CSE 417
Algorithms and Complexity

Winter 2023
Lecture 15
Divide and Conquer and Recurrences

Announcements

- Homework 6, Due Friday, Feb 17

Divide and Conquer

- Recurrences, Sections 5.1 and 5.2
- Algorithms
  - Median (Selection)
  - Fast Matrix Multiplication
  - Counting Inversions (5.3)
  - Multiplication (5.5)

Divide and Conquer : Merge Sort

```
Array Mergesort(Array a){
        n = a.Length;
        if (n <= 1)
                return a;
        b = Mergesort(a[0 .. n/2]);
        c = Mergesort(a[n/2+1 .. n-1]);
        return Merge(b, c);
}
```


Algorithm Analysis

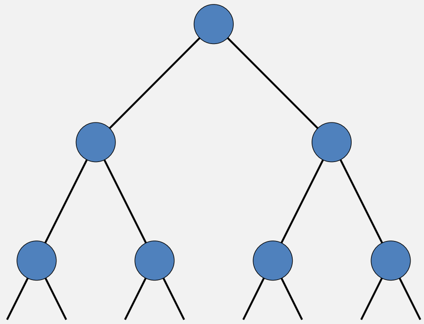
- Cost of Merge
- Cost of Mergesort

$T(n) = 2T(n/2) + cn; T(1) = c;$

Recurrence Analysis

- Solution methods
  - Unrolling recurrence
  - Guess and verify
  - Plugging in to a "Master Theorem"

Unrolling the recurrence



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

Substitution

Prove $T(n) <= n (\log_2 n + 1)$ for $n >= 1$

Induction:
Base Case:

Induction Hypothesis:

A better mergesort (?)

- Divide into 3 subarrays and recursively sort
- Apply 3-way merge

What is the recurrence?

Unroll recurrence for $T(n) = 3T(n/3) + n$

$T(n) = aT(n/b) + f(n)$

T(n) = T(n/2) + cn

Where does this recurrence arise?

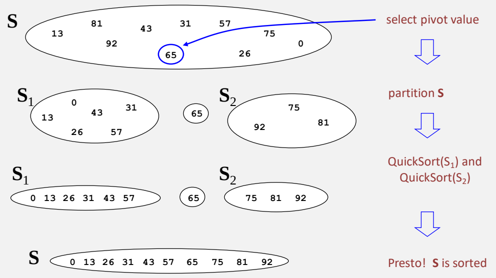
Quicksort

QuickSort(S):

1. Pick an element *v* in **S**. This is the ***pivot*** value.
2. Partition **S**-{*v*} into two disjoint subsets, $\mathbf{S}_1$ and $\mathbf{S}_2$ such that:
   - elements in $\mathbf{S}_1$ are all < *v*
   - elements in $\mathbf{S}_2$ are all > *v*
3. Return concatenation of QuickSort($\mathbf{S}_1$), *v*, QuickSort($\mathbf{S}_2$)

Recursion ends if Quicksort( ) receives an array of length 0 or 1.

The steps of Quicksort



Picking the pivot

- Choose the first element in the subarray
- Choose a value that might be close to the middle
  – Median of three
- Choose a random element

Recurrence for Quicksort

$$QS(n) = \sum_{i=1}^{n} \frac{1}{n} \{QS(i-1) + QS(n-i)\}$$

Computing the Median

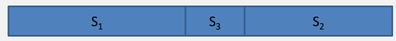
- Given n numbers, find the number of rank n/2
- One approach is sorting
  – Sort the elements, and choose the middle one
  – Can you do better?

Problem generalization

- *Selection*, given n numbers and an integer k, find the k-th largest

Select(A, k)

Select(A, k){
    Choose element x from A
    S_1 = {y in A | y < x}
    S_2 = {y in A | y > x}
    S_3 = {y in A | y = x}
    if ($|S_2|$ >= k)
        return Select(S_2, k)
    else if ($|S_2| + |S_3|$ >= k)
        return x
    else
        return Select(S_1, k - $|S_2|$ - $|S_3|$)
}



Randomized Selection

- Choose the element at random
- Analysis can show that the algorithm has expected run time O(n)

T(n) = T(n/2) + cn

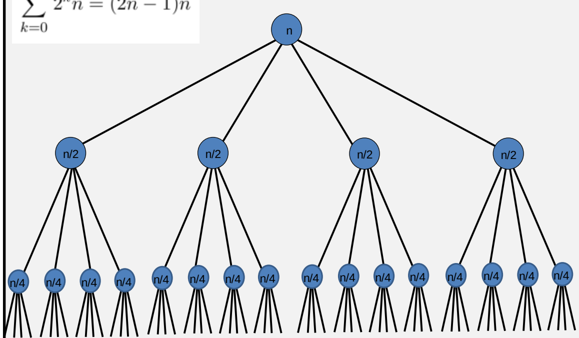
Where does this recurrence arise?

Solving the recurrence exactly

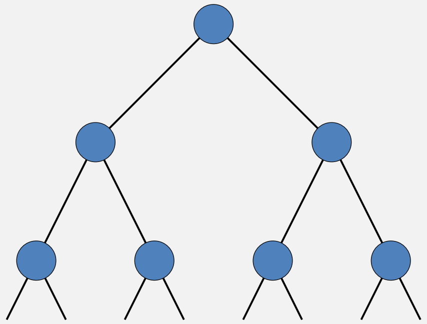
T(n) = 4T(n/2) + n

Total Work

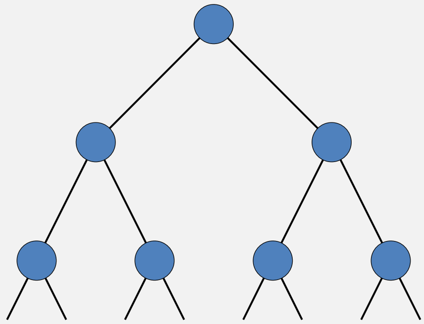
$$\sum_{k=0}^{\log n} 2^k n = (2n-1)n$$



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



$T(n) = 2T(n/2) + n^{1/2}$



Recurrences

- Three basic behaviors
  - Dominated by initial case
  - Dominated by base case
  - All cases equal – we care about the depth

What you really need to know about recurrences

- Work per level changes geometrically with the level
- Geometrically increasing ($x > 1$)
  - The bottom level wins
- Geometrically decreasing ($x < 1$)
  - The top level wins
- Balanced ($x = 1$)
  - Equal contribution

Classify the following recurrences (Increasing, Decreasing, Balanced)

- $T(n) = n + 5T(n/8)$
- $T(n) = n + 9T(n/8)$
- $T(n) = n^2 + 4T(n/2)$
- $T(n) = n^3 + 7T(n/2)$
- $T(n) = n^{1/2} + 3T(n/4)$