**Аттестция 1 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:

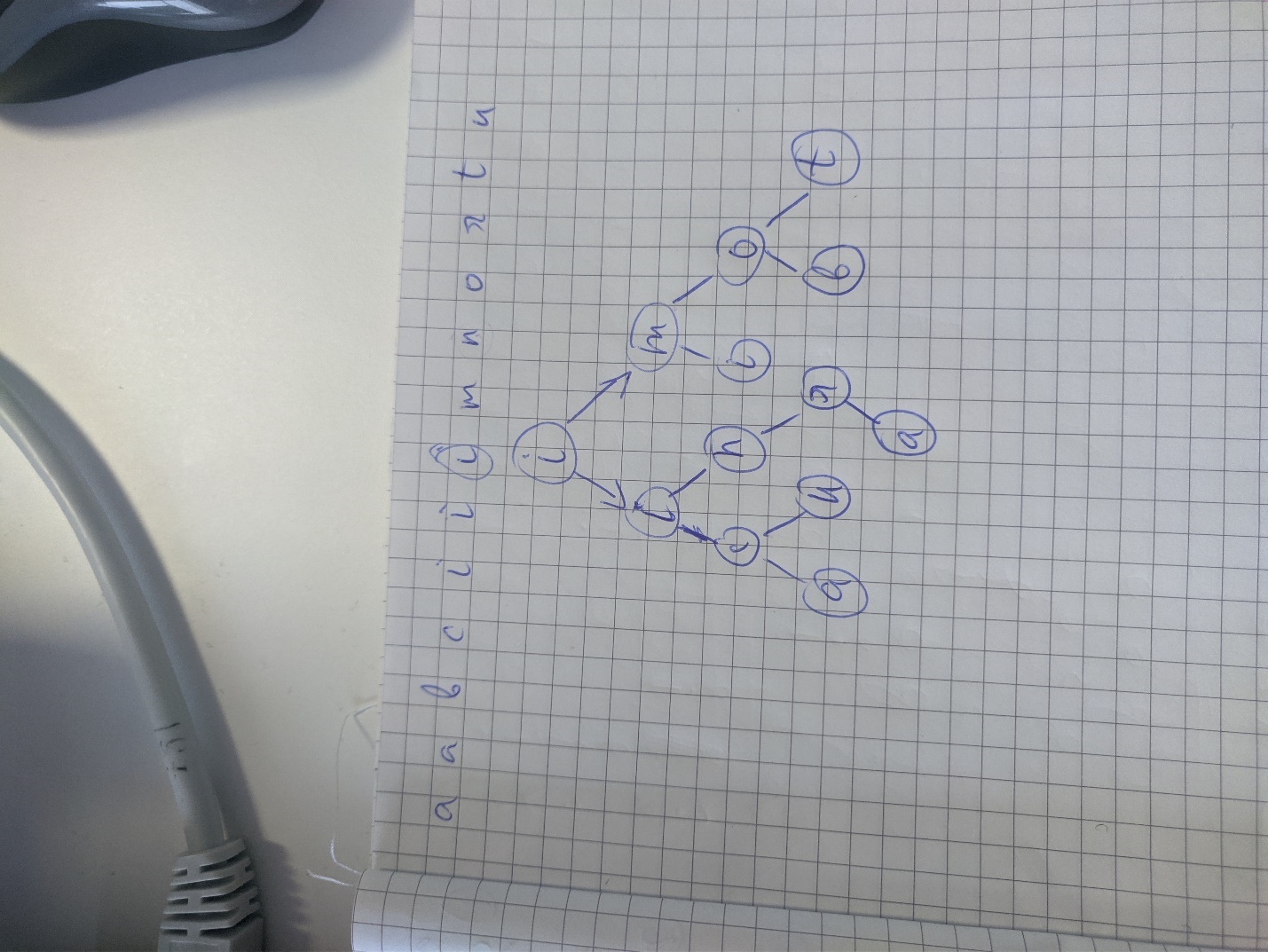
private static char[] GetNameArray() => new[]  
{  
 'c', 'i', 'o', 'b', 'a', 'n', 'u', 'a', 'r', 't', 'i', 'o', 'm', 'i'  
};  
  
private static void Main()  
{  
 char[] myArray = GetNameArray();  
 char[] mySortedArray = GetNameArray().BubbleSort();  
  
 Console.WriteLine("Изначальный массив: ");  
 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:



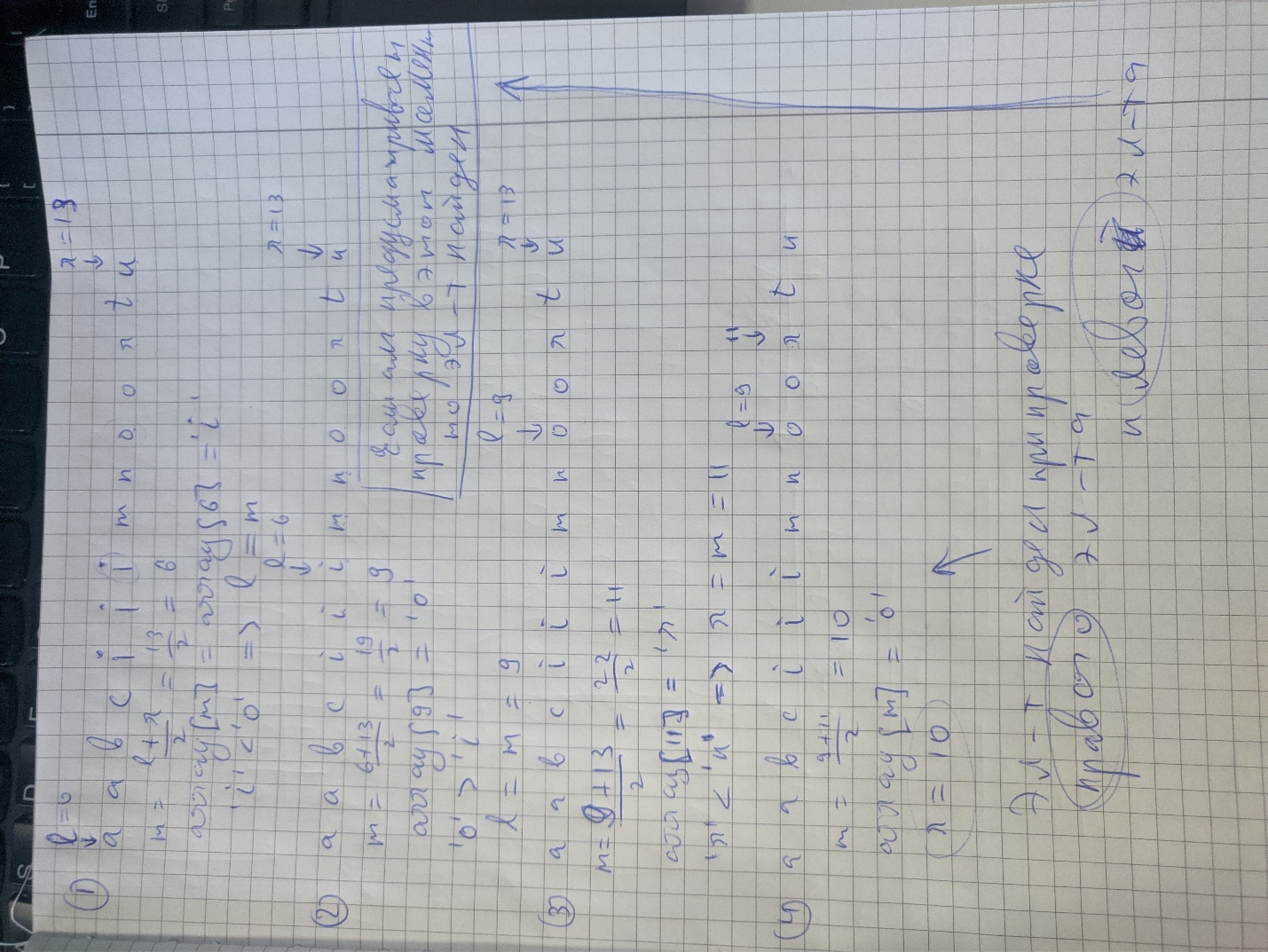
**Задания**

1. Из элементов персонального массива MyArray постройте древовидно структурированный неупорядоченный массив. Нарисуйте бинарное дерево соответствующее построенному массиву. Посчитайте сколько сопоставлений (сравнений) будет выполнено для поиска в этом массиве элемента ‘m’ (до его достижения, либо до достижения конца массива или тупиковой позиции в случае его не-нахождения). (20 б.)



1. Примените метод Бинарного Поиска, элемента ‘o’ 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). (10 б.)

Подробное описание работы метода:



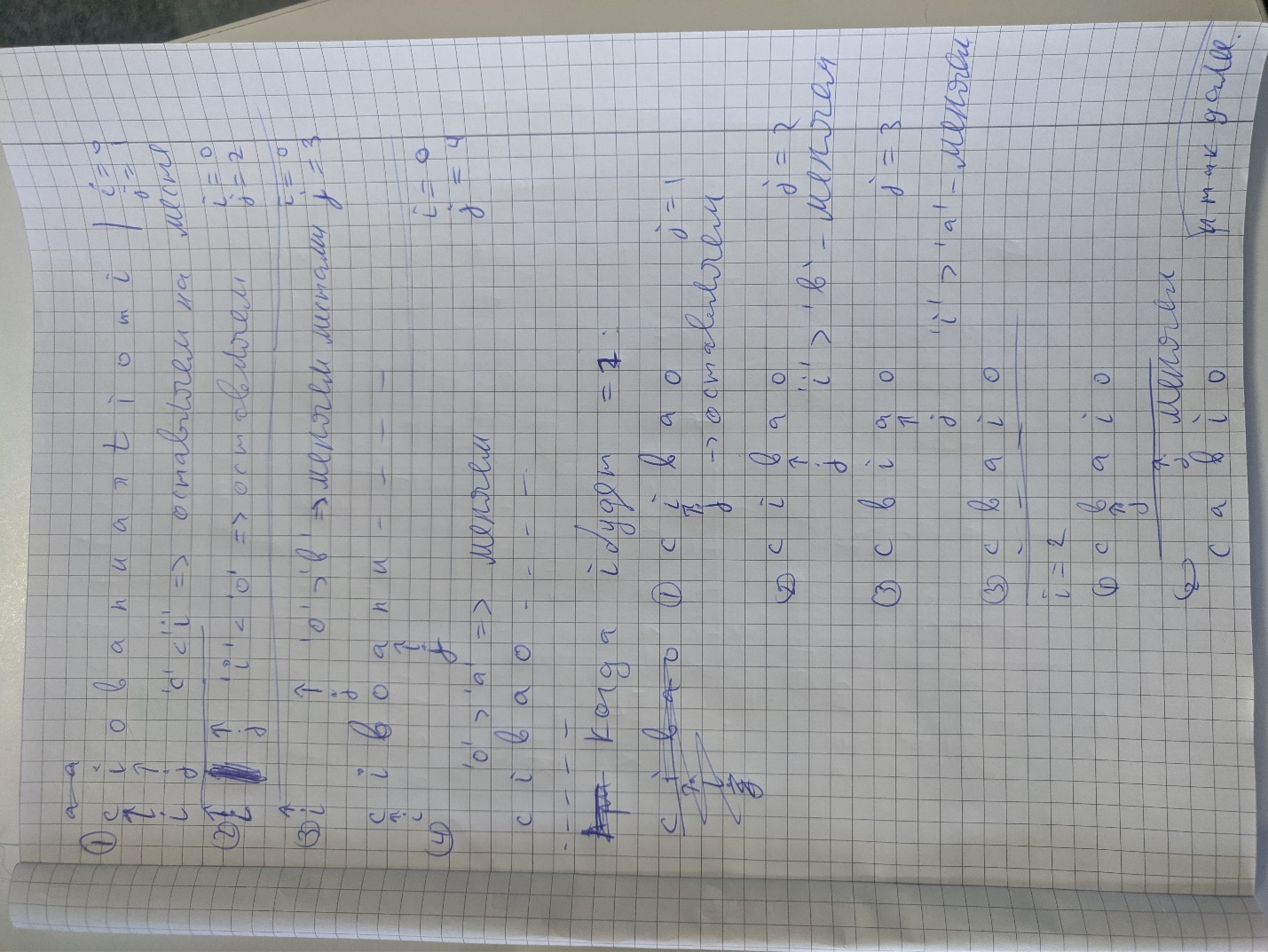
Реализация алгоритма на языке C#:

public static int BinarySearch(this char[] collection, char character)  
{  
 int leftIndex = 0;  
 int rightIndex = collection.Length - 1;  
  
 while (collection[leftIndex] != character)  
 {  
 var middle = (int) Math.Round((leftIndex + rightIndex) / 2.0);  
  
 if (collection[middle] > character)  
 {  
 rightIndex = middle;  
 }  
 else  
 {  
 leftIndex = middle;  
 }  
 }  
  
 return leftIndex;  
}

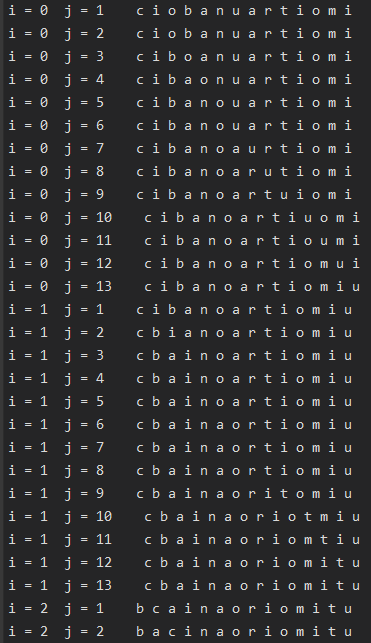
Напомню, что алгоритм работает только с отсортированным массивом.

1. Примените Пузырьковый Метод (BubbleSort) над вашим персональным массивом MyArray, для сортировки элементов в порядке возрастания. Укажите детально каждое прохождение 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.)

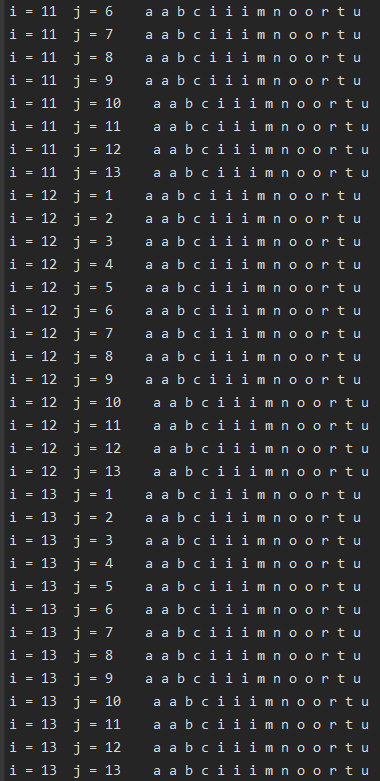
Подробное описание работы алгоритма:



Вывод массива на каждой итерации:



Конец работы алгоритма:



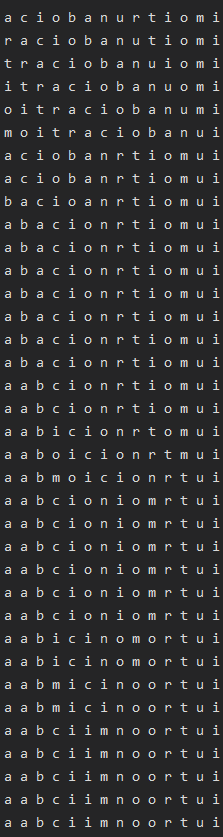
1. Примените Сортировку Вставкой (InsertionSort) над вашим персональным массивом MyArray, для сортировки элементов в порядке возрастания. Укажите детально каждое прохождение массива данным методом. Для каждого прохода, укажите количество сопоставлений и перестановок, которые были произведены для его получения (для исходного массива ничего не указывается). (20 б.)

public static char[] InsertionSort(this char[] collection)  
{  
 for (int i = 1; i < collection.Length; i++)  
 {  
 for (int j = i - 1; j > 0; j--)  
 {  
 if (collection[i] < collection[j])  
 {  
 var current = collection[i];  
  
 for (int k = i; k > j; k--)  
 {  
 collection[k] = collection[k - 1];  
 }  
  
 collection[j] = current;  
 }  
  
 collection.DisplayInline();  
 }  
 }  
  
 return collection;  
}

1. Примените Сортировку Выбором (SelectionSort) над вашим персональным массивом MyArray, для сортировки элементов в порядке возрастания. Укажите детально каждое прохождение массива данным методом. Для каждого прохода, укажите количество сопоставлений и перестановок, которые были произведены для его получения (для исходного массива ничего не указывается). (20 б.)

public static char[] SelectionSort(this char[] collection)  
{  
 for (int i = 0; i < collection.Length; i++)  
 {  
 for (int j = i; j < collection.Length; j++)  
 {  
 if (collection[j] < collection[i])  
 {  
 var c = collection[j];  
 collection[j] = collection[i];  
 collection[i] = c;  
 }  
 }  
 }  
  
 return collection;  
}

1. Опишите алгоритм Быстрой Сортировки (QuickSort) – принцип, шаги, условие остановки, теоретическа сложность и иллюстрируйте выполнение алгоритма для сортировки массива MyArray. (15 б.)



Сложность порядка O(nlogn)

Код программы:

public static char[] QuickSort(this char[] collection)  
{  
 int lastIndex = collection.Length - 1;  
  
 var areas = new List<Tuple<int, int>>  
 {  
 new(0, lastIndex)  
 };  
  
 while (areas.Any())  
 {  
 var (left, right) = areas.First();  
  
 var pivotIndex = (right + left) / 2;  
 var pivot = collection[pivotIndex];  
  
 for (int i = pivotIndex - 1; i >= left; i--)  
 {  
 if (collection[i] > pivot)  
 {  
 var current = collection[i];  
 //Сдвиг всего влево  
 for (int j = i; j < right - 1; j++)  
 {  
 collection[j] = collection[j + 1];  
 }  
  
 collection[right - 1] = current;  
  
 pivotIndex--;  
 }  
 }  
  
 for (int i = pivotIndex + 1; i < right; i++)  
 {  
 if (collection[i] < pivot)  
 {  
 var current = collection[i];  
  
 for (int j = i; j >= left + 1; j--)  
 {  
 collection[j] = collection[j - 1];  
 }  
  
 collection[left] = current;  
  
 pivotIndex++;  
 }  
  
 collection.DisplayInline();  
 }  
  
 areas.RemoveAt(0);  
 if (right - pivotIndex > 1)  
 {  
 var area2 = new Tuple<int, int>(pivotIndex, right);  
 areas.Insert(0, area2);  
 }  
  
 if (pivotIndex - left > 1)  
 {  
 var area1 = new Tuple<int, int>(left, pivotIndex);  
 areas.Insert(0, area1);  
 }  
 }  
  
 return collection;  
}