```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;
public class Main {

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        String[] vals = br.readLine().split(" ");
        int n = Integer.parseInt(vals[0]);
        int m = Integer.parseInt(vals[1]);
    }
}
```

1. Minimum stack

Find a smallest element of the stack in O(1)

```
Stack
        stack<pair<int, int>> st;
Adding an element
        int new_min = st.empty( ) ? new_elem : min(new_elem, st.top( ).second);
        st.push({new_elem, new_min});
Removing an element
        int removed_element = st.top( ).first;
        st.pop( )
Finding the minimum
        int minimum = st.top( ).second

Queue (method 1) (non-decreasing)
        deque<int> q;
Adding an element
        if ( !q.empty( ) && q.front( ) == q.back( ) > new_element)
                q.pop_back( );
        q.push_back(new_element);
Removing an element
        if (!q.empty( ) && q.front( ) == remove_element)
                q.pop_front( );

Queue (method 2)
        deque<Pair<int, int>> q;
        int cnt_added = 0;
        int cnt_removed = 0;
Adding an element
        while ( !q.empty() && q.back().first > new_element
                q.pop_back( )
        q.push_back({new_elemnt, cnt_added});
        cnt_added++;
Removing an element
        if (!q.empty( ) && q.front( ).second == cnt_removed)
                q.pop_front( );
        cnt_removed++;
```

Queue (2 stacks)
stack<pair<int, int>> s1, s2;
Adding an element
```
int prevMin = s1.empty( ) ? new_element : min(new_element, s1.top( ).second);
s1.push({new_element, prevMin});
```
Removing an element
```
if (s2.empty( ) {
    while (!s1.empty( )) {
        int element = s1.top( ).first;
        s1.pop( );
        int minimum = s2.empty( ) ? element : min(element, s2.top( ).second);
        s2.push({element, minimum}P;
    }
}
int remvoe_element = s2.top( ).first;
s2.pop( );
```
Finding the minimum:
```
if (s1.empty( ) || s2.empty( ))
    minimum = s1.emtpy( ) ? s2.top( ).second : s1.top( ).second;
else
    minimum = min(s1.top( ).second, s2.top( ).second);
```

2. Sparse Table
answer most range queries in O(logn)
answer range minimum (maximum) queries in O(1) time
*array has to be immutable

create:
```
long long st[k+1][MAXN]; // k is 25, MAXN arr len
std::copy(array.begin(), array.end(), st[0]);

for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j++)
        st[i][j] = st[i-1][j] + st[i-1][j + (1 << (i-1))];
```

Range sum query O(K) = O(log MAXN)
```
long long sum = 0;
for (int i = K; i >= 0; i—)
    if ((1 << i) <= R-L + 1)
        sum += st[i][L];
        L += 1 << i;
```
Range Minimum Queries (RMQ)
min(L, R) = min(st[i][L], st[i][R - 2^i + 1]), where i = log(R - L + 1)
precompute log
```
int lg[MAXN+1];
lg[1] = 0;
for (int i = 2; i <= MAXN; i++)
    lg[i] = lg[i/2] + 1;
```
minimum:
```
int i = lg[R - L + 1];
int minimum = min(st[i][L], st[i][R - (1 << i) + 1]);
```

3. Disjoint Set Union

make_set(v): creates a new set of v
union(a, b): merge set containing a, and set containing b
find_set(v): find leader of set containing v
(all are nearly constant time)

optimization: path compression, union by rank/size

```
make_set(int v) {
        parent[v] = v;
        size[v] = 1;
}

int find_set(int v) {
        if (v == parent[v])
                return v;
        return parent[v] = find_set(parent[v]);
}

void union_sets(int a, int b) {
        a = find_set(a);
        b = find_set(b);
        if (a != b)
                if (size[a] < size[b])
                        swap(a, b)
                parent[b] = a;
                size[a] += size[b];
}
```

4. Fenwick Tree / Bit Indexed Tree
logn update & logn range query
O(N) memory

```
public class BIT {
   private int[] arr;

   public BIT(int n) {
      arr = new int[n+1];
      Arrays.fill(arr, 0);
   }

   public BIT(int[] a) {
      arr = new int[a.length + 1];
      for (int i = 0; i < a.length; i++) {
         int idx = i+1;
         arr[idx] += a[i];
         int iidx = idx + lsb(idx);
         if (iidx < arr.length) {
            arr[iidx] += arr[idx];
         }
      }
   }

   public int preSum(int i) {
```

```
        int sum = 0;
        while (i > 0) {
            sum += arr[i];
            i = i - lsb(i);
        }

        return sum;

    }

    public void add(int i, int v) {
        while (i < arr.length) {
            arr[i] += v;
            i = i + lsb(i);
        }
    }

    private int lsb(int n) {
        return n & -n;
    }
}
```

5. Sqrt Decomposition

O(logn) extra space
O(logn) time

```
// input data
int n;
vector<int> a (n);

// preprocessing
int len = (int) sqrt (n + .0) + 1; // size of the block and the number of blocks
vector<int> b (len);
for (int i=0; i<n; ++i)
    b[i / len] += a[i];

// answering the queries
for (;;) {
    int l, r;
  // read input data for the next query
    int sum = 0;
    for (int i=l; i<=r; )
        if (i % len == 0 && i + len - 1 <= r) {
            // if the whole block starting at i belongs to [l, r]
            sum += b[i / len];
            i += len;
        }
        else {
            sum += a[i];
            ++i;
        }
}
```

6. Segment Tree

```
int n, t[4*MAXN];

void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = t[v*2] + t[v*2+1];
    }
}

int sum(int v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && r == tr) {
        return t[v];
    }
    int tm = (tl + tr) / 2;
    return sum(v*2, tl, tm, l, min(r, tm))
           + sum(v*2+1, tm+1, tr, max(l, tm+1), r);
}

void update(int v, int tl, int tr, int pos, int new_val) {
    if (tl == tr) {
        t[v] = new_val;
    } else {
        int tm = (tl + tr) / 2;
        if (pos <= tm)
            update(v*2, tl, tm, pos, new_val);
        else
            update(v*2+1, tm+1, tr, pos, new_val);
        t[v] = t[v*2] + t[v*2+1];
    }
}

// Lazy propagation
void build(int a[], int v, int tl, int tr) {
    if (tl == tr) {
        t[v] = a[tl];
    } else {
        int tm = (tl + tr) / 2;
        build(a, v*2, tl, tm);
        build(a, v*2+1, tm+1, tr);
        t[v] = 0;
    }
}

void update(int v, int tl, int tr, int l, int r, int add) {
    if (l > r)
        return;
    if (l == tl && r == tr) {
```

```
        t[v] += add;
    } else {
        int tm = (tl + tr) / 2;
        update(v*2, tl, tm, l, min(r, tm), add);
        update(v*2+1, tm+1, tr, max(l, tm+1), r, add);
    }
}

int get(int v, int tl, int tr, int pos) {
    if (tl == tr)
        return t[v];
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return t[v] + get(v*2, tl, tm, pos);
    else
        return t[v] + get(v*2+1, tm+1, tr, pos);
}
```