You want to travel on a straight line from city A to city B which is N miles away from A. For concreteness, imagine a line with A being at 0 and B being at N. Each day you can travel at most d miles (where 0 < d < N), after which you need to stay at an expensive hotel. There are n such hotels between 0 and N, located at points 0 < a_1 < a_2 < \ldots < a_n = N (the last hotel is in B). Luckily, you know that |a_{i+1} - a_i| \leq d for any i (with a_0 = 0), so that you can at least travel to the next hotel in one day. Your goal is to complete your travel in the smallest number of days (so that you do not pay a fortune for the hotels).

Consider the following greedy algorithm: “Each day, starting at the current hotel a_i, travel to the furthest hotel a_j s.t. |a_j - a_i| \leq d, until eventually a_n = N is reached”. I.e., if several hotels are within reach in one day from your current position, go to the one closest to your destination.

(a) (6 points) Formally argue that this algorithm is correct using the “Greedy Stays Ahead” method.

(Hint: Think how to define F_i(Z). For this problem, the name of the method is really appropriate.)

Solution: **************** INSERT YOUR SOLUTION HERE ****************

(b) (6 points) Formally argue that this algorithm is correct using the “Local Swap” method. More concretely, given some hypothetical optimal solution Z of size k and the solution Z* output by greedy, define some solution Z_1 with the following two properties: (1) Z_1 is no worse than Z; (2) Z_1 agrees with greedy in the first day travel plan. After Z_1 is defined, define Z_2 s.t.: (1) Z_2 is no worse than Z_1; (2) Z_2 agrees with greedy in the first two days travel plan. And so on until you eventually reach greedy.

Solution: **************** INSERT YOUR SOLUTION HERE ****************
Recall, Fibonacci numbers are defined by $f_0 = f_1 = 1$ and $f_i = f_{i-1} + f_{i-2}$ for $i \geq 2$.

(a) (2 points) What is the optimal Huffman code for the following set of frequencies which are the first 8 Fibonacci numbers.

Solution: ***************** INSERT YOUR SOLUTION HERE ************ *

(b) (4 points) Let $S_1 = 2 = f_0 + f_1$ and $S_i = S_{i-1} + f_i = \ldots = f_i + f_{i-1} + \ldots f_1 + f_0$ (for $i > 1$) be the sum of the first $i$ Fibonacci numbers. Prove that $S_i = f_{i+2} - 1$ for any $i \geq 1$.

Solution: ***************** INSERT YOUR SOLUTION HERE ************ *

(c) (4 points) Generalize your solution to part (a) to find the shape of the optimal Huffman code for the first $n$ Fibonacci numbers. Formally argue that your tree structure is correct, by using part (b).

Solution: ***************** INSERT YOUR SOLUTION HERE ************ *
You have \( n \) items of weights \( w_1, \ldots, w_n \) that you want to pack. You can use boxes of capacity \( u \) each. (You can assume that \( w_i \leq u \) for all \( i \).) You are allowed to put \textit{at most two} items in a box (even if more than two could fit!). Your goal is to pack all \( n \) items using the \textit{smallest possible} total number of boxes. Consider the following idea for a greedy solution:

1. Take the lightest remaining item \( a \);
2. find the heaviest item \( b \) (if it exists) that will fit in one box with \( a \);
3. Pack these two elements \( a \) and \( b \) into one box (if no \( b \) exists, just pack \( a \));
4. repeat steps (1)-(3) with the remaining items.

(a) (6 points) Assume that the input weights are already sorted: \( w_1 \leq w_2 \leq \ldots \leq w_n \). Write the fastest pseudocode you can find to implement the above described solution. (You will get full credit only for an \( O(n) \) algorithm.)

Solution: ****************** INSERT YOUR SOLUTION HERE ******************

(b) (6 points) Prove the correctness of this algorithm using Local Swap technique.

\textbf{(Hint:} Think of the box containing the lightest item \( a \) in the optimum solution.\textbf{)}

Solution: ****************** INSERT YOUR SOLUTION HERE ******************

(c) \textbf{(Extra credit:} \( ? \) points) Help the authors of this problem and come up with GASA proof for this algorithm.
Recall the following two PFEs $E_1$ and $E_2$ discussed in class. Given a message $m$ of length $n$ (known to encoder), $E_1(m) = 0^n \ 1 \ m$; $E_2(m) = 0^{\lceil \log_2 n \rceil} \ 1 \ \langle n \rangle_2 \ m$, where $\langle n \rangle_2$ is the binary representation of integer $n$. Namely, $E_1$ encodes in unary the length of the string and then the actual string, while the better encoding $E_2$ starts with encoding the length $n$ using $E_1$, and then putting actual string $m$. Recall also the overhead of $E_1$ is $n + 1$ and the overhead of $E_2$ is $2\lceil \log_2 n \rceil + 1 = 2 \log_2 n + O(1)$.

(a) (5 points) Extend the above idea one more iteration to design an encoding $E_3$ having overhead of $\log_2 n + 2 \log_2 \log_2 n + O(1)$.

Solution: ****************** INSERT YOUR SOLUTION HERE ******************

(b) (5 points) Now write a fully recursive procedure (instead of stopping it after a constant number of iterations) to get an encoding $E^*$ having overhead

$$\log_2 n + \log_2 \log_2 n + \log_2 \log_2 \log_2 n + \ldots + O(1)$$

where the number of terms in the sum is the number of times one can take $\log_2$ of $n$ until one reaches 1 (called inverse Ackerman function).

Solution: ****************** INSERT YOUR SOLUTION HERE ******************

---

Turns out this overhead is the best possible one can achieve in any PFE.