H3.SOLUTION

PROBLEM 1 *Compute the FFT on the values* $(5,8,1,2,2,0,2,2)$. *Illustrate the steps for the first level of recursion, you can assume the base case occurs at* $n=4$. *You can leave your answers in terms of* $\omega_1,\omega_3,\omega_5,\omega_7$, *i.e., without multiplying those roots out.*

Solution: Recall that the purpose of the FFT is to evaluate the polynomial

$$A(x) = 5 + 8x + 1x^2 + 2x^3 + 2x^4 + 2x^6 + 2x^7$$

on the 8 roots of unity which are

ω_0	ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7
1	$\frac{1}{\sqrt{2}}+\frac{i}{\sqrt{2}}$	i	$\frac{-1}{\sqrt{2}}+\frac{i}{\sqrt{2}}$	-1	$\frac{-1}{\sqrt{2}}+\frac{-i}{\sqrt{2}}$	$-i$	$\frac{1}{\sqrt{2}}+\frac{-i}{\sqrt{2}}$

Following the lecture method, we first divide A into two polynomials of half the degree

$$A_e(x) = 5 + 1x + 2x^2 + 2x^3$$
$$A_o(x) = 8 + 2x + 2x^3$$

The FFT calls itself on both of these smaller polynomials. Those recursive calls evaluate A_e, A_o on the 4 roots of unity $\omega_0,\omega_2,\omega_4,\omega_6$ which are $\{1, i, -1, -i\}$. These calls return:

$$\begin{array}{lccccccc} & & 1 & i & -1 & -i & \\ \text{FFT on } A_e\text{:} & \{ & 10, & 3-i, & 4, & 3+i & \} \\ \text{FFT on } A_o\text{:} & \{ & 12, & 8, & 4, & 8 & \} \end{array}$$

Finally, in order to combine these sub-results, we use the equation

$$A(x) = A_e(x^2) + xA_o(x^2)$$

$$\begin{aligned}
A(\omega_0) &= A_e(\omega_0^2) + \omega_0 A_o(\omega_0^2) = A_e(1) + 1A_o(1) = 10 + 1\cdot 12 = \underline{22} \\
A(\omega_1) &= A_e(\omega_1^2) + \omega_1 A_o(\omega_1^2) = A_e(i) + \omega_1 A_o(i) = \underline{(3-i) + 8\omega_1} \\
A(\omega_2) &= A_e(i^2) + iA_o(i^2) = \underline{4+4i} \\
A(\omega_3) &= A_e(-i) + \omega_3 A_o(-i) = \underline{(3+i) + 8\omega_3} \\
A(\omega_4) &= A_e(1) - A_o(1) = 10 - 12 = \underline{-2} \\
A(\omega_5) &= A_e(i) + \omega_5 A_o(i) = \underline{(3-i) + 8\omega_5} \\
A(\omega_6) &= A_e(-1) - iA_o(-1) = \underline{4-4i} \\
A(\omega_7) &= A_e(-i) + \omega_7 A_o(-i) = \underline{(3+i) + 8\omega_7}
\end{aligned}$$

PROBLEM 2 *Lasers!*

The NASA Near Earth Object Program lists potential future Earth impact events that the JPL Sentry System has detected based on currently available observations. Sentry is a highly automated collision monitoring system that continually scans the most current asteroid catalog for possibilities of future impact with Earth over the next 100 years.

This system allows us to predict that i years from now, there will be x_i tons of asteroid material that has near-Earth trajectories. In the mean time, we can build a space laser that can blast asteroids. However, each laser blast will require *exajoules* of energy, and so there will need to be a recharge period on the order of *years* between each use of the laser. The longer the recharge period, the stronger the laser blast; e.g. after j years of charging, the laser will have enough power to obliterate d_j tons of asteroid material. This problem explores the best way to use such a laser.

Example Suppose $(x_1, x_2, x_3, x_4) = (1, 10, 10, 1)$ and $(d_1, d_2, d_3, d_4) = (1, 2, 4, 8)$. The best solution is to fire the laser at times $3, 4$ in order to blast 5 tons of asteroids.

(a) Construct an instance of the problem on which the following "greedy" algorithm returns the wrong answer:

```
BadLaser($(x_1, \dots, x_n), (d_1, \dots, d_n)$)
1  Compute the smallest $j$ such that $d_j \geq x_n$. Set $j = n$ if no such $j$ exists.
2  Shoot the laser at time $n$.
3  if $n > j$ then BadLaser($(x_1, \dots, x_{n-j}), (d_1, \dots, d_{n-j})$).
```

Intuitively, the algorithm figures out how many years (j) are needed to blast all the material in the last time slot. It shoots during that last time slot, and then accounts for the j years required to recharge for that last slot, and recursively considers the best solution for the smaller problem of size $n - j$.

Solution: Consider the problem $(1, 10, 10, 2)$ and the same array $(d_1, \dots, d_4)$ from above. The optimal solution is still $(3, 4)$, but BadLaser instead outputs times $(2, 4)$ which zaps only $4 < 5$.

(b) Given an array holding x_i and d_i, describe an algorithm that blasts the most asteroid material. Before presenting the pseudo-code for the algorithm, present a DP equation that characterizes the optimum choice as we have done in class. Analyze the running time of your solution.

Solution: It is always optimal to zap during the last time slot. The only question is how long to charge for the last zap. The optimal solution at time n maximizes the following

$$\text{BEST}_n = \max_{i=1}^{n} \{ \ \min(d_i, x_n) + \text{BEST}_{n-i} \ \}$$

In other words, we must choose the number of seconds i that we wait to recharge the laser before we fire at time n. If we charge the laser for i years, then the total blast will be the last blast $\min(d_i, x_n)$ plus the amount of asteroid blasted with the best solution that ends at time $n - i$. It is important to take the min between d_i, x_n since there might not be d_i tons of material to zap at time n.

Once we have had the insight into the recursive substructure of the problem, it is fairly straightforward to design an algorithm that starts with the smallest problem size and moves up:

```
LASER($(x_1, \ldots, x_n), (d_1, \ldots, d_n)$)
1  Initialize memory $best[]$ of size $n+1$
2  best[0] = 0
3  for each $j$ from 1 to $n$, $best[j] \leftarrow$ the max $val$ of:
4    for each $i$ from 1 to $j$
5      $val \leftarrow min(d_i, x_j) + best[j-i]$
6  return $best[n]$
```

Notice that it takes $\Theta(i)$ time to compute BEST_i. Therefore, this algorithm requires $\Theta(n^2)$ time.

PROBLEM 3 *Price Run*

Given a list of closing stock ticker prices $p_1, p_2, \ldots, p_n$, devise an $O(n^2)$ algorithm that finds the longest (not necessarily consecutive) streak of prices that increase or stays the same. For example, given the prices $2, 5, 2, 6, 3, 3, 6, 7, 4, 5$, there is the streak $2, 5, 6, 6, 7$ of prices that increase or stay the same, but an even longer streak is $2, 2, 3, 3, 4, 5$.

(Challenge: by using both dynamic programming and binary search, you can solve this problem in $O(n \log n)$ time.)

Solution: Let us define a variable LONG_i to be the length of the longest subsequence of prices that only increase or stay the same in height among the prices from $p_1, \ldots, p_i$ that ends with price i.

How can we define LONG_i in terms of LONG_{i-1}? First observe that $\text{LONG}_1 = 1$. Next, note that if p_i is larger than p_j, then i can be added to the longest subsequence of prices that ended at j to form a longer subsequence.

$$\text{LONG}_i = \max_{j=1}^{i-1} \begin{cases} 1 & \text{if } p_j > p_i \\ \text{LONG}_j + 1 & \text{if } p_j \leq p_i \end{cases} \tag{1}$$

In other words, if p_i is no smaller than p_j, then one can form a sequence of length $1 + \text{TALL}_j$ by considering ..., p_j, p_i where the ... represents the prices in the longest sequence that ends at price j.

Finally, the longest overall sub-sequence is simply $\max_{i=1}^{n}\{\text{LONG}_i\}$. By also recording the value of j used in computing each LONG_i, one can backtrack to find the actual sequence of prices. Computing each value LONG_i takes $O(n)$. There are n such values; the last step of finding the max takes $\Theta(n)$. Thus, the overall running time is $\Theta(n^2)$.

```
PRICES($h_1, \ldots, h_n$)
1  $\text{LONG}_1 \leftarrow 1$
2  for $i = 2$ to $n$
3      compute $\text{LONG}_i$ according to Eq. (1)
4      record argmin index $j$
5  Compute $L$ such that $t_L = \max\{\text{LONG}_1, \ldots, \text{LONG}_n\}$
   ▷ Backtrack to find sequence of prices
```