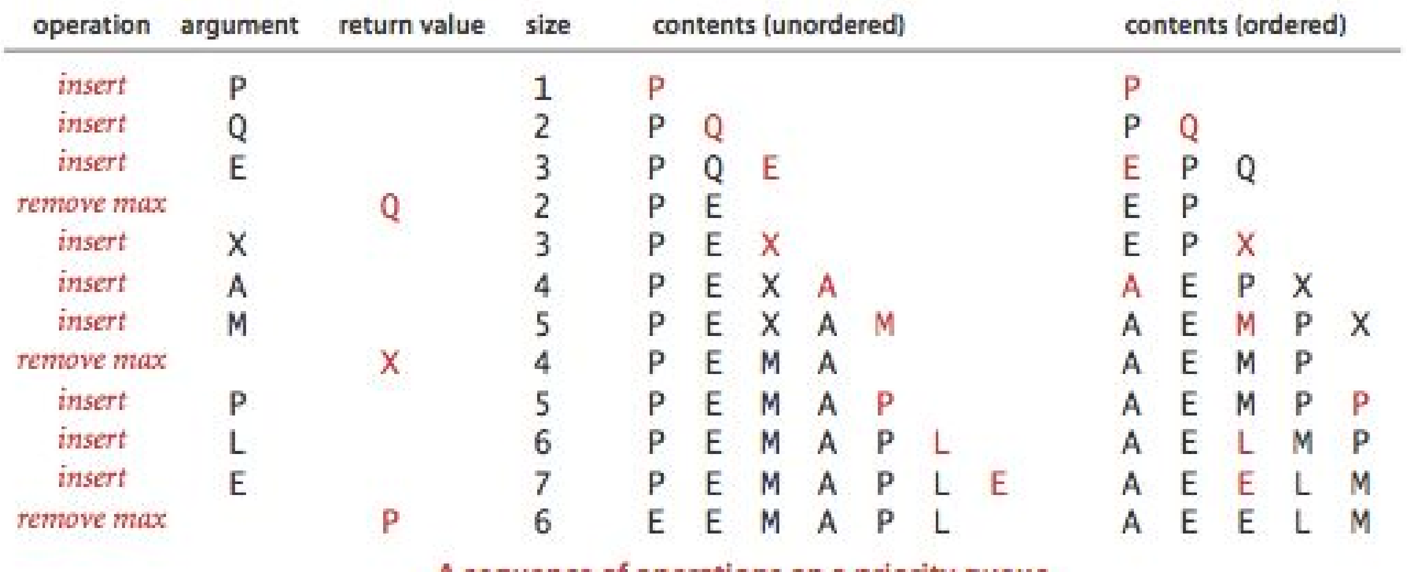
GeeksMan
Queue Data Structure
Lesson 2

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 :

```cpp
#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)
  {
      cout << "\nQueue overflow no more elements can be inserted";
    return;
  }
  if ((front == -1) && (rear == -1))
  {
    front++;
    rear++;
    q[rear] = data;
    return;
  }
  else
    check(data);
     rear++;
}

void dq()
{
  if (front == -1 && rear == -1)
  {
      cout << "Underflow\n";
  }
  else if (front == rear)
  {
    front = -1;
    rear = -1;
  }
  else
  {
        front++;
    }
}
void display()
{
    if (front == rear && rear == -1)
    {
          cout << "Queue is empty\n" ;
    }
    else
    {
        int i;
        for (i = front ; i<=rear ;i++)
        {
            cout << q[i] << " ";
        }
    }
}
void peek()
{
    if (front == -1 && rear == -1)
    {
          cout << "Queue is empty\n";
    }
    else
    {
        cout << "Front = " << q[front];
    }
}
int main()
```

```
{
    while(1)
    {
    int a;
        cout << "Choose any option:\n";
        cout << "Enter 1 to enqueue\n";
        cout << "Enter 2 to deque\n";
        cout << "Enter 3 to display\n";
        cout << "Enter 4 to get the front element\n";
        cout << "Enter 5 to exit\n";
    cin >> a;
        switch(a)
        {
        case 1:
            {
                int x;
                cout << "Enter the element you want to enqueue: ";
                cin >> x;
                enqueue(x);
                break;
            }
        case 2:
        {
                dq();
                break;
            }
        case 3:
            {
                display();
                break;
            }
        case 4:
            {
                peek();
                break;
            }

        case 5:
            {
                exit(1);
            }
        }
        }

    }
```

Priority Queue in STL

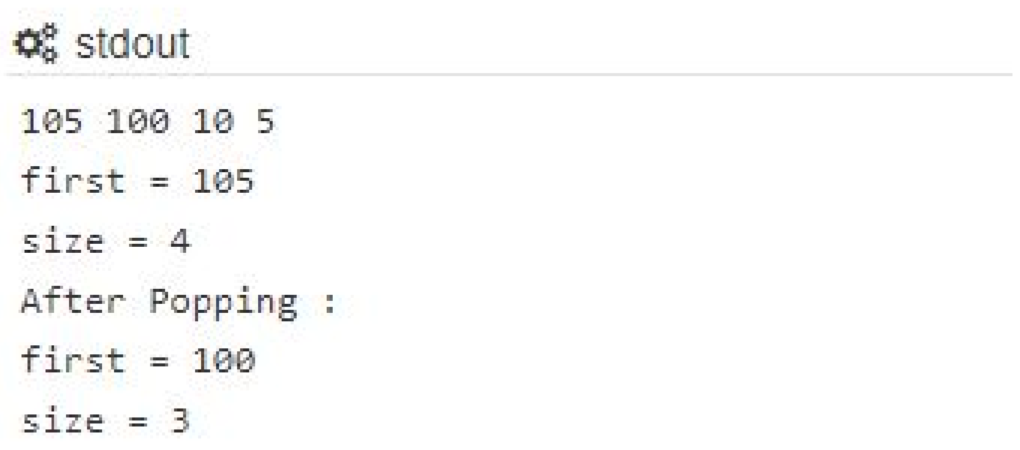
- **empty()** function returns whether the queue is empty.
- **size()** function returns the size of the queue.
- **top()** Returns a reference to the top most element of the queue
- **push(g)** function adds the element 'g' at the end of the queue.
- **pop()** function deletes the first element of the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

Example :

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   priority_queue<int>pq;
   pq.push(10);
   pq.push(100);
   pq.push(5);
   pq.push(105);
   priority_queue<int>q = pq;
   while(q.size() != 0)
   {
      cout << q.top() <<" ";
      q.pop();
   }
   cout << "\nfirst = " << pq.top();
   cout << "\nsize = " << pq.size();
```

```cpp
    pq.pop();
    cout << "\nAfter Popping : ";
    cout << "\nfirst = " << pq.top();
    cout << "\nsize = " << pq.size();

}
```



stdout

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

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 ≤ **T** ≤ 10

1 ≤ **N** ≤ 105

0 ≤ **K** ≤ 105

0 ≤ ***Ai*** ≤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<iostream>
#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;
}
```

Generate Binary Numbers

Given a number **N**. The task is to generate and print all binary numbers with decimal values from **1 to N**.

Input : 2 **Output:** 1 10

Input : 5 **Output:** 1 10 11 100 101

ALGORITHM :

1) Create an empty queue.

2) Enqueue the first binary number "1" to queue.

3) Now run a loop upto n for generating binary numbers.

- Dequeue and Print the front of the queue.
- Append "0" at the end of the front item and enqueue it.
- Append "1" at the end of the front item and enqueue it.

SOLUTION :

```cpp
#include <bits/stdc++.h>
using namespace std;
void fun(int n)
{
   queue<string>q;
   q.push("1");
   while(q.size() != n)
   {
       q.push(q.front() + '0');
       q.push(q.front() + '1');
       cout << q.front() << " ";
       q.pop();
   }
   if(q.size() != 0)
   {
       cout << q.front();
       q.pop();
   }
}
int main()
{int t,n;
   cin >> t;
   while(t--)
   {
       cin >> n;
       fun(n);
       cout <<endl;
   }
        return 0;}
```

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.

SOLUTION :

```
#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;

    while (end != start || bal < 0)
    {
        while (bal < 0 && start != end)
        {
            bal -= arr[start].petrol - arr[start].distance;
            start = (start + 1) % n;

            if (start == 0)
            return -1;
        }
        bal += arr[end].petrol - arr[end].distance;

        end = (end + 1) % n;
    }
    return start;
    }
```

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.

```
for(int i = 1 ; i <= n ; i++)
    {
        int j = i;
        while(j--)
        {
                    q.push(q.front());
            q.pop();
        }

        a[q.front()] = i;
        q.pop();
    }
```