# Demux Summer Hackathon -3

## Solution

## 1. Queue using Two Stacks

In this problem we have to implement queue using two stacks. First stack will be used to insert the element **(stack_insert**) and the second stack will be used to delete and display the element **(stack_delete).**

**Queries :**

- x - Insert x into **stack_insert.**
- A **stack_delete** is empty by one pop elements from **stack_insert** and push them into **stack_delete,** this way the order of the elements is reversed and the first inserted element is on the top of **stack_delete** and the last inserted element is at the bottom of **stack_delete**. This way we can delete the first inserted element easily.
- If **stack_delete** is empty, fill it in the same way as second query and display the top element of **stack_delete**, else simply display the top element of **stack_delete.**

**Code:**

```
import java.io.*;
import java.util.*;

class queue_using_two_stacks {

public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Stack<Integer> stack_insert = new Stack<>();
        Stack<Integer> stack_delete = new Stack<>();
        int q = sc.nextInt();
        while(q-- > 0) {
                int type = sc.nextInt();
                if(type == 1) {
                        int x = sc.nextInt();
                        stack_insert.push(x);
                } else if(type == 2) {
                        if( stack_delete.isEmpty() ) {
                                while( !stack_insert.isEmpty() ) {
                                        stack_delete.push( stack_insert.pop() );
```

```
                    }
                }
                stack_delete.pop();
            } else {
                if( stack_delete.isEmpty() ) {
                    while( !stack_insert.isEmpty() ) {
                        stack_delete.push( stack_insert.pop() );
                    }
                }
                System.out.println( stack_delete.peek() );
            }
        }
}
}
```

## *2.* Sherlock and Array

Problem: Find if there exists an element in the array, such that, the sum of elements on its left is equal to the sum of element on its right.

Formally, find an i, such that, **A1 + A2 +...+ Ai-1 = Ai+i + Ai+2 +-.. + An.**

**Solution:**

**Approach 1:**

We first count total = A1 + A2 +...+ An.

Now if for an i, let's define sum = A1 + A2 +...+ An

If **sum = total — sum — Ai+1**, the answer is YES. ‘

So, we maintain a sum variable in second pass which we keep updating and checking.

**Approach 2:**

We keep a prefix array pre, where **pre[i] = A1 + A2+...+Ai**. Traversing over i, we can easily check whether sum on left is equal to sum on right.

We can use the following property to calculate prefix array**, pre**, in **O(N)** steps.

**pre[i] = 0 if i == 0**
**pre[i] = pre[i-1] + A[i] else**

**C++**

```
#include<bits/stdc++.h>
using namespace std;
#define assn(n,a,b) assert(n<=b && n>=a)
typedef long long LL;
LL pre[100005];
int main()
{
   int t;
   cin >> t;
   assn(t,1,10);
   while(t--)
   {
      LL n,i,j,flag=0,x;
      cin >> n;
      assn(n,1,100000);
      for(i=1; i<=n; i++)
      {
         cin >> x;
         assn(x,1,20000);
         // store pre[i]= sum of all elements till index i.
         pre[i]=pre[i-1]+x;
      }
      for(i=1; i<=n; i++)
      {
         // check if sum to left is same as sum to right
         if(pre[i-1]==(pre[n]-pre[i]))flag=1;
      }
      if(flag)cout << "YES\n";
```

```
        else cout << "NO\n";
    }
    return 0;
}
```

## 3. Find Merge Point of Two Lists

To calculate the merge point, first calculate the difference in the sizes of the linked lists. Move the pointer of the smaller linked list by this difference. Increment both pointers till you reach the merge point.

**Code:**

```
int getCount(Node* head)
{
  Node* current = head;
  int count = 0;

  while (current != NULL)
  {
    count++;
    current = current->next;
  }

  return count;
}

int getNode(int d, Node* head1, Node* head2)
{
  int i;
  Node* current1 = head1;
  Node* current2 = head2;

  for(i = 0; i < d; i++)
  {
    if(current1 == NULL)
    {  return -1; }
    current1 = current1->next;
  }

  while(current1 !=  NULL && current2 != NULL)
  {
    if(current1 == current2)
      return current1->data;
    current1= current1->next;
    current2= current2->next;
  }

  return -1;
}

int FindMergeNode(Node *headA, Node *headB)
{
    // Complete this function
    // Do not write the main method.
    int c1 = getCount(headA);
  int c2 = getCount(headB);
  int d;

  if(c1 > c2)
  {
    d = c1 - c2;
    return getNode(d, headA, headB);
  }
  else
  {
    d = c2 - c1;
    return getNode(d, headB, headA);
  }
}
```

## 4. Quicksort 1 - Partition

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
```

```
#include <assert.h>

/* Head ends here */
void partition(int ar_size, int *  ar) {

int f,temp=ar[0],j=0,k=0,i;
    int ar1[ar_size],ar2[ar_size];
    for(i=0;i<ar_size;i++)
    {
        if(ar[i]<temp)
        {
            ar1[j]=ar[i];
            j++;
        }
        if(ar[i]>temp)
        {
            ar2[k]=ar[i];
            k++;
           //printf("k=%d\n",k);
        }

    }
  /*  ar1[j]=temp;
    j++;
        for(i=0;i<k;i++)
        {
         ar1[j]=ar2[i];
            j++;
        }*/
   // printf("j=%d\n",j);
    for(i=0;i<j;i++)
    printf("%d ",ar1[i]);
    printf("%d ",temp);
    for(i=0;i<k;i++)
        printf("%d ",ar2[i]);

}

/* Tail starts here */
int main() {

   int _ar_size;
scanf("%d", &_ar_size);
int _ar[_ar_size], _ar_i;
for(_ar_i = 0; _ar_i < _ar_size; _ar_i++) {
   scanf("%d", &_ar[_ar_i]);
}

partition(_ar_size, _ar);

   return 0;
}
```

## 5. Deque-STL

```
#include <iostream>
#include <deque>

using namespace std;

// A Dequeue (Double ended queue) based method for printing maixmum element of
// all subarrays of size k
void printKMax(int arr[], int n, int k)
{
    // Create a Double Ended Queue, Qi that will store indexes of array elements
    // The queue will store indexes of useful elements in every window and it will
    // maintain decreasing order of values from front to rear in Qi, i.e.,
    // arr[Qi.front[]] to arr[Qi.rear()] are sorted in decreasing order
    std::deque<int>  Qi(k);

    /* Process first k (or first window) elements of array */
    int i;
    for (i = 0; i < k; ++i)
    {
        // For very element, the previous smaller elements are useless so
        // remove them from Qi
        while ( (!Qi.empty()) && arr[i] >= arr[Qi.back()])
            Qi.pop_back();  // Remove from rear

        // Add new element at rear of queue
        Qi.push_back(i);
    }

    // Process rest of the elements, i.e., from arr[k] to arr[n-1]
    for ( ; i < n; ++i)
    {
        // The element at the front of the queue is the largest element of
        // previous window, so print it
        cout << arr[Qi.front()] << " ";

        // Remove the elements which are out of this window
        while ( (!Qi.empty()) && Qi.front() <= i - k)
            Qi.pop_front();  // Remove from front of queue

        // Remove all elements smaller than the currently
        // being added element (remove useless elements)
        while ( (!Qi.empty()) && arr[i] >= arr[Qi.back()])
            Qi.pop_back();

         // Add current element at the rear of Qi
        Qi.push_back(i);
    }

    // Print the maximum element of last window
    cout << arr[Qi.front()] << endl;
}

// Driver program to test above functions
int main()
{
    int t;
    cin >> t;
    while(t>0) {
    int n,k;
    cin >> n >> k;
    int i;
    int arr[n];
    for(i=0;i<n;i++)
    cin >> arr[i];
    printKMax(arr, n, k);
    t--;
    }
    return 0;
}
```