Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
Федерального государственного бюджетного образовательного учреждения
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Факультет ИТР

Кафедра ПИн

ЛАБОРАТОРНАЯ РАБОТА №2

По Работа с массивами.

Руководитель

Кульков Я.Ю.
(фамилия, инициалы)

(подпись) (дата)

Студент ПИн - 121
(группа)

Ермилов М.В.
(фамилия, инициалы)

(подпись) (дата)

Муром 2023

Лабораторная работа №2

Тема: Работа с массивами.
Цель: Изучение принципов работы массивов на примере алгоритмов сортировки.
Задачи:

1. Вынести каждый из рассмотренных методов в отдельные функции.
2. Сгенерировать три массива с количеством элементов не менее 10 000:
• Полностью отсортированный массив значений
• Полностью случайных набор значений
• Отсортированный массив, в котором первые 10% от общего числа элементов случайные
3. Замерить время сортировки для каждого из массивов используя подготовленные функции.
4. Замерить время сортировки массивов методом sort() класса Arrays
5. Сравнить результаты и сделать выводы.

Таблица 1 - Время работы сортировок

	Случ. массив	Отсорт. массив	Массив 10% случ.
Сорт. пузырьком	0.308 сек.	0.001 сек.	0.012 сек.
Сорт. вставками	0.065 сек.	0 сек.	0.017 сек.
Сорт. выбором	0.034 сек.	0.087 сек.	0.025 сек.
Сорт. слиянием	0.005 сек.	0.002 сек.	0.001 сек.
Быстрая сорт.	0.003 сек.	0.014 сек.	0.016 сек.
Стандартная сорт.	0.011 сек.	0.0 сек.	0.001 сек.

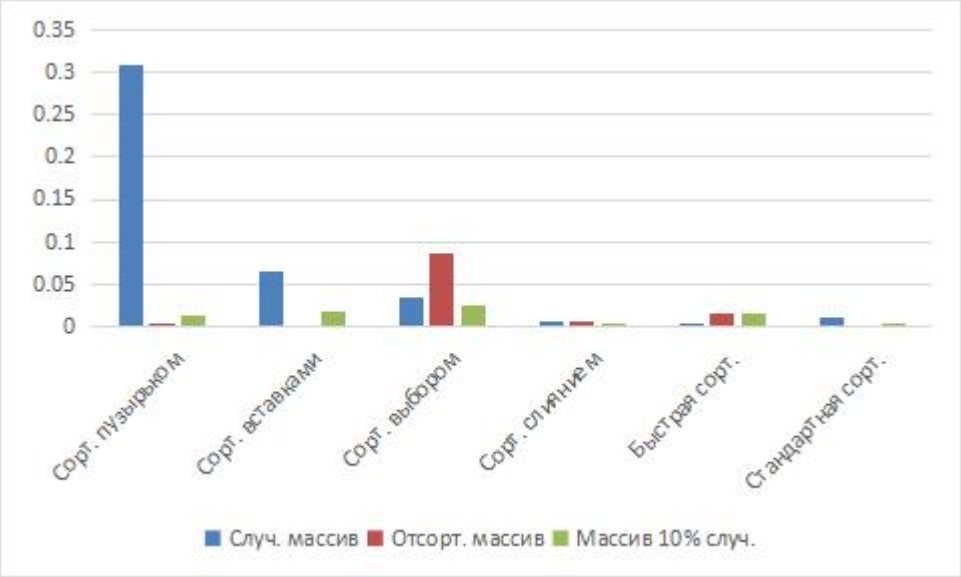

Рисунок 1 - Гистограмма таблицы 1

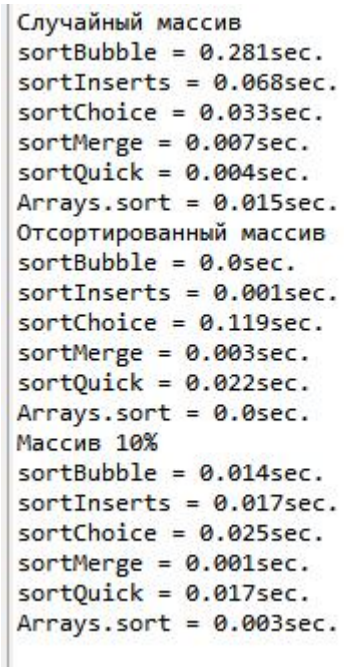

Рисунок 2 - пример работы первого задания

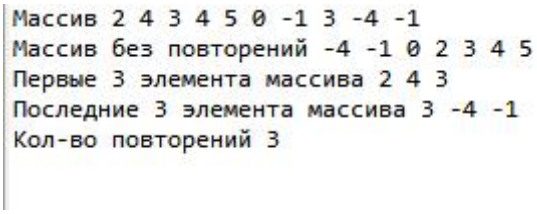

Рисунок 3 - пример работы второго задания

Листинг программы:

```
import java.util.Arrays;
import java.util.Random;

public class Main
{
	static Random random = new Random();
	public static void main(String[] args)
	{
		int[] array = randomArray(10000, -1000, 1000);
		System.out.println("Случайный массив");
		sortAll(array);

		System.out.println("Отсортированный массив");
		Arrays.sort(array);
		sortAll(array);

		System.out.println("Массив 10%");
		randomArray(array, array.length / 10, -1000, 1000);
		sortAll(array);
```

```
        //testII();
    }

    static void testI()
    {
        int[] array = randomArray(10, -5, 5);
        int[] arr = array.clone();
        arrayPrintln(arr);


        arr = array.clone();
        sortBubble(arr);
        arrayPrintln(arr);


        arr = array.clone();
        sortInserts(arr);
        arrayPrintln(arr);


        arr = array.clone();
        sortChoice(arr);
        arrayPrintln(arr);


        arr = array.clone();
        sortMerge(arr);
        arrayPrintln(arr);


        arr = array.clone();
        sortQuick(arr);
        arrayPrintln(arr);
    }

    static void testII()
    {
        int[] array = randomArray(10, -5, 5);
        System.out.print("Массив ");
        arrayPrintln(array);
        System.out.print("Массив без повторений ");
        arrayPrintln(removeDuplicates(array));
        System.out.print("Первые 3 элемента массива ");
        arrayPrintln(getFirst(array, 3));
        System.out.print("Последние 3 элемента массива ");
        arrayPrintln(getLast(array, 3));
        System.out.print("Кол-во повторений ");
        System.out.println(countIdentic(array));
    }

    public static void sortAll(int[] array)
    {
        long start;
        long end;
        double time;


        start = System.currentTimeMillis();
        sortBubble(array.clone());
        end = System.currentTimeMillis();
        time = (end - start) / 1000d;
        System.out.println("sortBubble = " + time + "sec.");
```

```
        start = System.currentTimeMillis();
        sortInserts(array.clone());
        end = System.currentTimeMillis();
        time = (end - start) / 1000d;
        System.out.println("sortInserts = " + time + "sec.");


        start = System.currentTimeMillis();
        sortChoice(array.clone());
        end = System.currentTimeMillis();
        time = (end - start) / 1000d;
        System.out.println("sortChoice = " + time + "sec.");


        start = System.currentTimeMillis();
        sortMerge(array.clone());
        end = System.currentTimeMillis();
        time = (end - start) / 1000d;
        System.out.println("sortMerge = " + time + "sec.");


        start = System.currentTimeMillis();
        sortQuick(array.clone());
        end = System.currentTimeMillis();
        time = (end - start) / 1000d;
        System.out.println("sortQuick = " + time + "sec.");


        start = System.currentTimeMillis();
        Arrays.sort(array.clone());
        end = System.currentTimeMillis();
        time = (end - start) / 1000d;
        System.out.println("Arrays.sort = " + time + "sec.");
    }



    public static void randomArray(int[] array, int randCount, int min, int max)
    {
        if(randCount <= array.length)
        {
            if(min > max)
            {
                int a = max;
                max = min;
                min = a;
            }
            max -= min - 1;
            for(int i = 0; i < randCount; i++)
            {
                array[i] = random.nextInt(max) + min;
            }
        }
    }
    public static int[] randomArray(int length, int min, int max)
    {
        if(min > max)
        {
            int a = max;
            max = min;
```

```
                min = a;
        }
        max -= min - 1;
        int[] array = new int[length];
        for(int i = 0; i < length; i++)
        {
                array[i] = random.nextInt(max) + min;
        }
        return array;
}
public static void arrayPrintln(int[] array)
{
        for(int i = 0; i < array.length; i++)
        {
                System.out.print(array[i] + " ");
        }
        System.out.println();
}

public static void sortBubble(int[] array)
{
        boolean sorted = false;
        int temp;
        while(!sorted)
        {
                sorted = true;
                for (int i = 0; i < array.length - 1; i++)
                {
                        if (array[i] > array[i+1])
                        {
                                temp = array[i];
                                array[i] = array[i+1];
                                array[i+1] = temp;
                                sorted = false;
                        }
                }
        }
}
public static void sortInserts(int[] array)
{
        for (int i = 1; i < array.length; i++)
        {
                int current = array[i];
                int j = i - 1;
                while(j >= 0 && current < array[j])
                {
                        array[j+1] = array[j];
                        j--;
                }
                array[j+1] = current;
        }
}
public static void sortChoice(int[] array)
{
        for (int i = 0; i < array.length; i++)
        {
                int min = array[i];
                int minId = i;
                for (int j = i+1; j < array.length; j++)
                {
                        if (array[j] < min)
                        {
```

```
                        min = array[j];
                        minId = j;
                    }
                }
                // замена
                int temp = array[i];
                array[i] = min;
                array[minId] = temp;
        }
    }
    public static void sortMerge(int[] array)
    {
        sortMerge(array, 0, array.length - 1);
    }
    public static void sortQuick(int[] array)
    {
        sortQuick(array, 0, array.length - 1);
    }

    public static int[] removeDuplicates(int[] array)
    {
        if (array == null || array.length == 0)
        {
            return array;
        }
        array = array.clone();
        Arrays.sort(array);
        int[] uniqueValues = new int[array.length];
        int uniqueIndex = 0;

        for (int i = 0; i < array.length; i++)
        {
            if (i == 0 || array[i] != array[i - 1])
            {
                uniqueValues[uniqueIndex] = array[i];
                uniqueIndex++;
            }
        }
        int[] arr = new int[uniqueIndex];
        for(int i = 0; i < uniqueIndex; i++)
        {
            arr[i] = uniqueValues[i];
        }
        return arr;
    }
    public static int[] getFirst(int[] array, int n)
    {
        int[] result = new int[n];
        System.arraycopy(array, 0, result, 0, n);
        return result;
    }
    public static int[] getLast(int[] array, int n)
    {
        int[] result = new int[n];
        System.arraycopy(array, array.length - n, result, 0, n);
        return result;
    }
    public static int countIdentic(int[] array)
    {
      return array.length - removeDuplicates(array).length;
    }
```

```
	static void sortMerge(int[] array, int left, int right)
	{
		if (right <= left) return;
		int mid = (left+right)/2;
		sortMerge(array, left, mid);
		sortMerge(array, mid+1, right);
		merge(array, left, mid, right);

	}
	static void merge(int[] array, int left, int mid, int right)
	{
		int lengthLeft = mid - left + 1;
		int lengthRight = right - mid;
		int leftArray[] = new int [lengthLeft];
		int rightArray[] = new int [lengthRight];
		// копируем отсортированные массивы во временные
		for (int i = 0; i < lengthLeft; i++)
			leftArray[i] = array[left+i];
		for (int i = 0; i < lengthRight; i++)
			rightArray[i] = array[mid+i+1];
		// итераторы содержат текущий индекс временного подмассива
		int leftIndex = 0;
		int rightIndex = 0;
		// копируем из leftArray и rightArray обратно в массив
		for (int i = left; i < right + 1; i++)
		{
			// если остаются нескопированные элементы в R и L, копируем
минимальный
			if (leftIndex < lengthLeft && rightIndex < lengthRight)
			{
				if (leftArray[leftIndex] < rightArray[rightIndex])
				{
					array[i] = leftArray[leftIndex];
					leftIndex++;
				}
				else
				{
					array[i] = rightArray[rightIndex];
					rightIndex++;
				}
			}
			// если все элементы были скопированы из rightArray, скопировать
остальные из leftArray
			else if (leftIndex < lengthLeft)
			{
				array[i] = leftArray[leftIndex];
				leftIndex++;
			}
			// если все элементы были скопированы из leftArray, то скопировать
остальные из rightArray
			else if (rightIndex < lengthRight)
			{
				array[i] = rightArray[rightIndex];
				rightIndex++;
			}
		}
	}
	static int partition(int[] array, int begin, int end) {
		int pivot = end;
		int counter = begin;
```

```
            for (int i = begin; i < end; i++)
            {
                  if (array[i] < array[pivot])
                  {
                        int temp = array[counter];
                        array[counter] = array[i];
                        array[i] = temp;
                        counter++;
                  }
            }
            int temp = array[pivot];
            array[pivot] = array[counter];
            array[counter] = temp;
            return counter;
      }
      static void sortQuick(int[] array, int begin, int end)
      {
            if (end <= begin) return;
            int pivot = partition(array, begin, end);
            sortQuick(array, begin, pivot-1);
            sortQuick(array, pivot+1, end);
      }

}
```