GeeksMan
Queue Data Structure
Lesson 3

Maximum of all subarrays of size k

Given an array A and an integer K . Find the maximum for each and every contiguous subarray of size K.

Input:

The first line of input contains an integer T denoting the number of test cases. The first line of each test case contains a single integer N denoting the size of array and the size of subarray K. The second line contains N space-separated integers denoting the elements of the array.

Output:

Print the maximum for every subarray of size k.

Example:

Input:

2

9 3

1 2 3 1 4 5 2 3 6

10 4

8 5 10 7 9 4 15 12 90 13

Output:

3 3 4 5 5 5 6

10 10 10 15 15 90 90

ALGORITHM:

Create a deque of size k, which will keep track of the greatest element of the subarray .

Traverse all elements of the array one by one and keep updating the deque.

Check that your deque is sorted always with the greatest element at front and smallest at back.

SOLUTION :

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
void solve()
{
    int n,k,i;
    cin>>n>>k;
    int A[n];
    for(i=0;i<n;i++)
    {
        cin>>A[i];
    }
    deque<int> q(k);
    for(i=0;i<k;i++)
    {
        while(!q.empty()&& A[i]>=A[q.back()])
        {
            q.pop_back();
        }
        q.push_back(i);
    }
```

```
22.  for(;i<n;i++)
23.  {
24.      cout<<A[q.front()]<<" ";
25.      while(!q.empty()&& q.front()<=(i-k))
26.      {
27.          q.pop_front();
28.      }
29.      while(!q.empty()&& A[i]>=A[q.back()])
30.      {
31.          q.pop_back();
32.      }
33.      q.push_back(i);
34.  }
35.   cout<<A[q.front()];
36.}
37.int main()
38.{
39.    int t;
40.    cin>>t;
41.    while(t--)
42.    {
43.      solve();
44.      cout<<endl;
45.    }
46.    return 0;
47.}
```

Chinky and diamonds

One day Chinky was roaming around the park and suddenly she found N bags with diamonds.
The i'th of these bags contains Ai diamonds. She felt greedy and started to pick up the bag very fastly. But due to quick movement, she drops it on the ground. But as soon as she drops the bag, a genie appears in front of her and he increases the number of diamonds in the bag suddenly!
Now the bag which was used to contain P diamonds , now contains [P/2] diamonds, where [p] is the greatest integer less than p. Then genie gave her the time K minutes in which she can take as many as diamonds. In a single minute , she can take all the diamonds in a single bag, regardless of the number of diamonds in it. Find the maximum number of diamonds that Chinky can take with her.

Input:

First line contains an integer **T**. **T** test cases follow.First line of each test case contains two space-separated integers **N** and **K**.Second line of each test case contains N space-separated integers,the number of diamonds in the bags.

Output:

Print the answer to each test case in a new line.

Constraints:

$1 \leq \mathbf{T} \leq 10$

$1 \leq \mathbf{N} \leq 105$

$0 \leq \mathbf{K} \leq 105$

$0 \leq \mathbf{Ai} \leq 105$

Example:

Input:

1

5 3

2 1 7 4 2

Output:

14

Explanation:

The state of bags is:

2 1 7 4 2

Chinky takes all diamonds from Third bag (7). The state of bags becomes:

2 1 3 4 2

Chinky takes all diamonds from Fourth bag (4). The state of bags becomes:

2 1 3 2 2

Chinky takes all diamonds from Third bag (3). The state of bags becomes:

2 1 1 2 2

Hence, the Chinky takes 7+4+3= 14

Algorithm :

1. Make a priority queue , and push all the elements in it.
2. At every instance the largest element 'a' is added in sum and is replaced by a/2 in the priority queue .
3. Step 2 is carried out k times and the final answer is reported.

SOLUTION :

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int t;
   cin>>t;
   while(t--)
   {
      int n,k;
      cin>>n>>k;
      priority_queue<int> pq;
      for(int i=0;i<n;i++)
      {
         int x;
         cin>>x;
         pq.push(x);
      }
      long long int sum=0;
      for(int i=0;i<k;i++)
      {
         int a= pq.top();
         pq.pop();
         sum+=a;
         a=a/2;
         pq.push(a);
      }
      cout<<sum<<endl;  }
   return 0;
}
```

Card Rotation

Given a sorted deck of cards numbered 1 to N.

1) We pick up 1 card and put it on the back of the deck.

2) Now, we pick up another card , it turns out to be card numbered 1 , we put it outside the deck.

3) Now we pick up 2 cards and put them on the back of the deck.

4) Now, we pick up another card and it turns out to be card numbered 2 , we put it outside the deck. ...

We perform this step till the last card.

If such arrangement of decks is possible, output the arrangement, if it is not possible for a particular value of N then output -1.

Input:

The first line of the input contains the number of test cases 'T', after that 'T' test cases follow.

Each line of the test case consists of a single line containing an integer 'N'.

Output:

If such arrangement of decks is possible, output the arrangement, if it is not possible for a particular value of n then output -1.

Constraints:

1 <= T <= 100;

1<= N<= 1000;

Example:

Input :

2

4

5

Output :

2 1 4 3

3 1 4 5 2

Explanation:

Test Case 1: We initially have [2 1 4 3]

In Step1, we move the first card to the end. Deck now is: [1 4 3 2]

In Step2, we get 1. Hence we remove it. Deck now is: [4 3 2]

In Step3, we move the 2 front cards ony by one to the end ([4 3 2] -> [3 2 4] -> [2 4 3]).Deck now is: [2 4 3]

In Step4, we get 2. Hence we remove it. Deck now is: [4 3]

In Step5, the following sequence follows: [4 3] -> [3 4] -> [4 3] -> [3 4]. Deck now is: [3 4]

In Step6, we get 3. Hence we remove it. Deck now is: [4]

Finally, we're left with a single card and thus, we stop.

Solution :

1. Create a queue and enqueue all 1 to n numbers in it.
2. Traverse the queue , and for each element , enqueue the front element and dequeue the front element , and enter the elements in the array correspondingly.

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

int main()
{
      int t;
   cin >> t;
   while(t--)
   {
      int n;
      cin >> n;
      int a[n];
      queue<int>q;
      for(int i = 1 ; i <=n ;i++)
         q.push(i);
      for(int i = 1 ; i <= n ; i++)
      {
         int j = i;
         while(j--)
         {
                              q.push(q.front());
            q.pop();
```

```
24.             }
25.
26.             a[q.front()] = i;
27.             q.pop();
28.         }
29.             for(int i = 1 ; i <=n ;i++)
30.                 cout << a[i]<< " ";
31.         cout << endl;
32.     }
33.         return 0;
34. }
```

Circular tour

Suppose there is a circle. There are N petrol pumps on that circle. You will be given two sets of data.
1. The amount of petrol that every petrol pump has.
2. Distance from that petrol pump to the next petrol pump.
Find a starting point where the truck can start to get through the complete circle without exhausting its petrol in between.
Note : Assume for 1 litre petrol, the truck can go 1 unit of distance.

Input:
N = 4
Petrol = 4 6 7 4
Distance = 6 5 3 5

Output:
1

Explanation:

There are 4 petrol pumps with
amount of petrol and distance to next
petrol pump value pairs as {4, 6}, {6, 5},
{7, 3} and {4, 5}. The first point from
where truck can make a circular tour is
2nd petrol pump. Output in this case is 1
(index of 2nd petrol pump).

Expected Time Complexity: O(N)
Expected Auxiliary Space : O(N)

ALGORITHM:

1. **O(N*N) Solution :**

Consider every petrol pump as a starting point and see if there is a possible tour.

If we find a starting point with a feasible solution, we return that starting point.

2. **O(N) Solution :**

An efficient approach is to use a Queue to store the current tour. First enqueue the first petrol pump to the queue .

Keep enqueueing petrol pumps till we either complete the tour, or the current amount of petrol becomes negative .

If the amount becomes negative , then we keep dequeuing petrol pumps until the queue becomes empty.

```
#include<bits/stdc++.h>
using namespace std;
struct petrolPump
{
    int petrol;
    int distance;
};
int tour(petrolPump [],int );
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        int n;
        cin>>n;
        petrolPump p[n];
        for(int i=0;i<n;i++)
            cin>>p[i].petrol>>p[i].distance;
        cout<<tour(p,n)<<endl;
    }
}

int tour(petrolPump arr[],int n)
{
    int start = 0;
    int end = 1;

    int bal = arr[start].petrol - arr[start].distance;

```

```
31.   while (end != start || bal < 0)
32.   {
33.       while (bal < 0 && start != end)
34.       {
35.           bal -= arr[start].petrol - arr[start].distance;
36.           start = (start + 1) % n;
37.
38.           if (start == 0)
39.           return -1;
40.       }
41.       bal += arr[end].petrol - arr[end].distance;
42.
43.       end = (end + 1) % n;
44.   }
45.   return start;
46.}
```