Worksheets up front (take 3sheets)

CS381 PSO 3

TAs: Justin Zhang, Aditya Gandhi
Segyul Park, Sage Stefonic

Problem (Warm up Questions/Test Your Understanding).

How to Multiply Numbers

1. Compute $5 \times 13 =$ _____ in $O\left(n^{\log_2 3}\right)$ time.

2. Okay smart guy, in $O\left(n^{\log_2 7}\right)$ time, compute

$$\begin{bmatrix}(1+3i) & (2+7i)\\(2+i) & (4+i)\end{bmatrix} \times \begin{bmatrix}(13+16i) & (13-34i)\\(-5-10i) & (-5+15i)\end{bmatrix} = \underline{\hspace{3cm}}.$$

3. Solve the following recurrences:

   - $T(n) = 3T(n/2) + n$;
   - $T(n) = 7T(n/2) + n$.

   How do these recurrences relate to the first two questions?

FFT → divide and conquer

$w_{2n}^2 = w_n$

4. (True/[✓ False]) We can multiply two polynomials $p, q \in \mathbb{C}[x]$ of max degree n in $O(n \log n)$ time by running FFT ~~once~~. twice ✓

5. How is divide-and-conquer used in FFT?

6. There is a nice diagram for multiplying two polynomials $p, q \in \mathbb{C}[x]$ of max degree n using FFT.

$$\begin{tikzcd}(p,q) \arrow[r, "O(n\log n)"] & (\overset{\text{FFT}}{F_{2n}p}, \overset{\text{FFT}}{F_{2n}q}) \arrow[r, "O(n)"] & \overset{p*q;}{F_{2n}r} \arrow[r, "O(n\log n)", "\text{i-FFT}"'] & F_{2n}^{*}F_{2n}r\end{tikzcd}$$

coeffients of r = p*q

Explain the middle $O(n)$ step. Justify that the middle step does indeed take at most $O(n)$ time.

Dynamic Programming

1. What are the steps you should take when solving any DP problem?

2. List the required properties of a *metric* $d(x, y)$:

   - (Non-negative). _______________
   - (Reflexive). _______________
   - (Symmetric). _______________
   - (Triangle Inequality). _________________________

3. Recall the recursive case of edit distance (supposing $x[1] \neq y[1]$).

$$\texttt{edit}(x[1\ldots m], y[1\ldots n]) = \min\begin{cases}1 + \texttt{edit}((x[2\ldots m], y[1\ldots n]) & \underline{\hspace{3cm}}\\ 1 + \texttt{edit}((x[1\ldots m], y[2\ldots n]) & \underline{\hspace{3cm}}\\ 1 + \texttt{edit}((x[2\ldots m], y[2\ldots n]) & \underline{\hspace{3cm}}\end{cases}$$

Fill in each blank with which edit (insertion, deletion) each case corresponds to. When $x[1] = y[1]$, how do the above cases change?

4. How big is our “cache” for $\texttt{edit}(x[1\dots m], y[1\dots n])$? That is, how many sub-problems are there for edit distance?

5. Write the recursive algorithm for longest increasing subsequence $\texttt{Lis}(i)$ for array $A[1\dots n]$, where $\texttt{Lis}(i)$ computes the longest subsequence of $A[i,\dots,n]$ *that starts wtih* $A[i]$:

$$\texttt{Lis}(i) = \begin{cases} \underline{\hspace{2.5em}} & \text{if i = n} \\ \underline{\hspace{2.5em}} & \text{if } A[j] \leq A[i] \text{ for all } j > i \\ \underline{\hspace{13em}} & \text{for all } j > i \text{ such that } A[j] > A[i] \end{cases}$$

With caching/dp, how many subproblems? What is the work done per subproblem? Finally, what is the total work done?

$(2n-1)^2 \times$

Problem (FFT Problem). Suppose we have n linear (degree-1) polynomials

$$(a_1 + b_1 x) \times (a_2 + b_2 x), \times \ldots, (a_n + b_n x) = (\_\_\_\_) \quad nx$$

Design and analyze an algorithm to compute the product of these polynomials using FFT.

Guiding Questions

1. First some basics. Test your understanding of FFT on the previous page. What is the time complexity of the naive algorithm in this case (multiplying each polynomial one by one)? What is the max degree of the product of n linear polynomials?

   n

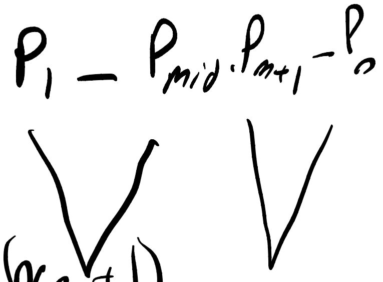
2. What would a recursive spec look like for this problem?

   rec($P_1, P_2, \ldots, P_n$) : Given n linear polys. $P_1, \ldots, P_n$, Compute the product $P_1 \times P_2 \ldots \times P_n$.
   input

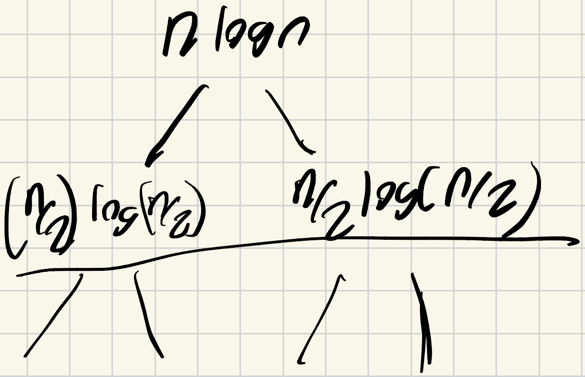
3. How can I use this recursive spec to “divide and conquer?”

```
def rec(P_1, ..., P_n):
        if(n=1): return P_1.
        M = ⌊n/2⌋

    P_1' = rec(P_1, ..., P_m) // max deg (n/2 + 1)
    P_2' = rec(P_{m+1}, ..., P_n) // max deg (n/2 + 1)
    return P_1' × P_2'
           // O(n log n)
```



$T(n) = 2T(n/2) + O(n \log n) \rightarrow O(n \log^2 n)$



= 1st level: $n\log(n/2)$

⋮

2nd level: $n\log(n/4)$

⋮

ith : $n\log(n/2^i)$

$$\sum_{i=1}^{\log n} n\log(n/2^i) = n\sum_{i=1}^{\log n}\log(n/2^i)$$

$$= n\sum(\log n - \log 2^i)$$

$$= n\sum_{i=1}^{\log n}(\log n - \underline{i})$$

$$\leq n\sum_{i=1}^{\log n}\log n$$

$$= n\Big(\underbrace{\log^2 n}_{O(\log^2 n)} - \frac{\log^2 n - \log n}{2}\Big) \qquad = O(n\log^2 n)$$

Problem (FFT Problem). Suppose we have n linear (degree-1) polynomials

$$(a_1 + b_1 x), (a_2 + b_2 x), \ldots, (a_n + b_n x)$$

Design and analyze an algorithm to compute the product of these polynomials using FFT.

(scrap paper)

Problem (DP Problems). A sequence of number $x_1, \ldots, x_k$ is strictly increasing if $x_i < x_{i+1}$ for $i = 1. \ldots, (k-1)$.

(1, 3, 7, 9) ✓

Let $A[1, \ldots, n]$ be an array of n integers.

1. Compute the length of the longest increasing subsequence of A. (1, 3, 2, 7) ✗
2. Compute the maximum sum of any strictly increasing subsequence of A.

Guide

Recall the steps for solving DP problems in the textbook. I've paraphrased them below:

1. Give a concise description of the DP algorithm.
2. Write out a pseudocode/formula for how to use the algorithm (recursively).
3. Explicitly mention how to apply caching/DP. What are your subproblems?
4. What is the runtime? (Often, this can be calculated as the formula below)

   $$\text{Time taken} = (\#\ \text{subproblems}) \times (\text{non-recursive work per subproblem})$$

5. Often times, the DP algorithm differs slightly from the problem spec (e.g., longest subsequence of A is specified by `lis`(A) but depends on DP `Lis`(i)). Write down how to implement the actual problem spec with the DP algorithm.
6. Give an induction proof for correctness, if needed.

1. LI(A): given array A, compute the longest incr. subsequence of A.

↕

[1, 2, 7, 3, 5]

LI[A] = 4

LIS(2) = 3

LIS(3) = 1

dp: LIS(i) = the length of the longest incr. subsequence of A[i, n] that starts with A[i]

[1, 2, 7, 1, 3, 5]

LIS(2)

↳ LIS(3)

LIS(4)

2.

$$LIS(i) = \begin{cases} 1 & \text{if } i = n \quad // A[n,n] = A[n]. \\ \max\left\{1 + LIS(j) \text{ for all } j > i, \text{ and } A[j] > A[i].\right\} \end{cases}$$

3. def LI(A):

```
    return $\max_{i \le n} LIS(i)$ // over all start index i, choose max.
```

Problem (DP Problems). A sequence of number $x_1, \ldots, x_k$ is strictly increasing if $x_i < x_{i+1}$ for $i = 1. \ldots, (k-1)$.

Let $A[1, \ldots, n]$ be an array of n integers.

1. Compute the length of the longest increasing subsequence of A.
2. Compute the maximum sum of any strictly increasing subsequence of A.

(scrap paper)

4. a) runtime of LIS(i)

(# subproblems) × non recursive work.

$n \times O(n)$

$= \underline{O(n^2)}$

b) LI(A) = // use DP to cache LIS(i) calls.

return $\max_{i \leq n} \underline{LIS(i)}$

$= O(n^2)$ // $O(n^2)$ unique LIS calls.