Worth: 8% **Due:** By 5:59pm on Tuesday 3 November

Remember to write the *full name* and *student number* of *every group member* prominently on your submission.

---

*Please read and understand the policy on Collaboration given on the Course Information Sheet. Then, to protect yourself, list on the front of your submission **every** source of information you used to complete this homework (other than your own lecture and tutorial notes). For example, indicate clearly the **name** of every student from another group with whom you had discussions, the **title and sections** of every textbook you consulted (including the course textbook), the **source** of every web document you used (including documents from the course webpage), etc.*

*For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks **will** be deducted for incorrect or ambiguous use of notation and terminology, and for making incorrect, unjustified, ambiguous, or vague claims in your solutions.*

---

1. Give an algorithm for the following problem. The input is a sequence of n numbers $\{x_1, x_2, \ldots, x_n\}$, another sequence of n numbers $\{y_1, y_2, \ldots, y_n\}$, and a number z. Your algorithm should determine whether or not $z \in \{x_i + y_j \mid 1 \leq i, j \leq n\}$. You should use universal hashing families, and your algorithm should run in expected time $O(n)$.

   Provide justification that your algorithm is correct and runs in the required time. Be very clear about which theorems from class and/or the text you are using, and how.

2. We can implement a queue Q using two stacks H and T as follows: Think of the stack T as containing the "tail" of the queue (i.e., the recently inserted items), with the most recently inserted item at the top. Think of the stack H as containing the "head" of the queue (i.e., the older items), with the oldest item of the queue at the top. Conceptually, Q consists of T and H placed "back-to-back".

   To EnQueue an item x on Q, we actually Push x into T. To DeQueue an item, we Pop H, provided H is not empty. If H is empty, we transfer the items of T to H (by popping each item of T and then pushing it into H), and then Pop H.

   These algorithms are given below in pseudo-code. (We assume that initially the stacks H and T are empty, and that the function StackEmpty(T) returns **true** if T is an empty stack and **false** otherwise.)

```
EnQueue(Q, x)
    Push(T, x)
```

```
DeQueue(Q)
    if StackEmpty(H) then
        loop
            exit when StackEmpty(T)
            x := Pop(T)
            Push(H, x)
        end loop
    end if
    return Pop(H)
```

Assume that each Push, Pop and StackEmpty operation takes $\Theta(1)$ time.

(a) What is the worst-case time complexity of a single operation in a sequence of m EnQueue and DeQueue operations? Derive matching upper and lower bounds. That is, define an initial situation by describing what H and T look like at the start, and then define a sequence of m operations, where the sequence consists of EnQueue's and DeQueue's. Then show that one of the operations in the sequence (probably the last operation) will have the claimed worst-case time. For the upper bound, show that no operation in any m-operation sequence can ever take more time than the claimed worst-case time.

(b) Use the accounting method to prove that the amortised time complexity of each operation in a sequence of m EnQueue and DeQueue operations is $O(1)$.

To solve this problem, first give a credit scheme indicating how many credits to allocate to each EnQueue and DeQueue operataion. Secondly, state the credit invariant, and thirdly, prove the credit invariant.

3. Recall that the doubling method enables the implementation of a stack without placing a limit on the size of the stack, such that the amortized complexity of each operation is $O(1)$. Every time the array gets full, a new array is allocated whose size is twice the size of the old array, and the old array is copied to the new array.

(a) Suppose we change the implementation so that the size of the new array is 3/2 times the size of the old array. What is the time complexity of a sequence of m operations in the worst-case? Justify your answer.

(b) Suppose we change the implemntation so that the size of the new array is 50 plus the size of the old array. What is the time complexity of a sequence of m operations in the worst-case? Justify your answer.