CS660: Algorithms - Lecture 3

Instructor: Hoa Vu

San Diego State University

Strassen's algorithm

- For simplicity, assume n is a power of 2.

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

- Then,

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

- Where
  $P_1 = A(F - H), P_2 = (A + B)H, P_3 = (C + D)E, P_4 = D(G - E)$
  $P_5 = (A + D)(E + H), P_6 = (B - D)(G + H), G_7 = (A - C)(E + F).$
- The new running time is $T(n) = 7T(n/2) + O(n^2)$. By master theorem, the running time is $O(n^{\log_2 7}) \approx O(n^{2.81})$.

Quick Sort

- After partitioning, we have two subproblems
  - Sort $A[1 \ldots r-1]$
  - Sort $A[r+1 \ldots n]$
- In the worse case, $r = 2$ (or $r = n$). Then,

$$T(n) = T(n-1) + O(n)$$

  which is $O(n^2)$ (worse than merge sort).
- Idea: pick a pivot randomly which results in running time $O(n \log n)$ with high probability (we will cover this later if time permits).

Quick Select

- Given an array of n numbers. Find the the kth smallest element.
- $MomSelect(A[1 \ldots n], k)$:
  - If $n = 1$, return $A[1]$.
  - Pick a pivot $A[p]$.
  - Call $r \leftarrow Partition(A, p)$.
  - If $r = k$, return $A[r]$.
  - Else if $r > k$, recursively call $MomSelect(A[1 \ldots r-1], k)$.
  - Else if $r < k$, recursively call $MomSelect(A[r+1 \ldots n], k-r)$.
- Worst case running time? $O(n^2)$.

Quick Select

- How to pick a good pivot?
- Divide the array into $\lceil n/5 \rceil$ blocks of size 5.
- In each block, find the median.
- Find the median of the medians (recursively), use that as a pivot.
- The pivot is larger than at least $(n/5)/2 = n/10$ block medians and therefore, larger than at least $3n/10$ elements.
- Similarly, the pivot is smaller than at least $3n/10$ elements.
- After partitioning, the left half and the right half has size at most $7n/10$. In particular,
  $A[1 \ldots r-1]$ has at most 7n/10 elements
  $A[r+1 \ldots n]$ has at most 7n/10 elements

Quick Select

```
MomSelect(A[1..n], k):
  if n ≤ 25   ⟨⟨or whatever⟩⟩
      use brute force
  else
      m ← ⌈n/5⌉
      for i ← 1 to m
          M[i] ← MedianOfFive(A[5i − 4..5i])  ⟨⟨Brute force!⟩⟩
      mom ← MomSelect(M[1..m], ⌊m/2⌋)          ⟨⟨Recursion!⟩⟩

      r ← Partition(A[1..n], mom)

      if k < r
          return MomSelect(A[1..r − 1], k)      ⟨⟨Recursion!⟩⟩
      else if k > r
          return MomSelect(A[r + 1..n], k − r)  ⟨⟨Recursion!⟩⟩
      else
          return mom
```

- Therefore, $MomSelect(A[1 \ldots n], k)$ recursively calls
  - $MomSelect(A[1 \ldots \lceil n/2 \rceil], \lfloor n/2 \rfloor)$,
  - $MomSelect(A[1 \ldots r-1], k)$ or $MomSelect(A[r+1 \ldots n], k-r)$.
- Recurrence: $T(n) = T(n/2) + T(7n/10)$. Show that $T(n) = O(n)$.

Practice

26. Suppose you are given a $2^n \times 2^n$ checkerboard with one (arbitrarily chosen) square removed. Describe and analyze an algorithm to compute a tiling of the board by without gaps or overlaps by **L**-shaped tiles, each composed of 3 squares. Your input is the integer n and two n-bit integers representing the row and column of the missing square. The output is a list of the positions and orientations of $(4^n - 1)/3$ tiles. Your algorithm should run in $O(4^n)$ time. *[Hint: First prove that such a tiling always exists.]*

Practice

Given an array of size n, find the majority element. The majority element is the element that appears more than $\lfloor n/2 \rfloor$ times.
Design an $O(n)$ time algorithm.

Dynamic Programming

- Consider the Fibonacci sequence. $F_0 = 0, F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$.
- Recursion algorithm: $F(n)$: return $F(n-1) + F(n-2)$ (also handle the base case $n = 0$ and $n = 1$).
- Running time? Exponential. $T(n) = T(n-1) + T(n-2) + 1 > 2T(n-2)$.

Dynamic Programming

- What's wrong with recursion? Recompute a term too many times.

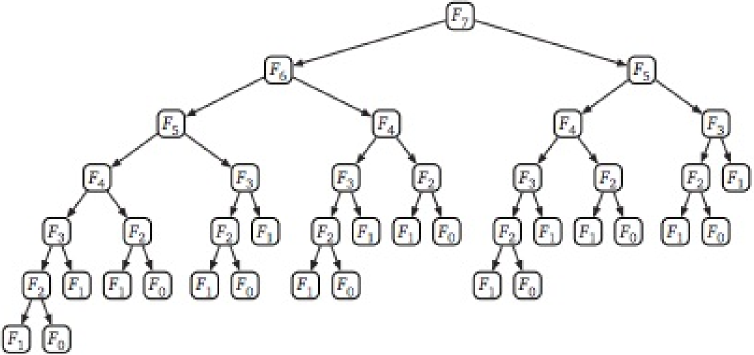

Figure 3.1. The recursion tree for computing F_7; arrows represent recursive calls.

Dynamic Programming

- Dynamic programming: remember the answer.
- $F[0] = 0$, $F[1] = 1$.
- For $i = 2$ to n: $F[i] \leftarrow F[i-1] + F[i-2]$.
- Running time? $O(n)$ if you assume that you can do multiplication in constant time. However, this actually runs in $O(n^2)$ time since the nth Fibonacci number requires $O(n)$ bits to represent and adding two such numbers requires $O(n)$ time.