COMP 3711 – Design and Analysis of Algorithms
2021 Spring Semester – Written Assignment # 1
Distributed: Feb 9, 2021
Due: Feb 21, 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/36180/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 questions (2) and (3) means that you must include clear comments *inside* your pseudocode.

- 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 CASS 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: [25 pts] In class, we "solved" the recurrence

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

by simplifying and assuming that n was a power of two. This permitted replacing Equation (1) with

$$T(n) \le 2T(n/2) + n \quad \text{and} \quad T(1) = 1.$$

The solution to this homework problem will show, at least for two recurrences, why such simplifications are "legal".

(a) Let $c > 0$ be some constant integer. Let $T(n)$ be a function satisfying

$$\forall n > 2, \quad T(n) \le T\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + c \quad \text{and} \quad T(1) = T(2) = 1. \tag{2}$$

(i) Prove using the expansion method that $T(3^k) = O(k)$.

(ii) Let $S(n)$ be some nondecreasing function of n.
You are told that $S(n)$ also satisfies $S(3^k) = O(k)$.
Prove that $S(n) = O(\log_3 n)$.

(iii) For all $n \ge 1$, set $R(n) = \max_{1 \le i \le n} T(i)$. Prove that

$$\forall n > 1, \quad R(n) \le R\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + c.$$

(iv) Using (i), (ii) and (iii), prove that if $T(n)$ satisfies Equation (2) then $T(n) = O(\log n)$.

(b) We learned in class that if $T(n)$ satisfies Equation (1) then

$$T(2^k) = O(k2^k). \tag{3}$$

(v) Let $S(n)$ be some nondecreasing function of n.
You are told that $S(n)$ also satisfies $S(2^k) = O(k2^k)$.
Prove that $S(n) = O(n \log_2 n)$.

(vi) For all $n \ge 1$, set $R(n) = \max_{1 \le i \le n} T(i)$.
Prove that if $T(n)$ satisfies Equation (1) then

$$\forall n > 1, \quad R(n) \le R\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + R\left(\left\lceil \frac{n}{2} \right\rceil\right) + n.$$

(vii) Using Equation (3), (v) and (vi), prove that $T(n)$ defined by Equation (1) satisfies $T(n) = O(n \log_2 n)$.

Requirements, Hints and Comments:

- Prove each of parts (i) - (vii) separately in order.
  Make sure that you clearly label each part.

  Leave a few empty lines between each part.
  Solutions not following this format will have points deducted.

- Read the solution to the Practice Homework on the Tutorial web page before attempting this problem. That will provide you with techniques and proof structures that will be applicable here.

- You should assume the correctness of earlier parts when solving later ones. For example, when solving part (iv), you should assume the correctness of parts (i), (ii) and (iii).

- k and n will always denote non-negative integers.

- $F(n)$ being a *non-decreasing function* means $\forall n$, $F(n) \le F(n+1)$.

- $f(n) = O(g(n))$ means that

  there exists $c > 0$ and n_0 such that $\forall n > n_0$, $f(n) \le cg(n)$.

  Your proofs in parts (i), (ii) (iv), (v) and (vii) should explicitly identify the values of c and n_0 used.

- $T(3^k) = O(k)$ and $T(2^k) = O(k2^k)$ mean, respectively, that

  there exists $c > 0$ and k_0 such that $\forall k > k_0$, $T(3^k) \le ck$.

  there exists $c' > 0$ and k_0' such that $\forall k > k_0'$, $T(2^k) \le c'k2^k$.

- In part (ii) you only know that $S(n)$ is a non-decreasing function satisfying $S(3^k) = O(k)$. Your proof may not use any other assumptions about $S(n)$.

  Hint: Use the fact that for $x \ge 3$, $\log_3 3x = 1 + \log_3 x \le 2\log_3 x$.

- Similarly, in part (v), you may only use the facts that $S(n)$ is a non-decreasing function satisfying $S(2^k) = O(k2^k)$.

Problem 2: [22 pts]

Let $A[1 \ldots n]$ be an array with n items. A' is *array A right-flipped at k* with $1 \le k \le n$ if

$$A'[i] = \begin{cases} A[i] & \text{if } i < k \\ A[n+k-i] & \text{if } i \ge k \end{cases}$$

Suppose that A originally contains n distinct numbers sorted in increasing order, i.e., $A[1] < A[2] \cdots < A[n-1] < A[n]$. You are told that A has been right-flipped for some value of k. Design an $O(\log n)$ time algorithm that returns the value of k.

As an example, the algorithm would return $k = 5$ for the array below since A was a sorted array that was right-flipped at $k = 5$.

i	1	2	3	4	5	6	7	8
$A[i]$	1	3	5	7	20	18	15	10

(a) (i) Write your algorithm down in clear documented pseudocode.

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

(b) Prove (explain) why your algorithm is correct.

(c) Let $T(n)$ be the worst case number of comparisons performed by your algorithm on an array of size n.
Derive a recurrence relation for $T(n)$
(explain how you derived the recurrence relation).
Explain why this recurrence relation implies $T(n) = O(\log n)$.

Note: For (c) it suffices to show that

$$T(n) \le T\left(\frac{n}{2}\right) + c \qquad (4)$$

for some value $c > 0$. You can then quote the "fact" from class that Equation (4) implies $T(n) = O(\log n)$. If you can not show that $T(n)$ satisfies Equation (4), then you must prove from first principles that whatever recurrence relation you derived implies $T(n) = O(\log n)$ for all values of n.

See the next page for requirements, hints and general comments:

Requirements, Hints and Comments:

- Consider using a variant of binary search to solve this problem. Tutorial problem SS1 describes another variant of binary search.
- The Practice Homework on the Tutorial page describes how to prove the correctness of binary search. You can use that as a pattern upon which to base your solution to Part (b).
- The term *Documented Pseudocode* in Part (a) means that you must include clear comments *inside* your pseudocode.
- 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.
- 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. That is quite common.
- Your algorithm should assume that $n \geq 2$. It does not have to deal with the special case of $n = 1$ (which is always technically right-flipped by $k = 1$).

  You may assume that the value of n is known by the algorithm (so you don't need to use a function such as *size(A)* in your code).
- Recall that $1 \leq k \leq n$. We suggest that you carefully check that your algorithm correctly handles the boundary cases $k = 1$ and $k = n$. Your intuitively logical solution might not work in one or both of those boundary cases.
- The expression
$$T(n) \leq T\left(\frac{n}{2}\right) + c$$
  is technically undefined when n is not even. It is used in Equation (4) as shorthand to denote any of
$$\begin{aligned} T(n) &\leq T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + c \\ T(n) &\leq T\left(\left\lceil \frac{n}{2} \right\rceil\right) + c, \\ T(n) &\leq T\left(\max\left(\left\lceil \frac{n}{2} \right\rceil, \left\lfloor \frac{n}{2} \right\rfloor\right)\right) + c. \end{aligned}$$
  All of those recurrences imply $T(n) = O(\log n)$.

Problem 3: [25 pts]

Let A be an array of n elements. A *heavy element* of A is any element that appears more than $2n/5$ times, i.e., is more than 40% of the items. For example, if $n = 11$ then a heavy element must appear at least 5 times.

Note that an array might contain no heavy items, one heavy item or two heavy items. It can not contain more.

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

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

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

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

A contains no heavy items. B contains the unique heavy item 3. C contains the two heavy items 3 and 6.

Design an $O(n \log n)$ time **divide-and-conquer algorithm** for finding the heavy items 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

$$\forall n > 1, \quad T(n) \le T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + c_1 n \quad \textit{and} \quad T(1) = c_2 \qquad (5)$$

for some $c_1, c_2 \ge 0$ you may immediately conclude that $T(n) = O(n \log n)$ without providing any further explanation. This is is essentially what you proved in Problem (1) part (b).

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 the next page 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 the items in set S must appear at least $\frac{2}{5}(j - i + 1)$ times in $A[i \ldots j]$ and that this implies that a (sub)array can not contain more than two heavy items.*
- The only comparisons allowed are equalities. Inequalities are not permitted. More specifically:
  - You MAY test an equality, e.g., by writing "If $A[i] = A[j]$" or "If $A[i] = x$".
  - You MAY NOT test an inequality, e.g., by writing "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 inequalities).
  - In particular you MAY NOT solve the problem by sorting the array and then counting all of the items with the same value.
- The call $Heavy(1, n)$ should run in $O(n \log n)$ time and return the set of heavy items for the entire array.
- Similar to Problem 2, 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.
- Again similar to Problem 2, 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.
- Set Notation: Your pseudocode may use standard set terminology. This will simplify the exposition
  - Your procedure should return a set. You can write this as $S = Heavy(i, j)$
  - Recall that $|S|$ denotes the size, i.e., number of items in a set. E.g, if $S = \{a, b\}$ then $|S| = 2$. If $S = \emptyset$ (the empty set), then $|S| = 0$.
  - "$\cup$" denotes the union of 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 $S = \{a, b, c\}$.
  - Set operations are not constant time. The time used by operation $S = S_1 \cup S_2$ will be $O((|S_1| + 1)(|S_2| + 1))$.
    When counting the number of operation used by the algorithm, you need to make this running time explicit.

  - You may write pseudocode code in the form

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

    where $Work(x)$ is some code dependent upon x.
    If $S = \{x_!, \ldots, x_k\}$, the time required to run this code will be the total of the work required to run all of $Work(x_1)$, $Work(x_2), \ldots, Work(x_k)$.

- Part (b) should be 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.

Problem 4 [14 pts]

For each pair of expressions (A, B) below, indicate whether A is O, Ω, or Θ of B. Note that zero, one, or more of these relations may hold for a given pair. List the most appropriate relation.

It often happens that some students will get the directions wrong, so please write out the relation in full, i.e., you should write exactly one of the terms or write that *none of the relations is satisfied.* More explicitly, if any of the relationships are satisfied you should write the appropriate

$$A = O(B), \quad A = \Omega(B), \quad \text{or} \quad A = \Theta(B)$$

and not just $O(B)$, $\Omega(B)$ or $\Theta(B)$, omitting the A.

(a) $A = n^2 + n^2 \log n,\ B = 5n - 7n^3 + 2n^4$;

(b) $A = \log_{100}((n+100)!),\ B = \ln n^n$;

(c) $A = (\sqrt[3]{2})^{\frac{1}{\log_n 10}},\ B = 2^{\sqrt{3 \log_2 n}}$;

(d) $A = \sum_{i=1}^{n} k^3,\ B = 12 \times \binom{n}{4}$;

(e) $A = n^6 2^{n(\log n)^3},\ B = n^8 + 2021^{2020^{2019}}$;

(f) $A = \sum_{k=1}^{n} \frac{1}{k(k+1)},\ B = \ln \sum_{k=1}^{n} \frac{1}{k}$;

(g) $A = n(1 + (-1)^n),\ B = n(1 + (-1)^{n+1})$.

Write one solution per line.

Hints and Comments:

- If only one relationship holds, then the most appropriate relationship is just that relationship. If $A = \Theta(B)$ then you should write $A = \Theta(B)$ (even though both $A = O(B)$ and $A = \Omega(B)$ are both also valid). As examples,
  - If $A_1 = 10n$ and $B_1 = n^2$ then the answer should be "$A_1 = O(B_1)$".
  - If $A_2 = 10n^2$ and $B_2 = n^2 - 20n$ then the answer should be "$A_2 = \Theta(B_2)$".
- It is NOT necessary to provide justifications for your answer. But if you do not provide any justifications and you are wrong, we can not give you partial credit.

Problem 5 [14 pts] Give asymptotic upper bounds for $T(n)$ satisfying the following recurrences. Make your bounds as tight as possible.

Note that "tight upper bound" means the best possible Big-Oh answer. So, $T(n) = O(n^5)$ would not be considered a fully correct answer if $T(n) = O(n^4)$ as well.

You must prove your results from scratch using either the expansion or tree method shown in class. Show all of your work.

For (a) you may assume that n is a power of 4; for (b) you may assume that it is a power of 3.

(a) $T(1) = 1; \quad T(n) = 6T(n/4) + n^2$ for $n > 1$.

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