Problem specification. We are given an array $A$ of numbers representing prices of some stock at the beginning of trading on consecutive trading days. For example, the following array:

$A = [17, 18, 20, 22, 23]$

indicates that on Day 1 of some five-day period of time, the stock price was 17, on Day 2 - 18, etc., ending with the price of 23 on Day 5.

The BestBuySell problem is defined as follows: *given an array $A$ as described above, return a pair of numbers $(a, b)$ such that both $a$ and $b$ are in $A$, $a$ shows up in the array $A$ before $b$, and $b-a$ is the largest profit that could be made by buying one share of the stock on any day, and selling in on any other day* (i.e., to maximize profit, we want to buy the stock on the day its price was $a$ and sell it on the day its price was $b$).

Our goal is to design a Divide-and-Conquer algorithm for this problem that has $O(n)$ runtime (we will need to prove it) that solves the BestBuySell problem (we will need to prove this too).

We do this in a number of steps.

**Notation:** Let $A = [a_1, ..., a_n]$ and let a pair $(x, y)$ be the output of an algorithm solving the BestBuySell problem. We call $x$ the **buy value** and the day on which the price of that stock was $x$ - the **buy day**. Similarly, we call $y$ the **sell value** and the day on which the price of the stock reached $y$ (after it reached $x$ at an earlier day) the **sell day**.
Step 1.
Most Divide-and-Conquer algorithm design exercises start by trying to split the problem of size $n$ into two separate subproblems of size $n/2$. Consider an array

$$A = [a_1, a_2, ..., a_n]$$

Give some examples of array A, such that:

- **Case 1.** the best profit achieved on this array is in the first half of the array (i.e., both the buy and the sell days are in the first half of the array).

  $$A = [\quad]$$

- **Case 2.** the best profit achieved on array A is in the second half of the array (i.e., both the buy and the sell days are in the first half of the array).

  $$A = [\quad]$$

- **Case 3.** the best profit achieved on array A has the buy day in the first half of the array, and the sell day in the second half of the array.

  $$A = [\quad]$$
Step 2. Consider the situations covered in Case 3 on Step 1. Formulate a rule that would define how to select the appropriate buy day and the appropriate sell day.

That is, answer the following question: if the solution of the BestBuySell problem on array $A$ has the buy day in the first half of $A$ and the sell day in the second half of $A$, then how can we define the buy day and the sell day in this case?

Step 3. Formulate the property that you have established on Step 2 as a statement of a Lemma.

Lemma:
Step 4. Prove formally the Lemma from Step 3.

Proof.
Step 5. Now, let us start developing the **BestBuySell** divide-and-conquer algorithm. We start with the base case.

Describe the condition on the input array $A$ you need to check in order to determine that you are in a base case.

Describe the values you want to return in the base case. How many values are you returning? What is each value?

Write the full pseudocode for the base case.
Step 6. Write the pseudocode for the divide step of the divide-and-conquer algorithm. (I.e., write the part of the BestBuySell algorithm that sets up and executes the recursive calls, and retrieves the results of the recursive calls). Make sure that all values returned by the recursive calls are named explicitly in your pseudocode.
**Step 7.** At this point, we have the base case for our algorithm, and we have the divide portion of the inductive case - i.e., the portion of the algorithm that prepares and issues the recursive calls, and collects the results of the recursive calls. What is left is to build the combine step.

Describe, in your own words, how you want to proceed with the combine step. What are the checks you are planning to make? What will you return for each of the paths in your algorithm (based on how the checks went)?
Step 8. Write the pseudocode for the *combine (conquer)* part of the divide-and-conquer BestBuySell algorithm. This part of the pseudocode must result in a return of the necessary values for each execution path.
Step 9. Analyze the performance of your **BestBuySell** algorithm. How many recursive calls are being made? Buy how much does each call reduce the size of the problem? How much effort is spent combining the outputs of all the recursive calls?

Based on your analysis, provide the recurrence relation describing the runtime of your algorithm.

Does this recurrence relation fall under the purview of the Master Theorem? If yes - solve the recurrence relation