[Analysis of Assignment 2 Problem Statement 5 2](#_Toc32778247)

[Program Logic 2](#_Toc32778248)

[Iterative Method 2](#_Toc32778249)

[Divide and Conquer Method 2](#_Toc32778250)

[Time Complexity 4](#_Toc32778251)

[Iterative Method 4](#_Toc32778252)

[Divide and Conquer Method 4](#_Toc32778253)

[Space Complexity 4](#_Toc32778254)

# Analysis of Assignment 2 Problem Statement 5

## Program Logic

The program computes the maximum profit upon buying and subsequently selling one unit of stock via two different methods.

### Iterative Method

* Consider min\_day as a prospective buy day (for which a future stock price will decide whether this can be claimed as a real buy day.), initially set to 0th day
* Consider initial profit = 0.
* From day 1 onwards, start calculating profit.
* If, for any day, the profit obtained (current day value – min\_day value) is higher than previous profit value, set the buy\_day to min\_day, sale\_day to current day)
* If at any point the value is lower than min\_day value (prospective buy day value), consider current day as a new prospective buy day.
* repeat.

Example with the dataset given in the assignment:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Day | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Price | 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | min day | max profit | Buy day | Sell day |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 0 | 0 | 0 | 1 |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 0 | **13** | **0** | **1** |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 0 | 13 | 0 | 1 |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | **3** | 13 | 0 | 1 |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 3 | **20** | **3** | **4** |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 3 | 20 | 3 | 4 |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 3 | 20 | 3 | 4 |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | **7** | 20 | 3 | 4 |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 7 | 20 | 3 | 4 |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 7 | **38** | **7** | **9** |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 7 | 38 | 7 | 9 |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 7 | **43** | **7** | **11** |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 7 | 43 | 7 | 11 |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 7 | 43 | 7 | 11 |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 7 | 43 | 7 | 11 |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 7 | 43 | 7 | 11 |
| 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 | 7 | 43 | 7 | 11 |

(\*) value updates are marked with orange color, blue color indicates the current iteration of the for-loop.

### Divide and Conquer Method

* Calculate a profits array by calculating day1 value - day0 value, day2 value - day1 value and so on.
* Recursively, divide the profit array in two parts, continue the process till we have only 1 element left.
* Start merging the left and right subarrays such that only the maximum subarray sum of left, right or left-to-right is retained for next level. Here:

left\_total is the maximum profit subarray on the left of the midpoint,

right\_total is the maximum profit subarray on the right of the midpoint, and

lr\_total is the maximum profit subarray from midpoint to left combined with midpoint to right, i.e. across the midpoint.

* The algorithm also returns the start and end index of the profit array, along with the max profit value.
* In case of a negative maximum/profit value, it returns 0.
* Additionally, note that the stock buy\_date index shall be 1 less than the start index returned by the max\_profit algorithm.

As an example, with the given dataset in the problem statement, it is recursively divided into subparts as follows:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Day | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Price | 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 |
| Profit | 0 | 13 | -3 | -25 | 20 | -3 | -16 | -23 | 18 | 18 | -7 | 12 | -5 | -22 | 15 | -4 | 7 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Stage/Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Divide | 0 | 13 | -3 | -25 | 20 | -3 | -16 | | -23 | 18 | 20 | -7 | 12 | -5 | -22 | 15 | -4 | 7 |
| 0 | 13 | -3 | -25 | 20 | -3 | -16 | | -23 | 18 | 20 | -7 | 12 | -5 | -22 | 15 | -4 | 7 |
| 0 | 13 | -3 | -25 | 20 | -3 | -16 | | -23 | 18 | 20 | -7 | 12 | -5 | -22 | 15 | -4 | 7 |
| 0 | 13 | -3 | -25 | 20 | -3 | -16 | | -23 | 18 | 20 | -7 | 12 | -5 | -22 | 15 | -4 | 7 |
| 0 | 13 |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |
| Merge | max (0, 13, 0+13) = 13 | | -3 | -25 | 20 | -3 | | -16 | -23 | 18 | 20 | -7 | 12 | -5 | -22 | 15 | -4 | 7 |
| max (13, -3, 13+0) = 13 | | | max (-25, 20, 0+20) = 20 | | max (-3, -16, 0+0) = 0 | | | max (-23, 18, 0+18) = 18 | | max (20, -7, 20+0) = 20 | | max (12, -5, 12+0) = 12 | | max (-22, 15, 0+15) = 15 | | max (-4, 7, 0+7) = 7 | |
| max (13, 20, 10+0) = 20 | | | | | max (0, 18, 0+0) = 18 | | | | | max (20, 12, 13+12) = 25 | | | | max (15, 7, 15+3) = 18 | | | |
| max (20, 18, 20+0) = 20 | | | | | | | | | | max (25, 18, 20+0) = 25 | | | | | | | |
| max (20, 25, 18+25) = 43 | | | | | | | | | | | | | | | | | |

The convention max (l, r, t) means, max (left\_max, right\_max, left\_total + right\_total)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Left half |  |  | Right half |

## Time Complexity

### Iterative Method

The iterative method uses a single for-loop to find the maximum profit. All the statements inside the for-loop use constant time complexity. Hence given **n** data points, the time complexity of the iterative method is:

###### O(n)

### Divide and Conquer Method

The divide and conquer method recursively divides the input dataset into two parts, while the corresponding merge parts runs for O(n) complexity to merge the left and right subparts.

The recurrence relation for divide and conquer algorithm can be written as:

###### T(n) = 2\*T(n/2) + n

###### where n > 1

Solving the above with master theorem yields:

###### T(n) = O (n log(n))

## Space Complexity

Both the Iterative and Divide / Conquer methods use a single, globally available list to store the input data set. The same list is passed (by object reference) to all the recursive calls in Divide and Conquer algorithm, so the space usage remains constant.

Hence for both the algorithms, the space complexity for **n** input data points:

###### O(n)