Problem 3-1 (Stock Profit) 10 (+6) Points

Sometimes, computing “extra” information can lead to more efficient divide-and-conquer algorithms. As an example, we will improve on the solution to the problem of maximizing the profit from investing in a stock (page 68-74).

Suppose you are given an array $A$ of $n$ integers such that entry $A[i]$ is the value of a particular stock at time interval $i$. The goal is to find the time interval $(i, j)$ such that your profit is maximized by buying at time $i$ and selling at time $j$. For example, if the stock prices were monotone increasing, then $(1, n)$ would be the interval with the maximal profit ($A[n] - A[1]$). More formally, the current formulation of the problem has the following input/output specification:

**Input:** Array $A$ of length $n$.


(a) (6 Points) Suppose you change the input/output specification of the stock problem to also compute the largest and the smallest stock prices:

**Input:** Array $A$ of length $n$.

**Output:** Indices $i \leq j$ maximizing ($A[j] - A[i]$), and indices $\alpha, \beta$ such that $A[\alpha]$ is a minimum of $A$ and $A[\beta]$ is a maximum of $A$.

Design a divide-and-conquer algorithm for this modified problem. Make sure you try to design the most efficient “conquer” step, and argue why it works. How long in your conquer step? (Hint: When computing optimal $i$ and $j$, think whether the “midpoint” $n/2$ is less than $i$, greater than $j$ or in between $i$ and $j$.)

(b) (4 Points) Formally analyze the runtime of your algorithm and compare it with the runtime of the solution for the Stock Profit problem in the book.

(c) (Extra Credit: 6 Points) Design a direct, non-recursive algorithm for the Stock Profit problem which runs in time $O(n)$. Write its pseudocode. Ideally, you should have a single “For $i = 1$ to $n$” loop, and inside the loop you should maintain a few “useful counters”. Formally argue the correctness of your algorithm. (Hint: Scan the array left to right and maintain its running minimum and the best solution found so far. Under which conditions would the best current solution be improved when scanning the next array element?)
Problem 3-2 (Local Minimum) 10 Points

A local minimum of an array $A[1], \ldots, A[n]$ is an index $i \in \{1, \ldots, n\}$ such that either (a) $i = 1$ and $A[1] \leq A[2]$; or (b) $i = n$ and $A[n] \leq A[n-1]$; or (c) $1 < i < n$ and $A[i] \leq A[i-1]$ and $A[i] \leq A[i+1]$. Note that every array has at least (and possibly more than) one local minimum, since the “global” minimum of the entire array is also a local minimum. Design an $O(\log n)$ divide-and-conquer algorithm to find some local minimum of a given (unsorted) array $A$ of size $n$.

(Hint: Think of binary search for inspiration.)

Problem 3-3 (Identity Element in an Array) 10 Points

Let $A$ be an array with $n$ distinct integer elements in sorted order. Consider the following algorithm $\text{IdFind}(A, j, k)$ that finds an $i \in \{j \ldots k\}$ such that $A[i] = i$, or returns FALSE if no such element $i$ exists.

1 $\text{IdFind}(A, j, k)$
2 \hspace{1em} If $j > k$ Return FALSE
3 \hspace{1em} Set $i := \ldots$
4 \hspace{1em} If $A[i] = \ldots$ Return \ldots
5 \hspace{1em} If $A[i] < \ldots$ Return $\text{IdFind}(A, \ldots, \ldots)$
6 \hspace{1em} Return $\text{IdFind}(A, \ldots, \ldots)$

(a) (3 points) Fill in the blanks (denoted \ldots) to complete the above algorithm.

(b) (5 points) Prove correctness and analyze the running time of the algorithm.

(Notice the emphasis on Prove, you can’t just say “my algorithm works because it works”.)

(c) (2 points) Does the algorithm work if the elements of $A$ are not distinct? Why or why not?

Problem 3-4 (QuickSort vs Insertion Sort) 10 Points

We say that an array $A$ is $c$-nice, where $c$ is a constant (think 100), if for all $1 \leq i, j \leq n$, such that $j - i \geq c$, we have that $A[i] \leq A[j]$. For example, 1-nice array is already sorted. In this problem we will sort such $c$-nice $A$ using INSERTION SORT and QUICKSORT, and compare the results.

(a) (4 Points) In asymptotic notation (remember, $c$ is a constant) what is the worst-case running time of INSERTION SORT on a $c$-nice array? Be sure to justify your answer.

(b) (5 Points) In asymptotic notation (remember, $c$ is a constant) what is the worst-case running time of QUICKSORT on a $c$-nice array? Be sure to justify your answer.

(c) (1 Points) Which algorithm would you prefer?