**Аттестция 2 Чобану Артём I1902**

Входные данные Создайте 10-ти элементный одномерный массив MyArray состоящий из эллементов типа char, заполненный латинскими буквами вашей фамилии. Если ваша фамилия состоит из менее 10-ти букв, используйте используйте ваше имя и отчество. Также, создайте массив MySortedArray который совпадает с отсортированным массивом MyArray.

Примеры

1. Если ваша фамилия «Добровольская» («Dobrovoliscaia»), тогда: MyArray = { d, o, b, r, o, v, o, l, i, s} MySortedArray = { b, d, i, l, o, o, o, r, s, v}

2. Если ваша фамилия «Петров» («Petrov»), то добавьте ваше имя, например «Дан» («Dan»), и начало отчества «Иванович» («Ivanovici»): MyArray = { p, e, t, r, o, v, d, a, n, i} MySortedArray = { a, d, e, i, n, o, p, r, t, v}

Код метода Main:

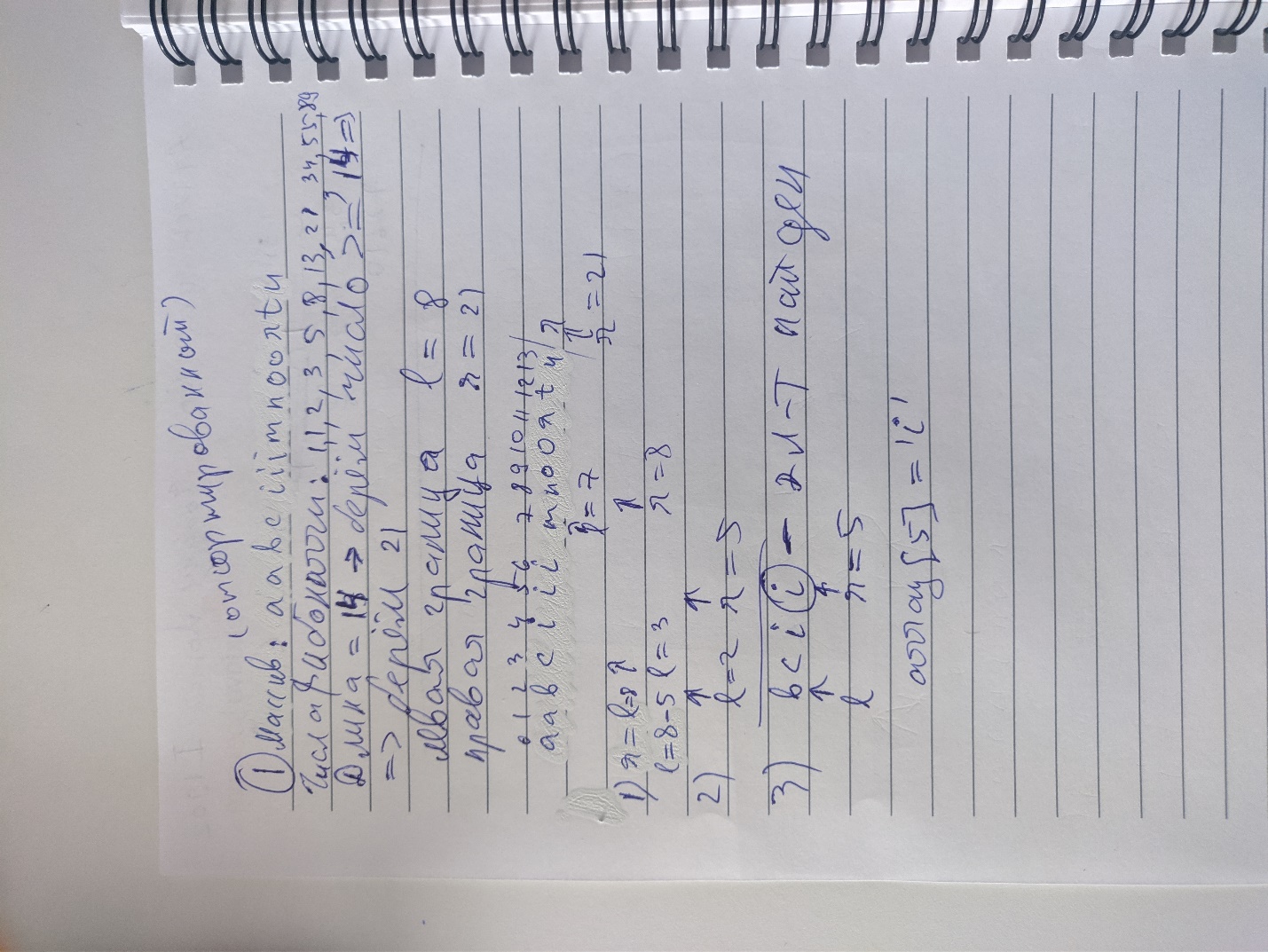
private static char[] GetNameArray() => new[]  
{  
 'c', 'i', 'o', 'b', 'a', 'n', 'u', 'a', 'r', 't', 'i', 'o', 'm', 'i'  
};  
  
private static void Main()  
{  
 char[] myArray = GetNameArray();  
 char[] mySortedArray = GetNameArray().BubbleSort();  
  
 Console.WriteLine("Изначальный массив: ");  
 myArray.DisplayInline();  
 Console.WriteLine("Отсортированный массив: ");  
 mySortedArray.DisplayInline();  
  
 var oPosition = mySortedArray.BinarySearch('o');  
 Console.WriteLine($"Символ о находится на позиции: {oPosition}");  
  
 Console.WriteLine();  
 myArray.BubbleSortWithDisplaying();  
}

Отсортированный массив:

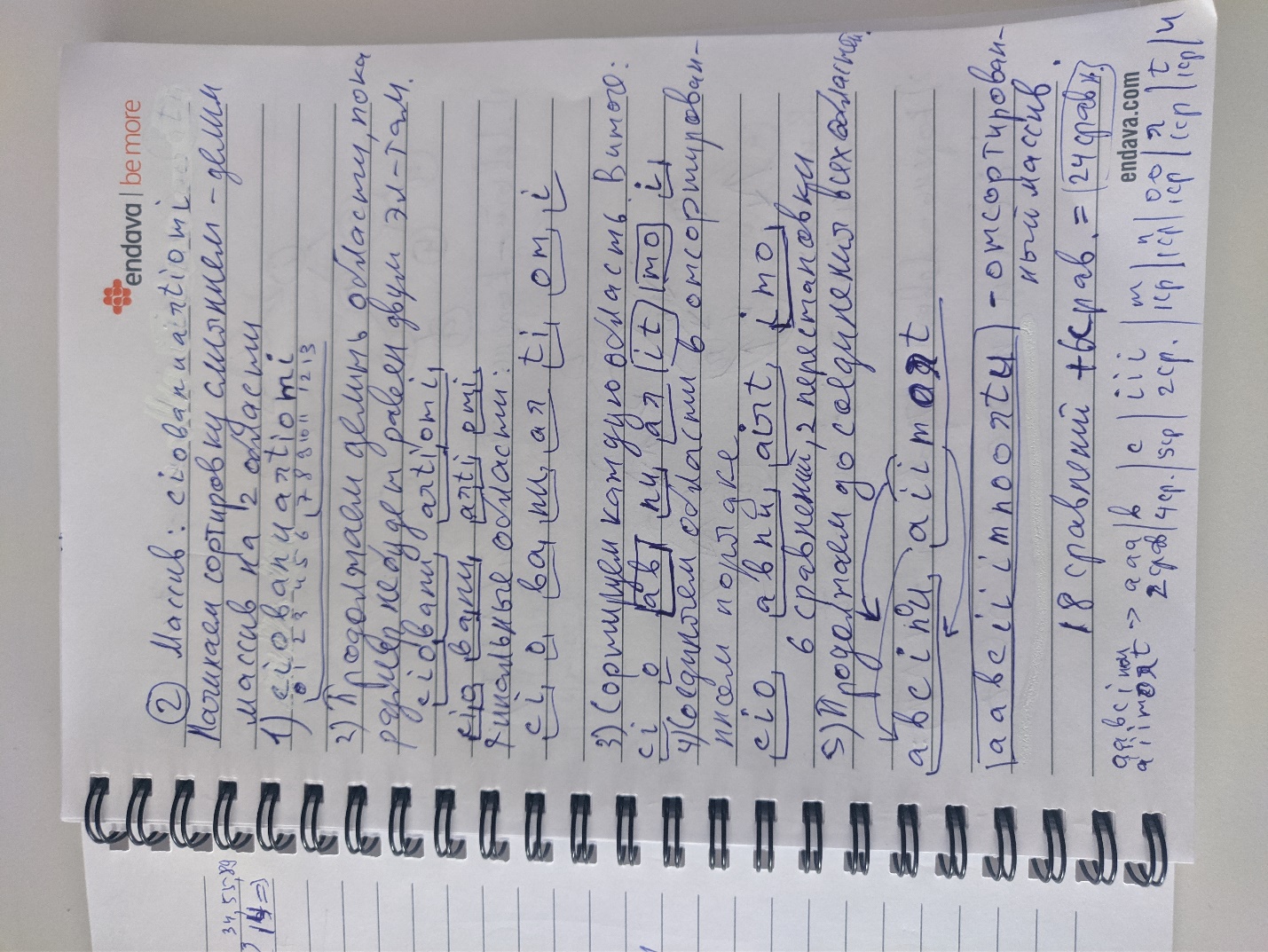


**Задания**

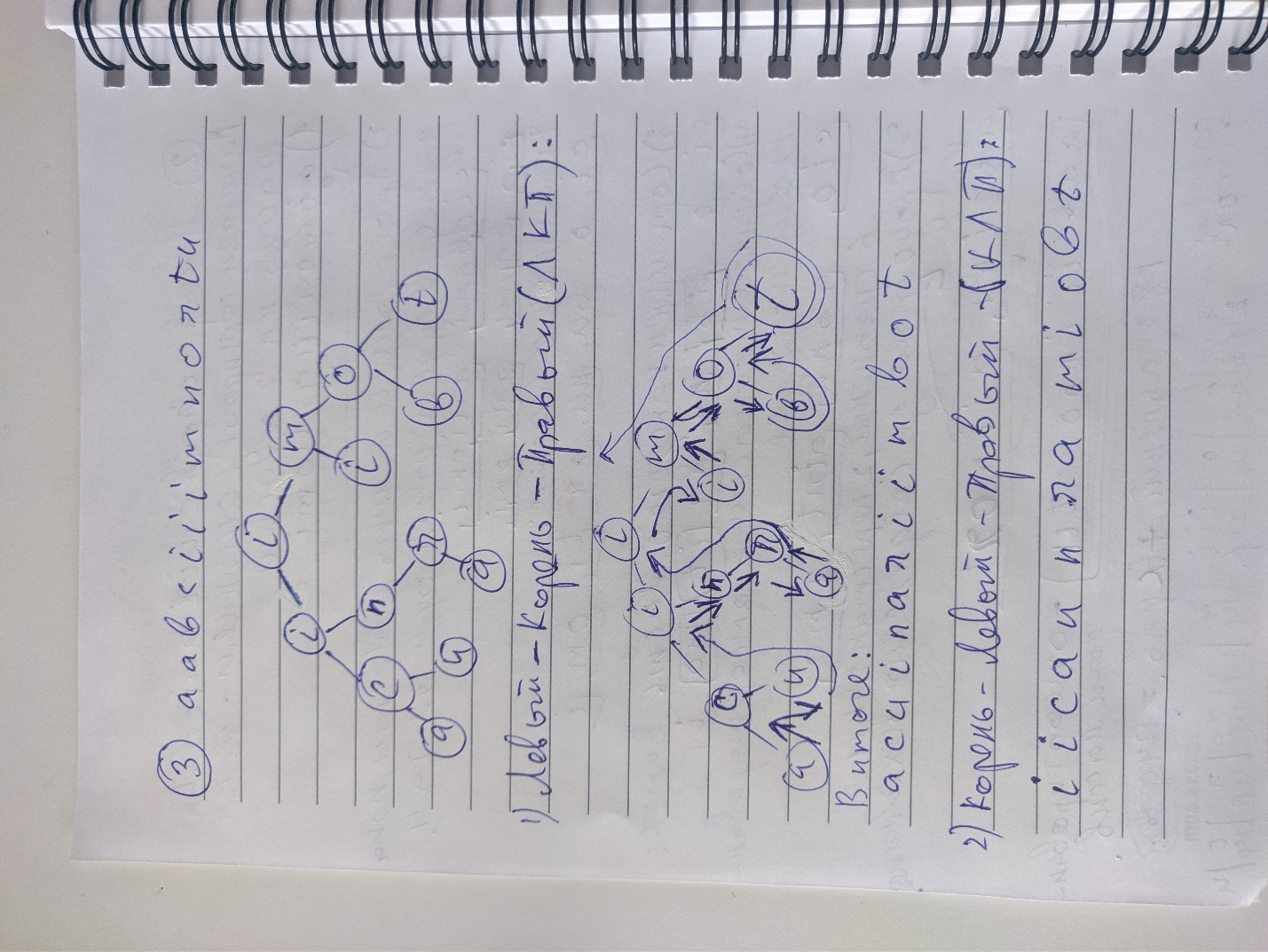
1. Примените метод Фибоначчи для поиска элемента ‘a’ в вашем персональном массиве MySortedArray. Перечислите все элементы из массива MySortedArray с которыми был сопоставлен искомый элемент в процесе поиска (в порядке сопоставления), независимо от результата поиска (был ли найден элемент, или нет). (15 б.)

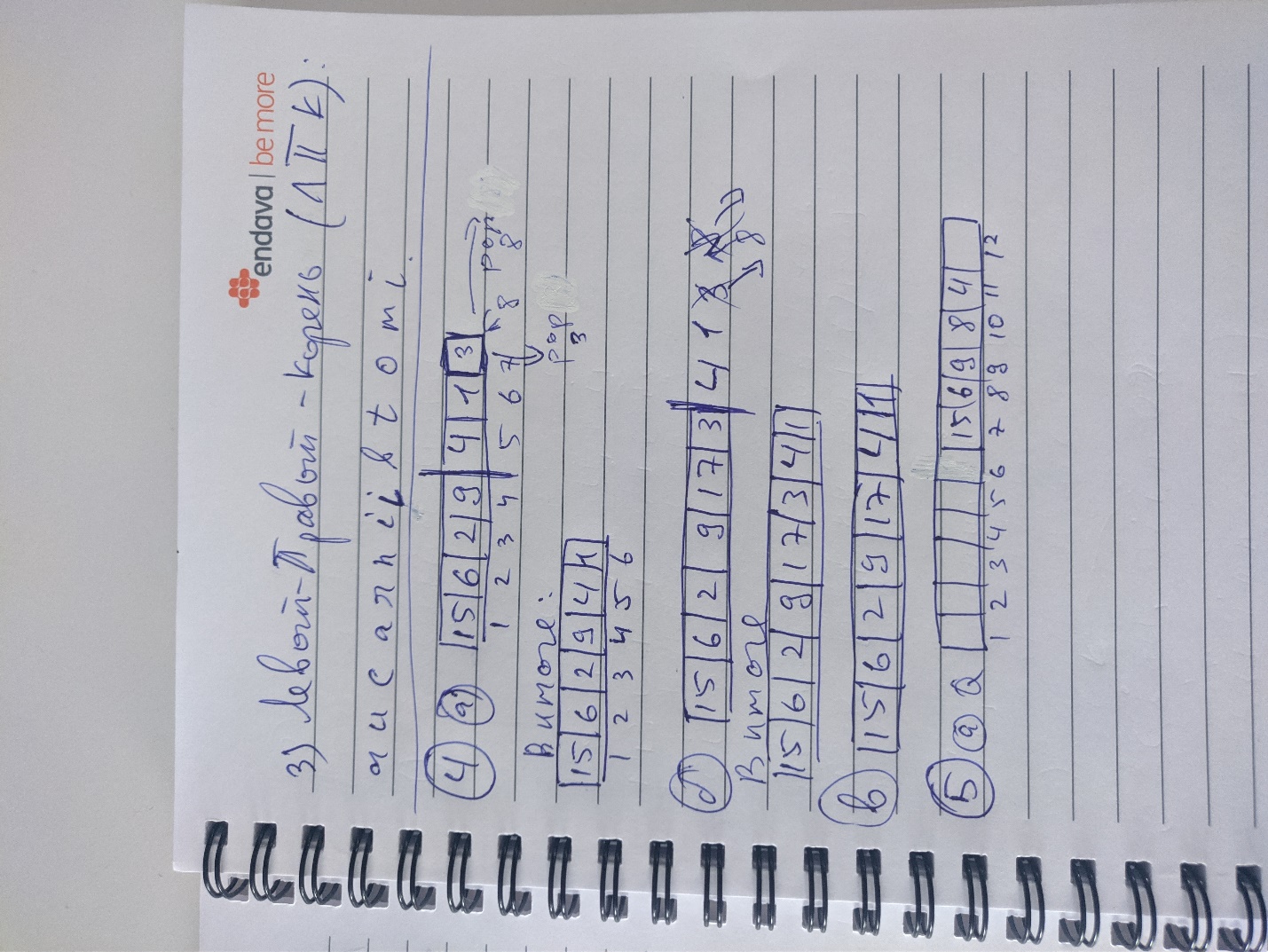


1. Примените Сортировку слиянием (MergeSort) над вашим персональным массивом MyArray, сортируя элементы в алфавитном порядке. Укажите детально каждое прохождение массива данным методом. Для каждого прохода, укажите количество сравнений и перестановок, которые были произведены для его получения (для исходного массива ничего не указывается). (15 б.)

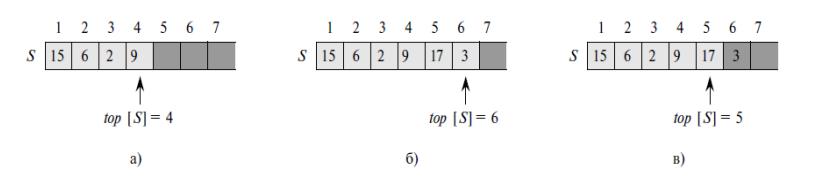


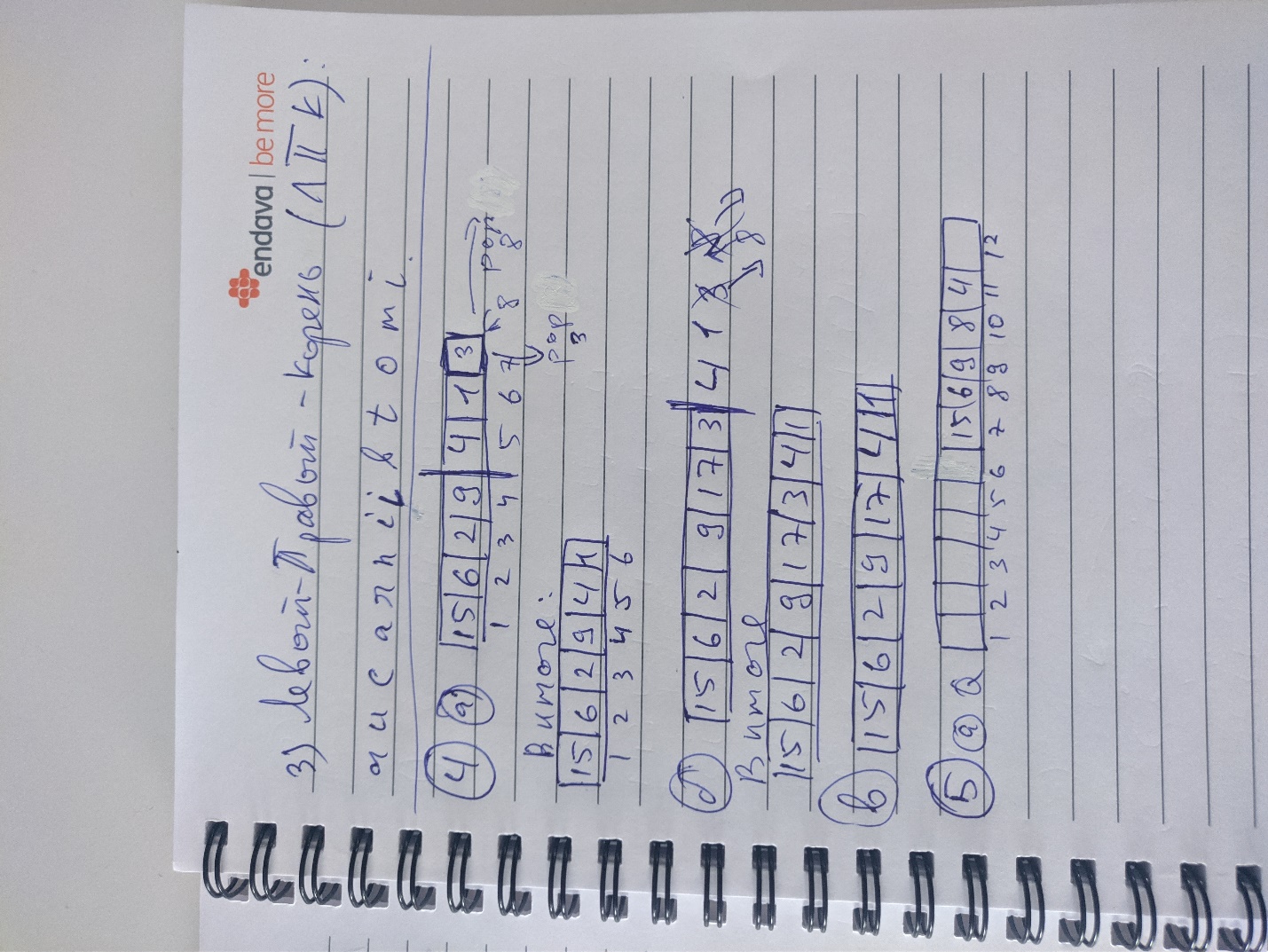
1. Из элементов персонального массива MyArray постройте бинарное дерево поиска. Проиллюстрируйте выполнение трёх типов обходов: «лкп», «клп», «лпк»массива данным методом. Для каждого прохода, укажите количество сопоставлений и перестановок, которые были произведены для его получения (для исходного массива ничего не указывается). (15 б.)



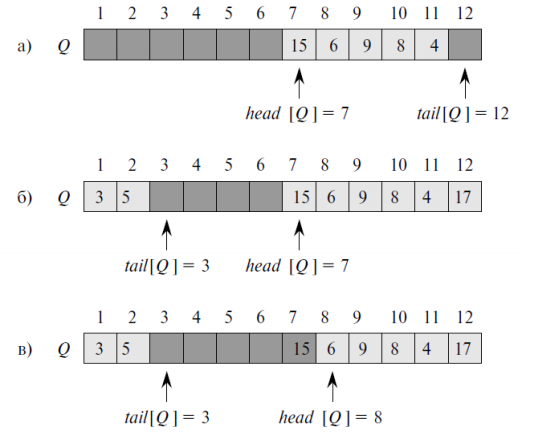


1. Используя в качестве модели следующий рисунок, проиллюстрируйте результат воздействия на изначально пустой стек S, хранящийся в Page 2 of 2 массиве S[1..6], операций PUSH(S, 4), PUSH(S, 1), PUSH(S, 3), POP(S), PUSH(S, 8) и POP(S). (10 б.)





1. Используя в качестве модели следующий рисунок, проиллюстрируйте результат воздействия на изначально пустую очередь Q, хранящуюся в массиве Q[1..6], операций ENQUEUE(Q, 4), ENQUEUE(Q, 1), ENQUEUE(Q, 3), DEQUEUE(Q), ENQUEUE(Q, 8) и DEQUEUE(Q). (10 б.)



1. Покажите, как реализовать очередь с помощью двух стеков. Проанализируйте время работы операций, которые выполняются с ее элементами. (10 б.)
2. Напишите код программы в которой создается линейный динамический двусвязный список, элементов типа char. Список будет храниться одним единственным указателем на любой его элемент. Левый край списка будет считаться его началом, правый край – его концом. Напишите следующие методы: а) Позиционирование текущего указателя на первый элемент списка, б) Вывод элементов списка слева-направо, начиная с текущего элемента, в) Добавление одного нового элемента в начало списка. В методе main, создайте пустой список. Затем добавьте первые четыре элемента вашего персонального массива MyArray, используя метод (в). Затем выведите содержимое списка, используя методы (а) и (б). Укажите что будет выведено на консоль. (25 б.)

private static void Main()  
{  
 Range range1 = new Range(-1, 1);  
 Range range2 = new Range(2, 4);  
 Range range3 = new Range(5, 6);  
 Range range4 = new Range(7, 8);  
  
 var array = new Array<int>(range1, range2, range3, range4);  
  
 for (int i = 0; i < array.Length; i++)  
 {  
 array.Elements[i] = i;  
 }  
  
 int n = 0;  
 for (int i1 = range1.L; i1 <= range1.H; i1++)  
 {  
 for (int i2 = range2.L; i2 <= range2.H; i2++)  
 {  
 for (int i3 = range3.L; i3 <= range3.H; i3++)  
 {  
 for (int i4 = range4.L; i4 <= range4.H; i4++)  
 {  
 array.SetValueByIlliffesVector(n, i1, i2, i3, i4);  
 Console.Write($"[{i1},{i2},{i3},{i4}] = {array.GetValueByIlliffesVector(i1, i2, i3, i4)} ");  
  
 n++;  
 }  
  
 Console.WriteLine();  
 }  
 }  
 }  
  
 Console.WriteLine();  
  
 PrintMeasuredTime(() => array.GetRowValue(-1, 2, 5, 7), "Direct access by rows");  
 PrintMeasuredTime(() => array.GetColumnValue(-1, 2, 5, 7), "Direct access by columns");  
 PrintMeasuredTime(() => array.GetValueByDefiningVector(-1, 2, 5, 7), "Defining vector access time");  
 PrintMeasuredTime(() => array.GetValueByIlliffesVector(-1, 2, 5, 7), "Vector Illiffe access");  
}  
  
private static void PrintMeasuredTime(Func<int> action, string message)  
{  
 var beginTime = DateTime.Now;  
 action();  
 var endTime = DateTime.Now;  
 Console.WriteLine($"{message} {(endTime - beginTime).TotalMilliseconds} ms");  
}

public class Range  
{  
 public int L { get; }  
 public int H { get; }  
 public int Difference { get; }  
  
 public Range(int l, int h)  
 {  
 L = l;  
 H = h;  
 Difference = h - l;  
 }  
}

public class Array<TValue>  
{  
 private List<int> DefiningVector { get; }  
  
 private List<List<List<List<TValue>>>> IlliffesVector { get; }  
  
 public int Length { get; }  
  
 public Range Range1 { get; }  
 public Range Range2 { get; }  
 public Range Range3 { get; }  
 public Range Range4 { get; }  
  
 public TValue[] Elements { get; }  
  
 public Array(Range range1, Range range2, Range range3, Range range4)  
 {  
 Range1 = range1;  
 Range2 = range2;  
 Range3 = range3;  
 Range4 = range4;  
  
 Length = (range1.Difference + 1) \*  
 (range2.Difference + 1) \*  
 (range3.Difference + 1) \*  
 (range4.Difference + 1);  
 Elements = new TValue[Length];  
  
 DefiningVector = new int[15].ToList();  
  
 DefiningVector[0] = 4;  
 DefiningVector[1] = range1.L;  
 DefiningVector[2] = range1.H;  
 DefiningVector[3] = range2.L;  
 DefiningVector[4] = range2.H;  
 DefiningVector[5] = range3.L;  
 DefiningVector[6] = range3.H;  
 DefiningVector[7] = range4.L;  
 DefiningVector[8] = range4.H;  
 DefiningVector[13] = 1;  
 DefiningVector[12] = range4.Difference + 1;  
  
  
 DefiningVector[11] = (range3.Difference + 1) \* DefiningVector[12];  
 DefiningVector[10] = (range2.Difference + 1) \* DefiningVector[11];  
 DefiningVector[14] = range1.L \* DefiningVector[10] +  
 range2.L \* DefiningVector[11] +  
 range3.L \* DefiningVector[12] +  
 range4.L \* DefiningVector[13];  
  
 IlliffesVector = new List<List<List<List<TValue>>>>();  
 for (int i1 = range1.L; i1 <= range1.H; i1++)  
 {  
 IlliffesVector.Add(new List<List<List<TValue>>>());  
  
 for (int i2 = range2.L; i2 <= range2.H; i2++)  
 {  
 int currenti1 = i1 - range1.L;  
 IlliffesVector[currenti1].Add(new List<List<TValue>>());  
  
 for (int i3 = range3.L; i3 <= range3.H; i3++)  
 {  
 var currenti2 = i2 - range2.L;  
 IlliffesVector[currenti1][currenti2].Add(new List<TValue>());  
  
 for (int i4 = range4.L; i4 <= range4.H; i4++)  
 {  
 var currenti3 = i3 - range3.L;  
  
 IlliffesVector[currenti1][currenti2][currenti3].Add(default);  
 }  
 }  
 }  
 }  
 }  
  
 public void SetValueByIlliffesVector(TValue value, int i1, int i2, int i3, int i4)  
 {  
 IlliffesVector[i1 - Range1.L][i2 - Range2.L][i3 - Range3.L][i4 - Range4.L] = value;  
 }  
  
 public TValue GetValueByIlliffesVector(int i1, int i2, int i3, int i4)  
 => IlliffesVector[i1 - Range1.L][i2 - Range2.L][i3 - Range3.L][i4 - Range4.L];  
  
  
 public TValue GetRowValue(int i1, int i2, int i3, int i4)  
 {  
 int d3 = Range4.Difference + 1;  
 int d2 = (Range3.Difference + 1) \* d3;  
 int d1 = (Range2.Difference + 1) \* d2;  
  
 int finalIndex = i1 - Range1.L \* d1 +  
 (i2 - Range2.L) \* d2 +  
 (i3 - Range3.L) \* d3 +  
 (i4 - Range4.L);  
  
 return Elements[finalIndex];  
 }  
  
 public TValue GetColumnValue(int i1, int i2, int i3, int i4)  
 {  
 int d2 = Range1.Difference + 1;  
 int d3 = (Range2.Difference + 1) \* d2;  
 int d4 = (Range3.Difference + 1) \* d3;  
  
 int finalIndex = i1 - Range1.L +  
 (i2 - Range2.L) \* d2 +  
 (i3 - Range3.L) \* d3 +  
 (i4 - Range4.L) \* d4;  
  
 return Elements[finalIndex];  
 }  
  
 public TValue GetValueByDefiningVector(int i1, int i2, int i3, int i4)  
 {  
 int finalIndex = i1 \* DefiningVector[10] +  
 i2 \* DefiningVector[11] +  
 i3 \* DefiningVector[12] +  
 i4 \* DefiningVector[13]  
 - DefiningVector[14];  
  
 return Elements[finalIndex];  
 }  
}

