sum_{k=1}^m k &= \frac{m(m+1)}{2} \\
 \sum_{k=1}^{m+1} k &= \frac{(m+1)(m+2)}{2} \\
 \sum_{k=1}^m k + \sum_{m+1}^{m+1} &= \frac{(m)(m+1)}{2} + (m+1) \\
 &= \frac{(m)(m+1)}{2} + (m+1)
 \end{aligned}$$