Best Time to Buy and Sell Stock

You are given an array `prices` where `prices[i]` is the price of a given stock on the `i`th day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return `0`.

Example 1:

```
Input: prices = [7,1,5,3,6,4]

Output: 5

Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
```

Approach 1: Brute Force

```
public class Solution {
   public int maxProfit(int prices[]) {
      int maxprofit = 0;
      for (int i = 0; i < prices.length - 1; i++) {
         for (int j = i + 1; j < prices.length; j++) {
            int profit = prices[j] - prices[i];
            if (profit > maxprofit)
               maxprofit = profit;
         }
      }
      return maxprofit;
   }
}
```

Complexity Analysis

- Time complexity: $O(n^2)$. Loop runs $\dfrac{n (n-1)}{2}$ times.
- Space complexity: $O(1)$. Only two variables - maxprofit and profit are used.

Approach 2:

This problem can be solved using greedy approach. To maximize the profit we have to minimize the buy cost and we have to sell it on maximum price.

Follow the steps below to implement the above idea:

Declare a buy variable to store the buy cost and max_profit to store the maximum profit.

Initialize the buy variable to first element of profit array.

Iterate over the prices array and check if the current price is minimum or not.

If the current price is minimum then buy on this ith day.

If the current price is greater then previous buy then make profit from it and maximize the max_profit.

Finally return the max_profit.

```
int maxProfit(int prices[], int n)
{
   int buy = prices[0], max_profit = 0;
   for (int i = 1; i < n; i++) {

      // Checking for lower buy value
      if (buy > prices[i])
         buy = prices[i];

      // Checking for higher profit
      else if (prices[i] - buy > max_profit)
         max_profit = prices[i] - buy;
   }
   return max_profit;
}
```

Time Complexity: O(N). Where **N** is the size of prices array.
Auxiliary Space: O(1). We do not use any extra space.