

**Министерство образования Республики Беларусь  
Учреждение образования «Полоцкий государственный  
университет имени Евфросинии Полоцкой»**

Кафедра технологий  
программирования

Алгоритмы и структуры данных  
Отчет по лабораторной работе №1

Выполнил

Ланцев Евгений Николаевич.  
21-ИТ-1, ФИТ

Проверил

преподаватель  
Виноградова А.Д.

Полоцк  
2022 г.

## Модуль № 2

### “Числа Фибоначчи”

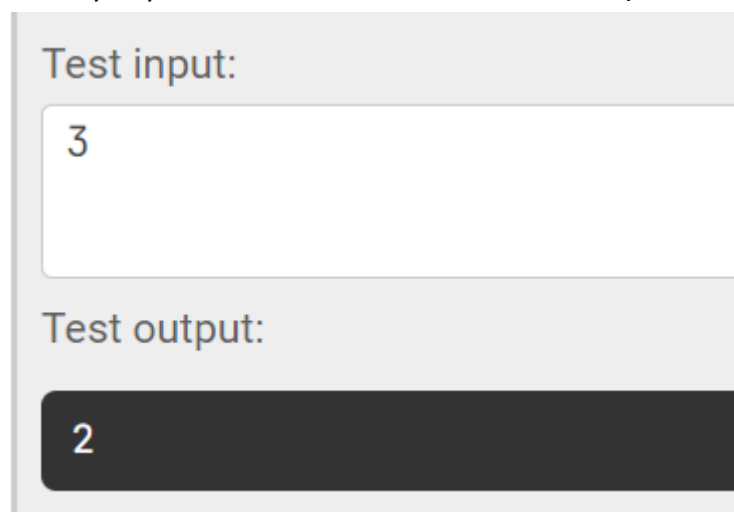
**Задача 1:** Дано целое число  $1 \leq n \leq 40$ , необходимо вычислить  $n$ -е число Фибоначчи.

### Решение

```
using System;
static class Program
{
    private static void Main()
    {
        int n = int.Parse(Console.ReadLine());
        int r = 0;
        int i = 0;
        int k = 1;
        while(i < n)
        {
            int tmp = r;
            r += k;
            k = tmp;
            i++;
        }

        Console.WriteLine(r);
    }
}
```

*Листинг программы- Подсчет  $n$ -ого числа фибоначчи*



The image shows a graphical user interface for a program. It has two main sections: 'Test input:' and 'Test output:'. The 'Test input:' section contains a text box with the number '3'. The 'Test output:' section contains a dark rectangular box with the number '2' in white. The interface is styled with light gray backgrounds and rounded corners.

*Рисунок 1 - Результат работы программы*

**Задача 2:** Дано число  $1 \leq n \leq 10^7$  необходимо найти последнюю цифру  $n$ -го числа Фибоначчи.

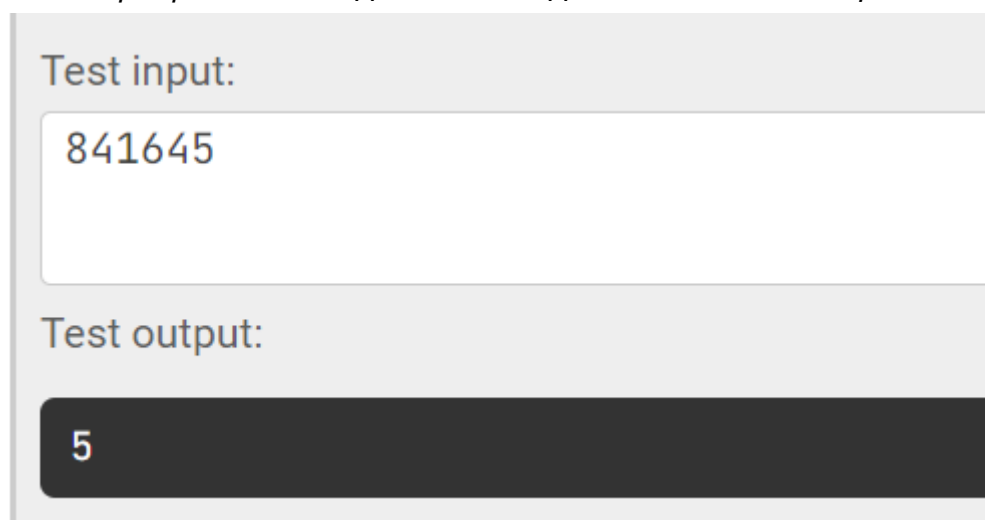
## Решение

```
#include <cassert>
#include <iostream>

class Fibonacci {
public:
    static int get_last_digit(int n) {
        assert(n >= 1);
        // put your code here
        return n;
    }
};

int main(void) {
    int n;
    std::cin >> n;
    std::cout << Fibonacci::get_last_digit(n) << std::endl;
    return 0;
}
```

*Листинг программы - Подсчет последнего числа числа фибоначчи*



The image shows a screenshot of a program execution interface. It has a light gray background. At the top, there is a label "Test input:" in a dark gray font. Below it is a white rectangular input field containing the number "841645". Below the input field is another label "Test output:" in a dark gray font. Below this label is a dark gray rectangular output field containing the number "5".

*Рисунок 2 - Результат работы программы*

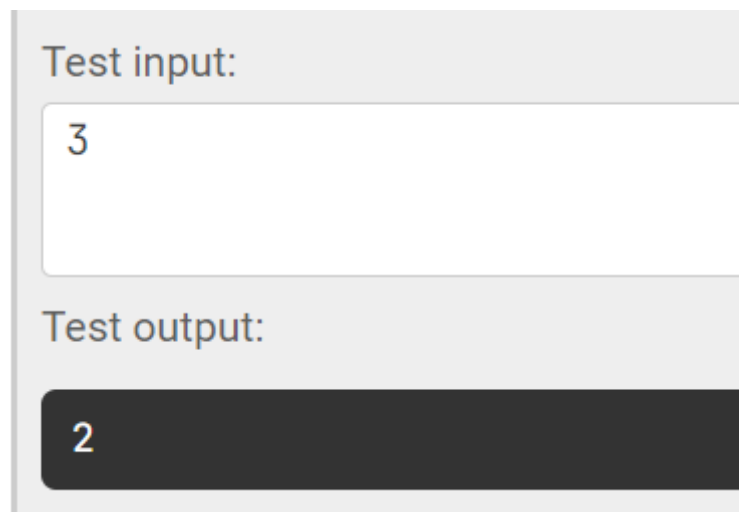
**Задача 3:** По данным двум числам  $1 \leq a, b \leq 2 \cdot 10^9$ , найдите их наибольший общий делитель.

## Решение

```
using System;
static class Program
{
    private static void Main()
    {
        int n = int.Parse(Console.ReadLine());
        int r = 0;
        int i = 0;
        int k = 1;
        while(i < n)
        {
            int tmp = r;
            r += k;
            k = tmp;
            i++;
        }

        Console.WriteLine(r);
    }
}
```

*Листинг программы- Подсчет n-ого числа фибоначчи*



The image shows a graphical user interface for a program. It has a light gray background. At the top, there is a label "Test input:" in a dark gray font. Below it is a white text input field with rounded corners, containing the number "3". Below the input field is another label "Test output:" in a dark gray font. Below this label is a dark gray rectangular box with rounded corners, containing the number "2" in a white font.

*Рисунок 1 - Результат работы программы*

**Задача 4:** По данным двум числам  $1 \leq a, b \leq 2 \cdot 10^2$  найдите их наибольший общий делитель.

## Решение

```
def gcd(a, b):  
    if a % b == 0 or b % a == 0:  
        return min(a, b)  
    else:  
        if a >= b:  
            return gcd(a % b, b)  
        else:  
            return gcd(a, b % a)  
  
def main():  
    a, b = map(int, input().split())  
    print(gcd(a, b))  
  
if __name__ == "__main__":  
    main()
```

*Листинг программы - Подсчет наибольшего общего делителя*



The image shows a graphical user interface for a program. It has a light gray background. At the top, there is a label "Test input:" in a dark gray font. Below it is a white text input field with rounded corners, containing the text "10 2". Below the input field, there is another label "Test output:" in a dark gray font. Below this label is a dark gray rectangular box with rounded corners, containing the number "2" in a white font.

*Рисунок 3 - Результат работы программы*

## Модуль № 4

### “Жадные алгоритмы”

**Задача 1:** По данным  $n$  отрезкам необходимо найти множество точек минимального размера, для которого каждый из отрезков содержит хотя бы одну из точек. В первой строке дано число  $1 \leq n \leq 100$  отрезков. Каждая из последующих  $n$  строк содержит по два числа, задающих начало и конец отрезка. Выведите оптимальное число  $m$  точек и сами  $m$  точек. Если таких множеств точек несколько, выведите любое из них.

### Решение

```
static class Program
{
    static List<int> Dots(List<Tuple<int, int>> segments)
    {
        List<Tuple<int, int>> res_segments = new List<Tuple<int, int>>();
        while (segments.Count > 0)
        {
            if (segments.Count < 2)
            {
                res_segments.Add(segments.Last());
                segments.RemoveAt(segments.Count - 1);
            }
            else
            {
                Tuple<int, int> a = segments[0], b = segments[1];
                if (b.Item1 <= a.Item2)
                {
                    var left = b.Item1;
                    var right = b.Item2 <= a.Item2 ? b.Item2 : a.Item2;
                    Tuple<int, int> overlapping = new Tuple<int, int>(left,
right);

                    segments = segments.GetRange(2, segments.Count - 2);
                    segments.Insert(0, overlapping);

                }
                else
                {
                    res_segments.Add(segments[0]);
                    segments = segments.GetRange(1, segments.Count - 1);
                }
            }
        }

        List<int> result = new List<int>();
```

```

        foreach (var x in res_segments)
        {
            result.Add(x.Item2);
        }

        return result;
    }

    static void Main()
    {
        List<Tuple<int, int>> segments = new List<Tuple<int, int>>();
        int n = int.Parse(Console.ReadLine());

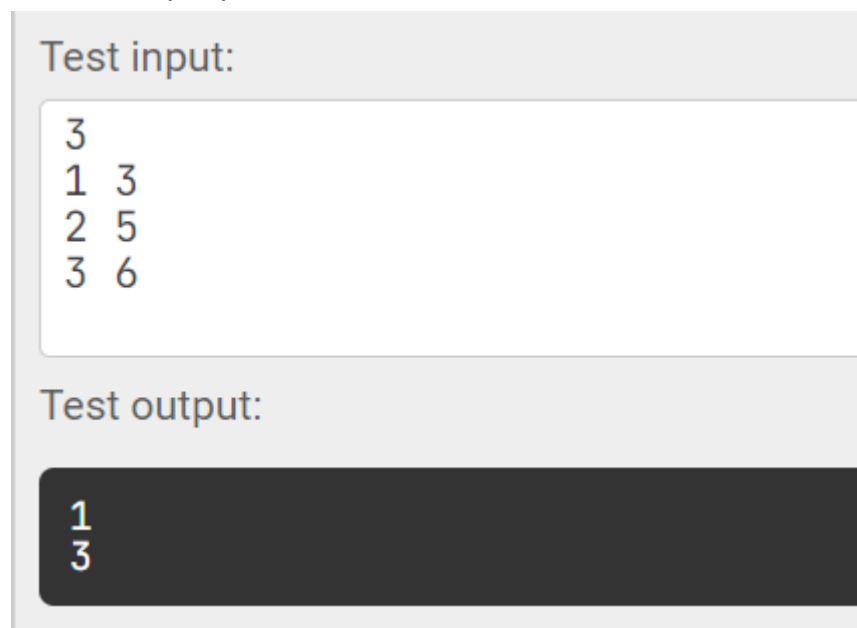
        for (int i = 0; i < n; i++)
        {
            string[] segment = Console.ReadLine().Split(' ');
            segments.Add(new Tuple<int, int>(int.Parse(segment[0]),
int.Parse(segment[1])));
        }

        segments = segments.OrderBy(x => x.Item1).ThenBy(x =>
x.Item2).ToList();
        var result = Dots(segments);

        Console.WriteLine(result.Count);
        result.ForEach(x =>
        {
            Console.Write(x + " ");
        });
    }
}

```

*Листинг программы - Оптимальное количество точек*



*Рисунок 4 - Результат работы программы*





```

        {
            _value += items[0].Value;
            Capacity -= items[0].Volume;
        }
        else
        {
            _value += Capacity * items[0].ValuePerVolume;
            break;
        }
    }
    items = items.GetRange(1, items.Count - 1);
}

return _value;
}

static void Main()
{
    string[] backpackInput = Console.ReadLine().Split(' ');
    List<Item> items = new List<Item>();

    Backpack backpack = new Backpack(backpackInput[1]);
    for (int i = 0; i < int.Parse(backpackInput[0]); i++)
    {
        items.Add(new Item(Console.ReadLine().Split(' ')));
    }

    items = items.OrderByDescending(x => x.ValuePerVolume).ToList();
    Console.WriteLine($"{backpack.CalculateValue(items):F3}");
}
}

```

### Листинг программы- Решение задачи

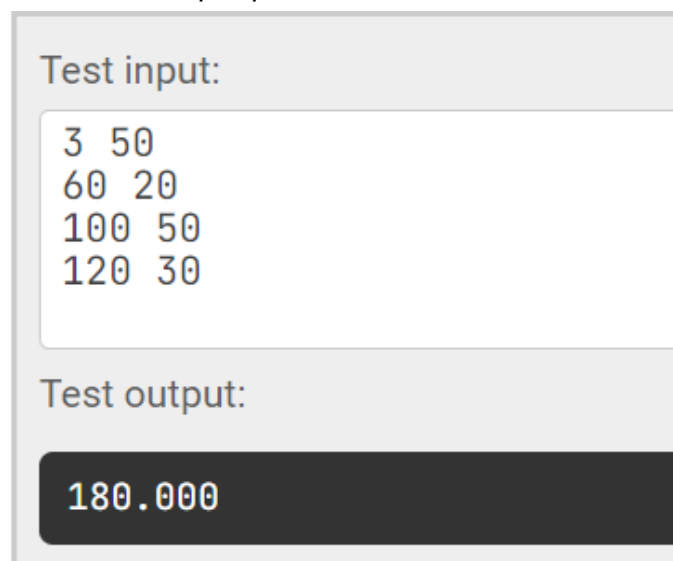


Рисунок 5 - Результат работы программы

**Задача 3:** По данному числу  $1 \leq n \leq 10$  найдите максимальное число  $k$ , для которого  $n$  можно представить как сумму  $k$  различных натуральных слагаемых. Выведите в первой строке число  $k$ , во второй –  $k$  слагаемых.

## Решение

```
using System.Collections.Generic;
using System;
using System.Linq;

static class Program
{
    static void Main()
    {
        int n = int.Parse(Console.ReadLine());
        List<int> k = new List<int>();

        int i = 1;
        int sum = 0;
        while (sum < n)
        {
            k.Add(i);
            sum += i;
            if (sum > n)
            {
                sum -= k.Last();
                k.RemoveAt(k.Count - 1);
                while (sum != n)
                {
                    sum -= k[k.Count - 1];
                    k[k.Count - 1] = k[k.Count - 1] + 1;
                    sum += k[k.Count - 1];
                }
            }

            i++;
        }

        Console.WriteLine(k.Count);
        k.ForEach(x =>
        {
            Console.Write($"{x} ");
        });
    }
}
```

*Листинг программы - Решение задачи*

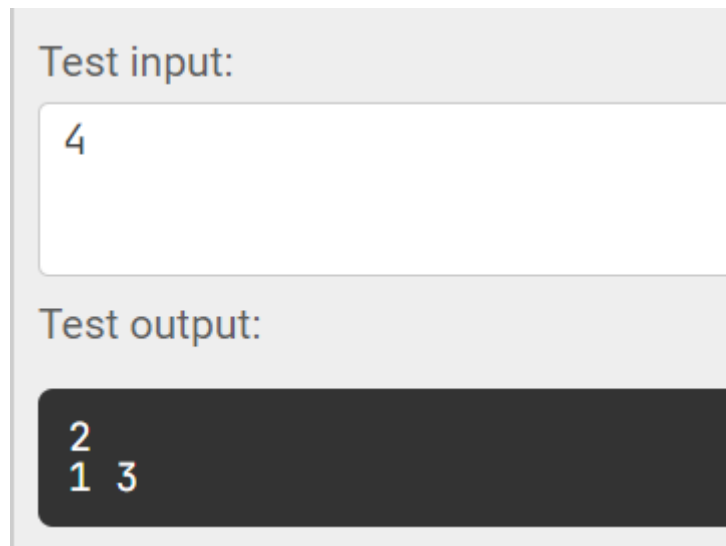


Рисунок 6 - Результат работы программы

**Задача 4:** По данной непустой строке  $ss$  длины не более  $10^4$ , состоящей из строчных букв латинского алфавита, постройте оптимальный беспрефиксный код. В первой строке выведите количество различных букв  $kk$ , встречающихся в строке, и размер получившейся закодированной строки. В следующих  $kk$  строках запишите коды букв в формате "letter: code". В последней строке выведите закодированную строку.

## Решение

```
using System.Net;
using System.Security.Cryptography.X509Certificates;
using System.Linq;
using System.Collections.Generic;
using System;
static class Program
{
    static string Result = string.Empty;
    private class Node
    {
        public char? Value;

        public Node? Left, Right;

        public char VAL;
        public string Code = string.Empty;

        public Node(char value)
        {
            Value = value;
            VAL = value;
        }
    }
}
```

```

    }

    public void AddNeighbour(Node left, Node right)
    {
        Value = null;
        Left = left;
        Right = right;
    }

    public void AddToCode(string dir)
    {
        Code = dir + Code;
        Left?.AddToCode(dir);
        Right?.AddToCode(dir);
    }
}

private static void Main()
{
    string n = Console.ReadLine();

    Dictionary<char, int> letters = new();

    for (int i = 0; i < n.Length; i++)
    {
        // Ищем уникальные символы
        if (!letters.ContainsKey(n[i]))
            // Добавляем в словарь, и считаем количество символов для
приоритета
            letters.Add(n[i], n.Count(x => x == n[i]));
    }

    // отдельный словарь для сортированных узлов
    Dictionary<Node, int> sorted = new();
    foreach (var c in letters.OrderBy(k => k.Value))
    {
        sorted.Add(new Node(c.Key), c.Value);
    }

    /*
    foreach (var c in sorted)
    {
        Console.WriteLine(c.Key.Value + " " + c.Value);
    }
    */

    Dictionary<Node, int> saveNodes = new();

    if (sorted.Count == 1)
    {

```

```

    var s = sorted.First();

    s.Key.Code = "0";
    saveNodes.Add(s.Key, s.Value);
}

while (sorted.Count != 1)
{
    // Выделяем минимальные узлы, сохраняем и удаляем их
    var left = MinBy(sorted);
    sorted.Remove(left.Key);
    var right = MinBy(sorted);
    sorted.Remove(right.Key);

    var temp = new Node('t');
    temp.AddNeighbour(left.Key, right.Key);
    sorted.Add(temp, left.Value + right.Value);

    left.Key.AddToCode("1");
    right.Key.AddToCode("0");

    if (left.Key.Value is not null)
    {
        saveNodes.Add(left.Key, left.Value);
    }

    if (right.Key.Value is not null)
    {
        saveNodes.Add(right.Key, right.Value);
    }

    // Снова сортируем по возрастанию
    sorted = SortByValue(sorted);
}

Dictionary<char, string> map = new();
foreach (var sn in saveNodes.OrderByDescending(x => x.Value))
{
    map.Add(sn.Key.VAL, sn.Key.Code);
}

string res = String.Empty;
foreach (var v in n)
{
    res += map[v];
}
Console.WriteLine($"{letters.Count} {res.Length}");
foreach (var sn in saveNodes.OrderByDescending(x => x.Value))
{
    Console.WriteLine($"{sn.Key.Value}: {sn.Key.Code}");
}

```

```

        Console.WriteLine(res);
    }

    private static Dictionary<Node, int> SortByValue(Dictionary<Node, int>
letters)
    {
        Dictionary<Node, int> sorted = new();
        foreach (var c in letters.OrderBy(k => k.Value))
        {
            sorted.Add(c.Key, c.Value);
        }

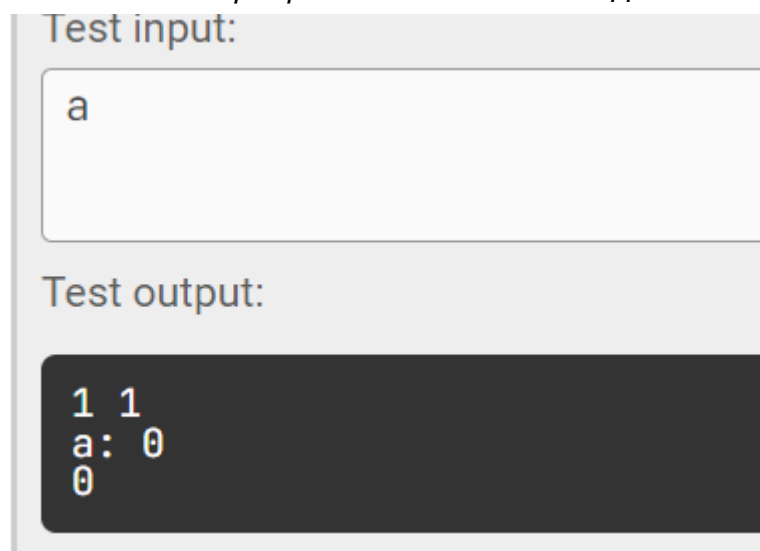
        return sorted;
    }

    private static KeyValuePair<Node, int> MinBy(Dictionary<Node, int>
letters)
    {
        Dictionary<Node, int> sorted = new();
        foreach (var c in letters.OrderBy(k => k.Value))
        {
            return new KeyValuePair<Node, int>(c.Key, c.Value);
        }

        return new KeyValuePair<Node, int>();
    }
}

```

*Листинг программы - Решение задачи*



*Рисунок 7 - Результат работы программы*

**Задача 5:** Восстановите строку по её коду и беспрефиксному коду символов. В первой строке входного файла заданы два целых числа  $k$  и  $l$  через пробел — количество различных букв, встречающихся в строке, и размер получившейся закодированной строки, соответственно. В следующих  $k$  строках записаны коды букв в формате "letter: code". Ни один код не является префиксом другого. Буквы могут быть перечислены в любом порядке. В качестве букв могут встречаться лишь строчные буквы латинского алфавита; каждая из этих букв встречается в строке хотя бы один раз. Наконец, в последней строке записана закодированная строка. Исходная строка и коды всех букв непусты. Заданный код таков, что закодированная строка имеет минимальный возможный размер.

## Решение

```
using System.Collections.Generic;
using System.Linq;

public class Program
{
    public static void Main()
    {
        Dictionary<string, char> map = new Dictionary<string, char>();

        string lengths = Console.ReadLine();
        var l = lengths.Split(" ");
        int uniqueChars = Int32.Parse(l.First());

        for (int i = 0; i < uniqueChars; i++)
        {
            var input = Console.ReadLine().Split(": ");
            map.Add(input.Last(), input.First()[0]);
        }

        string coded = Console.ReadLine();
        string decoded = String.Empty;

        string token = string.Empty;
        foreach (var ch in coded)
        {
            token += ch;
            if (map.ContainsKey(token))
            {
                decoded += map[token];
                token = String.Empty;
            }
        }
    }
}
```

```

        Console.WriteLine(decoded);
    }
}

```

### Листинг программы - Решение задачи

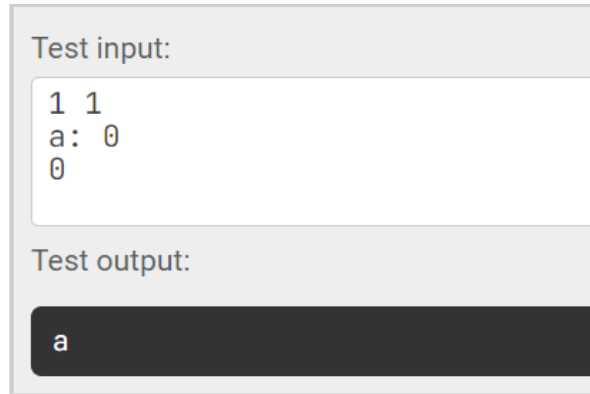


Рисунок 8 - Результат работы программы

**Задача 5:** Первая строка входа содержит число операций  $n$ . Каждая из последующих  $n$  строк задают операцию одного из следующих двух типов:

- Insert  $x$  — целое число;
- ExtractMax.

Первая операция добавляет число  $x$  в очередь с приоритетами, вторая — извлекает максимальное число и выводит его.

## Решение

```

using System;
using System.Collections;

public class MainClass
{
    public static void Main()
    {
        var input = Console.ReadLine();
        var commandCount = int.Parse(input);

        PriorityQueue<int> queue = new PriorityQueue<int>();
        for(int i=0; i<commandCount; i++)
        {

```



```

var commands = Console.ReadLine()!.Split(' ');
switch (commands[0])
{
    case "Insert":
    {
        var value = int.Parse(commands[1]);
        queue.Add(value, value); break;
    }
    case "ExtractMax": Console.WriteLine(queue.Poll()); break;
}
}

class PriorityQueue<T>
{
    struct Item<T>
    {
        public T value;
        public int priority;

        public Item(T value, int priority)
        {
            this.value = value;
            this.priority = priority;
        }

        public bool Compare(Item<T> item)
        {
            return priority > item.priority;
        }
    }

    static ArrayList items = new ArrayList();

    private void BinAdd(Item<T> item)
    {
        int i = -1;
        int j = items.Count;
        int avr;
        while(i + 1 < j)
        {
            avr = (i + j) >> 1;
            if(item.Compare((Item<T>)items[avr])) {
                j = avr;
            }
            else i = avr;
        }
        items.Insert(++i, item);
    }

    // Return the number of items in the queue.
    public int Count
    {

```

```

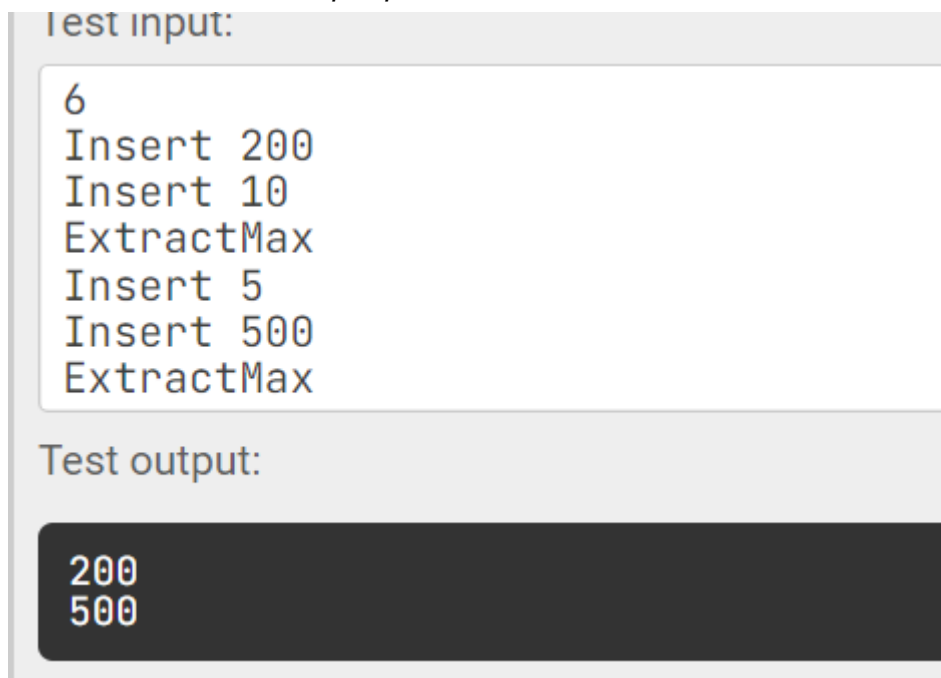
        get
        {
            return items.Count;
        }
    }

    // Add an item to the queue.
    public void Add(T new_value, int new_priority)
    {
        BinAdd(new Item<T>(new_value, new_priority));
    }

    // Remove the item with the largest priority from the queue.
    public T Poll()
    {
        Item<T> top_item = (Item<T>)items[0];
        items.Remove(top_item);
        return top_item.value;
    }
}

```

*Листинг программы - Решение задачи*



*Рисунок 9 - Результат работы программы*

## “Разделяй и властвуй”

**Задача 1:** В первой строке даны целое число  $1 \leq n \leq 10$  и массив  $A[1...n]$  из  $n$  различных натуральных чисел, в порядке возрастания, во второй — целое число, не превышающих  $10^9$ . Для каждого  $i$  от 1 до  $n$  необходимо вывести индекс  $1 \leq j \leq n$ , для которого  $A[j] = b - iA[i]$ , или -1, если такого  $j$  вывести “нет”.

## Решение

```
using System;
using System.Collections.Generic;

public class MainClass
{
    private static Dictionary<int,int> FillList(){
        string[] input = Console.ReadLine().Split(' ');
        Dictionary<int,int> list = new Dictionary<int,int>();
        for(int i = 1; i <= int.Parse(input[0]); i++){
            list.Add(int.Parse(input[i]), i);
        }

        return list;
    }

    private static Dictionary<int,int> FillListUn(){
        string[] input = Console.ReadLine().Split(' ');
        Dictionary<int,int> list = new Dictionary<int,int>();
        for(int i = 1; i <= int.Parse(input[0]); i++){
            list.Add(i, int.Parse(input[i]));
        }

        return list;
    }

    public static void Main()
    {
        Dictionary<int,int> x = FillList();
        Dictionary<int,int> y = FillListUn();

        foreach(KeyValuePair<int,int> i in y){
            if (x.ContainsKey(i.Value)){
                Console.WriteLine($"{x[i.Value]} ");
            }
            else
            {
                Console.WriteLine("-1 ");
            }
        }
    }
}
```

*Листинг программы- Двоичный поиск*

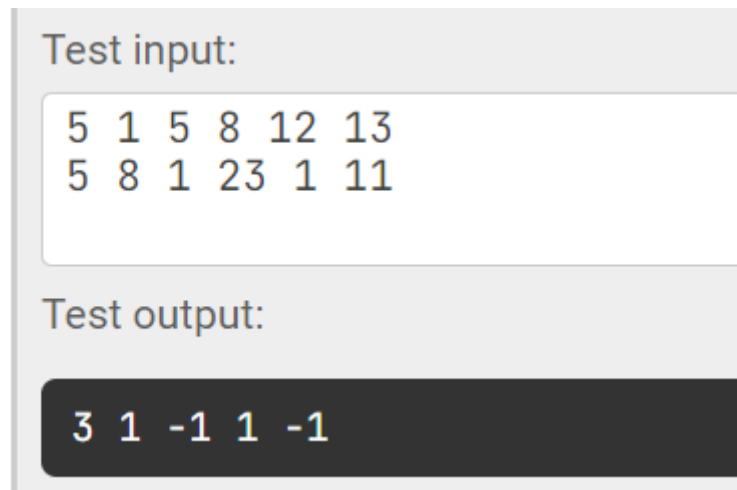


Рисунок 10 - Результат работы программы

**Задача 2:** Первая строка содержит число  $1 \leq n \leq 10^5$ , вторая — массив  $A[1 \dots n]$   $A[1 \dots n]$ , содержащий натуральные числа, не превосходящие  $10^9$ .

Необходимо посчитать число пар индексов  $1 \leq i < j \leq n$ , для которых  $A[i] > A[j]$   $A[i] > A[j]$ . (Такая пара элементов называется инверсией массива. Количество инверсий в массиве является в некотором смысле его мерой неупорядоченности: например, в упорядоченном по неубыванию массиве инверсий нет вообще, а в массиве, упорядоченном по убыванию, инверсию образуют каждые два элемента.)

```
using System;
```

```
//подсчет инверсий через сортировку слиянием
```

```
public class MainClass
```

```
{
```

```
    public static long Merge(int[] arr, int begin, int middle, int end)
```

```
    {
```

```
        //суммарный размер участка
```

```
        int n = end - begin + 1;
```

```
        //вспомогательный массив для слияния
```

```
        int[] arr_aux = new int[n];
```

```
        //индекс для прохода по левой половине
```

```
        int left = begin;
```

```
        //индекс для прохода по правой половине
```

```
        int right = middle + 1;
```

```
        //подсчет количества инверсий где один элемент пары из левой  
        //половины, а другой из правой
```

```
        long inversions = 0;
```

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

```
        {
```

```
            if (left > middle)
```

```
            {
```

```
                //если левый массив закончился
```

```
                //просто копируем оставшееся в правый
```

```

        arr_aux[i] = arr[right];
        right++;
    }
    else if (right > end)
    {
        //если правый массив закончился
        //просто копируем оставшееся в левом
        arr_aux[i] = arr[left];
        left++;
    }
    else if (arr[left] <= arr[right])
    {
        //наименьший элемент левого массива меньше либо равен
        //наименьшему элементу правого массива
        //инверсий нет
        arr_aux[i] = arr[left];
        left++;
    }
    else
    {
        //наименьший элемент левого массива больше наименьшего
        //элемента правого массива
        arr_aux[i] = arr[right];
        right++;
        //считаем количество инверсий (равно количеству элементов
        //оставшихся в левом массиве)
        inversions += middle - left + 1;
    }
}

//копируем вспомогательный массив в основной
for(int i=0; i<n; i++)
{
    arr[begin+i] = arr_aux[i];
}

return inversions;
}

public static long MergeSort(int[] arr, int begin, int end)
{
    if (begin >= end) return 0;
    int middle = begin + (end-begin)/2;
    long inversions = 0;
    //подсчет количества инверсий в левой части массива
    inversions += MergeSort(arr, begin, middle);
    //подсчет количества инверсий в правой части массива
    inversions += MergeSort(arr, middle+1, end);
    //подсчет количества инверсий где один элемент пары из левой
    //половины, а другой из правой
    inversions += Merge(arr, begin, middle, end);
    return inversions;
}

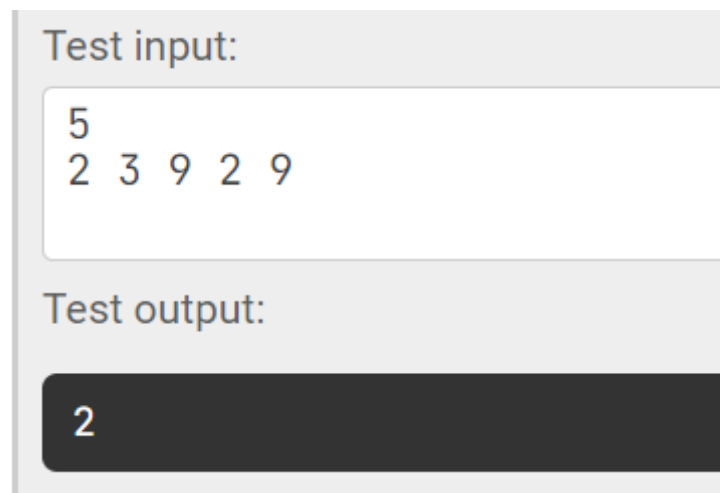
```

```

public static void Main()
{
    int n = Int32.Parse(Console.ReadLine());
    int[] arr = Array.ConvertAll(Console.ReadLine().Split(' '),
Int32.Parse);
    Console.Write(MergeSort(arr, 0, n-1));
}
}

```

*Листинг программы- Подсчет инверсий*



*Рисунок 11 - Результат работы программы*

**Задача 3:** В первой строке задано два целых числа  $1 \leq n \leq 50000$  и  $1 \leq m \leq 50000$  — количество отрезков и точек на прямой, соответственно. Следующие  $n$  строк содержат по два целых числа  $a_i$  — координаты концов отрезков. Последняя строка содержит  $m$  целых чисел — координаты точек. Все координаты не превышают  $10^8$  по модулю. Точка считается принадлежащей отрезку, если она находится внутри него или на границе. Для каждой точки в порядке появления во вводе выведите, скольким отрезкам она принадлежит.

## Решение

```

using System;

public class MainClass
{
    //деление массива на 3 части для быстрой сортировки
    public static int Partition3(int[] arr, int left, int right, out int
pivot_end)
    {
        //меняем местами первый элемент и случайный

```

```

int random_index = new Random().Next(left, right);
int pivot_value = arr[random_index];
arr[random_index] = arr[left];
arr[left] = pivot_value;

int j = left; //менее опорного
int k = right; //более опорного
int i = left; //счетчик для прохода
int tmp;

while (i <= k)
{
    if (arr[i] < pivot_value)
    {
        tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;
        j++;
        i++;
    }
    else if (arr[i] > pivot_value)
    {
        tmp = arr[i];
        arr[i] = arr[k];
        arr[k] = tmp;
        k--;
    }
    else
    {
        i++;
    }
}

pivot_end = k;
return j;
}

public static void QuickSort(int[] arr, int left, int right)
{
    if (left >= right)
    {
        return;
    }

    int pivot_end;
    int pivot_begin = Partition3(arr, left, right, out pivot_end);

    QuickSort(arr, left, Math.Max(pivot_begin - 1, left));
    QuickSort(arr, Math.Min(pivot_end + 1, right), right);
}

//поиск по отсортированному массиву, содержащему координаты

```

```

    //возвращает количество элементов содержащих координату меньше либо
    //равной заданной
    //либо строго меньше заданной если strict_comparison == true
    public static int BinarySearch_continuous(int[] array, int begin, int
end, int value, bool strict_comparison)
    {
        //если диапазон из одного элемента
        if (begin == end)
        {
            //если заданная координата меньше единственного элемента
            if(array[begin] > value)
            {
                return begin;
            }

            //если заданная координата больше единственного элемента
            else if(array[begin] < value)
            {
                return begin+1;
            }

            //если единственный элемент равен заданной координате и
сравнение строгое
            else if(array[begin] == value && strict_comparison == true)
            {
                return begin;
            }

            //если единственный элемент равен заданной координате и
сравнение нестрогое
            else
            {
                return begin+1;
            }
        }
        //если в диапазоне больше одного элемента
        else
        {
            //середина диапазона
            int midpoint = begin + (end-begin)/2;
            if(array[midpoint] < value)
            {
                return BinarySearch_continuous(array, midpoint+1, end,
value, strict_comparison);
            }
            else if(array[midpoint] > value)
            {
                return BinarySearch_continuous(array, begin,
Math.Max(midpoint-1,begin), value, strict_comparison);
            }
            else if(array[midpoint] == value && strict_comparison == true)
            {

```



```

        return BinarySearch_continuous(array, begin,
Math.Max(midpoint-1,begin), value, strict_comparison);
    }
    else
    {
        return BinarySearch_continuous(array, midpoint+1, end,
value, strict_comparison);
    }
}

}

public static void Main()
{
    int[] temp_arr = Array.ConvertAll(Console.ReadLine().Split(' '),
Int32.Parse);
    int num_segments = temp_arr[0];
    int num_points = temp_arr[1];
    int[] left_ends = new int[num_segments];
    int[] right_ends = new int[num_segments];
    for (int i = 0; i < num_segments; i++)
    {
        temp_arr = Array.ConvertAll(Console.ReadLine().Split(' '),
Int32.Parse);
        left_ends[i] = temp_arr[0];
        right_ends[i] = temp_arr[1];
    }
    int[] points = Array.ConvertAll(Console.ReadLine().Split(' '),
Int32.Parse);

    //сортируем массив левых концов отрезков и массив правых концов
отрезков
    QuickSort(left_ends,0,num_segments-1);
    QuickSort(right_ends,0,num_segments-1);

    foreach (int point in points)
    {
        //количество отрезков, левые концы которых лежат левее заданной
точки либо на ней (нестрогое сравнение)
        int num_left_ends = BinarySearch_continuous(left_ends, 0,
num_segments-1, point, false);
        //количество отрезков, правые концы которых лежат левее заданной
точки (строгое сравнение)
        int num_right_ends = BinarySearch_continuous(right_ends, 0,
num_segments-1, point, true);
        Console.Write((num_left_ends - num_right_ends).ToString() + '
');
    }
    Console.Write('\n');
}
}

```

### Листинг программы - Точки и отрезки

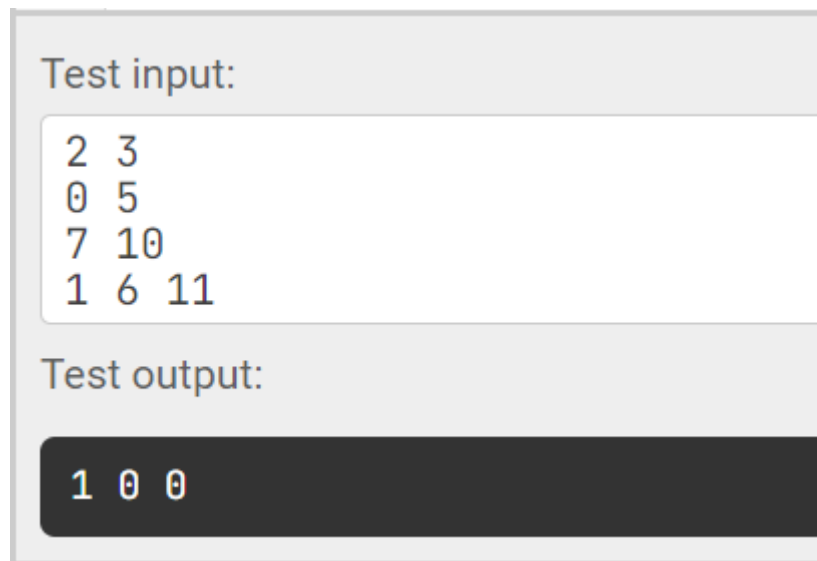


Рисунок 12 - Результат работы программы

**Задача 4:** Первая строка содержит число  $1 \leq n \leq 10^4$ , вторая —  $n$  натуральных чисел, не превышающих 10. Выведите упорядоченную по не убыванию последовательность этих чисел.

### Решение

```
public class MainClass
{
    public static void Main()
    {
        Console.ReadLine();
        var input = Console.ReadLine().Split('
').Select(int.Parse).ToArray();

        var counts = new int[11];

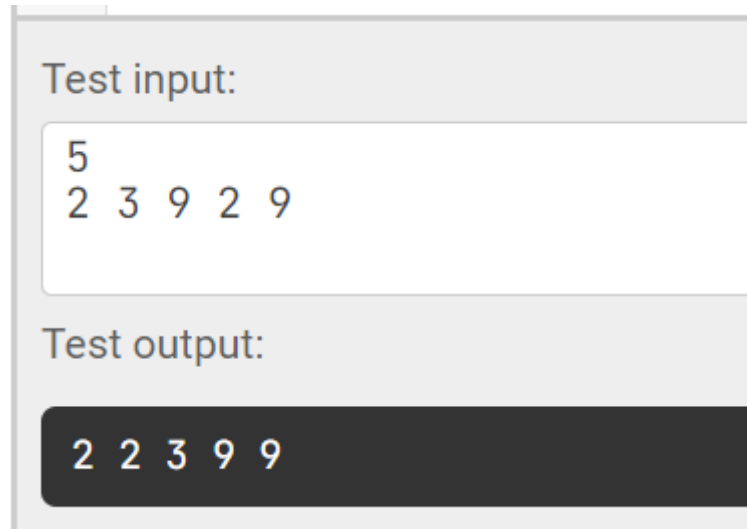
        for(int i = 0; i < input.Length; i++)
        {
            counts[input[i]]++;
        }

        int index = 0;
        var output = new int[input.Length];

        for(int i = 0; i < counts.Length; i++)
        {
            for(int j=0; j < counts[i]; j++)
```

```
        {  
            output[index++] = i;  
        }  
    }  
  
    Console.WriteLine($"{string.Join(" ", output)}");  
}  
}
```

*Листинг программы- сортировка подсчетом*



*Рисунок 13 - Результат работы программы*