Program 3 Report Jack Casey

Stock Trading Profit Maximization

Problem Statement

Given a sequence of stock prices, find the maximum profit that can be obtained by buying a stock at a given day and selling it on a later day. The goal is to design an efficient algorithm using the divide and conquer strategy to solve this problem.

Divide and Conquer Approach

Design

The divide and conquer approach for this problem involves the following steps:

1. Divide the sequence of stock prices into two halves.
2. Recursively find the maximum profit in the left half and the right half.
3. Find the maximum profit that can be obtained by buying in the left half and selling in the right half (crossing subproblem).
4. Return the maximum of the three profits obtained in steps 2 and 3.

The base case for the recursion is when the sequence contains only one or two elements. In this case, we can directly compute the maximum profit.

Pseudo Code

```
function maxProfit(prices, start, end)
    if start == end
        return 0
    if end - start == 1
        return max(0, prices[end] - prices[start])

    mid = (start + end) / 2
    leftProfit = maxProfit(prices, start, mid)
    rightProfit = maxProfit(prices, mid + 1, end)
    crossProfit = maxCrossingProfit(prices, start, mid, end)
    return max(leftProfit, rightProfit, crossProfit)

function maxCrossingProfit(prices, start, mid, end)
    leftMin = prices[mid]
    rightMax = prices[mid + 1]

    for i = mid downto start
        leftMin = min(leftMin, prices[i])
    for i = mid + 1 to end
        rightMax = max(rightMax, prices[i])

    return max(0, rightMax - leftMin)
```

Time Complexity Analysis

Let T(n) be the time complexity of the `maxProfit` function for a sequence of length n.
The recurrence equation for T(n) is: $T(n) = 2T(\frac{n}{2}) + \Theta(n)$
Using the Master Theorem, we can solve the recurrence equation:
a = 2, b = 2, f(n) = $\Theta(n)$
Since f(n) = $\Theta(n)$ and $n^{\log_b(a)} = n^{\log_2(2)} = n$, we have: f(n) = $\Theta(n^{\log_b(a)})$
Therefore, the time complexity of the divide and conquer approach is T(n) = $\Theta(n \log n)$.

Linear Time Divide and Conquer (Extra Credit)

Design

To achieve a linear time complexity of $\Theta(n)$ using divide and conquer, we can modify the approach as follows:

1. Divide the sequence into two halves.
2. Recursively find the maximum profit in the left half and the right half, along with the minimum price in the left half and the maximum price in the right half.
3. Compute the maximum profit that can be obtained by buying in the left half and selling in the right half using the minimum price in the left half and the maximum price in the right half.
4. Return the maximum of the profits obtained in steps 2 and 3.

Pseudo Code

```
function maxProfit(prices, start, end)
    if end - start ≤ 2
        return maxProfitBaseCase(prices, start, end)

    mid = (start + end) / 2
    leftResult = maxProfit(prices, start, mid)
    rightResult = maxProfit(prices, mid + 1, end)
    crossProfit = prices[rightResult.maxRightIndex] -
prices[leftResult.minLeftIndex]

    if leftResult.maxProfit ≥ rightResult.maxProfit and
leftResult.maxProfit ≥ crossProfit
        return leftResult
    else if rightResult.maxProfit ≥ crossProfit
        return rightResult
    else
        return {
            buyIndex: leftResult.minLeftIndex,
            sellIndex: rightResult.maxRightIndex,
            maxProfit: crossProfit,
            minLeftIndex: min(leftResult.minLeftIndex,
rightResult.minLeftIndex),
            maxRightIndex: max(leftResult.maxRightIndex,
rightResult.maxRightIndex)
        }

function maxProfitBaseCase(prices, start, end)
    if end - start == 1
        return {
            buyIndex: start,
            sellIndex: end,
            maxProfit: prices[end] - prices[start],
            minLeftIndex: prices[start] ≤ prices[end] ? start : end,
            maxRightIndex: prices[start] ≤ prices[end] ? end : start
        }
    else if end - start == 2
        profit1 = prices[end] - prices[start]
        profit2 = prices[end - 1] - prices[start]
        profit3 = prices[end] - prices[start + 1]

        if profit1 ≥ profit2 and profit1 ≥ profit3
            buyIndex = start
            sellIndex = end
            maxProfit = profit1
        else if profit2 ≥ profit3
            buyIndex = start
            sellIndex = end - 1
            maxProfit = profit2
        else
            buyIndex = start + 1
            sellIndex = end
            maxProfit = profit3

        minLeft = min(prices[start], prices[end - 1], prices[end])
        maxRight = max(prices[start], prices[end - 1], prices[end])

        minLeftIndex = prices[start] == minLeft ? start : (prices[end - 1]
== minLeft ? end - 1 : end)
        maxRightIndex = prices[start] == maxRight ? start : (prices[end -
1] == maxRight ? end - 1 : end)

        return {
            buyIndex: buyIndex,
            sellIndex: sellIndex,
            maxProfit: maxProfit,
            minLeftIndex: minLeftIndex,
            maxRightIndex: maxRightIndex
        }
```

Time Complexity Analysis

Let T(n) be the time complexity of the `maxProfit` function for a sequence of length n using the linear divide and conquer approach.

The recurrence equation for T(n) can be written as:

- $T(n) = 2T(\frac{n}{2}) + \Theta(1)$ for $n > 2$

- $T(n) = \Theta(1)$ for $n \leq 2$

In this case, the recurrence equation is slightly different from the previous one because the base case handles sequences of length 1 and 2 in constant time, and the recursive case divides the problem into two subproblems of roughly equal size.

Using the Master Theorem, we can solve the recurrence equation:
a = 2, b = 2, $f(n) = \Theta(1)$

Since $f(n) = \Theta(1)$ and $n^{\log_b(a)} = n^{\log_2(2)} = n$, we have: $f(n) = O(n^{\log_b(a)-\epsilon})$ for some constant $\epsilon > 0$.

Therefore, the time complexity of the linear divide and conquer approach falls into case 2 of the Master Theorem, and we have:
$T(n) = \Theta(n^{\log_b(a)}) = \Theta(n)$

So, the time complexity of the linear divide and conquer approach for finding the maximum profit in stock trading is $\Theta(n)$, which is a linear time complexity.

Decrease and Conquer (Extra Credit)

Design

A decrease and conquer approach can also be used to solve this problem in $\Theta(n)$ time complexity. The idea is to iterate through the sequence of prices and keep track of the minimum price seen so far and the maximum profit that can be obtained by selling at the current price.

Pseudo Code

```
function maxProfit(prices)
    min = prices[0]
    maxProfit = 0
    buyPos = 0
    sellPos = 1
    minIndex = 0

    for i = 1 to prices.length - 1
        if prices[i] < min
            min = prices[i]
            minIndex = i
        if prices[i] - min > maxProfit
            maxProfit = prices[i] - min
            buyPos = minIndex
            sellPos = i

    return {
        buyPos: buyPos,
        sellPos: sellPos,
        maxProfit: maxProfit
    }
```

Time Complexity Analysis

The time complexity of this approach is $\Theta(n)$ since it involves a single pass through the sequence of prices.The time complexity can be expressed as $T(n) = 1 + \sum_{i=1}^{n-1} 2 = 2(n-1) + 1 = 2n - 3 \in \Theta(n)$

Conclusion

The divide and conquer strategy provides an efficient solution to the stock trading profit maximization problem. The basic approach has a time complexity of $\Theta(n \log n)$, which is achieved by dividing the problem into smaller subproblems and combining their solutions.

However, with some modifications, it is possible to achieve a linear time complexity of $\Theta(n)$ using divide and conquer. This is done by recursively finding the maximum profit, minimum price in the left half, and maximum price in the right half, and then computing the maximum profit that can be obtained by buying in the left half and selling in the right half.

Furthermore, a decrease and conquer approach can also solve the problem in $\Theta(n)$ time complexity by iterating through the sequence of prices and keeping track of the minimum price seen so far and the maximum profit that can be obtained by selling at the current price.

In summary, the divide and conquer strategy provides a solid foundation for solving the stock trading profit maximization problem efficiently, with the potential for further optimizations to achieve a linear time complexity.