CS660: Algorithms - Lecture 4

Instructor: Hoa Vu

San Diego State University

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: $T(n) = T(n/2) + T(7n/10) + O(n)$. Show that $T(n) = O(n)$.

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.

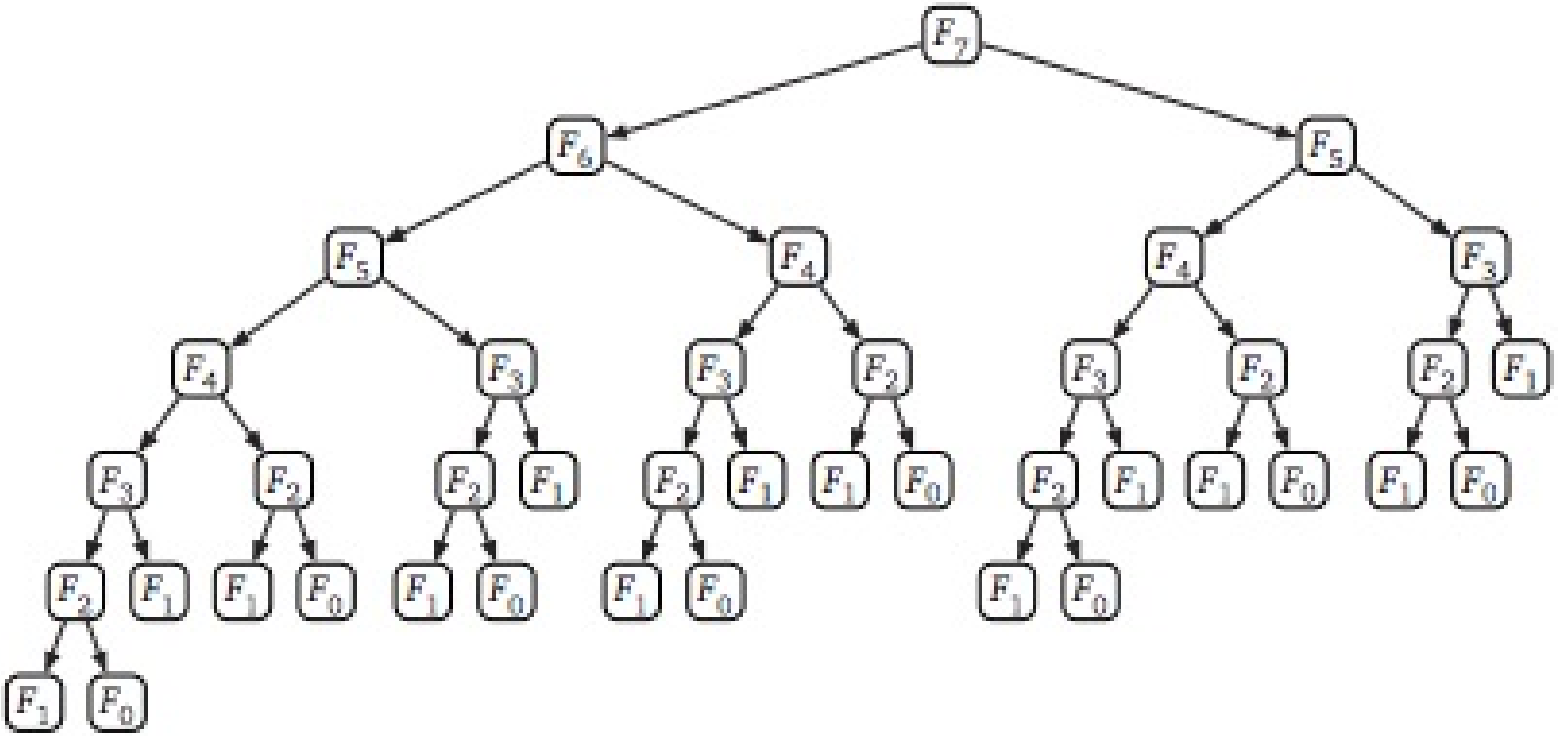

Figure 3.1. The recursion tree for computing F_7; arrows represent recursive calls.

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.

Longest Increasing Subsequence

- Given a list of numbers $A[1 \ldots n]$. Find the longest increasing subsequence.
- We define $LIS[i]$ as the length of the longest increasing subsequence that ends at $A[i]$.
- Let $A[0] = -\infty$.
- Observe that the length of the longest increasing subsequence ending at $A[i]$ is equal to the length of the longest increasing subsequence ending at $A[j]$ for some $j < i$ where $A[j] \leq A[i]$.
- Hence, $LIS[i] = 1 + \max_{0 \leq j < i : A[j] \leq A[i]} LIS[j]$. (how to turn this into code?)
- How to return the actual subsequence?
- Running time? $O(n^2)$.

Subset sum

- Given a list of integers $A[1 \ldots n]$ and a target integer T.
- Decide if there is a subset of the numbers that sum to T. Return the subset if there exists one. Otherwise, output FALSE. Find the longest increasing subsequence.
- Let $SS[i, K] = \mathit{true}$ if some subsets of $A[1 \ldots i]$ sums to K.
- Then $SS[i, K] = \mathit{true}$ if one of the following two cases happens:
  a) $A[i]$ is in the subset that sums to K or
  b) $A[i]$ is not in the subset that sums to K.
- So we have

$$SS[i] = \mathit{true} \text{ iff } SS[i-1, K] = \mathit{true} \text{ or } SS[i-1, K - A[i]] = \mathit{true}.$$

- How to turn the above into actual code?
- Running time? $O(nT)$.