

ВВЕДЕНИЕ

Сортировка - это процесс упорядочивания данных в определенном порядке по тем или иным признакам. С точки зрения программирования, сортировка относится к алгоритмам, которые переставляют элементы в некотором наборе данных, чтобы они следовали в каком-либо порядке, например, по возрастанию или убыванию.

Сортировка является одной из наиболее распространенных и важных задач в области алгоритмов и программирования, поскольку она позволяет эффективно упорядочивать данные, что облегчает их дальнейший поиск, обработку и анализ.

Существует множество различных сортировочных алгоритмов, каждый из которых имеет свои преимущества и недостатки в зависимости от различных факторов.

В данной лабораторной работе требуется изучить и реализовать алгоритмы сортировки: сортировка Шелла, быстрая сортировка, сортировка подсчетами.

Для достижения поставленных целей лабораторной работы необходимо выполнить следующие задачи:

- рассмотреть и изучить сортировки Шелла и подсчетами, а также быструю сортировку;
- реализовать каждую из трёх сортировок;
- составить блок-схемы алгоритмов;
- рассчитать трудоемкость алгоритмов сортировки;
- экспериментально проверить работу алгоритмов;
- на основании проделанной работы сделать выводы.

1 Аналитическая часть

1.1 Сортировка Шелла

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии gap. После этого процедура повторяется для некоторых меньших значений d , а завершается сортировка Шелла упорядочиванием элементов при $gap = 1$ – то есть обычной сортировкой вставками. Эффективность сортировки Шелла в некоторых ситуациях обеспечивается тем, что элементы «быстрее» встают на свои места [1, с.269].

1.2 Быстрая сортировка

Общая идея алгоритма быстрой сортировки состоит в том, что необходимо сначала выбрать из массива элемент, называемый опорным - это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность. Далее нужно сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие». Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы [2, с.198].

1.3 Сортировка подсчетом

Алгоритм сортировки подсчетом является методом сортировки, основанным на подсчете количества элементов с определенными

значениями. Он эффективно справляется с сортировкой целых чисел в заданном диапазоне.

Суть сортировки подсчетом состоит в том, что сначала происходит подсчет количества вхождений каждого значения в исходном массиве. Создается массив счетчиков, где каждый индекс соответствует значению элемента, а значение в ячейке – это количество вхождений этого значения в массиве.

Затем, с помощью этого массива счетчиков, вычисляется позиция каждого уникального элемента в отсортированном массиве. Это достигается путем суммирования значений в ячейках массива счетчиков до текущей позиции. Далее создается дополнительный массив, куда помещаются элементы из исходного массива в соответствующие позиции, определенные с помощью массива счетчиков.

Финальным шагом является замена исходного массива отсортированным массивом.

Вывод

На данном этапе была изучена теоретическая база сортировок. Кратко описаны идея и принцип работы каждой из трех представленных сортировок.

2 Конструкторская часть

2.1 Разработка алгоритмов

В соответствии с аналитической частью можно составить блок-схемы каждого алгоритма сортировки.

На рисунке 1 представлена блок-схема алгоритма сортировки Шелла.

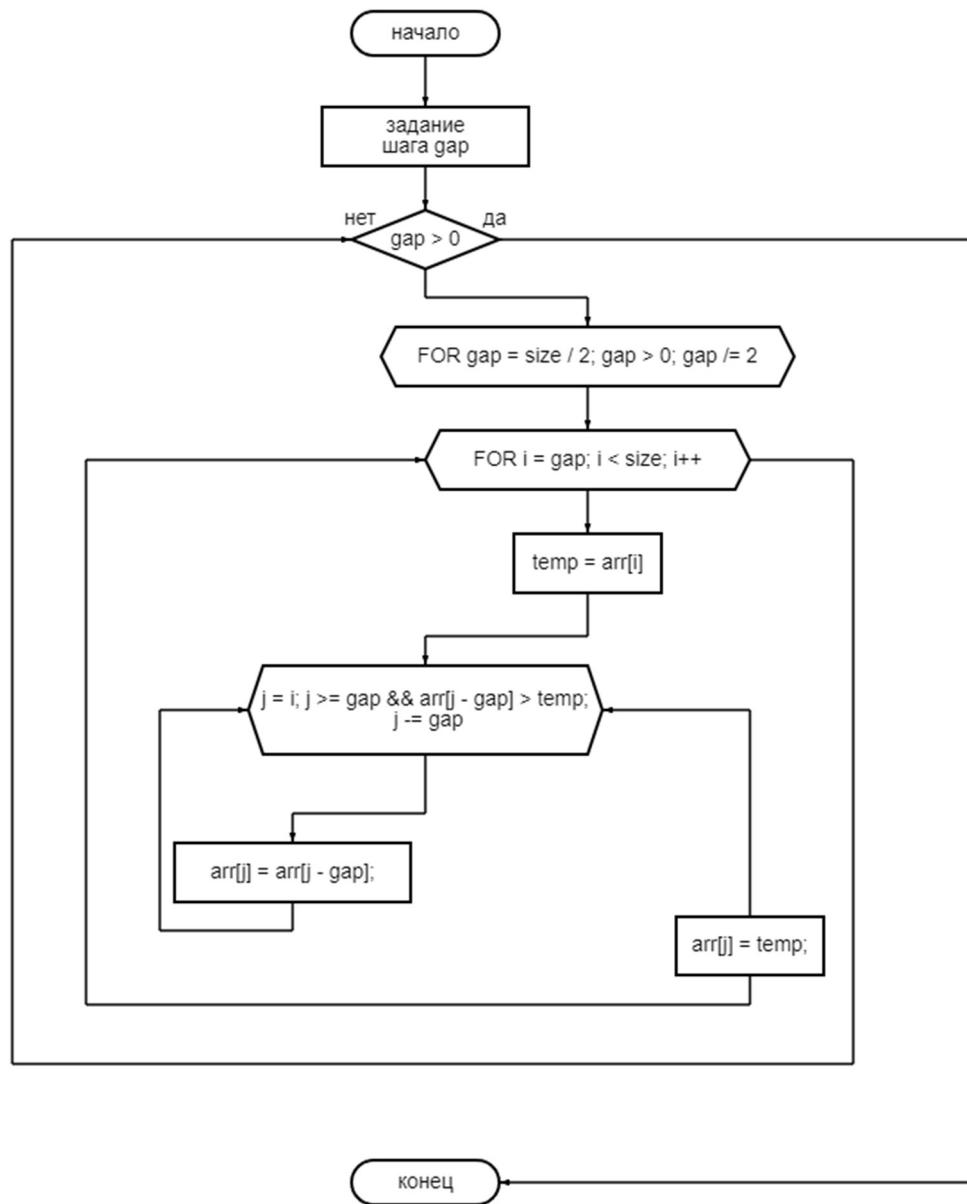


Рисунок 1 –Блок-схема алгоритма сортировки Шелла

На рисунке 2 приведена блок-схема алгоритма быстрой сортировки.

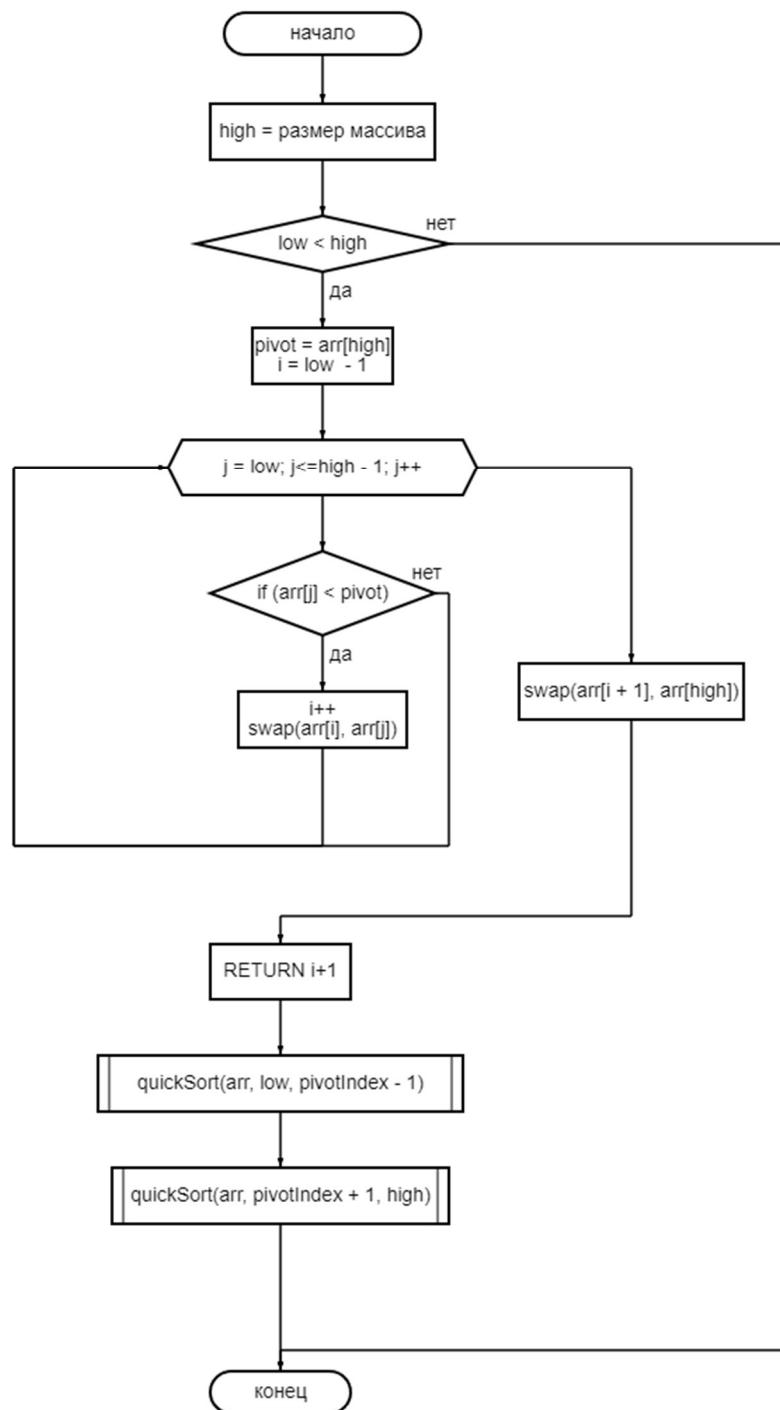


Рисунок 2 – Блок-схема алгоритма быстрой сортировки

На рисунке 3 приведена блок-схема алгоритма сортировки подсчетами.

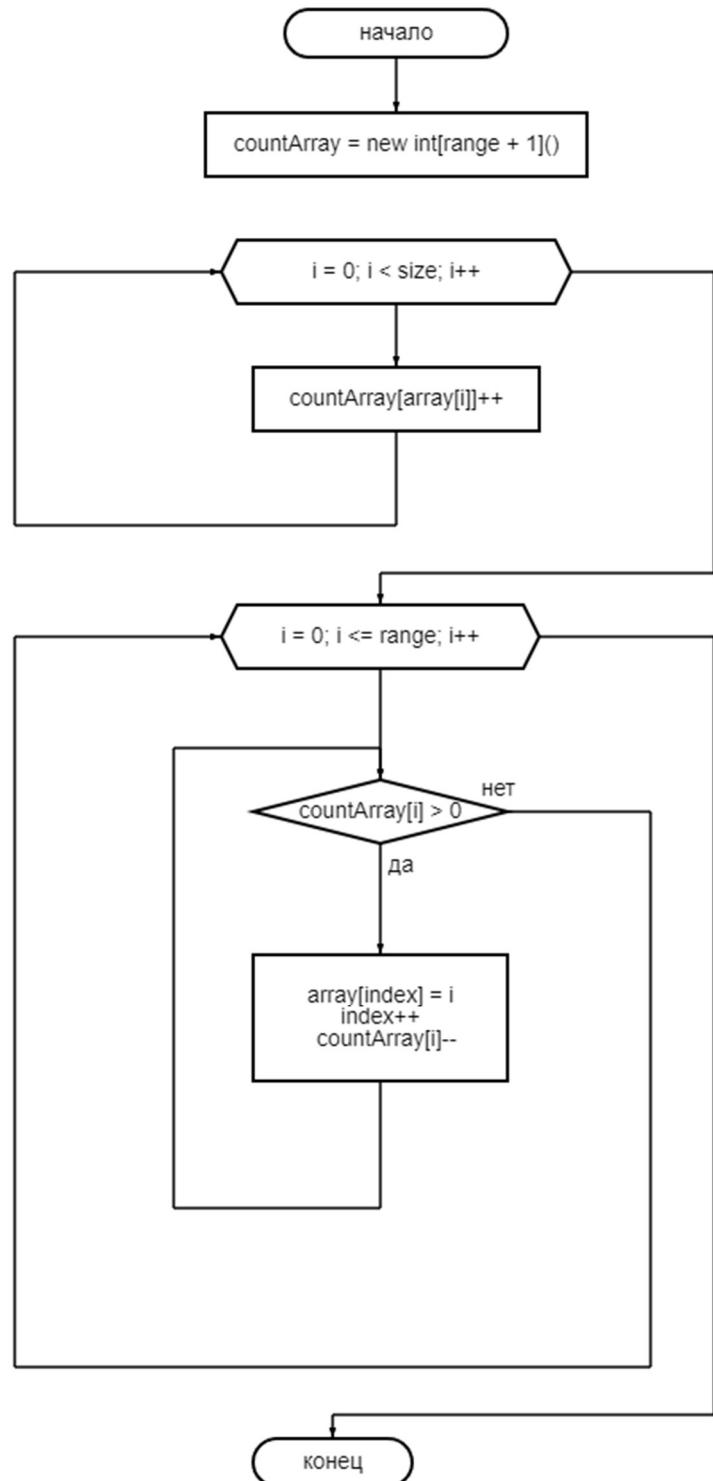


Рисунок 3 – Блок-схема алгоритма сортировки подсчетами

2.2 Трудоемкость алгоритмов

Для последующего вычисления трудоемкости алгоритмов необходимо ввести модель вычислений:

- 1) Пусть: +, -, /, %, ==, !=, <, >, <=, >=, [], *, ++, -- – трудоемкость 1;
- 2) Трудоемкость оператора выбора if условие then A else B рассчитывается как на рисунке 4;

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases}$$

Рисунок 4 - Трудоемкость оператора выбора if условие then A else B

- 3) Трудоемкость цикла for рассчитывается как на рисунке 5;

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инициализации}} + f_{\text{сравнения}})$$

Рисунок 5 - Трудоемкость оператора for

- 4) Трудоемкость вызова функции равна 0.

Произведём вычисление трудоёмкости алгоритма для каждой сортировки. В таблице 1 приведены трудоемкости алгоритмов для лучшего и худшего случаев.

Таблица 1 – Расчёт трудоемкости алгоритмов сортировки

Вид сортировки	Трудоемкость алгоритма сортировки	
	Худший случай	Лучший случай
<i>Сортировка Шелла</i>	$O(n * (\log(n))^2)$	$O(n)$
<i>Быстрая сортировка</i>	$O(n^2)$	$O(n * \log(n))$
<i>Сортировка подсчетами</i>	$O(n+M)$	$O(n)$

Вывод

С учетом теоретических данных, полученных из аналитического раздела, разработаны блок-схемы необходимых алгоритмов сортировки. Кроме того, проведена теоретическая оценка их трудоемкости, которая согласуется с предполагаемыми сложностями данных алгоритмов.

3 Экспериментальная часть

3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход подаётся массив из целых чисел;
- на выходе — отсортированные при помощи сортировок массивы.

3.2 Технические характеристики

Технические характеристики устройства, на котором происходило тестирование:

- Операционная система: Windows 10 Pro 64 бит
- Память: 16 ГБ
- Процессор: Intel Core Pentium G4560 3.5 ГГц

Тестирование проводилось на ПК, включенном в сеть электропитания. Во время тестирования ПК был нагружен только встроенными приложениями окружения рабочего стола, окружением рабочего стола, а также непосредственно системой тестирования.

3.3 Тестирование программ

В таблице 2 приведены функциональные тесты для программ, реализующих алгоритмы сортировки – все тесты были пройдены успешно.

Таблица 2 – Тестирование функций

Входной массив	Выходной массив	Ожидаемый результат
6, -100, 300, 500, -1000, 0, 1, 23	-1000, -100, 0, 1, 6, 23, 300, 500	-1000, -100, 0, 1, 6, 23, 300, 500
324, 0, -1, 1, -2, 2, 137, 65, -988, 13	-988, -2, -1, 0, 1, 2, 13, 65, 137, 324	-988, -2, -1, 0, 1, 2, 13, 65, 137, 324
1, 1, 2, 3, 3, -1, -1, 0, 0, -77	-77, -1, -1, 0, 0, 1, 1, 2, 3, 3	-77, -1, -1, 0, 0, 1, 1, 2, 3, 3
{Пустой массив}	{Пустой массив}	{Пустой массив}
1, 75, 50, 0	0, 1, 50, 75	0, 1, 50, 75
0	0	0

В таблице 3 приведено время выполнения каждой сортировки.

Таблица 3 – Время выполнения программ сортировок

Размер массива	Среднее время работы, с		
	Сортировка Шелла	Быстрая сортировка	Сортировка подсчетами
10	0.003 с	0.021 с	0.00000012 с
100	0.02 с	0.177 с	0.000071 с
1000	0.15 с	2.27 с	0.001 с

Стоит уточнить, что так как в сортировке подсчетами возможный диапазон чисел составляет только неотрицательные целые числа, было принято решение проводить тесты на среднее время работы сортировок с массивом, включающего в себя целые числа от 0 до 100 включительно.

Вывод

В результате эксперимента были разработаны и протестированы сортировка Шелла, сортировка подсчетами, быстрая сортировка. По итогам проводимого тестирования на время с указанными параметрами на самой быстрой сортировкой является сортировка подсчетами, что соответствует теории из аналитической части работы: сортировка подсчетами в сравнении с другими сортировками отлично себя зарекомендовала, поскольку тестирование проводилось на относительно небольшом промежутке с неотрицательными целыми значениями чисел, меньших 100, что для данной сортировки будет являться лучшим условием.

Несмотря на то, что быстрая сортировка является одним из самых быстрых известных универсальных алгоритмов сортировки, она обладает рядом недостатков. По результатам эксперимента можно добавить, что использовать быструю сортировку имеет смысл с различными модификациями в зависимости от поставленной задачи.

Сортировка Шелла по итогам тестирования оказалась результативнее быстрой сортировки, что в целом соответствует вычисленным трудоемкостям алгоритмов.

Обобщая, можно сказать, что выводы по итогу эксперимента в достаточно полной мере соответствуют теоретическим трудоемкостям.

ЗАКЛЮЧЕНИЕ

В процессе выполнения лабораторной работы были проведены анализ, подробное изучение, реализация и тестирование различных алгоритмов сортировки, таких как сортировка Шелла, быстрая сортировка и сортировка подсчетом, были выполнены следующие задачи:

- рассмотрены и изучены сортировки Шелла и подсчетами, а также быструю сортировку;
- реализована каждая из трёх сортировок;
- составлена блок-схема алгоритмов;
- рассчитана трудоемкость алгоритмов сортировки;
- экспериментально проверена работа алгоритмов;
- на основании проделанной работы сделаны выводы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Роберт Седжвик. Алгоритмы на C++. Фундаментальные алгоритмы и структуры данных. Algorithms in C++. — М.: «Вильямс», 2011. — 269 с.
2. Т.Кормен, Ч.Лейзерсон, Р.Ривест, К.Штайн – Алгоритмы. Построение и анализ, 2013. – 198 с.
3. Полный справочник по C++, 4-е издание C++: The Complete Reference, 4th Edition. — М.: «Вильямс», 2011. — 591 с.
4. Сортировка Шелла [Электронный ресурс]. Режим доступа:
<https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%BA%D0%BB%D0%BB%D0%B0> (Дата обращения: 23.05.2023)
5. С.Пратта. Язык программирования C++. Лекции и упражнения, 2012. – 196 с.

ПРИЛОЖЕНИЕ А

Листинг программы

Листинг 1 – Алгоритм сортировки Шелла

```
#include <iostream>

using namespace std;

short int errs = 0;

//Сортировка Шелла
void shellSort(int arr[], int size)
{
    //Начало с большого промежутка и дальнейшее уменьшение
    for (int gap = size / 2; gap > 0; gap /= 2)
    {
        //Сортировка вставки по элементам, разделенным промежутком
        for (int i = gap; i < size; i++)
        {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
            {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }
}

//Функция вывода массива
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

void inputArray(int *&arr, int &size)
{
    std::cout << "Enter array size: ";
    std::cin >> size;
    if(size != 0){

        //Выделение памяти для массива
        arr = new int[size];

        std::cout << "Enter array: ";
        for (int i = 0; i < size; i++)
        {
            std::cin >> arr[i];
        }
    } else {

```

```

        errs = 1;
        std::cout << "ENTERED EMPTY ARRAY " << std::endl;
    }
}

int main()
{
    int *arr;
    int size;

    inputArray(arr, size);
    if(errs == 0) {
        std::cout << "Input array: ";
        printArray(arr, size);

        shellSort(arr, size);

        std::cout << "Output array: ";
        printArray(arr, size);

        // Освобождение памяти
        delete[] arr;
    }
    return 0;
}

```

Листинг 2 – Алгоритм быстрой сортировки

```

#include <iostream>

using namespace std;

short int errs = 0;

//Функция обмена значениями двух переменных
void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}

//Функция для разбиения массива на разделы и возврата сводного индекса
int partition(int arr[], int low, int high)
{
    //Самый правый элемент выбирается в качестве опорного
    int pivot = arr[high];
    int i = low - 1;

    //Разбиение массива на разделы
    for (int j = low; j <= high - 1; j++)
    {
        if (arr[j] < pivot)
        {
            i++;
            swap(arr[i], arr[j]);
        }
    }
}

```

```

        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

//Быстрая сортировка
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pivotIndex = partition(arr, low, high);

        //Рекурсивная сортировка "подмассивов"
        quickSort(arr, low, pivotIndex - 1);
        quickSort(arr, pivotIndex + 1, high);
    }
}

//Функция вывода массива
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

void inputArray(int *&arr, int &size)
{
    std::cout << "Enter array size: ";
    std::cin >> size;
    if(size != 0){

        //Выделение памяти для массива
        arr = new int[size];

        std::cout << "Enter array: ";
        for (int i = 0; i < size; i++)
        {
            std::cin >> arr[i];
        }
    } else {
        errs = 1;
        std::cout << "ENTERED EMPTY ARRAY " << std::endl;
    }
}

int main()
{
    int *arr;
    int size;

    inputArray(arr, size);

    if(errs == 0) {
        std::cout << "Input array: ";
        printArray(arr, size);
    }
}

```

```

        quickSort(arr, 0, size - 1);

        std::cout << "Output array: ";
        printArray(arr, size);

        //Освобождение памяти
        delete[] arr;
    }

    return 0;
}

```

Листинг 3 – Алгоритм сортировки подсчетами

```

#include <iostream>

using namespace std;

short int errs = 0;

void countingSort(int array[], int size, int range)
{
    // Создание массива для подсчета частоты элементов
    int *countArray = new int[range + 1]();

    // Подсчет частоты элементов
    for (int i = 0; i < size; i++)
    {
        countArray[array[i]]++;
    }

    // Восстановление отсортированного массива
    int index = 0;
    for (int i = 0; i <= range; i++)
    {
        while (countArray[i] > 0)
        {
            array[index] = i;
            index++;
            countArray[i]--;
        }
    }

    // Освобождение памяти
    delete[] countArray;
}

void inputArray(int *&arr, int &size)
{
    std::cout << "Enter array size: ";
    std::cin >> size;
    if(size != 0){

        //Выделение памяти для массива
        arr = new int[size];

        std::cout << "Enter array: ";
    }
}

```

```

        for (int i = 0; i < size; i++)
        {
            std::cin >> arr[i];
        }
    } else {
        errs = 1;
        std::cout << "ENTERED EMPTY ARRAY " << std::endl;
    }
}

void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
}

int main()
{
    int size;
    int *array;

    inputArray(array, size);

    if(errs == 0) {
        //Нахождение максимального элемента в массиве
        int maxElement = array[0];
        for (int i = 1; i < size; i++)
        {
            if (array[i] > maxElement)
            {
                maxElement = array[i];
            }
        }

        std::cout << "Input array ";
        printArray(array, size);

        //Сортировка подсчетом
        countingSort(array, size, maxElement);

        //Вывод отсортированного массива
        std::cout << "Output array ";
        printArray(array, size);

        // Освобождение памяти
        delete[] array;
    }

    return 0;
}

```