Minor
Akshay Iyer
Roll number 190070006
Hackerrank ID- h190070006
email ID on hackkerrank- 190070006@iitb.ac.in

I pledge on my honor that I have not given or received any unauthorized assistance on this assignment or any previous task.

Q1:

My goals were that every time the number of elements in the array became half of its original capacity, I decreased the capacity of the array to its previous capacity. I achieved this goal by using an if statement in the remove function, and creating the shrink function that performs the above operation. I did this by copying the elements into an array of half the original capacity, and replacing the old array with that. I also set the variable that denotes the capacity of the array, to half its previous value with the above step.

Q2:

Initially I thought that we would start from the head pointer and go till the index to be deleted, and then rearrange the pointers there accordingly. But this method was very inefficient and it had been explicitly mentioned to use the fastest method possible in the question.
Thus, I implemented a different method. Instead of deleting the node at the address given, I just tansferred the value stored in the next node, to this node. Then I deleted the next node, and rearranged the connections of the previous pointer, to point to the next to next element.
We needed the address of the previous node to set its next pointer, which wouldn't be possible to get from only the address of the current pointer.
Thus by transferring the value of the next node into the current one, I eliminated the need to change the next pointer of the previous node, as it points to the current pointer anyway.
I then rearranged the next pointers accordingly.

Q3:

This was a pretty good question.

We had also been told to use a stack. So I figured that as long as a group of numbers was in ascending order, only the last element matters as if any number is greater than the last element, it greater than all the other elements too. Thus I decided that my stack should store only the indices of the numbers that I have a doubt, might be greater than the number we are finding the span of.

Thus, we push every index onto the stack, but each time the number we are checking the span of, is greater than the last number of the stack, we pop that value from the stack as it becomes irrelevant as any subsequent number, if greater than the current number, will also be greater than numbers at those popped indices in the stack.
At the end, to find the span, we compare the current number with all of the numbers in the doubtful indices, and if we find a number greater than the current number, we subtract its index from the index of the current number to get our span.

Q4:

I tried to initially make a double linked list myself, in which the last element had a next index pointing back to the first element. Like a circular doubly linked list. I did everything right, but there was this weird bug caused by a memory leak somewhere, which I for the life of me could not figure out. I figured the exact reason and place which was causing the problem, yet I had no clue how to rectify it. None of the TA's knew how to help either. Thus it caused a segmentation fault in most test cases.(I have pasted the code for this below, after Q6)

But the above method would not have passed all the test cases without the bug too, as insertion and deletion would have a worst case time complexity of O(n^2). Thus I thought of implementing another dll which was like a binary tree. Every node would have a left tree pointer(pointing to the midpoint on the left), a right tree pointer(pointing to the midpoint on the right), a value(stored at that index), and an index(of the number). The head pointer would point towards the middle element and keep jumping to the middle element of the second half/first half etc, by comparing the indexes of the nodes. This is kind of like a binary tree and works because indices of the nodes will be in ascending order, hence we can use that to find a particular index. I was figuring out insertion and deletion for this too, but due to lack of time, and the approaching deadline, I couldn’t complete this.

Hence I finally decided to use vectors and vector functions to do it, as that was the quickest way I could think of passing all the test cases before the deadline. Please give me my 20 percent.

Q5 and Q6 are as below-

Akshay Iyer 190070006 Minor

Date :..............................
MON TUE WED THU FRI SAT SUN
□ □ □ □ □ □ □

DSA Assignment 3

5) Crude method -

Maximum size of the stack is n entries. Worst case, for any single step, we will have to copy all n of these elements into a new backup.

Thus $C_n = O(n)$ $\therefore O(n^2)$ is total time

Amortized time is $O(n)$.

Aggregate method -

ΣC_i is what we need to find.

In the worst case scenario, every n^{th} step, we will have to copy the maximum size of n elements into a new backup. Every other push/pop step takes a time of 1.
Thus, after p stack operations,

$\Sigma C_i = p + \frac{p(n)}{n} = 2p$. Thus $\Sigma C_i = O(p)$

But now, amortized complexity $= \frac{\Sigma C_i}{\text{total no. of operations}}$

$$= \frac{O(n)}{n} = O(1).$$

Thus O(1) is the amortized complexity.

6) Now we know that ~~each~~ exactly one of the elements of Y are repeated, in the set of f(x) for $x \in X$.

Thus we would like to find the repeated value. The easiest way to do this would be to find the sum of f(x) $\forall\ x \in X$.

Let the repeated element of Y be y.

$$\therefore\ 1+2+3+\dots+N+y = \Sigma f(x_i)$$

$$\therefore\ \Sigma f(x_i) = \frac{N(N+1)}{2} + y$$

$$\therefore \text{ The repeated element } y = \Sigma f(x_i) - \frac{N(N+1)}{2}$$

We can find $\Sigma f(x_i)$ easily with a single loop, with a total time complexity of O(N).

subtracting $\frac{N(N+1)}{2}$ will take O(1) time.

Once we know y, we conduct a simple linear search to find the two values of x that give the same value of f(x). This will take at most, time O(N).

Hence the total time taken to complete the algorithm is O(N).

<u>Pseudo code</u> -

```
ADT K        s=0
def findy(N):
    for i in range(1, N):
        s= K.f(i) +s
    s = s- ((N(N+1))/2)
    return(s)
```

```
N1= whatever value of N has been given
y= findy(N1)                a=0
for i in range(1,N1):
    Linkedlist1[0] = i
    Linkedlist1[1] = K.f(i)
    if Linkedlist1[1] == y and    a==0 :
        Linkedlist2[0] = i
        Linkedlist2[1] = Linkedlist1[1]
    a =1
    elif Linkedlist1[1] ==y:
        break;
```

OR.

We could do it in a much less elegant way in $O(n^2)$.

Fix x_1 and position 1, check for all other x_2 if any $f(x_2) = f(x_1)$.

If not, then fix x_1 at position 2. check for x_2, such that $f(x_2) = f(x_1)$ for all x_2 beyond x_1.

Similarly for x_1 fixed at position 3 and so on.

We can do this as we do not care about time complexity in this question, and this approach satisfies all the space complexity conditions.

time complexity $= 1 + 2 + 3 + \dots n-1$

$$= \frac{n(n-1)}{2} = O(n^2)$$

Code for Q4 first approach(Please give partial marks for this if possible I tried very hard to do this)

```
#include <map>
#include <set>
#include <list>
#include <cmath>
#include <ctime>
#include <deque>
#include <queue>
#include <stack>
#include <string>
#include <bitset>
#include <cstdio>
#include <limits>
#include <vector>
#include <climits>
#include <cstring>
#include <cstdlib>
#include <fstream>
#include <numeric>
#include <sstream>
#include <iostream>
#include <algorithm>
#include <unordered_map>
#include <bits/stdc++.h>
using namespace std;

// Link list node
class Node {
public:
    int value;
    Node* next;
    Node* prev;
};

/*
headPtr is a reference to the head node(i.e. pointer to pointer) and
deleteNodePtr is the node which is to be deleted. You can see the Node definition above.
It is guaranteed that deleteNodePtr will not point to the last element.
*/
void deleteNode(Node** headPtr, int j) {
    // Write your code here
    if(*headPtr==NULL)
        return;
    Node* deleteNodePtr= *headPtr;

    for(int i=0; i<j; i++)
    {
        deleteNodePtr = deleteNodePtr->next;
    }

    Node* temp= deleteNodePtr->prev;
    Node* temp1= deleteNodePtr->next;
    temp->next = temp1;
    temp1->prev = temp;
    if(j==0)
    {
        *headPtr=temp1;
    }
    Node* d= *headPtr;
    cout<<(((d->next)->next)->value)<<endl;
    //cout<<((temp1->prev)->value)<<endl;
    if(temp!=NULL)
    {
    delete(temp);
    }
    if(temp1!=NULL)
    {
    delete(temp1);
    }
    if(deleteNodePtr!=NULL)
    {
    delete(deleteNodePtr);
    }
    if(d!=NULL)
    {
    delete(d);
    }

}

void insertNode(Node** headPtr, int x, int j) {
    // Write your code here
    Node* newNode = new Node();
    newNode->value = x;
    Node* temp= *headPtr;
    for(int i=0; i<j; i++)
    {
        temp = temp->next;
    }
    newNode->next = temp;
    newNode->prev = temp->prev;
    temp->prev = newNode;
    Node* temp2=newNode->prev;
    temp2->next = newNode;
    free(temp);
    free(temp2);
    if(j==0)
    {
        *headPtr=newNode;
    }


}

int getNode(Node** headPtr, int j) {
    // Write your code here
    Node* temp= * headPtr;
    for(int i=0; i<j; i++)
    {
        temp = temp->next;
    }
    return(int(temp->value));
}

void rotate(Node** head, int r) {
    // Write your code here
    if(r==-1)
    {
    *head=(*head)->next;
    }
    else if(r==1)
    {
    *head=(*head)->prev;
    }
}


/* You Should NOT Modify Anything Below This */
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    Node* head = NULL;
    int n;
    cin>>n;
    int n1;
    cin>>n1;
    vector <int> values(n);
    vector <Node*> pointers(n);
    for(int i=0;i<n;i++)
    {
        cin>>values[i];
    }


    Node* newNode = new Node();
    newNode->value = values[0];
    pointers[0] = newNode;
    for(int i=1;i<=n-1;i++)
    {

        // creating the node in the linkedList
        Node* newNode = new Node();
        newNode->value = values[i];
        pointers[i] = newNode;
        newNode->prev=pointers[i-1];
        pointers[i-1]->next=pointers[i];
    }
    head=pointers[0];
    pointers[n-1]->next=pointers[0];
    pointers[0]->prev=pointers[n-1];





    for(int j=0;j<n1;j++)
    {
    char c;
    cin>>c;
    if(c=='I')
    {
      int k;
      cin>>k;
      int k2;
      cin>>k2;
      insertNode(&head,k2,k);
    }
    if(c=='D')
    {
      int k;
      cin>>k;
      deleteNode(&head,k);
    }
    if(c=='R')
    {
      int k;
      cin>>k;
      rotate(&head,k);
    }
    if(c=='P')
    {
      int k;
      cin>>k;
      cout<<getNode(&head,k);
      cout<<endl;

    }
    Node* d= head;
    cout<<(((d->next)->next)->value)<<endl;
    if(d!=NULL)
    {
    delete(d);
    }
    }
}
```