COMP 3711 – Design and Analysis of Algorithms
2021 Fall Semester – Written Assignment # 2
Distributed: Sept 23, 2021
Due: Oct. 4, 2021, 11:59
Corrected Sept 27, 2021

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* 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.

- Sept, 27, 2021. Typo in Problem 2 (3) definition on p 5 corrected.
  $\{(x,y) : 1 \le x \le n,\ 1 \le y \le n\}$ was fixed to be $\{(x,y) : 1 \le x \le m,\ 1 \le y \le m\}$.

Problem 1: [30 pts]

Let A be an array of n elements. A *heavy element* of A is any element that appears more than 3/20 times, i.e., is more than 15% of the items. For example, if $n = 14$ then a heavy element is one that appears at least $3 = \lceil \frac{3}{20} \cdot 14 \rceil$ times.

As examples, consider the arrays A, B and C below:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$A[i]$	2	6	3	5	6	3	8	8	2	7	4	9	9	4

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$B[i]$	3	3	3	5	6	3	8	3	6	7	4	9	9	4

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$C[i]$	3	3	3	6	6	3	8	3	6	6	6	9	9	9

A contains no heavy items. B contains the unique heavy item 3. C contains the three heavy items 3, 6 and 9.

Design an $O(n \log n)$ time **divide-and-conquer algorithm** for finding all the heavy items (if any) in an array.

Your solution should be split into the following three parts. Write the answer to each part separately.

(a) (i) Write documented pseudocode for a procedure $Heavy(i, j)$ that returns the *set* of heavy items in subarray $A[i \dots j]$.

(ii) Below the pseudocode, provide a description in words (as opposed to code) of what your algorithm is doing.

(b) Explain (prove) why your algorithm is correct.

(c) Let $T(n)$ be the worst case number of total operations of all types performed by your algorithm. Derive a recurrence relation for $T(n)$ (explain how you derived the recurrence relation).

Show that $T(n) = O(n \log n)$.

Note: If the recurrence relation is in the form

$$\forall n > 1, \quad T(n) \le T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + c_1 n \quad \textit{and} \quad T(1) = c_2 \qquad (1)$$

for some $c_1, c_2 \ge 0$ you may immediately conclude that $T(n) = O(n \log n)$ without providing any further explanation other than saying you are calling the Master Theorem (and explaining which case of the master theorem it is and why).

Otherwise, you must prove from first principles that whatever recurrence relation you derived implies $T(n) = O(n \log n)$ for all values of n.

See below for requirements, hints and general comments.

Requirements, Hints and Comments:

- *Hint: The solution to Tutorial problem DC12 will help you get started.*
- *Hint: Let $S = Heavy(i, j)$. Prove that subarray $A[i \dots j]$ cannot contain more than 6 heavy items. You will need this fact in your analysis*
- **Sorting is not permitted.**
  The only comparisons allowed **to items in the array** are equalities. Inequalities are not permitted. More specifically:
  - You MAY write "If $A[i] = A[j]$" or "If $A[i] = x$".
  - You MAY NOT write "If $A[i] < A[j]$" or "If $A[i] < x$".
    Note that this immediately implies that you can NOT sort the array (since sorting requires inequality comparisons).
  - In particular you MAY NOT solve the problem by sorting the array and then counting all of the items with the same value.
- Part (a) requires two different types of explanations as to what your algorithm is doing. The first is the documentation IN the pseudocode. The second is the explanation AFTER the pseudo-code. BOTH must be provided.
- Part (b) is a proof of correctness. Write it as you would a mathematical proof. Explicitly state any facts upon which the proof depends. Incomplete proofs will have points deducted. See the Practice Homework on the Tutorial page for pointers as to how this should be structured.
- Answer Parts (a) and (b) separately. Do not worry if your explanation in Part (a) and your proof of correctness in Part (b) overlap somewhat.
- We are specifically asking for a divide-and-conquer algorithm here, e.g., a generalization of DC12. We are aware that faster non divide-and-conquer algorithms do exist and you might be able to find one by Googling. Such algorithms will not be accepted as answers. The purpose of this exercise is to use the divide-and-conquer approach to solve the problem.
- The call $Heavy(1, n)$ should run in $O(n \log n)$ time and return a *set*. You can do this by writing $Return(S)$, where S is a set.
- *Set Notation:* Your pseudocode will need to use a *set data structure* with associated *set operations*. You should use standard mathematical notation for writing set operations (do not make up your own set notation or use a notation from a programming language you know).

  See the next page for more information on how to write set operations in pseudocode along with associated running times.

<u>*Set notation in pseudocode:*</u> In what follows, S, S_1, S_2 are all sets.

Recall that sets do not contain repeated items.
So, if $S_1 = \{a, b\}$, $S_2 = \{b, c\}$ and $S = S_1 \cup S_2$, then $S = \{a, b, c\}$.

$|S|$ denotes the size of S, e.g, $|S_1| = |S_2| = 2$, $|S_1 \cup S_2| = 3$, $|S_1 \cap S_2| = 1$.
If $S = \emptyset$ (the empty set), then $|S| = 0$.

"$\cup$" denotes the union of sets. '$\cap$" denotes the intersection of sets.

$S_1 \setminus S_2$ (equivalently, $S_1 - S_2$) is the set S_1 with all items in $S_1 \cap S_2$ removed.

- Create sets using the notation "`Create Set` S".
  This creates empty set S in $O(1)$ time.
- Evaluating $|S|$ requires $O(1)$ time.
- Setting $S_1 = S_2$ will require $O(|S_2| + 1)$ time.
- You can add element x to set S by writing $S = S \cup \{x\}$.
- Evaluating $S = S_1 \cup S_2$, $S = S_1 \cap S_2$ and $S = S_1 \setminus S_2$ all require $O((|S_1| + 1)(|S_2| + 1))$ time.
- Evaluating the statement "$x \in S$" requires $O(|S| + 1)$ time and returns TRUE if x is in S and FALSE otherwise.
- You may write pseudocode in the form

  ```
  $\forall x \in S$ do
      $Work(x)$
  ```

  where $Work(x)$ is some code dependent upon x.

  If $S = \{x_1 \ldots, x_k\}$, the time required to run this code will be $O(1)$ plus the total of the work required to run all of $Work(x_1)$, $Work(x_2), \ldots, Work(x_k)$.

Example 1: The following code finds the unique items in array A:

```
Create Set $S$
For $i = 1$ to $n$ do
    $S = S \cup \{A[i]\}$
```

It uses $O(n \cdot m)$ time, where m is the number of unique items in $A[1 \ldots n]$.

Example 2: The following code constructs $S_1 \cap S_2$:

```
Create Set $S$
$\forall x \in S_1$ do
    If $x \in S_2$ then
        $S = S \cup \{x\}$
```

It runs in $O((|S_1| + 1)(|S_2| + 1))$ time.

Problem 2 [28 pts] **Indicator Random Variables**

$A[1 \dots n]$ is an array. In addition to A you are given 2 other structures that are built from A: a binary tree T and a $m \times m$ 2D matrix M. In each of these structure we also define $N(v)$, the set of neighbors of item v. Each of these structures and their neighbor definitions are defined below. See the diagram on the next page for examples.

(1) Array A: This is just the standard array.
Neighbors are the items to the left and right of i.

$$N(i) = \begin{cases} \{i-1, i+1\} & \text{if } 1 < i < n \\ \{2\} & \text{if } i = 1 \\ \{n-1\} & \text{if } i = n \end{cases}$$

(2) Binary Tree T :
In this case we assume that $n = 2^k - 1$ for some integer $k \geq 0$.
The correspondence between A and T is given below (and is also in the Heapsort slides page 9 onwards. Please see those for more explanation).
Each node in the tree corresponds to some element of array A. Node i will have value $T[i] = A[i]$.

The root of the tree is node 1, and has no parent.
The parent of node $i \neq 1$ is node $\lfloor i/2 \rfloor$. The left and right children of node i are nodes $2i$ and $2i+1$, if they exist

$$N_T(i) = \begin{cases} \{2, 3\} & \text{if } v = 1 \\ \{2i, 2i+1, \lfloor i/2 \rfloor\} & \text{if } 1 < i < (n+1)/2 \\ \{\lfloor i/2 \rfloor\} & \text{if } (n+1)/2 \leq i \leq n \end{cases}$$

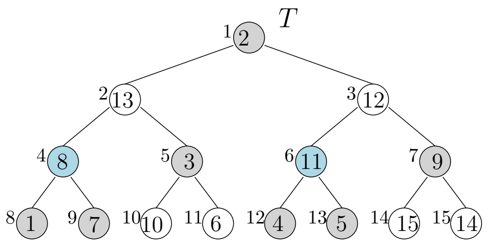
(3) $m \times m$ 2D matrix M.
In this case we assume that $n = m^2$.
$M(i, j) = A[(i-1)m + j]$. An element's neighbors are its (at most 4) vertical and horizontal neighbors. That is,

$$N_M((i,j)) = \{(i+1, j),\ (i-1, j),\ (i, j+1),\ (i, j-1)\} \bigcap \{(x, y)\ :\ 1 \leq x \leq m,\ 1 \leq y \leq m\}.$$

Now solve the six subproblems below. See the end of this problem for an explanation of what "Derive the expected number" means and how your solutions should be structured.

Local minima are shaded gray; saddle points, light blue.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$A[i]$	2	13	12	8	3	11	9	1	7	10	6	4	5	15	14



A has 5 local minima and 6 saddle points;
T has 7 local minima and 2 saddle points.

---

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$A[i]$	16	6	14	7	11	10	13	4	1	8	15	9	3	2	12	5

M

$i \backslash j$	1	2	3	4
1	16	6	14	7
2	11	10	13	4
3	1	8	15	9
4	3	2	12	5

A has 6 local minima and 4 saddle points; M has 5 local minima and 3 saddle points.

For all of the six problems assume that A is a uniform random permutation of $< 1, 2, \ldots, n >$. This means that every one of the $n!$ possible permutations is equally likely to occur.

The corresponding T and M are built from A as previously described.

The solutions to all six of the problems must use the indicator random variable technique to derive the result.

(a) **Counting local minima.**

(i) $A[i]$ is a *local minimum* in A if it is smaller than all of its neighbors, i.e, $A[i] < \min_{v \in N(i)} A[v]$.
Derive the expected number of local minima in A.
Hint: Let X_i be the indicator random variable for the event that $A[i]$ is a local minimum. What is $E(X_i)$ (this might depend upon i)? How does knowing the $E(X_i)$ let you answer the problem?

(ii) $T[i]$ is a *local minimum* in T if it is smaller than all of its neighbors, i.e, $T[i] < \min_{v \in N_T(i)} T[v]$.
Derive the expected number of local minima in T.

(iii) $M(i, j)$ is a *local minimum* in M if it is smaller than all of its neighbors, i.e, $M(i, j) < \min_{(x,y) \in N_M(i,j)} M(x, y)$.
Derive the expected number of local minima in M.

(b) **Counting Saddle points.**

(i) $A[i]$ is a *saddle point* in A if $i \neq 1$, $i \neq n$, and $A[i]$ is larger than one of its neighbors and smaller than the other one of its neighbors.
Derive the expected number of saddle points in A.

(ii) $T[i]$ is a *saddle point* in T if $1 < i < (n+1)/2$ and $T[i]$ is smaller than its parent but larger than both of its children.
Derive the expected number of saddle points in T.

(iii) $M(i,j)$ is a *saddle point* in M if $M(i,j)$ is smaller than all of its neighbors in the same row and larger than all of its neighbors in the same column.
Derive the expected number of saddle points in M.

Structure of the Solution: The solution to each of the six subproblems must be written separately, with space between the solution to each subproblem

For each subproblem, you should derive the expected number of local minima or saddle points. as follows, using a three part solution:

(A) First clearly define the indicator random variable(s) that you will be using to solve this problem.

(B) Next, derive the expectation of each such indicator random variable.

(C) Finally, solve the full subproblem, by deriving a closed formula in n (no summations "$\sum$" allowed) for the requested value.

Problem 3 [24 pts] (Using Selection)

Note: Recall the Selection problem of calculating Select(A, p, r, i) that we learned in class. We actually learned a randomized *linear, i.e., $O(r - p + 1)$, time algorithm for solving Selection. We also stated that a* deterministic *linear time algorithm for solving this problem exists. For the sake of this problem, you may assume that you have been given this deterministic linear time algorithm and can use it as a subroutine. (calling it exactly as Select(A, p, r, i)). You may not use any other algorithm as a subroutine without explicitly providing the code and proving correctness from scratch.*

The *Manhattan Distance* between two 2-dimensional points $p = (x, y)$ and $p' = (x', y')$ is

$$d_1(p, p') = |x - x'| + |y - y'|.$$

In what follows you are given real numbers $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$. They are NOT necessarily sorted.

Define the n two-dimensional points $p_i = (x_i, y_i)$. The *Manhattan Warehouse problem* is to find a point $p = (x, y)$ that minimizes the average distance to all the p_i. That is, to find p such that

$$\forall p' = (x', y') \in \Re^2, \quad \sum_{i=1}^{n} d_1(p, p_i) \leq \sum_{i=1}^{n} d_1(p', p_i).$$

(Dividing both sides of the equation by $\frac{1}{n}$ gives the average.)

The goal of this problem is to design a $O(n)$ worst case time algorithm for solving the Manhattan Warehouse problem. We split this into two parts.

(A) Solve the one-dimensional problem.

Given (unsorted) $x_1, \ldots x_n$, define the function

$$f(x) = \sum_{i=1}^{n} |x - x_i|.$$

Call $\bar{x}$ a *center* if it minimizes $f(x)$, i.e.,

$$\forall x \in \Re, \quad f(\bar{x}) \leq f(x).$$

(a) Prove that there exist z_1, z_2 such that $z_1 \leq z_2$ and

(i) $f(x)$ is monotonically decreasing for $x \in [-\infty, z_1]$.

(ii) $f(z_1) = f(x) = f(z_2)$ for all $x \in [z_1, z_2]$.

(iii) $f(x)$ is monotonically increasing for $x \in [z_2, \infty]$.

Note that it is possible that $z_1 = z_2$.

(b) Give an $O(n)$ time algorithm for finding a center of n one-dimensional points $x_1, \ldots, x_n$.
First give documented pseudocode for your algorithm.
In addition, below your algorithm, explain in words/symbols what your algorithm is doing

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

(d) Explain why your algorithm runs in $O(n)$ time.

(B) Solve the Manhattan Warehouse Problem.

(a) Give an $O(n)$ time algorithm for solving the Manhattan Warehouse Problem given n points $p_1, p_2, \ldots, p_n$ where $p_i = (x_i, y_i)$.
First give documented pseudocode for your algorithm.
In addition, below your algorithm, explain in words/symbols what your algorithm is doing

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

(c) Explain why your algorithm runs in $O(n)$ time.

Notes:

- *For simplicity, you may assume that none of the x_i or y_i repeat, i.e., there is no pair i, j such that $x_i = x_j$ or $y_i = y_j$.*
- *Part A(a) is only meant as a hint to get you started. Note that if you know z_1, z_2, then (you must prove this) any $x \in [z_1, z_2]$ will be a solution to the one-dimensional center problem.*

  A more detailed understanding of the behavior of $f(x)$, that lets you find z_1, z_2, therefore permits solving the remainder of A. You may, if you like, fully analyze the behavior of $f(x)$ in A(a), write this as a mathematical lemma and then quote that lemma in A(c).
- *We strongly recommend that before trying to solve part A you choose 5 or 6 points x_i and then graph the resultant function $f(x)$. Seeing the diagram should help you solve the problem.*
- *The algorithm for part (B) can use the result of part (A) as a subroutine.*
- *The main work in solving this problem is in understanding how to solve it using selection. Once you understand that, the actual pseudocode might be quite short.*

P4: [18 pts] **Recurrence Relations**

Give asymptotic upper bounds for $T(n)$ satisfying the following recurrences. Make your bounds as tight as possible. For example, if $T(n) = \Theta(n^2)$ then $T(n) = O(n^2)$ is a tight upper bound but $T(n) = O(n^2 \log n)$ is not. Your upper bound should be written in the form $T(n) = O(n^\alpha (\log n)^\beta)$ where α, β are appropriate constants.

A correct answer will gain full credits. It is not necessary to show your work BUT, if your answer was wrong, showing your work steps may gain you partial credits.

If showing your work, you may quote theorems shown in class.
For (a), (b) and (c) you may assume that n is a power of 2;
for (d) you may assume that n is a power of 4;
for (e) that n is a power of 7; for (f) that n is a power of 3

(a) $T(1) = 1; T(n) = 4T(n/2) + n^2\sqrt{n}$ for $n > 1$.

(b) $T(1) = 1; T(n) = 9T(n/2) + n^3 \log_2 n$ for $n > 1$.

(c) $T(1) = 1; T(n) = 4T(n/2) + \sum_{i=1}^{n} i$ for $n > 1$.

(d) $T(1) = 1; T(n) = 3T(n/4) + 1$ for $n > 1$.

(e) $T(1) = 1; T(n) = 2T(n/7) + 2^{\log_7 n}$ for $n > 1$.

(f) $T(1) = 1; T(n) = 9T(n/3) + \log_3 (n!)$ for $n > 1$.