Data Structure and Algorithm, Spring 2021

Homework 1

Due: 13:00:00, Tuesday, April 13, 2021

TA E-mail: dsa_ta@csie.ntu.edu.tw

Rules and Instructions

- Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.
- In Homework 1, the problem set contains 6 problems and is divided into two parts, the non-programming part (Problems 1, 2, 3) and the programming part (Problems 4, 5, 6).
- For problems in the non-programming part, you should combine your solutions in ONE PDF file. Your file should generally be legible with a white/light background—using white/light texts on a dark/black background is prohibited. Your solution must be as simple as possible. At the TAs' discretion, solutions which are too complicated will be regarded as incorrect. Moreover, if you would like to use any theorem which is not mentioned in the classes, please include its proof in your solution.
- The PDF file for the non-programming part should be submitted to Gradescope as instructed, and you should use Gradescope to tag the pages that correspond to each subproblem to facilitate the TAs' grading. Failure to tagging the correct pages of the subproblem can cause losing part or all of the scores on the subproblem.
- For the programming part, you should have visited the *DSA Judge* (https://dsa-2021.csie.org) and familiarized yourself with how to submit your code via the judge system in Homework 0.1126.
- For problems in the programming part, you should write your code in C programming language, and then submit the code via the judge system. You can submit up to 10 times per day for each problem. The judge system will compile your code with

```
gcc main.c -static -O2 -std=c11
```

- Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from. For *each non-programming problem*, you have to specify the references (the Internet URL you consulted with or the people you

discussed with) on the first page of your solution for that problem; for *each programming problem*, you have to specify the references on the first lines (comments) of your code (`main.c`) for that problem.

- Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.
- The score of the part that is submitted after the deadline will get some penalties according to the following rule:

$$\text{LateScore} = \max\left(0, \frac{86400 \cdot 5 - \text{DelayTime (sec.)}}{86400 \cdot 5}\right) \times \text{OriginalScore}$$

- If you have questions about HW1, please go to the course forum and discuss (*strongly preferred*, which will provide everyone a better learning experience). If you really need an email answer, please follow the rules outlined below to get a fast response:
  - The subject should contain two tags, `"[HW1]"` and `"[Px]"`, specifying the problem where you have questions. For example, `"[HW1][P1] Are m, n in subproblem 4 positive?"`. Adding these tags allows the TAs to track the status of each email and to provide faster responses to you.
  - If you want to provide your code segments to us as part of your question, please upload it to Gist or similar platforms and provide the link. Screenshots or code segments directly included in the email is discouraged and may not be reviewed.

Non-Programming Part

Problem 1 - Complexity (80 + 20 pts)

Warning: Conciseness makes everyone's life easier. You should answer Problem 1 within 3 pages, or you will get some penalties. Also, make sure your answer is clear and easy to read for others.

Suppose that f and g are real-valued functions defined on $\mathbb{R}^+$, and $g(x)$ is strictly positive for all sufficiently large x (formally, there is some x_1 such that $g(x) > 0$ for all $x > x_1$). Recall the following asymptotic notations:

- $f(x) = O(g(x))$ if and only if there exist positive numbers c and x_0 such that
$$|f(x)| \leq cg(x) \text{ for all } x > x_0.$$
- $f(x) = \Omega(g(x))$ if and only if there exist positive numbers c and x_0 such that
$$|f(x)| \geq cg(x) \text{ for all } x > x_0.$$
- $f(x) = \Theta(g(x))$ if and only if $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$.

Note that the definitions here may be slightly different to other versions in different textbooks, but they may come in handy in some problems of this section. We ask you to use our definitions here to answer the subproblems below.

Now, imagine that you are on the magic train heading to the empire of God 127, the master of algorithm. To worship God 127, you should have some skills to evaluate the complexity of simple function calls.

Read the following pseudo code carefully. Write down the time complexity for each of the following function using the Θ notation along with an expression of n. Your answer should be as simple as possible. For instance, $\Theta(\frac{1}{n} + \lg(n^{543}) + 8763n^2)$ should be written as $\Theta(n^2)$. Your answer should also come with a *brief* explanation.

1. (5 pts)

```
FUNC-A(n)
1  sum = 0, i = 1
2  while sum < n
3        sum = sum + i
4        i = i + 1
```

2. (5 pts)

```
FUNC-B(n)
1  m = 2
2  while m < n
3        m = m * m
```

3. (10 pts)

```
FUNC-C(n)
1  if n > 87506055
2        FUNC-C(n − 1)
3        FUNC-C(n − 1)
4        FUNC-C(n − 1)
5        FUNC-C(n − 1)
6  elseif n > 0
7        FUNC-C(n − 1)
8        FUNC-C(n − 1)
9        FUNC-C(n − 1)
```

Finally, you arrives at the entrance of 127's empire. However, the soldiers won't let you pass the entrance if you are not familiar with asymptotic notations. For the next six subproblems, consider n to be a positive integer and $f(n), g(n), i(n), j(n)$ to be *positive*, *monotonically increasing* functions with *non-negative* domains.[1] Prove or disprove each of the statement. You should provide a formal proof if you believe the statement to be true, or a counterexample if you believe the statement to be false.

4. (10 pts) $f(n) + g(n) = \Theta\big(\max(f(n), g(n))\big)$.

5. (10 pts) If $f(n) = O\big(i(n)\big)$ and $g(n) = O\big(j(n)\big)$, then $f(n) \cdot g(n) = O\big(i(n) \cdot j(n)\big)$.

6. (10 pts) If $f(n) = O\big(g(n)\big)$, then $2^{f(n)} = O\big(2^{g(n)}\big)$.

Did you have fun warming up? Now the soldiers are giving you more challenging ones. Try your best! Some calculus tricks may help.

7. (10 pts) The harmonic series $\sum_{k=1}^{n} \frac{1}{k} = \Theta(\lg n)$.

8. (10 pts) $\lg(n!) = \Theta(n \lg n)$.

[1] A function f is monotonically increasing if for all n_1, n_2 such that $n_1 < n_2$, we have $f(n_1) \leq f(n_2)$.

Finally, you get to meet God 127 face to face. He decides to give you the final trial with a recursive function to test if you are able to pass this course. ;-)

9. (10 pts) Define

$$f(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2f(\lfloor \frac{n}{2} \rfloor) + n \lg n, & \text{otherwise} \end{cases}$$

   Please find the tightest bound of $f(n)$ with Θ notation. You should provide a proof to get credits. Answers without any proof will receive NO credits.

Congratulations on completing all the proof—*or not?* Those mathematically inclined can try two more subproblems below. Those subproblems are harder and do not carry many points, but tackling them surely gives you some self-satisfaction!

10. (Bonus 10 pts) Let $\{f_k(n)\}_{k=1}^{\infty}$ be a sequence of real-valued functions defined on $\mathbb{R}^+$ such that $f_k(n) = O(n^2)$ for $k \in \mathbb{N}$, then $\sum_{k=1}^{n} f_k(n) = O(n^3)$.

11. (Bonus 10 pts) The following pseudo code is the famous Euclidean algorithm for computing the greatest common divisor—hmm, where have you seen this task? ;-)

    EUCLIDEAN-GCD(m, n)

    1 **while** $m \neq 0$
    2 $\quad\quad (m, n) = (n \bmod m, m)$
    3 **return** n

    Please show that the worst time complexity for this algorithm is $O(\lg(m + n))$.

Problem 2 - Stack / Queue (60 pts)

First, please design an algorithm that reverses the content of a *source* queue using an additional *helper* queue that is initially empty. For example, consider a source queue originally contains [1,2,3], where 3 is at the front of the queue. After running the algorithm, the source queue should become [3,2,1], where 1 is at the front, with the helper queue empty. The algorithm can only use $O(1)$-extra space in addition to the two queues. You have to guarantee that the algorithm runs in $O(n^2)$-time where n is the number of elements in the queue. In addition to the usual *enqueue*, *dequeue*, and *front* of the queue, you may assume that the queues above also supports *size*, which returns the current number of elements in the queue.

1. (10 pts) Show your algorithm with any pseudo code.

2. (10 pts) Prove that your algorithm in the previous subproblem runs in $O(n^2)$-time.

Next, let's play with another data structure. Whereas a stack allows insertion and deletion of elements at only one end, and a queue allows insertion at one end and deletion at the other end, a *deque* (double-ended queue) allows insertion and deletion at both ends. Please use any pseudo code to implement a deque by using two stacks and at most $O(1)$-extra space, and answer the following five subproblems. A deque should support four operations, as listed below. You can assume that there is no invalid operation like popping from an empty deque.

3. (10 pts) Illustrate your design, in particular, how you achieve each of the four operations below (preferably with figures).

Then, briefly describe the time complexity of each operation under your design—the faster the better.

4. (5 pts) `void push_front(int x)`: Insert an element to the front end of the deque.

5. (5 pts) `void push_back(int x)`: Insert an element to the back end of the deque.

6. (5 pts) `int pop_front()`: Remove and return the element at the front end of the deque.

7. (5 pts) `int pop_back()`: Remove and return the element at the back end of the deque.

Finally, let's have more fun with the stack. Arguably the simplest way to implement a stack is to use a consecutive array and push to the "end" of the consecutive region. However, the array-based one is limited by the allocated capacity of the underlying array. One way to conquer the limitation is to dynamically enlarge the array when it is full, which is commonly called a dynamic-array implementation.

8. (10 pts) Consider the following dynamic-array implementation of a stack; please analyze the time needed for $\boldsymbol{n}$ consecutive push operations. You can assume that each `realloc` operation takes $O(m)$-time where m is the number of elements in the array.

```
struct Stack {
    int top;
    int capacity;
    int *arr;
}

struct Stack *createStack() {
    struct Stack *S = malloc(sizeof(struct Stack));
    S->capacity = 1;
    S->top = -1;
    S->arr = (int*)malloc(S->capacity * sizeof(int));
    return S;
}

int isFullStack(struct Stack *S) {
    return (S->top == S->capacity-1);
}

void enlarge(struct Stack *S) {
    int new_capacity = (S->capacity * 3);
    S->capacity = new_capacity;
    S->arr = (int*)realloc(S->arr, new_capacity * sizeof(int));
}

void push(struct Stack *S, int data) {
    S->arr[++S->top] = data;
    if (isFullStack(S))
        enlarge(S);
}
```

Problem 3 - Array / Linked Lists (60 pts)

For all subproblems below, your answers should include the following parts:

(a) Describe the workflow of your algorithm and briefly explain why it works.

(b) Analyze the time complexity and space complexity of your algorithm. If your algorithm is not efficient enough in time or in space, you may not get full points.

First, consider a read-only array A that contains n integers $\in \{0, 1, \ldots, n-1\}$, where "read-only" means that you cannot modify the contents of the array. A frog, starting from some legitimate initial position within A, jumps within $A[0], A[1], \ldots, A[n-1]$ by the following rule.

- If the frog is on the i^{th} position and $A[i] \neq i$, it will jump to the $(A[i])^{th}$ position.
- If the frog is on the i^{th} position and $A[i] = i$, it will stop.

1. (15 pts) Given the initial position of the frog, design an algorithm that computes whether the frog will stop somewhere or keep jumping forever.

2. (15 pts) Given an A such that $A[i] \neq i$, $\forall i \quad 0 \leq i < n$, which means that the frog is doomed to keep jumping forever from any initial position. Then, the frog will eventually jump into a "loop" of indices. Given the initial position of the frog, design an algorithm that computes the length of the loop.

   For example, $A = [1, 0, 4, 2, 3, 1, 4, 6]$.

   - When the initial position is on 0, 1, or 5, the frog jumps into the loop $0 \to 1 \to 0 \to 1 \cdots$. The length of the loop is 2.
   - When the initial position is on 2, 3, 4, 6, or 7, the frog jumps into the loop $2 \to 4 \to 3 \to 2 \to 4 \to 3 \cdots$. The length of the loop is 3.

Enough with the frog. The next subproblem is about the sorted array.

3. (15 pts) A is a strictly increasing array with n integers, where $A = [a_0, a_1, \cdots, a_{n-2}, a_{n-1}]$, $n \geq 3$. Now, consider the following definitions.

   - $A_{s,t} = [a_s, a_{s+1}, \cdots, a_{t-2}, a_{t-1}]$, where $0 \leq s < t \leq n$
   - $M_{s,t} =$ the median of $A_{s,t}$
   - $f(i, j) = \max(M_{0,i}, M_{i,j}, M_{j,n}) - \min(M_{0,i}, M_{i,j}, M_{j,n})$, where $0 < i < j < n$

In short, A is partitioned into three subarrays by index i and index j. Each subarray has its own median. $f(i, j)$ equals to "the maximum of three medians" minus "the minimum of three medians".

Design an algorithm to find a pair (i, j), where $0 < i < j < n$, such that $f(i, j)$ is minimized.

Finally, let's play with a new kind of linked list.

4. (15 pts) A circularly linked list is a linked list such that the last node of the linked list is connected to the `head` instead of some NIL (`NULL`) value. We call a node A decreasing in the circularly linked list if $A.next.value < A.value$.

   Consider a circularly linked list L with n integers, represented by some `head` node. Assume that there are two decreasing nodes within L. Design an algorithm that "sorts" all nodes within L such that `head` points to a new circularly linked list with only one decreasing node. We expect the algorithm to run in $O(n)$-time and $O(1)$-space.