**############################################################**

**1. Generate Binary Numbers**

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

**Example 1:**

**Input:**

N = 2

**Output:**

1 10

**Explanation:**

Binary numbers from

1 to 2 are 1 and 10.

**Example 2:**

**Input:**

N = 5

**Output:**

1 10 11 100 101

**Explanation:**

Binary numbers from

1 to 5 are 1 , 10 , 11 , 100 and 101.

**Your Task:**  
You only need to complete the **function**generate() that takes **N**as **parameter**and returns vector of strings denoting binary numbers.

**Expected Time Complexity :** O(N log2N)  
**Expected Auxilliary Space :**O(N log2N)

**Constraints:**  
1 ≤ N ≤ 106

vector<string> generate(int N)

{

queue<string> q;

vector<string> v;

int i = 1;

q.push("1");

while(i <= N)

{

string s = q.front();

v.push\_back(s);

q.pop();

q.push(s + "0");

q.push(s + "1");

i++;

}

return v;

}

**############################################################**

**2. Queue Push & Pop**

Given an array **arr[]**of size **N**, enqueue the elements of the array into a queue and then dequeue them.

**Example 1:**

**Input:**

N = 5

arr[] = 1 2 3 4 5

**Output:**

1 2 3 4 5

**Example 2:**

**Input:**

N = 7

arr[] = 1 6 43 1 2 0 5

**Output:**

1 6 43 1 2 0 5

**Your Task:**  
You don't need to read any input. Your task is to complete the functions **push()** and **\_pop()**. The function **push()** takes the array and its size as the input parameters and returns the queue formed, and the function **\_pop()**, takes the queue as the input parameter and prints the elements of the queue.

**Expected time complexity:** O(n)

**Expected space complexity:** O(n)

queue<int>\_push(int arr[],int n)

{ int i = 0;

queue<int> q;

while(i < n)

{ q.push(arr[i]);

i++;

}

return q;

}

void \_pop(queue<int>s)

{ while(!s.empty())

{ cout<<s.front()<<" ";

s.pop();

}

}

**############################################################**

**3. Stack using two queues**

Implement a Stack using two queues**q1** and**q2**.

**Example 1:**

**Input:**

push(2)

push(3)

pop()

push(4)

pop()

**Output:** 3 4

**Explanation:**

push(2) the stack will be {2}

push(3) the stack will be {2 3}

pop() poped element will be 3 the

  stack will be {2}

push(4) the stack will be {2 4}

pop()   poped element will be 4

**Example 2:**

**Input:**

push(2)

pop()

pop()

push(3)

**Output:** 2 -1

**Your Task:**

Since this is a function problem, you don't need to take inputs. You are required to complete the two methods **push()** which takes an integer **'x'** as input denoting the element to be pushed into the stack and **pop()** which returns the integer poped out from the stack(**-1** if the stack is empty).

**Expected Time Complexity:**O(1) for **push()**and O(N) for **pop()**(or vice-versa).  
**Expected Auxiliary Space:**O(1) for both **push()**and **pop()**.

**Constraints:**  
1 <=Number of queries <= 100  
1 <= values of the stack <= 100

/Function to push an element into stack using two queues.

void QueueStack :: push(int x)

{ q1.push(x);

}

//Function to pop an element from stack using two queues.

int QueueStack :: pop()

{

if(q1.empty())

return -1;

while(q1.size() > 1)

{

int x = q1.front();

q1.pop();

q2.push(x);

}

int data = q1.front();

q1.pop();

while(!q2.empty())

{

int x = q2.front();

q1.push(x);

q2.pop();

}

return data;

}

**############################################################**

**4. Queue using circular array**

Given **size** of a queue and **Q** query. The task is to perform operations according to the type of query. Queries can be of following types:

1)**1 element**: This means **push**the **element**into the queue (allowed only when queue is not full).  
2)**2**: This means **pop**the **element**at front from the queue (allowed only when queue is not empty).

**Input:**  
First line of input contains size of queue and Q queries. Next Q lines contains queries of any of the two types as given above.

**Output:**  
For each query, the task is to push and pop element and print "**1**" (without quotes) if operations succeeds, else print "**-1**" (without quotes).

**Constraints:**  
1 <= size <= 104  
1 <= Q <= 103

**User Task:**  
The task is to complete the functions **push**() and **pop**() which does push and pop according to the queue.

**Example:  
Input:**  
2  
4 6  
1 1  
1 1  
1 1  
1 1  
1 1  
2  
4 5  
2  
2  
1 1  
1 2  
1 3

**Output:**  
1  
1  
1  
1  
-1  
1  
-1  
-1  
1  
1  
1

**Explanation:**  
**Testcase 1:** Upto query number 4, queue is having space to push elements, so ouput is 1 till this query. After this, since queue is full and you want to push element, so output is -1.

**Testcase 2:** When queue is empty, and you want to pop out element, output is -1.

// element: element to be pushed into the queue

int Queue::push(int element){

int size = getSize();

// check for condition when queue is full

if(isFull){

return -1;

}

// do operation when queue is not full.

// Also, keep in mind of marking queue as

// full when rear becomes equal to front

else{

arr[rear] = element;

if((rear + 1) % size == front % size)

isFull = true;

rear++;

}

return 1;

}

// Function to pop elements from Queue

int Queue::pop(){

int size = getSize();

// check for the condition when queue is empty

if(isEmpty()){

return -1;

}

// If queue is not empty, do the required operation.

// Also, keep in mind to unmark the flag which represents

// queue is full or not

else {

if(front == rear)

front = rear = 0;

else if(front == size - 1)

front = 0;

else

front++;

}

return 1;

}

**############################################################**

**5. Maximium Sum**

You are Given an array **A[]** of  **N** Integers and an integer K. You have to choose one element one by one K   
times. You can only pick the element whose frequency is maximum. The element will be removed from the array after being chosen. If there are many elements with the maximum frequency then you can choose any. Your aim is to maximize the sum.

**Your Task:**

You don't need to read input or print anything. Your task is to complete the function**maximum\_sum()** which takes the N (number of elements in Array A) ,array A[] and K as input parameters and returns the maximum sum by choosing K elements.

**Example 1:**

**Input:**

N=5

K=3

A[]={1, 1, 2, 3, 3}

**Output:**

7

**Explanation:**

3+1+3=7

**Expected Time Complexity:** O(N\*LogN)  
**Expected Auxiliary Space:** O(N)

**Constraints:**  
1<=K<=N<=10^5  
1<=A[i]<=10^5

long maximum\_sum(int n, vector<int> arr, int k)

{

unordered\_map<int, int> mp;

for(int i = 0; i < arr.size(); i++)

mp[arr[i]]++;

priority\_queue<pair<int, int>> q;

for(auto i : mp)

q.push({i.second, i.first});

long sum = 0;

pair<int,int> p;

while(k--)

{ p = q.top();

q.pop();

sum += p.second;

q.push({--p.first, p.second});

/\*

priority\_queue<pair<int, int>> a = q;

while(!a.empty())

{ pair<int, int> t = a.top();

cout<<t.first<<" "<<t.second<<" ";

a.pop();

}

cout<<endl;

\*/

}

return sum;

}

**############################################################**

**6. Reverse First K elements of Queue**

Given an integer **K**and a [queue](http://www.geeksforgeeks.org/queue-data-structure/) of integers, we need to reverse the order of the first**K** elements of the queue, leaving the other elements in the same relative order.

Only following standard operations are allowed on queue.

* enqueue(x) : Add an item x to rear of queue
* dequeue() : Remove an item from front of queue
* size() : Returns number of elements in queue.
* front() : Finds front item.

**Example 1:**

**Input:**

5 3

1 2 3 4 5

**Output:**

3 2 1 4 5

**Explanation:**

After reversing the given

input from the 3rd position the resultant

output will be 3 2 1 4 5.

**Example 2:**

**Input:**

4 4

4 3 2 1

**Output:**

1 2 3 4

**Explanation:**

After reversing the given

input from the 4th position the resultant

output will be 1 2 3 4.

**Your Task:**  
**Complete**the **provided function** **modifyQueue**that takes **queue and k** as **parameters**and **returns**a **modified**queue. The **printing**is done **automatically**by the **driver code**.

**Expected TIme Complexity** : O(n)  
**Expected Auxilliary Space**: O(n)

**Constraints:**  
1 <= N <= 1000  
1 <= K <= N

**Note:**The **Input/Ouput** format and **Example** given are used for system's internal purpose, and should be used by a user for **Expected Output** only. As it is a function problem, hence a user should not read any input from stdin/console. The task is to complete the function specified, and not to write the full code.

queue<int> modifyQueue(queue<int> q, int k)

{

stack<int> stk;

queue<int> q2;

while(k--)

{ stk.push(q.front());

q.pop();

}

while(!q.empty())

{ q2.push(q.front());

q.pop();

}

while(!stk.empty())

{ q.push(stk.top());

stk.pop();

}

while(!q2.empty())

{ q.push(q2.front());

q2.pop();

}

return q;

}

**############################################################**

**7. Queue using two Stacks**

Implement a Queue using 2 stacks**s1** and**s2** .  
A Query **Q** is of 2 Types  
**(i)** 1 x (a query of this type means  pushing **'x'** into the queue)  
**(ii)** 2   (a query of this type means to pop element from queue and print the poped element)

**Example 1:**

**Input:**

5

1 2 1 3 2 1 4 2

**Output:**

2 3

**Explanation:**

In the first testcase

1 2 the queue will be {2}

1 3 the queue will be {2 3}

2   poped element will be 2 the queue

  will be {3}

1 4 the queue will be {3 4}

2   poped element will be 3.

**Example 2:**

**Input:**

4

1 2 2 2 1 4

**Output:**

2 -1

**Explanation:**

In the second testcase

1 2 the queue will be {2}

2   poped element will be 2 and

  then the queue will be empty

2   the queue is empty and hence -1

1 4 the queue will be {4}.

**Your Task:**  
You are required to complete the two methods **push** which take one argument an integer **'x'** to be pushed into the queue and **pop** which returns a integer poped out from other queue(-1 if the queue is empty). The **printing** is done **automatically**by the**driver code**.

**Expected Time Complexity** : O(1) for **push()**and O(N) for **pop()**or O(N) for **push()**and O(1) for **pop()**   
**Expected Auxilliary Space**: O(1).

**Constraints:**  
1 <=Q <= 100  
1 <= x <= 100

Note:The **Input/Ouput** format and **Example** given are used for system's internal purpose, and should be used by a user for **Expected Output** only. As it is a function problem, hence a user should not read any input from stdin/console. The task is to complete the function specified, and not to write the full code.

//Function to push an element in queue by using 2 stacks.

void StackQueue :: push(int x)

{ s1.push(x);

}

//Function to pop an element from queue by using 2 stacks.

int StackQueue :: pop()

{ if(s1.empty() == true)

return -1;

while(s1.size() > 1)

{ s2.push(s1.top());

s1.pop();

}

int data = s1.top();

s1.pop();

while(!s2.empty())

{ s1.push(s2.top());

s2.pop();

}

return data;

}

**############################################################**

**8. Maximum of all subarrays of size k**

Given an array arr[] of size N and an integer K. Find the maximum for each and every contiguous subarray of size K.

**Example 1:**

**Input:**

N = 9, K = 3

arr[] = 1 2 3 1 4 5 2 3 6

**Output:**

3 3 4 5 5 5 6

**Explanation:**

1st contiguous subarray = {1 2 3} Max = 3

2nd contiguous subarray = {2 3 1} Max = 3

3rd contiguous subarray = {3 1 4} Max = 4

4th contiguous subarray = {1 4 5} Max = 5

5th contiguous subarray = {4 5 2} Max = 5

6th contiguous subarray = {5 2 3} Max = 5

7th contiguous subarray = {2 3 6} Max = 6

**Example 2:**

**Input:**

N = 10, K = 4

arr[] = 8 5 10 7 9 4 15 12 90 13

**Output:**

10 10 10 15 15 90 90

**Explanation:**

1st contiguous subarray = {8 5 10 7}, Max = 10

2nd contiguous subarray = {5 10 7 9}, Max = 10

3rd contiguous subarray = {10 7 9 4}, Max = 10

4th contiguous subarray = {7 9 4 15}, Max = 15

5th contiguous subarray = {9 4 15 12},

Max = 15

6th contiguous subarray = {4 15 12 90},

Max = 90

7th contiguous subarray = {15 12 90 13},

Max = 90

**Your Task:**  
You dont need to read input or print anything. Complete the function **max\_of\_subarrays()** which takes the array, N and K as input parameters and returns a list of integers denoting the maximum of every contiguous subarray of size K.

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

**Constraints:**  
1 ≤ N ≤ 107  
1 ≤ K ≤ N  
0 ≤ arr[i] <= 107

vector <int> max\_of\_subarrays(int \*arr, int n, int k)

{

deque<int> dq;

vector<int> ans;

for(int i = 0; i < k; i++)

{ while(!dq.empty() && arr[i] > arr[dq.back()])

dq.pop\_back();

dq.push\_back(i);

}

ans.push\_back(arr[dq.front()]);

for(int i = k; i < n; i++)

{ if(dq.front() == i - k)

dq.pop\_front();

while(!dq.empty() && arr[i] > arr[dq.back()])

dq.pop\_back();

dq.push\_back(i);

ans.push\_back(arr[dq.front()]);

}

return ans;

}

**############################################################**

**9. Steps by Knight**

Given a square chessboard, the initial position of Knight and position of a target. Find out the minimum steps a Knight will take to reach the target position.

**Note:**  
The initial and the target position co-ordinates of Knight have been given accoring to 1-base indexing.

**Example 1:**

**Input:**

N=6

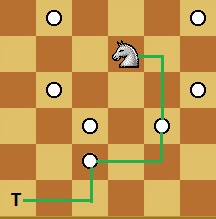
knightPos[ ] = {4, 5}

targetPos[ ] = {1, 1}

**Output:**

3

**Explanation:**



Knight takes 3 step to reach from

(4, 5) to (1, 1):

(4, 5) -> (5, 3) -> (3, 2) -> (1, 1).

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **minStepToReachTarget()** which takes the inital position of Knight (KnightPos), the target position of Knight (TargetPos) and the size of the chess board (N) as an input parameters and returns the minimum number of steps required by the knight to reach from its current position to the given target position.

**Expected Time Complexity:** O(N2).  
**Expected Auxiliary Space:** O(N2).

**Constraints:**  
1 <= N <= 1000  
1 <= Knight\_pos(X, Y), Targer\_pos(X, Y) <= N

int minStepToReachTarget(vector<int>&start, vector<int>&end,int n)

{

if(start[0] == end[0] && start[1] == end[1])

return 0;

int board[n][n];

for(int i = 0; i < n; i++)

for(int j = 0; j < n; j++)

board[i][j] = 0;

queue<pair<int, int>> q;

q.push(make\_pair(start[0] - 1, start[1] - 1));

while(!q.empty())

{ pair<int, int> cur = q.front();

int x = cur.first;

int y = cur.second;

q.pop();

if((x-1 >= 0 && x-1 < n) && (y-2 >= 0 && y-2 < n) && (board[x-1][y-2] == 0))

{ q.push(make\_pair(x-1, y-2));

board[x-1][y-2] = board[x][y] + 1;

}

if((x+1 >= 0 && x+1 < n) && (y-2 >= 0 && y-2 < n) && (board[x+1][y-2] == 0))

{ q.push(make\_pair(x+1, y-2));

board[x+1][y-2] = board[x][y] + 1;

}

if((x-2 >= 0 && x-2 < n) && (y-1 >= 0 && y-1 < n) && (board[x-2][y-1] == 0))

{ q.push(make\_pair(x-2, y-1));

board[x-2][y-1] = board[x][y] + 1;

}

if((x+2 >= 0 && x+2 < n) && (y-1 >= 0 && y-1 < n) && (board[x+2][y-1] == 0))

{ q.push(make\_pair(x+2, y-1));

board[x+2][y-1] = board[x][y] + 1;

}

if((x-2 >= 0 && x-2 < n) && (y+1 >= 0 && y+1 < n) && (board[x-2][y+1] == 0))

{ q.push(make\_pair(x-2, y+1));

board[x-2][y+1] = board[x][y] + 1;

}

if((x+2 >= 0 && x+2 < n) && (y+1 >= 0 && y+1 < n) && (board[x+2][y+1] == 0))

{ q.push(make\_pair(x+2, y+1));

board[x+2][y+1] = board[x][y] + 1;

}

if((x-1 >= 0 && x-1 < n) && (y+2 >= 0 && y+2 < n) && (board[x-1][y+2] == 0))

{ q.push(make\_pair(x-1, y+2));

board[x-1][y+2] = board[x][y] + 1;

}

if((x+1 >= 0 && x+1 < n) && (y+2 >= 0 && y+2 < n) && (board[x+1][y+2] == 0))

{ q.push(make\_pair(x+1, y+2));

board[x+1][y+2] = board[x][y] + 1;

}

}

return board[end[0]-1][end[1]-1];

}

**############################################################**