GeeksMan
Queue Data Structure
Lesson 2

Priority queue

Priority queue is an abstract data type similar to regular queue in which each element additionally has a "priority" associated with it. In a priority queue , an element with high priority is served before an element with low priority. In some implementations, if two elements have the same priority, they are served according to the order in which they were enqueued . (Priorities can be any **Comparable** values; in our examples, we'll just use numbers.)

A priority queue is different from a "normal" queue, because instead of being a "first-in-first-out" data structure , values come out in order by priority . A priority queue can be implemented using many of the data structures (an array, a linked list, or a binary search tree).

operation	argument	return value	size	contents (unordered)	contents (ordered)
insert	P		1	P	P
insert	Q		2	P Q	P Q
insert	E		3	P Q E	E P Q
remove max		Q	2	P E	E P
insert	X		3	P E X	E P X
insert	A		4	P E X A	A E P X
insert	M		5	P E X A M	A E M P X
remove max		X	4	P E M A	A E M P
insert	P		5	P E M A P	A E M P P
insert	L		6	P E M A P L	A E L M P
insert	E		7	P E M A P L E	A E E L M
remove max		P	6	E E M A P L	A E E L M

A sequence of operations on a priority queue

Let's code for the same :

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


#define n 5
int q[n];
int front = -1;
int rear = -1;
void check(int data)
{
   int i,j;
   for (i = 0; i <= rear; i++)
   {
      if (data >= q[i])
      {
         for (j = rear + 1; j > i; j--)
         {
            q[j] = q[j - 1];
         }
         q[i] = data;
         return;
      }
   }
   q[i] = data;
}
void enqueue(int data)
{
  if (rear >=n - 1)
```

```
30.  {
31.          cout << "\nQueue overflow no more elements can be inserted";
32.      return;
33.  }
34.  if ((front == -1) && (rear == -1))
35.  {
36.      front++;
37.      rear++;
38.      q[rear] = data;
39.      return;
40.  }
41.  else
42.      check(data);
43.       rear++;
44. }
45.
46. void dq()
47. {
48.  if (front == -1 && rear == -1)
49.  {
50.          cout << "Underflow\n";
51.  }
52.  else if (front == rear)
53.  {
54.      front = -1;
55.      rear = -1;
56.  }
57.  else
58.  {
59.      front++;
```

```
60.   }
61. }
62. void display()
63. {
64.   if (front == rear && rear == -1)
65.   {
66.         cout << "Queue is empty\n" ;
67.   }
68.   else
69.   {
70.      int i;
71.      for (i = front ; i<=rear ;i++)
72.      {
73.         cout << q[i] << " ";
74.      }
75.   }
76. }
77. void peek()
78. {
79.   if (front == -1 && rear == -1)
80.   {
81.         cout << "Queue is empty\n";
82.   }
83.   else
84.   {
85.      cout << "Front = " << q[front];
86.   }
87. }
88. int main()
89. {
```

```
90.  while(1)
91.  {
92.  int a;
93.      cout << "Choose any option:\n";
94.      cout << "Enter 1 to enqueue\n";
95.      cout << "Enter 2 to deque\n";
96.      cout << "Enter 3 to display\n";
97.      cout << "Enter 4 to get the front element\n";
98.      cout << "Enter 5 to exit\n";
99.  cin >> a;
100.     switch(a)
101.     {
102.     case 1:
103.        {
104.           int x;
105.           cout << "Enter the element you want to enqueue: ";
106.           cin >> x;
107.           enqueue(x);
108.           break;
109.        }
110.     case 2:
111.    {
112.           dq();
113.           break;
114.        }
115.     case 3:
116.        {
117.           display();
118.           break;
119.        }
```

```
120.      case 4:
121.          {
122.              peek();
123.              break;
124.          }
125.
126.      case 5:
127.          {
128.              exit(1);
129.          }
130.      }
131.      }
132.
133.  }
```

Priority Queue in STL

- **empty()** function returns whether the queue is empty.
- **size()** function returns the size of the queue.
- **top()** Returns a reference to the top most element of the queue
- **push(g)** function adds the element 'g' at the end of the queue.
- **pop()** function deletes the first element of the queue

Example :

```
1.  #include<bits/stdc++.h>
2.  using namespace std;
3.  int main()
4.  {
5.    priority_queue<int>pq;
6.    pq.push(10);
7.    pq.push(100);
8.    pq.push(5);
9.    pq.push(105);
10.   priority_queue<int>q = pq;
11.   while(q.size() != 0)
12.   {
13.      cout << q.top() <<" ";
14.      q.pop();
15.   }
16.   cout << "\nfirst = " << pq.top();
17.   cout << "\nsize = " << pq.size();
```

```
18.     pq.pop();
19.     cout << "\nAfter Popping : ";
20.     cout << "\nfirst = " << pq.top();
21.     cout << "\nsize = " << pq.size();
22.
23.  }
```

stdout

```
105 100 10 5
first = 105
size = 4
After Popping :
first = 100
size = 3
```

Min sum formed by digits

Given an array of digits (values are from 0 to 9), find the minimum possible sum of two numbers formed from digits of the array. All digits of a given array must be used to form the two numbers.

Input:

The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. First line of each test case contains an integer N denoting the size of the array. Next line of each test contains N space separated integers denoting the elements of the array.

Output:

For each test case output a single line containing the required sum.

Example:

Input

2

6

6 8 4 5 2 3

5

5 3 0 7 4

Output

604

82

Explanation:

The minimum sum is formed by numbers 358 and 246

ALGORITHM :

To minimize the sum of two numbers to be formed, we must divide all digits in two halves and assign half-half digits to them , in ascending order .

Push all the elements in a priority queue and retrieve min values from it , and form a number .

We return the sum of two formed numbers, which is our required answer.

SOLUTION :

```
1.  #include<iostream>
2.  #include<bits/stdc++.h>
3.  using namespace std;
4.  void solve()
5.  {
6.      int n;
7.      cin>>n;
8.      int A[n];
9.      int i;
10.     priority_queue<int> p;
11.     for(i=0;i<n;i++)
12.     {
13.         int x;
14.         cin >> x;
15.         p.push(x);
16.     }
17.     i=1;
18.     char c;
19.     string o,e;
20.     while(i<=n)
```

```
21.   {
22.         c=char(48+p.top());
23.       if(i%2)
24.       {
25.                           o=c+o;
26.       }
27.       else
28.       {
29.           e=c+e;
30.       }
31.       i++;
32.       p.pop();
33.   }
34.    int ev = stoi(e);
35.    int od = stoi(o);
36.    cout<<ev+od;
37.}
38.int main()
39.{
40.  int t;
41.  cin>>t;
42.  while(t--)
43.  {
44.      solve();
45.      cout<<endl;
46.  }
47.}
```

Deque in C++ Standard Template Library (STL)

Double ended queues are containers with the feature of expansion and contraction on both the ends . A queue data structure allows insertion only at the end and deletion from the front. This is like a queue in real life, wherein people are removed from the front and added at the back . Double ended queues are a special case of queues where insertion and deletion operations are possible at both the ends.

Methods of Deque:

1. **max_size():** Returns the maximum number of elements that a deque container can hold.
2. **push_front():** This function is used to push elements into a deque from the front.
3. **push_back():** This function is used to push elements into a deque from the back.
4. **pop_front() and pop_back():** pop_front() function is used to pop or remove elements from a deque from the front. pop_back() function is used to pop or remove elements from a deque from the back.
5. **front() and back():** front() function is used to reference the first element of the deque container. back() function is used to reference the last element of the deque container.

6. **empty() and size()** : empty() function is used to check if the deque container is empty or not. size() function is used to return the size of the deque container or the number of elements in the deque container.

```
#include <iostream>
#include <deque>
using namespace std;
void showdq(deque <int> dq)
{
    int l = dq.size();
    for (int i = 0; i < l; i++)
    {
        cout << dq.front() << " ";
        dq.pop_front();
    }
    cout << '\n';
}
int main()
{
    deque <int> dq;
    dq.push_back(10);
    dq.push_front(20);
    dq.push_back(30);
    dq.push_front(40);
    cout << "The elements in dq are : ";
    showdq(dq);
    cout << "\ndq.size() : " << dq.size();
    cout << "\ndq.max_size() : " << dq.max_size();
    cout << "\ndq.at(2) : " << dq.at(2);
    cout << "\ndq.front() : " <<dq.front();
```

```
27.  cout << "\ndq.back() : " << dq.back();
28.  cout << "\nAfter dq.pop_front() : \n";
29.  dq.pop_front();
30.  showdq(dq);
31.  cout << "\nAfter qz.pop_back() : \n";
32.  dq.pop_back();
33.  showdq(dq);
34.  return 0;
35.}
```

stdout

```
The elements in dq are : 40 20 10 30

dq.size() : 4
dq.max_size() : 4611686018427387903
dq.at(2) : 10
dq.front() : 40
dq.back() : 30
After dq.pop_front() :
20 10 30

After qz.pop_back() :
20 10
```

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);
    }
    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.}
```

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.

Card rotation

https://practice.geeksforgeeks.org/problems/card-rotation/0

Chinky and diamonds

https://practice.geeksforgeeks.org/problems/chinky-and-diamonds/0

First negative no. In every window of size k(sliding window)

https://practice.geeksforgeeks.org/problems/first-negative-integer-in-every-window-of-size-k/0

Generate binary no.

https://practice.geeksforgeeks.org/problems/generate-binary-numbers/0

Reverse first k element of queue

https://practice.geeksforgeeks.org/problems/reverse-first-k-elements-of-queue/1