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 $z \in \{x_i + y_j \mid 1 \le i, j \le n\}$. 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.

   ANSWER:

   First use universal hash maps all x_i $(1 \le i \le n)$ to n brackets. We know that INSERT for hashing takes $O(1)$; therefore, inserting n keys would take $O(n)$.

   Then we run search for $z - y_j$ $(1 \le j \le n)$. It is clearly SEARCH runs in expected time O(1) by the properties of universal hashing. The worst case is $z \notin \{x_i + y_j | 1 \le i, j \le n\}$ which we have to loop through all possible y_j $(1 \le j \le n)$. Since each SEARCH run in expected time $O(1)$, then worst case run time is in $O(n)$.

   Algorithm should run in expected time $O(n)$.

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) DEQUEUE(Q)

```
PUSH($T, x$)                    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.

ANSWER:

Assume T is a stack of n numbers, and H is a stack of y numbers, such that $n \geq 0 \wedge y \geq 0$.
Assume there is a sequence of m operations consisting of ENQUEUE's and DEQUEUE's.
Case 1, for ENQUEUE operation:

- Each ENQUEUE operation in m takes $\Theta(1)$ because it is simply pushing a number onto stack T, which by assumption takes $\Theta(1)$.

Case 2, for DEQUEUE operation:

- case 2.1: If stack H is not empty, then DEQUEUE operation in m will take $\Theta(1)$ since it is just a POP operation which takes $\Theta(1)$.
- case 2.2: If stack H is empty, then in the worst case, every operation except the last operation in m is ENQUEUE, then it will take $\Theta(2(m-1+n)+1) = \Theta(m+n)$ time. The original n number of elements in T and the $(m-1)$ number of ENQUEUE's will be popped from T and pushed into H, with each operation costing 2 executions. The last POP from H will cost an additional 1 execution.

  This is the worst case because if DEQUEUE was not the last operation in m, then in case 2.2 DEQUEUE will take less than $2(m-1+n)+1$ operations . Let's assume that DEQUEUE was xth operation in m, where x is not the last operation. Then $x > 1$ and $2(m-1) > 2(m-x)$; hence, $2(m-1+n)+1 > 2(m-x+n)+1$.
  The worst-case of a single operation happens when H is empty and T has n elements and sequence of m operations are $[EnQueue, EnQueue, EnQueue....EnQueue, DeQueue]$ ($m-1$ ENQUEUE's follow by 1 DEQUEUE) and the time complexity is in $\Theta(m+n)$.

(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 operation. Secondly, state the credit invariant, and thirdly, prove

the credit invariant.

ANSWER:

Each ENQUEUE will charge 3 credits.

Each DEQUEUE will change 1 credits.

Each PUSH and POP cost 1 credit.

Credit Invariant: At any step each element in H will have 0 credit, each element in T will have have 2 credits.

Proof by induction:

Base Case: Initially H and T are empty.

Induction Step: Assume all elements in H have 0 credit, all elements T have 2 credits.

– Case 1: ENQUEUE
  When we call ENQUEUE, H remain unchanged. In T ENQUEUE charges 3 credits and cost 1 credit for PUSH the element to T; hence the new item in T has 2 credits.
– Case 2: DEQUEUE
  case 2.1: H is not empty, then DEQUEUE charges 1 credits and cost that 1 credit for POP a element from H; hence, the rest elements in H remain unchanged. Also everything in T remain unchanged.
  case 2.2: H is empty, every item in T are popped and pushed onto H which cost all their 2 credits. Then using the 1 credit charges from DEQUEUE to POP a element from H. Now T is empty and every item in H has 0 credit.

Thus invariant is always true, so total charge for sequence is upper bound on total on cost. Then total charge at most be $3n$ so amortized cost per operation is 3.

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.

   ANSWER:

   Idea: Charge each operation 4 credits.
   – 1 credit (the "append-credit") will be spent when appending the element.
   – 1 credit (the "copy-credit") will be spent when copying the element to a new array.
   – 2 credits (the "recharge-credits") are used to recharge the old elements(first $\frac{2}{3}$ of the list) that have spent their "copy-credits".

   Credit Invariant: Each element in last $\frac{1}{3}$ of the array has 3 credits.

   Proof by induction:

   Base Case: Initially array is empty.

   Inductive Step:

- Array is not full, then 1 credits is spend for append and 3 credits stored on the item; hence, Invariant satisfied.
- Array is full, then make a new array. The last $\frac{1}{3}$ of the array using their "copy-credit" copy themselves to the new array and "recharge-credits" copy all items in first $\frac{2}{3}$ of array to the new list. Then add new item 3 credits. Since the last $\frac{1}{3}$ of the array only have this newly added item with 3 credits; hence, Invariant satisfied.

Therefore, a charge of 4 credits is enough. Then time complexity of a sequence of m operations in the worst-case is 4.

(b) Suppose we change the implementation 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.

Answer:

Idea: Charge each operation $\lceil \frac{n}{50} \rceil + 2$ credits, where n is the size of the old array.

- 1 credit (the "append-credit") will be spent when appending the element.
- 1 credit (the "copy-credit") will be spent when copying the element to a new array.
- $\lceil \frac{n}{50} \rceil$ credits (the "recharge-credits") are used to recharge the old elements (elements from old array) that have spent their "copy-credits".

Credit Invariant: Each element in last 50 elements of the array has $\lceil \frac{n}{50} \rceil + 1$ credits.

Proof by induction:

Base Case: Initially array is empty.

Inductive Step: (suppose the size of the array is $n + 50$, where n is size of its previous array.)

- Array is not full, then 1 credits is spend for append and $\lceil \frac{n}{50} \rceil + 1$ credits stored on the item; hence, Invariant satisfied.
- Array is full, then make a new array. The last 50 elements of the array use their "copy-credit" copy themselves to the new array and "recharge-credits" copy all first n items from array to the new list. Then add new item $\lceil \frac{n+50}{50} \rceil + 1$ credits. Since the last 50 elements of the array only have this newly added item with $\lceil \frac{n+50}{50} \rceil + 1$ credits and $n + 50$ is size of its previous array; hence, Invariant satisfied.

Therefore, a charge of $\lceil \frac{n}{50} \rceil + 2$ credits is enough. Then time complexity of a sequence of m operations in the worst-case is $O(n)$.