**Q.7: You are consultant working for a small computation-intensive investment**

**company, and they have the following type of problem they want to solve over and**

**over. A typical instance of the problem is the following. They are doing a simulation**

**in which they look at n consecutive days of a given stock, at some point in the past.**

**Let’s number the days = 1,2, . . . ; for each day i, they have a price "() per share**

**for the stock on that day (For simplicity, the price was same during each day).**

**Suppose during this time period, they wanted to buy 1000 shares on some day and sell**

**all these shares on some (later) day. They want to know: When should they have**

**bought and when should they have sold in order to have made as much money as**

**possible. (If there was no way to make money during the n days, you should report**

**this instead).**

**For example, suppose n = 3, p(1)=9, p(2)= 1, p(3)= 5. Then you should return**

**“Buy on day 2 and selling on day 3” means that they would have made $4 per share.**

**Show how to find the correct number I and j In time O(nlogn)**

Sol - We’ve seen a number of instances in this chapter where a brute force search over pairs of elements can be reduced to O(n log n) by divide and conquer.

Since we’re faced with a similar issue here, let’s think about how we might apply a divide-and-conquer strategy. A natural approach would be to consider the first n/2 days and the final n/2 days separately, solving the problem recursively on each of these two sets, and then figure out how to get an overall solution from this in O(n) time. This would give us the usual recurrence T(n) ≤ 2T n 2 + O(n), and hence O(n log n) . Also, to make things easier, we’ll make the usual assumption that n is a power of 2. This is no loss of generality: if n ′ is the next power of 2 greater than n, we can set p(i) = p(n) for all i between n and n ′ . In this way, we do not change the answer, and we at most double the size of the input (which will not affect the O() notation).

Now, let S be the set of days 1, . . . , n/2, and S ′ be the set of days n/2 + 1, . . . , n. Our divide-and-conquer algorithm will be based on the following observation: either there is an optimal solution in which the investors are holding the stock at the end of day n/2, or there isn’t. Now, if there isn’t, then the optimal solution is the better of the optimal solutions on the sets S and S ′ . If there is an optimal solution in which they hold the stock at the end of day n/2, then the value of this solution is p(j) − p(i) where i ∈ S and j ∈ S ′ . But this value is maximized by simply choosing i ∈ S which minimizes p(i), and choosing j ∈ S ′ which maximizes p(j).

Thus our algorithm is to take the best of the following three possible solutions

* The optimal solution on S.
* The optimal solution on S ′ .
* The maximum of p(j) − p(i), over i ∈ S and j ∈ S ′ . The first two alternatives are computed in time T(n/2), each by recursion, and the third alternative is computed by finding the minimum in S and the maximum in S ′ , which takes time O(n). Thus the running time T(n) satisfies T(n) ≤ 2T (n /2) + O(n), as desired. We note that this is not the best running time achievable for this problem. In fact, one can find the optimal pair of days in O(n) time using dynamic programming