Algorithms and Data Structures 2014/15
Coursework 1
Issue date: Monday, 29th September, 2014

The deadline for this coursework is 4pm, Friday 17th October, 2014. Submit your solution to the ITO in Appleton Tower. Your coursework should be submitted in clear handwriting (unless Special Circumstances prevent this for you). Remember that the School’s policy is that late coursework will not be accepted without good reason. If you have good reason, this goes through your PT.

This coursework should be your own individual work. You may discuss understanding of the questions with your classmates, but may not share solutions, or give strong hints. If you use any resources apart from the course slides/notes, or the book, you must cite these.

This exercise is worth 50% of the coursework for A&DS. The “marks” given in the margins of this sheet may be interpreted as percentages of the overall coursework credit for A&DS.

1. (10 marks in total) The investment firm Logit Consulting wants to evaluate its own investment history for a given stock against the optimum buy-sell for that stock.

The price history of a particular stock is represented by an array $A$ of length $n$, where each cell contains a date ($A[i].date$), a time ($A[i].time$), and the value ($A[i].val$) of the stock at that date and time. The array entries are presented in increasing order of date and time. The optimum buy-sell of the array is defined to be the value

$$ \max_{i,j:1 \leq i \leq j \leq n} \{(A[j].val - A[i].val)\}.$$

Logit Consulting needs an efficient algorithm to compute the optimum buy-sell for a given array.

(a) Give an example of a small array in which the optimum buy-sell neither involves the maximum value in the array, nor the minimum value in the array. [2 marks]

(b) Sketch a simple $O(n^2)$ algorithm to find the optimum buy-sell of a given array. [2 marks]

(c) Give a Divide-and-Conquer algorithm which finds the optimum buy-sell of a given array in $O(n \log(n))$ time.

For this part of the question, you must argue that your algorithm is correct, and you must justify that the running time is $O(n \log(n))$ (you may use the Master Theorem for this). [6 marks]
2. (24 marks in total) In this question we prove a fragment of the Master theorem. Suppose we are given a recurrence of the following form:

\[
T(n) = \begin{cases} 
1 & n = 1 \\
aT([n/2]) + nk & n \geq 2
\end{cases},
\]

(1)

for some fixed \(a, k \in \mathbb{N}, a \geq 1, k \geq 1\), where \(aT([n/2])\) is written to indicate \(a\) recursive calls, each being either size \([n/2]\) or \([n/2]\).

In this question we will prove the \(O(\cdot)\) bound of the Master Theorem for this recurrence (for the cases \(\lg(a) > k\) and \(\lg(a) = k\)).

(a) Prove by induction that \(T(n) < T(m)\) for all \(n, m \in \mathbb{N}\), \(n < m\) (this shows that \(T(\cdot)\) is monotonically increasing).

Remember to show this for general \(n, m\) (not just powers of 2), taking care to allow for any (fixed) combination of \(\lfloor \cdot \rfloor\) and \(\lceil \cdot \rceil\) options for the \(a\) recursive terms.

(b) Prove by induction that when \(n\) is a power of 2, then

\[T(n) = n^{\lg(a)} + nk^{\sum_{j=0}^{\lg(n)-1} (\frac{a}{2k})^j}\].

(c) Prove that when \(n\) is a power of 2, and when \(\lg(a) = k\), that

\[T(n) = n^{\lg(a)}(\lg(n) + 1)\].

Hint: You will need to use (b), or alternatively prove this by induction from (1).

(d) Prove that \(T(n) = O(n^{\lg(a)} \lg(n)) = O(n^{k} \lg(n))\) for the case of \(\lg(a) = k\) for all \(n \in \mathbb{N}\) from first principles.

Hint: For general \(n \in \mathbb{N}\), you will need to define an appropriate power-of-2 close to \(n\), use (a) to relate \(T(n)\) to \(T(\cdot)\) for that power-of-2, apply (c), and come up with an \(n_0\) and \(c\) for the \(O(\cdot)\) bound.

(e) Prove that when \(n\) is a power of 2, and when \(\lg(a) > k\), that

\[T(n) = \frac{2^k}{a - 2k} \left(\frac{a}{2k}\right)^{\lg(n)} - n^k\].

Hint: You will need to use the result of (b), or alternatively prove this by induction using (1). You may need to use the relationship between \(x^{\lg(n)}\) and \(n^{\lg(x)}\), plus other properties of logs, in your proof.

(f) Prove that \(T(n) = O(n^{\lg(a)})\) for the case of \(\lg(a) > k\) for all \(n \in \mathbb{N}\), giving details of \(n_0\) and \(c\) in the application of \(O\).

Hint: For general \(n \in \mathbb{N}\), you will need to define an appropriate power-of-2 close to \(n\), use (a) to relate \(T(n)\) to this close power-of-2, and then apply (e). You will then need to come up with an \(n_0\) and \(c\).
3. (16 marks in total) The question deals a variant of comparison-based sorting called Average sorting. We will be interested in upper and lower bounds.

Suppose that, instead of sorting an array completely, we only require that the elements increase on average. Formally, for any natural number $k \geq 1$, we say that the $n$-element array $A$ is $k$-sorted if, for every index $i = 1, 2, \ldots, n-k$, we have:

$$\frac{\sum_{j=i}^{i+k-1} A[j]}{k} \leq \frac{\sum_{j=i+1}^{i+k} A[j]}{k}.$$

(a) What does it mean for an array to be 1-sorted? [1 marks]

(b) Give a permutation of the numbers $1, 2, \ldots, 10$ that is 2-sorted, but is not sorted (according to the traditional definition). [3 marks]

(c) Now prove that an $n$-element array $A$ is $k$-sorted if and only if $A[i] \leq A[i + k]$ for all $i = 1, 2, \ldots, n-k$. [4 marks]

(d) Give an algorithm that $k$-sorts an $n$-element array in $O(n \lg(n/k))$ time. You do not have to prove this running time (though should write a few sentences justifying it). Hint: The fact mentioned in (c) will be useful in designing the Algorithm. Another good hint is to think about partitioning the array into $k$ different subarrays based on their “mod $k$-index”. [4 marks]

(e) Show that when $k$ is a constant, it takes $\Omega(n \lg n)$ time to $k$-sort an $n$-element array. [4 marks] Hint: It will be helpful to use the fact that an $n$-element array which is already $k$-sorted, can be sorted in $O(n \lg k)$ time. The material we cover in class about lower bounds for comparison-based sorting will also be helpful.