Code : 1  
package Topic\_19\_HashmapAndHeap;

import java.util.HashMap;

import java.util.Scanner;

public class A\_HighestFrequencyCharacter {

public static void main(String[] args) throws Exception {

Scanner scn = new Scanner(System.in);

String str = scn.next();

HashMap<Character, Integer> hm = new HashMap(); //1

for (int i = 0; i < str.length(); i++) {

char ch = str.charAt(i);

if (hm.containsKey(ch)) //2

{

int old = hm.get(ch);

int now = old + 1;

hm.put(ch, now);

} else { //3

hm.put(ch, 1);

}

}

char max = str.charAt(0); //4

for (Character key : hm.keySet()) //5

{

if (hm.get(key) > hm.get(max))

max = key;

}

System.out.println(max); //6

}

}

Code : 2  
package Topic\_19\_HashmapAndHeap;

import java.util.HashMap;

import java.util.Scanner;

public class B\_GetCommonElements1 {

public static void main(String[] args) throws Exception {

Scanner scn = new Scanner(System.in);

int n1 = scn.nextInt();

int[] a1 = new int[n1];

for (int i = 0; i < n1; i++) {

a1[i] = scn.nextInt();

}

int n2 = scn.nextInt();

int[] a2 = new int[n2];

for (int i = 0; i < n2; i++) {

a2[i] = scn.nextInt();

}

HashMap<Integer, Integer> hm = new HashMap();

for (int i = 0; i < n1; i++) {

hm = new HashMap();

}

for (int val : a1) {

if (hm.containsKey(val)) {

int old = hm.get(val);

int now = old + 1;

hm.put(val, now);

} else {

hm.put(val, 1);

}

}

for (int val : a2) {

if (hm.containsKey(val)) {

System.out.println(val);

hm.remove(val);

}

}

}

}

Code : 3  
package Topic\_19\_HashmapAndHeap;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

public class CustomPriotityQueue {

public static class MyPriorityQueue<T> {

ArrayList<T> data;

Comparator cmptr;

public MyPriorityQueue() {

data = new ArrayList<>();

cmptr = null;

}

public MyPriorityQueue(Comparator cmptr) {

data = new ArrayList<>();

this.cmptr = cmptr;

}

public void add(T val) {

data.add(val);

upheapify(data.size() - 1);

}

void upheapify(int i) {

if (i == 0) {

return;

}

int pi = (i - 1) / 2;

if (isSmaller(i, pi) == true) {

swap(pi, i);

upheapify(pi);

}

}

public T remove() {

if (data.size() == 0) {

System.out.println("Underflow");

return null;

}

swap(0, data.size() - 1);

T val = data.remove(data.size() - 1);

downheapify(0);

return val;

}

void downheapify(int i) {

int mini = i;

int lci = 2 \* i + 1;

if (lci < data.size() && isSmaller(lci, mini) == true) {

mini = lci;

}

int rci = 2 \* i + 2;

if (rci < data.size() && isSmaller(rci, mini) == true) {

mini = rci;

}

if (mini != i) {

swap(i, mini);

downheapify(mini);

}

}

public T peek() {

// write your code here

if (data.size() == 0) {

System.out.println("Underflow");

return null;

}

return data.get(0);

}

public int size() {

// write your code here

return data.size();

}

void swap(int i, int j) {

T ith = data.get(i);

T jth = data.get(j);

data.set(i, jth);

data.set(j, ith);

}

boolean isSmaller(int i, int j) {

T ith = data.get(i);

T jth = data.get(j);

if (cmptr != null) {

if (cmptr.compare(ith, jth) < 0) {

return true;

} else {

return false;

}

} else {

Comparable cith = (Comparable) ith;

Comparable cjth = (Comparable) jth;

if (cith.compareTo(cjth) < 0) {

return true;

} else {

return false;

}

}

}

}

static class Student implements Comparable<Student> {

String name;

int ht;

int wt;

int marks;

Student(String name, int ht, int wt, int marks) {

this.name = name;

this.ht = ht;

this.wt = wt;

this.marks = marks;

}

public String toString() {

return this.name + "-> " + this.ht + "," + this.wt + ", " + this.marks;

}

public int compareTo(Student other) {

return this.name.compareTo(other.name);

}

}

static class StudentHeightComparator implements Comparator<Student> {

public int compare(Student s1, Student s2) {

return s1.ht - s2.ht;

}

}

static class StudentWeightComparator implements Comparator<Student> {

public int compare(Student s1, Student s2) {

return s1.wt - s2.wt;

}

}

static class StudentMarksComparator implements Comparator<Student> {

public int compare(Student s1, Student s2) {

return s1.marks - s2.marks;

}

}

public static void main(String[] args) throws Exception {

Student[] students = new Student[5];

students[0] = new Student("Amit", 180, 75, 90);

students[1] = new Student("Sumit", 150, 85, 33);

students[2] = new Student("Neha", 185, 72, 99);

students[3] = new Student("Kunal", 165, 65, 75);

students[4] = new Student("Aryan", 177, 55, 88);

MyPriorityQueue<Student> pq = new MyPriorityQueue<>(Collections.reverseOrder());

MyPriorityQueue<Student> pqHt = new MyPriorityQueue<>(Collections.reverseOrder(new StudentHeightComparator()));

MyPriorityQueue<Student> pqWt = new MyPriorityQueue<>(new StudentWeightComparator());

MyPriorityQueue<Student> pqMarks = new MyPriorityQueue<>(new StudentMarksComparator());

for (Student student : students) {

pq.add(student);

pqHt.add(student);

pqWt.add(student);

pqMarks.add(student);

}

System.out.println("On the basis of name");

while (pq.size() > 0) {

Student student = pq.peek();

pq.remove();

System.out.println(student);

}

System.out.println("On the basis of height");

while (pqHt.size() > 0) {

Student student = pqHt.peek();

pqHt.remove();

System.out.println(student);

}

System.out.println("On the basis of weight");

while (pqWt.size() > 0) {

Student student = pqWt.peek();

pqWt.remove();

System.out.println(student);

}

System.out.println("On the basis of marks");

while (pqMarks.size() > 0) {

Student student = pqMarks.peek();

pqMarks.remove();

System.out.println(student);

}

}

}

Code : 4  
package Topic\_19\_HashmapAndHeap;

import java.util.HashMap;

import java.util.Scanner;

public class C\_GetCommonElements2 {

public static void main(String[] args) throws Exception {

Scanner scn = new Scanner(System.in);

int n1 = scn.nextInt();

int[] a1 = new int[n1];

for (int i = 0; i < n1; i++) {

a1[i] = scn.nextInt();

}

int n2 = scn.nextInt();

int[] a2 = new int[n2];

for (int i = 0; i < n2; i++) {

a2[i] = scn.nextInt();

}

HashMap<Integer, Integer> hm = new HashMap();

for (int val : a1) {

if (hm.containsKey(val)) {

int old = hm.get(val);

int now = old + 1;

hm.put(val, now);

} else {

hm.put(val, 1);

}

}

for (int val : a2) {

if (hm.containsKey(val)) {

System.out.println(val);

int old = hm.get(val);

if (old > 1)

hm.put(val, old - 1);

else

hm.remove(val);

}

}

}

}

Code : 5  
package Topic\_19\_HashmapAndHeap;

import java.util.HashMap;

import java.util.Scanner;

public class D\_LongestConsecutiveSequenceOfElements {

public static void main(String[] args) throws Exception {

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

int[] a = new int[n];

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

a[i] = scn.nextInt();

}

HashMap<Integer, Boolean> hm = new HashMap();

for (int val : a) {

hm.put(val, true);

}

for (int val : a) {

if (hm.containsKey(val - 1)) {

hm.put(val, false);

}

}

int mh = 0;

int mval = 0;

for (int val : a) {

if (hm.get(val) == true) {

int lh = 1;

int lval = val;

while (hm.containsKey(lval + lh)) {

lh++;

}

if (lh > mh) {

mh = lh;

mval = val;

}

}

}

for (int i = 0; i < mh; i++) {

System.out.println(mval + i);

}

}

}

Code : 6  
package Topic\_19\_HashmapAndHeap;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.PriorityQueue;

public class E\_KLargestElements {

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

int[] arr = new int[n];

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

arr[i] = Integer.parseInt(br.readLine());

}

int k = Integer.parseInt(br.readLine());

PriorityQueue<Integer> pq = new PriorityQueue<>();

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

if (i < k) {

pq.add(arr[i]);

} else {

if (arr[i] > pq.peek()) {

pq.remove();

pq.add(arr[i]);

}

}

}

while (pq.size() > 0) {

System.out.println(pq.remove());

}

}

}

Code : 7  
package Topic\_19\_HashmapAndHeap;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.PriorityQueue;

public class F\_SortedKArray {

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

int[] arr = new int[n];

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

arr[i] = Integer.parseInt(br.readLine());

}

int k = Integer.parseInt(br.readLine());

// Add first k+1 elements to the Priority Queue

PriorityQueue<Integer> pq = new PriorityQueue<>();

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

pq.add(arr[i]);

}

//Filter out the smallest element and move funnel to the next positions

for (int i = k + 1; i < arr.length; i++) {

System.out.println(pq.remove());

pq.add(arr[i]);

}

//Array is completely traversed, empty the funnel now

while (pq.size() > 0) {

System.out.println(pq.remove());

}

}

}

Code : 8  
package Topic\_19\_HashmapAndHeap;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.Collections;

import java.util.PriorityQueue;

public class G\_MedianPriotityQueue {

public static class MedianPriorityQueue {

PriorityQueue<Integer> left;

PriorityQueue<Integer> right;

public MedianPriorityQueue() {

left = new PriorityQueue<>(Collections.reverseOrder());

right = new PriorityQueue<>();

}

public void add(int val) {

if (right.size() > 0 && val > right.peek()) {

right.add(val);

} else {

left.add(val);

}

handleBalance();

}

private void handleBalance() {

if (left.size() - right.size() == 2) {

right.add(left.remove());

} else if (right.size() - left.size() == 2) {

left.add(right.remove());

}

}

public int remove() {

if (this.size() == 0) {

System.out.println("Underflow");

return -1;

} else if (left.size() >= right.size()) {

return left.remove();

} else {

return right.remove();

}

}

public int peek() {

if (this.size() == 0) {

System.out.println("Underflow");

return -1;

} else if (left.size() >= right.size()) {

return left.peek();

} else {

return right.peek();

}

}

public int size() {

return left.size() + right.size();

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

MedianPriorityQueue qu = new MedianPriorityQueue();

String str = br.readLine();

while (str.equals("quit") == false) {

if (str.startsWith("add")) {

int val = Integer.parseInt(str.split(" ")[1]);

qu.add(val);

} else if (str.startsWith("remove")) {

int val = qu.remove();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("peek")) {

int val = qu.peek();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("size")) {

System.out.println(qu.size());

}

str = br.readLine();

}

}

}

Code : 9  
package Topic\_19\_HashmapAndHeap;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.PriorityQueue;

public class H\_MergeKSortedList {

public static class Pair implements Comparable<Pair> {

int li;

int di;

int data;

Pair(int li, int di, int data) {

this.li = li;

this.di = di;

this.data = data;

}

public int compareTo(Pair o) {

return this.data - o.data;

}

}

public static ArrayList<Integer> mergeKSortedLists(ArrayList<ArrayList<Integer>> lists) {

ArrayList<Integer> rv = new ArrayList<>();

PriorityQueue<Pair> pq = new PriorityQueue<>();

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

Pair p = new Pair(i, 0, lists.get(i).get(0));

pq.add(p);

}

while (pq.size() > 0) {

Pair p = pq.remove();

rv.add(p.data);

p.di++;

if (p.di < lists.get(p.li).size()) {

p.data = lists.get(p.li).get(p.di);

pq.add(p);

}

}

return rv;

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int k = Integer.parseInt(br.readLine());

ArrayList<ArrayList<Integer>> lists = new ArrayList<>();

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

ArrayList<Integer> list = new ArrayList<>();

int n = Integer.parseInt(br.readLine());

String[] elements = br.readLine().split(" ");

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

list.add(Integer.parseInt(elements[j]));

}

lists.add(list);

}

ArrayList<Integer> mlist = mergeKSortedLists(lists);

for (int val : mlist) {

System.out.print(val + " ");

}

System.out.println();

}

}

Code : 10  
package Topic\_19\_HashmapAndHeap;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

public class I\_WritePriorityQueueUsingHeap {

public static class PriorityQueue {

ArrayList<Integer> data;

public PriorityQueue() {

data = new ArrayList<>();

}

public void add(int val) {

data.add(val);

upheapify(data.size() - 1);

}

private void upheapify(int i) {

if (i == 0) {

return;

}

int pi = (i - 1) / 2;

if (data.get(pi) > data.get(i)) {

swap(i, pi);

upheapify(pi);

}

}

private void swap(int i, int j) {

int ith = data.get(i);

int jth = data.get(j);

data.set(i, jth);

data.set(j, ith);

}

public int remove() {

if (this.size() == 0) {

System.out.println("Underflow");

return -1;

}

swap(0, data.size() - 1);

int val = data.remove(data.size() - 1);

downheapify(0);

return val;

}

private void downheapify(int i) {

int mini = i;

int li = 2 \* i + 1;

if (li < data.size() && data.get(li) < data.get(mini)) {

mini = li;

}

int ri = 2 \* i + 2;

if (ri < data.size() && data.get(ri) < data.get(mini)) {

mini = ri;

}

if (mini != i) {

swap(i, mini);

downheapify(mini);

}

}

public int peek() {

if (this.size() == 0) {

System.out.println("Underflow");

return -1;

}

return data.get(0);

}

public int size() {

return data.size();

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

PriorityQueue qu = new PriorityQueue();

String str = br.readLine();

while (str.equals("quit") == false) {

if (str.startsWith("add")) {

int val = Integer.parseInt(str.split(" ")[1]);

qu.add(val);

} else if (str.startsWith("remove")) {

int val = qu.remove();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("peek")) {

int val = qu.peek();

if (val != -1) {

System.out.println(val);

}

} else if (str.startsWith("size")) {

System.out.println(qu.size());

}

str = br.readLine();

}

}

}

Code : 11  
package Topic\_19\_HashmapAndHeap;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.LinkedList;

public class J\_WriteHashmap {

public static class HashMap<K, V> {

private class HMNode {

K key;

V value;

HMNode(K key, V value) {

this.key = key;

this.value = value;

}

}

private int size; // n

private LinkedList<HMNode>[] buckets; // N = buckets.length

public HashMap() {

initbuckets(4);

size = 0;

}

private void initbuckets(int N) {

buckets = new LinkedList[N];

for (int bi = 0; bi < buckets.length; bi++) {

buckets[bi] = new LinkedList<>();

}

}

public void put(K key, V value) throws Exception {

// write your code here

int bi = hashfn(key);

int di = getIndexWithinBucket(key, bi);

if (di != -1) {

HMNode node = buckets[bi].get(di);

node.value = value;

} else {

HMNode node = new HMNode(key, value);

buckets[bi].add(node);

size++;

}

double lambda = size \* 1.0 / buckets.length;

if (lambda > 2.0) {

rehash();

}

}

private void rehash() throws Exception {

LinkedList<HMNode>[] oba = buckets;

initbuckets(oba.length \* 2);

size = 0;

for (int i = 0; i < oba.length; i++) {

for (HMNode node : oba[i]) {

put(node.key, node.value);

}

}

}

private int hashfn(K key) {

int hc = key.hashCode();

return Math.abs(hc) % buckets.length;

}

public V get(K key) throws Exception {

// write your code here

int bi = hashfn(key);

int di = getIndexWithinBucket(key, bi);

if (di != -1) {

HMNode node = buckets[bi].get(di);

return node.value;

} else {

return null;

}

}

public boolean containsKey(K key) {

// write your code here

int bi = hashfn(key);

int di = getIndexWithinBucket(key, bi);

if (di != -1) {

return true;

} else {

return false;

}

}

private int getIndexWithinBucket(K key, int bi) {

int di = 0;

for (HMNode node : buckets[bi]) {

if (node.key.equals(key)) {

return di;

}

di++;

}

return -1;

}

public V remove(K key) throws Exception {

// write your code here

int bi = hashfn(key);

int di = getIndexWithinBucket(key, bi);

if (di != -1) {

HMNode node = buckets[bi].remove(di);

size--;

return node.value;

} else {

return null;

}

}

public ArrayList<K> keyset() throws Exception {

// write your code here

ArrayList<K> keys = new ArrayList<>();

for (int i = 0; i < buckets.length; i++) {

for (HMNode hm : buckets[i]) {

keys.add(hm.key);

}

}

return keys;

}

public int size() {

// write your code here

return size;

}

public void display() {

System.out.println("Display Begins");

for (int bi = 0; bi < buckets.length; bi++) {

System.out.print("Bucket" + bi + " ");

for (HMNode node : buckets[bi]) {

System.out.print(node.key + "@" + node.value + " ");

}

System.out.println(".");

}

System.out.println("Display Ends");

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

HashMap<String, Integer> map = new HashMap();

String str = br.readLine();

while (str.equals("quit") == false) {

if (str.startsWith("put")) {

String[] parts = str.split(" ");

String key = parts[1];

Integer val = Integer.parseInt(parts[2]);

map.put(key, val);

} else if (str.startsWith("get")) {

String[] parts = str.split(" ");

String key = parts[1];

System.out.println(map.get(key));

} else if (str.startsWith("containsKey")) {

String[] parts = str.split(" ");

String key = parts[1];

System.out.println(map.containsKey(key));

} else if (str.startsWith("remove")) {

String[] parts = str.split(" ");

String key = parts[1];

System.out.println(map.remove(key));

} else if (str.startsWith("size")) {

System.out.println(map.size());

} else if (str.startsWith("keyset")) {

System.out.println(map.keyset());

} else if (str.startsWith("display")) {

map.display();

}

str = br.readLine();

}

}

}