Disc Week 6

Agenda

1. DC - Finding $k^{th}$ smallest in $L$
2. DP - Subsequences

1. **Selection (Divide & Conquer)**

   **Input**: List $L$ and number $k$
   **Output**: $k^{th}$ smallest element of $L = \text{Select}(L,k)$

   *Note*: 1. $\text{Select}(L, \lfloor \frac{|L|}{2} \rfloor)$ returns median of $L$
           2. Sorting $L$ and returning $k^{th}$ element gives $O(n \log n)$ algorithm. Let's do better!

   *Idea*: Pick $v$ and split $L$ into

   $L < v \ , \ L = v \ , \ L > v$

   Then

   if $k \leq |L < v|$ look in left: $\text{Select}(L < v, k)$
   if $k > |L < v| + |L = v|$ look in right: $\text{Select}(L > v, k - |L < v| - |L = v|)$
   if $|L < v| < k \leq |L < v| + |L = v|$ return $v$
* Notice that if we can efficiently choose \( v \) so that 
\[
|\langle u, v \rangle|, |\langle v, v \rangle| \approx \frac{|L|}{2}
\]

\[
T(n) \leq T\left(\frac{n}{2}\right) + O(n) \Rightarrow O(n)
\]

* How do we pick \( v \) in \( O(n) \) time?

Ans: Randomly! (Randomly from \( L \) that is)

Note: Worst case is \( O(n^2) \) if we continually pick \( v \) to be the smallest or largest element of \( L \).

* Best case is \( O(n) \) if we pick the middle every time. What's the average case? Ans: \( O(n) \), why?

Call \( v \) "good" if it lies between the 25th and 75th percentile. \( v \) then has a 50% chance of being "good." On average we need to choose 2 \( v \)'s per iteration to find a good \( v \) which gives

\[
T(n) \leq T\left(\frac{3n}{4}\right) + O\left(2^2n\right) = O(n)!
\]

Expected running time
Dynamic Programming - Contiguous Subsequence

Input: a list of numbers $a_1, ..., a_n$, an
Value of
Output: a Contiguous Subsequence (CSS) of maximum sum.

Ex:

5, 15, -30, 10, -5, 40, 10

cont. SS

- 5, 15, 40 \rightarrow \text{NOT a CSS}
- 10, -5, 40, 10 \rightarrow \text{Max CSS}

How to solve DPs?

- Find a relation that tells you how to
  solve subproblems given the answers to
  smaller subproblems.

Idea: Let $T(j) = \"\text{Max CSS of } a_1 \text{ through } a_j \\text{ using } a_j\"$ (possibly of length zero)

Want: $\max_j T(j)$

Recurrence: $T(j) = \max \{0, a_j + T(j-1)\}$

Base: $T(0) = 0$
Alg

MaxCSS (a_1, ..., a_n)

1. T(0) = 0
2. For i = 1 to n
   \[ T(i) = \max \{ 0, a_i + T(i-1) \} \]
3. return \( \max T(j) \)

Runtime: \( O(n) \)

Correctness:

Claim: \( T(j) = \text{"Max CSS of } a_1 \text{ through } a_j \text{ using } a_j \text{ (and possibly of zero length)"} \)

If we are able to show the claim, then step 3 returns the \( \max T(j) \) which we know will be the overall max CSS since it must end at some \( a_j \) and be the max CSS that ends at that \( a_j \).

Let's prove the claim by induction. The base case \( T(0) = 0 \) is true since we only have the zero length CSS to consider. Assume \( T(j) = \text{"Max CSS ..."} \) for all \( j < k \).

We want to show \( T(k) = \text{"Max CSS ..."} \). There are two options:
1. \( a_k \) can be included in some positive CSS or
2. There are no CSS's which \( a_k \) is a part of. In case (2) the 0-length CSS must be the max and in case 1 \( a_k + T(k-1) \) must be the max CSS (since \( T(k-1) \) is a max CSS ending at \( a_{k-1} \)). This is exactly what we do at step 2. \( \Box \) The claim is shown.