**COMP 3711 – Design and Analysis of Algorithms**
**2021 Fall Semester – Written Assignment # 3**
**Distributed: October 7, 2021**
**Due: October 18, 2021, 11:59 PM**

Your solution should contain
(i) your name, (ii) your student ID #, and (iii) your email address
at the top of its first page.

Some Notes:

- Please write clearly and briefly. Your solutions should follow the guidelines given at
  *https://canvas.ust.hk/courses/38226/pages/assignment-submission-guidelines*

  In particular, your solutions should be written or printed on *clean* white paper with no watermarks, i.e., student society paper is not allowed.

- Please also follow the guidelines on doing your own work and avoiding plagiarism as described on the class home page.
  ***You must acknowledge individuals who assisted you, or sources where you found solutions.*** Failure to do so will be considered plagiarism.

- The term *Documented Pseudocode* in Problems 1 and 4 means that your pseudocode must contain documentation, i.e., comments, inside the pseudocode, briefly explaining what each part does.

- Many questions ask you to explain things, e.g., what an algorithm is doing, why it is correct, etc. To receive full points, the explanation must also be *understandable* as well as correct.

- Please make a *copy* of your assignment before submitting it. If we can't find your submission, we will ask you to resubmit the copy.

- Submit a SOFTCOPY of your assignment to Canvas by the deadline. The softcopy should be one PDF file (no word or jpegs permitted, nor multiple files).

  If your submission is a scan of a handwritten solution, make sure that it is of high enough resolution to be easily read. At least 300dpi and possibly denser.

**Problem 1:** **$k$-swapped arrays** [30 pts]

An $n$ element array $A[1 \ldots n]$ is **$k$-swapped** if $k$ is the *minimum* integer in $[0, n]$ satisfying the following:
for all $i, j \in [1, n]$ satisfying $i > j$ and $A[i] < A[j]$, we have $i - j \leq k$.

The problem is to sort a given $k$-swapped array in $O(n \log k)$ time.
Time in this problem refers to the number of comparisons performed by the algorithm.

In what follows $A$ is a $k$-swapped array of size $n$, where $k$ is provided as an extra parameter to the algorithm, and $k$ can be used in the code.

(a) (i) Write documented pseudocode for an $O(n \log k)$ time procedure to sort $k$-swapped array $A[1 \ldots n]$.
(ii) Below the pseudocode, provide a description in words (as opposed to code) of what your algorithm is doing.

(b) Prove why your algorithm is correct.
This should be a clear FORMAL mathematical style proof.

(c) Prove that your algorithm runs in $O(n \log k)$ time.

(d) In class we proved that any comparison-based algorithm requires $\Omega(n \log n)$ time. Why does this not contradict having an $O(n \log k)$ solution for this problem (when $k$ is much smaller than $n$).

(e) Prove that any algorithm for solving this problem requires $\Omega(n \log k)$ comparisons (so your solution is the best possible).

Rules, recommendations and hints

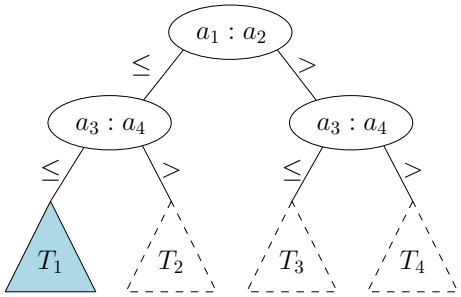
- In part (a), your algorithm may use any procedures or algorithms that we learned in class as a black-box subroutine (without having to provide code for that subroutine).
- In parts (b)-(d) you may use theorems, lemmas and facts that we learned in class or tutorial. If you use such theorems, lemmas and/or facts you must (i) explicitly state the text of the theorem/lemma/fact that you are using and (ii) identify exactly where in the class notes you are copying them from. Everything else must be proven from scratch.
- Hint. The solution to part (e) is in one of the tutorials (in a slightly different form).
- Proofs of correctness that are not formal and clear will have points deducted. There are multiple algorithms for this problem, some of which are quite intuitive. A main goal of this problem is to show that you can prove the correctness of an intuitive idea.
- Your proof of correctness may assume that all elements in the array are distinct.

**Problem 2 Decision Trees** [15 pts]

Recall that a comparison-based sorting algorithm can be represented in the (binary) decision tree model.

Following the worked example of Tutorial SS13, the figure below shows part of the decision tree for mergesort operating on a list of 4 numbers, $a_1$, $a_2$, $a_3$, $a_4$. Please expand subtree $T_1$, i.e., draw and label all of the edges, internal (comparison) nodes and leaves in subtree $T_1$.

Edges in the tree must be labelled with $\leq$ or $>$ and leaves must show the final sorted order.




**Problem 3 Radix sort** [16 pts]

You are given a set of 10 decimal integers in the range of 1 to 65535:

$$A = [29681, 53846, 43521, 39427, 32433, 35700, 30764, 16892, 52608, 19583].$$

(a) Please conduct Radix sort on A using Base 10. Illustrate your result after each step following the worked example on Page 36 in the 08_Linearsort lecture slides.

(b) Now convert these decimal integers to hexadecimal and conduct Radix sort again, this time using Base 16 (hexadecimal). Illustrate your result after each step following the worked example on Page 36 in the 08_Linearsort lecture slides.

*Note: a digit in hexadecimal is in the range of* $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F]$.
*To start you off, note that* 29681 *is* $73F1$ *in hexadecimal.*
*There are many decimal to hexadecimal converters available on the intern. We recommend you use one of them to covert the data rather than doing it by hand.*

**Problem 4 Greedy Covering** [29 pts]

Consider the following (unrealistic) scenario. You are running a political campaign and want to collect petition signatures in the park. Your research team tells you that everyone who goes to the park visits for one of $n$ known time intervals during the day (if they come for a particular interval, they stay for the entire time interval; note that different time intervals might overlap).

If you go to the park at any time during one of the intervals you will get the signatures of everyone there for that interval. You want to figure out a minimum sized set of times that you have to go to the park to get everyone's signature.

In computer science such a situation is called a *covering problem* and is modelled formally as described below.

- A real *interval* is $I = [s, f]$ where $s \leq f$
- A real *point* $x$ *covers* interval $I = [s, f]$ if $x \in I$, i.e., $s \leq x \leq f$.
- Let $\mathcal{I} = \{I_1, \ldots, I_n\}$ be a set of $n$ intervals and $A = \{a_1, a_2, \ldots a_k\}$ a set of points.

  Then "$A$ covers $\mathcal{I}$" if every interval in $\mathcal{I}$ is covered by at least one point in $A$. More formally, if, for every $I_j \in \mathcal{I}$ there exists $a_i \in A$ such that $a_i \in I_j$.
- $|A|$, the size of $A$, is just the number of points in $A$.

  $A$ is a *minimal cover* of $\mathcal{I}$ if $A$ is a smallest cover of $\mathcal{I}$.

  That is, for every cover $A'$ of $\mathcal{I}$, $|A| \leq |A'|$.

The input to this problem is the $2n$ numbers $s_1, f_1, s_2, f_2, \ldots, s_n, f_n$ with $s_j \leq f_j$, that define $\mathcal{I}$; $I_j = [s_j, f_j]$.

The problem is to construct a minimal cover for $\mathcal{I}$ in $O(n \log n)$ worst-case time. Your output should be a set $A$ which is a minimal cover.

When solving this problem you may use any algorithm we taught in class (but not the tutorial) as a black box as long as you quote which algorithm you are using. You can assume that any properties of that algorithm taught in class are correct as long as you quote what the properties are. Any other subroutines and their properties must be derived from scratch.

(a) Prove that there exists a minimal cover $A$ such that every point in $A$ is one of the $f_j$. That is, $A \subseteq \{f_1, \ldots, f_n\}$.

(b) Give documented pseudocode for your algorithm.

In addition, below your algorithm's pseudocode, explain in words/symbols what your algorithm is doing

(c) Prove correctness of your algorithm. Your proof must be mathematically formal and understandable.

*Note: Break your proof up into clear logical pieces and skip space between the pieces. Explicitly state what each part is assuming and proving.*

*For examples of such proofs please see the lecture notes and tutorials.*

*As seen in class, greedy algorithms tend to be simple. Their proofs of correctness are more complicated.*

(d) Explain why your algorithm runs in $O(n \log n)$ worst case time.

**P5:** [10 pts] **Huffman Coding**

The table below lists 10 letters ($a$ to $j$) and their frequencies in a document. Apply Huffman coding (following the algorithm on Page 11 of the 10_Huffman.pptx lecture notes) to construct an optimal codebook. Your solution should contain three parts:

(a) A full Huffman tree.

(b) The final codebook that contains the binary codewords representing $a$ to $j$ (sorted in the alphabetical order on $a$ to $j$).

(c) The codebook from part (b) but now sorted by the lengths of the codewords (in increasing order).

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $a_i$ | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ | $j$ |
| $f(a_i)$ | 36 | 21 | 58 | 16 | 17 | 29 | 45 | 27 | 26 | 28 |

**Rules, Recommendations and Hints:**

- In part (a) you should explicitly label each edge in the tree with a '0' or '1'. You should also label each leaf with its corresponding character $a_i$ and $f(a_i)$ value.
- Part (b) should be a table with 2 columns. The first column should be the letters $a$ to $j$. The second column should be the codeword associated with each character.
- In part (c) you should break ties using the alphabetical order of the characters. For example, if $a$, $d$ and $e$ all have codewords of length 5, then write the codeword for $a$ on top of the codeword for $d$ on top of the codeword for $e$.