Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Информатика

Семинар 5

Алгоритмы работы с массивами.

Материал подготовили:

Горбунов Д. В.

Кудрявцев М. А.

Тассов К. Л.

2022 г.

Содержание

1. Нахождение суммы элементов массива 3

2. Нахождение экстремумов в массиве 4

2.1 Нахождение значения максимального элемента 4

2.2 Нахождение индекса максимального элемента 6

2.3 Нахождение адреса максимального элемента 6

3. Замена элементов местами в массиве 7

4. Удаление элемента из массива 8

5. Вставка элемента в массив 9

6. Группировка элементов в массиве 9

7. Сортировка элементов массива 13

7.1 Полный перебор 13

7.2 Метод поиска минимального элемента 15

7.3 Метод “пузырек” 17

7.4 Метод вставки 18

Лабораторная работа 20

1. Нахождение суммы элементов массива

Рассмотрим алгоритм нахождения суммы элементов массива, показанный на рисунке 1.1.

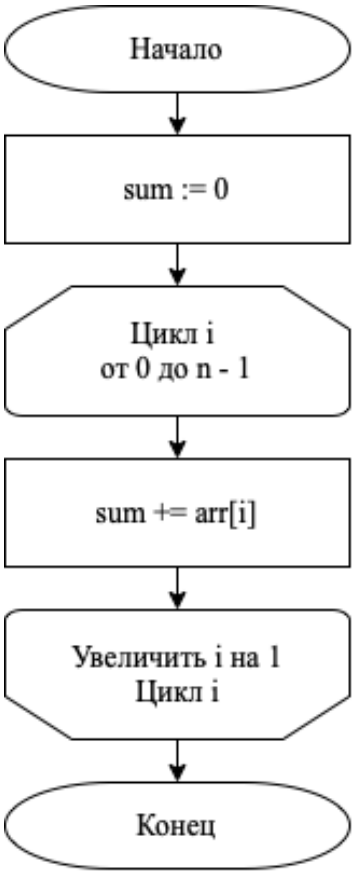

Рисунок 1.1. Схема алгоритма нахождения суммы элементов массива.

В переменной sum накапливаем сумму элементов массива, просматривая в цикле все элементы. Надо использовать правило: если мы что-то определяем то, по возможности, это надо инициализировать. Если этого не сделать, то в переменной может находиться «мусор» из памяти. Поэтому переменную sum предварительно инициализируем 0 или значением 1-го элемента массива (тогда просматривать со 2-го элемента).

Пример функции нахождения суммы элементов массива:

```
double sumElem(const double* arr, int n) {
    double sum = 0;
    for (int i = 0; i < n; i++)
        sum += arr[i];
    return sum;
}
```

2. Нахождение экстремумов в массиве

Рассмотрим алгоритм нахождения максимального элемента массива. Можно искать значение, индекс или адрес элемента. Если индекс или адрес элемента, то надо уточнять какого: первого, последнего и т.п. По индексу и адресу можно всегда найти значение, а по значению надо найти индекс, то надо организовывать поиск элемента.

2.1 Нахождение значения максимального элемента

Схема алгоритма поиска максимума в массиве показана на рисунке 2.1.

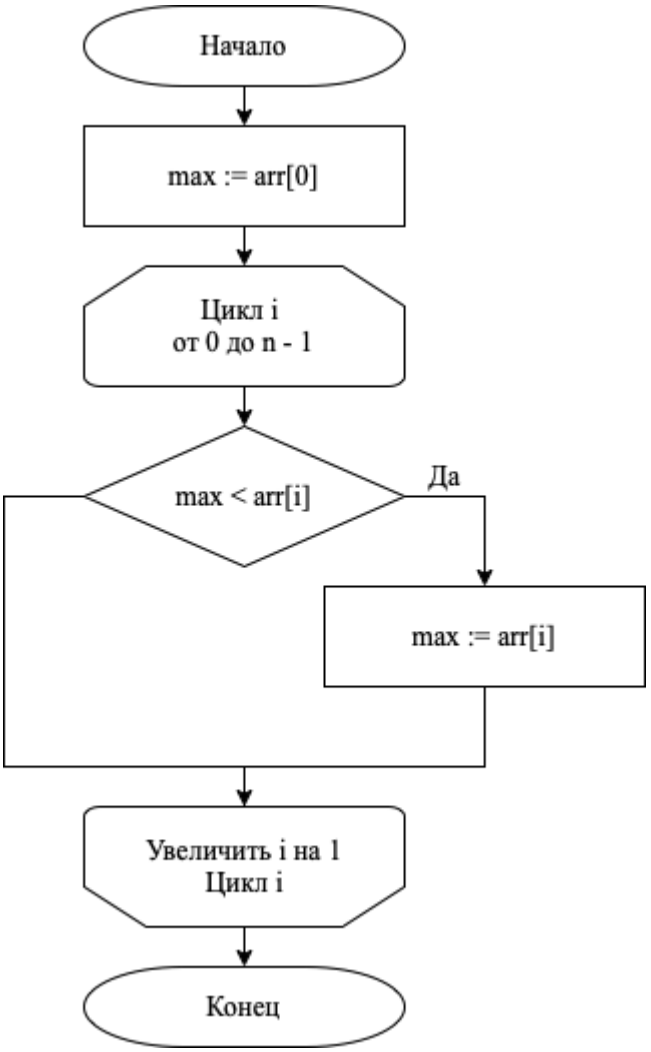

Рисунок 2.1. Схема алгоритма нахождения максимума в массиве.

Предположим, что максимальный элемент 1-ый. Присвоим переменной max его значение и будем сравнивать max со значениями всех остальных элементов. Если находится больший присваиваем его значение max.

Пример функции нахождения первого максимума в массиве:

```
double maxElem(const double *arr, int n) {
    double max = arr[0];
    for (int i = 1; i < n; i++) {
        if (max < arr[i])
```

```
            max = arr[i];
    }
    return max;
}
```

2.2 Нахождение индекса максимального элемента

Пример функции нахождения индекса 1-го максимума в массиве:

```
int indexOfMaxElem(const double* arr, int n) {
    int indexMax = 0;
    for (int i = 1; i < n; i++) {
        if (arr[indexMax] < arr[i])
            indexMax = i;
    }
    return indexMax;
}
```

2.3 Нахождение адреса максимального элемента

Пример функции нахождения адреса 1-го максимума в массиве:

```
double* pointerOfMaxElem(double* arr, int n) {
    double* pointerMax = arr;
    for (double *p = arr + 1; p - arr < n; p++) {
        if (*pointerMax < *p)
            pointerMax = p;
    }
    return pointerMax;
```

```
}
```

3. Замена элементов местами в массиве

Для замены элементов вводят вспомогательную переменную и в неё заносят значение одного из параметров, например, первого. Затем изменяют значение этого параметра на значение другого, а в другой заносят значение вспомогательной переменной. Замена элементов показана на рисунке 3.1.

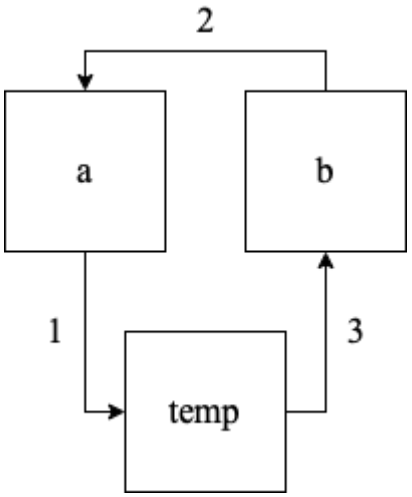

Рисунок 3.1. Замена элементов через вспомогательную переменную.

Пример функции замены элементов через вспомогательную переменную:

```
void swap(double* pa, double* pb) {
    double temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```

4. Удаление элемента из массива

Рассмотрим алгоритм удаления k-го элемента из массива с сохранением порядка следования элементов. Для этого на его место запишем следующий, а на место следующего – следующий за ним, и так далее до предпоследнего. В функцию удаления элемента имеет смысл передавать адрес удаляемого элемента и количество элементов до конца массива.

k-ый элемент показан на рисунке 4.1.

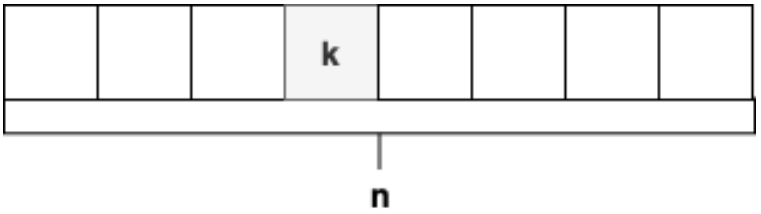

Рисунок 4.1. k-ый элемент.

Пример функции удаления элемента из массива:

```
int deleteElem(double* arr, int n) {
    for (int i = 0; i < n - 1; i++)
        arr[i] = arr[i + 1];
    return n - 1;
}
```

Вызов функции удаления элемента:

```
n = deleteElem(arr + k, n – k) + k;
```

5. Вставка элемента в массив

Под вставкой элемента подразумевается вставка на конкретное место в массиве без нарушения следования других элементов массива. Прежде чем вставлять элемент необходимо подготовить для него место, сдвинув элементы, начиная с места на которое надо вставить элемент, вправо на одну позицию.

Пример функции вставки элемента в массив:

```
int insertElem(double* arr, int n, double elem) {
    for (int i = n - 1; i >= 0; i--)
        arr[i + 1] = arr[i];
    arr[0] = elem;
    return n + 1;
}
```

Вызов функции вставки элемента:

```
n = insertElem(arr + k, n - k, elem) + k;
```

6. Группировка элементов в массиве

Рассмотрим алгоритм группировки элементов в массиве с перебором всех элементов, в котором положительные элементы должны быть в начале, все остальные после них.

Алгоритм группировки, где в массиве в начале лежат положительные числа, а в конце отрицательные и нули, показан на рисунке 6.1.

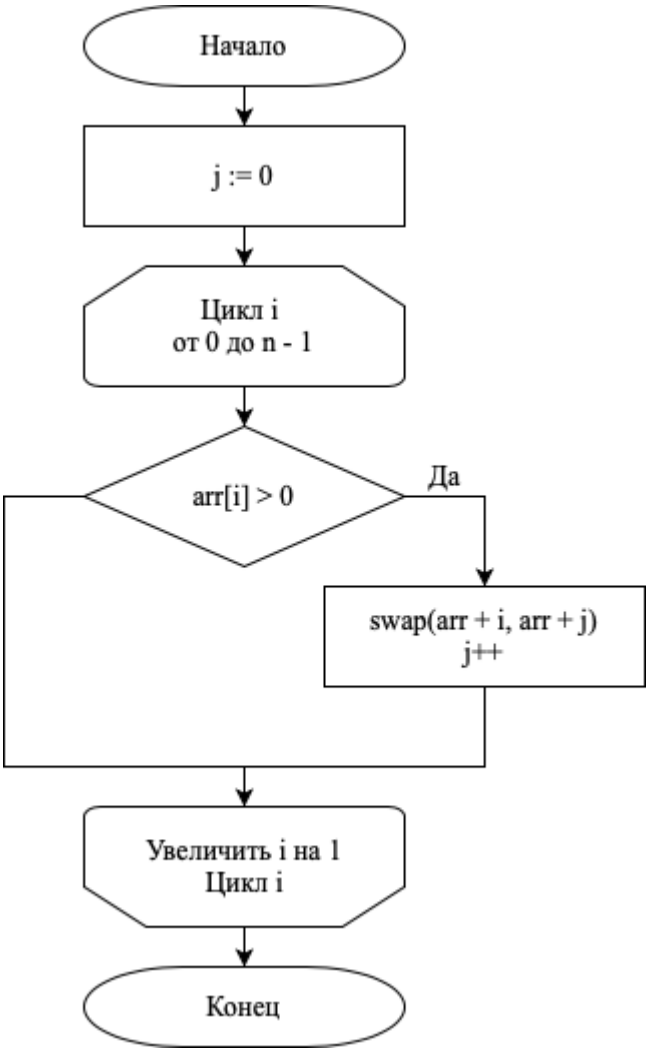

Рисунок 6.1. Схема алгоритма группировки.

Пример функции группировки элементов:

```
int groupElem(double* arr, int n) {
   int j = 0;
   for (int i = 0; i < n; i++)
      if (arr[i] > 0) {
         swap(arr + i, arr + j);
         j++;
      }
```

```
    return j;
}
```

Не трудно понять, что индекс j остановится на позицию первого не положительного элемента и является количеством положительных элементов.

Попробуем оптимизировать алгоритм. Будем идти указателями с двух концов массива навстречу друг другу. Возвращаемое значение i покажет количество положительных элементов.

Алгоритм оптимизированной группировки показан на рисунке 6.2.

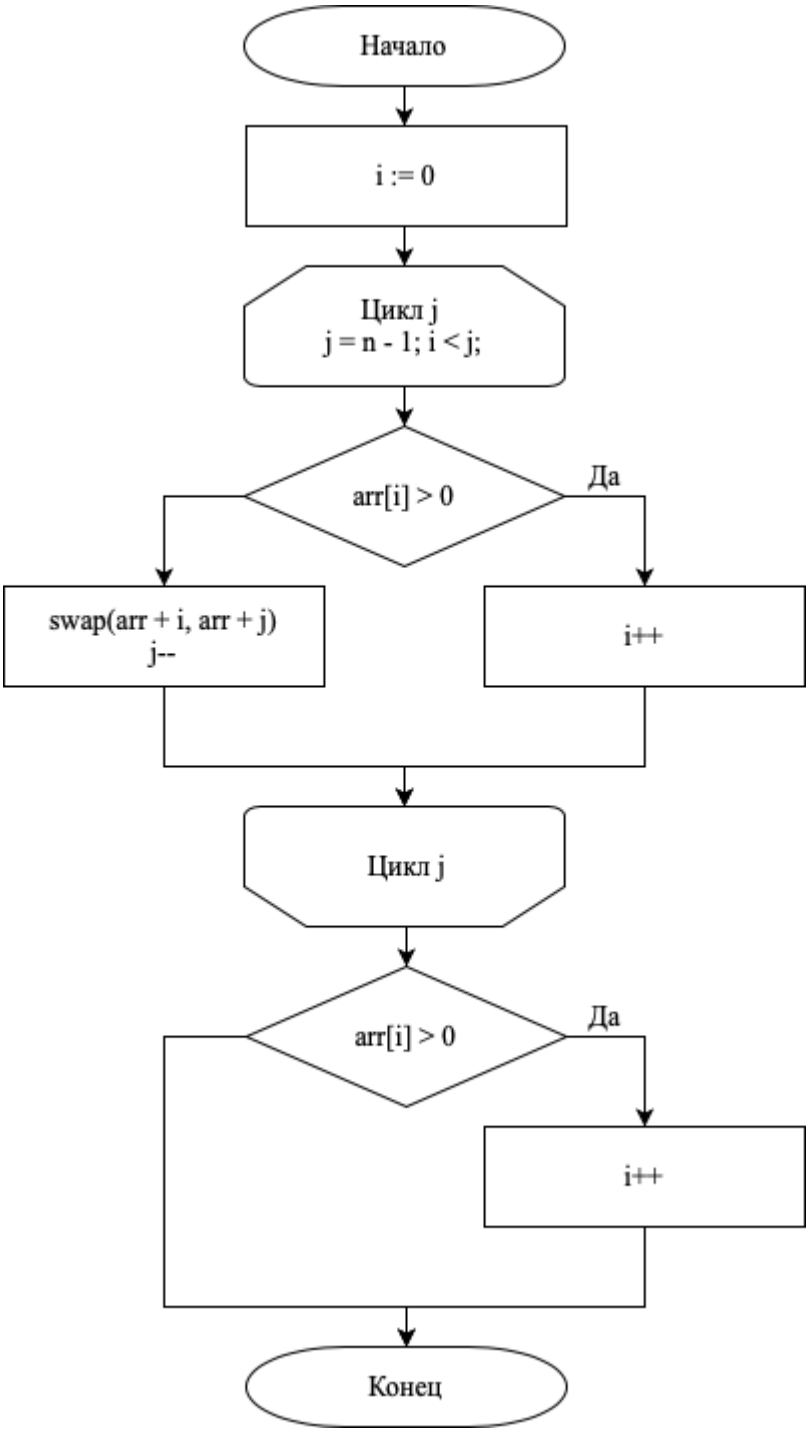

Рисунок 6.2. Схема оптимизированного алгоритма группировки.

Пример функции оптимизированной группировки элементов:

```
int optimizeGroupElem(double* arr, int n) {
    int i = 0;
    for (int j = n - 1; i < j;) {
        if (arr[i] > 0)
            i++;
        else {
            swap(arr + i, arr + j);
            j--;
        }
    }
    if (arr[i] > 0)
        i++;
    return i;
}
```

7. Сортировка элементов массива

7.1 Полный перебор

Будем располагать элементы по возрастанию. Сначала 1-ый элемент будем сравнивать со всеми последующими и, если находится элемент меньше первого, меняем их местами, затем 2-ой со всеми последующими и т.д. до предпоследнего. В результате получим отсортированный по возрастанию массив.

Алгоритм сортировки полным перебором по возрастанию показан на рисунке 7.1.

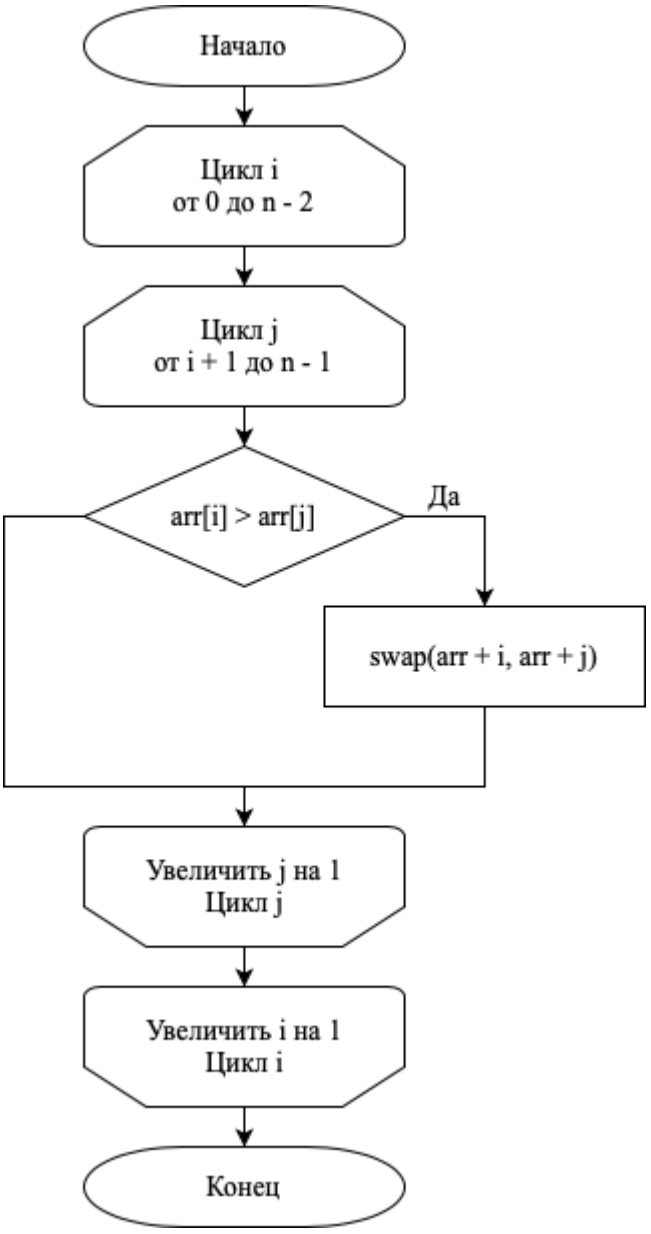

Рисунок 7.1. Схема алгоритма сортировки полным перебором.

Пример функции сортировки массива полным перебором по возрастанию:

```
void sort(double* arr, int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++)
            if (arr[i] > arr[j])
```

```
            swap(arr + i, arr + j);
    }
}
```

7.2 Метод поиска минимального элемента

Сравнивая i-ый элемент со всеми последующими, по существу мы ставим на i-ое место минимальный элемент из следующих за ним элементов. Оптимизируем алгоритм – будем искать индекс минимального элемента после i-го и менять его с ним.

Схема алгоритма сортировки массива методом поиска минимального элемента представлена на рисунке 7.2.

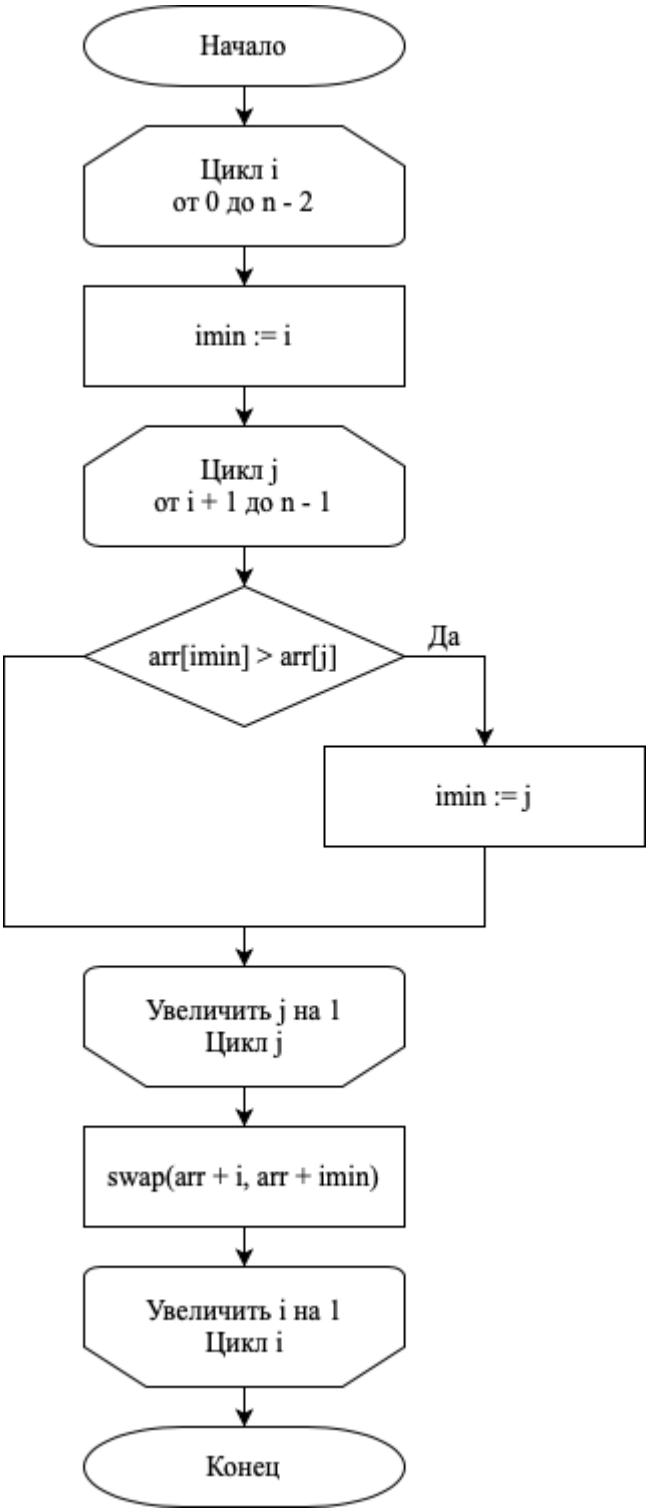

Рисунок 7.2. Схема алгоритма сортировки массива методом поиска минимального элемента.

Пример функции сортировки массива методом поиска минимального элемента:

```
void sort(double* arr, int n) {
    for (int i = 0; i < n - 1; i++) {
        int imin = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[imin] > arr[j])
                imin = j;
        }
        swap(arr + i, arr + imin);
    }
}
```

7.3 Метод “пузырек”

В этом алгоритме всегда сравниваются соседние элементы. Если следующий элемент меньше предыдущего, то они меняются местами. При первом проходе слева направо максимальный элемент встает в конец массива («всплывает» до нужной позиции, как пузырёк в воде) следующий проход выполняем до предпоследнего элемента и т.д. Перебор элементов прекращаем, когда при данном проходе не было ни одной замены или не осталось элементов для сравнения.

Схема алгоритма сортировки массива методом “пузырька” представлена на рисунке 7.3.

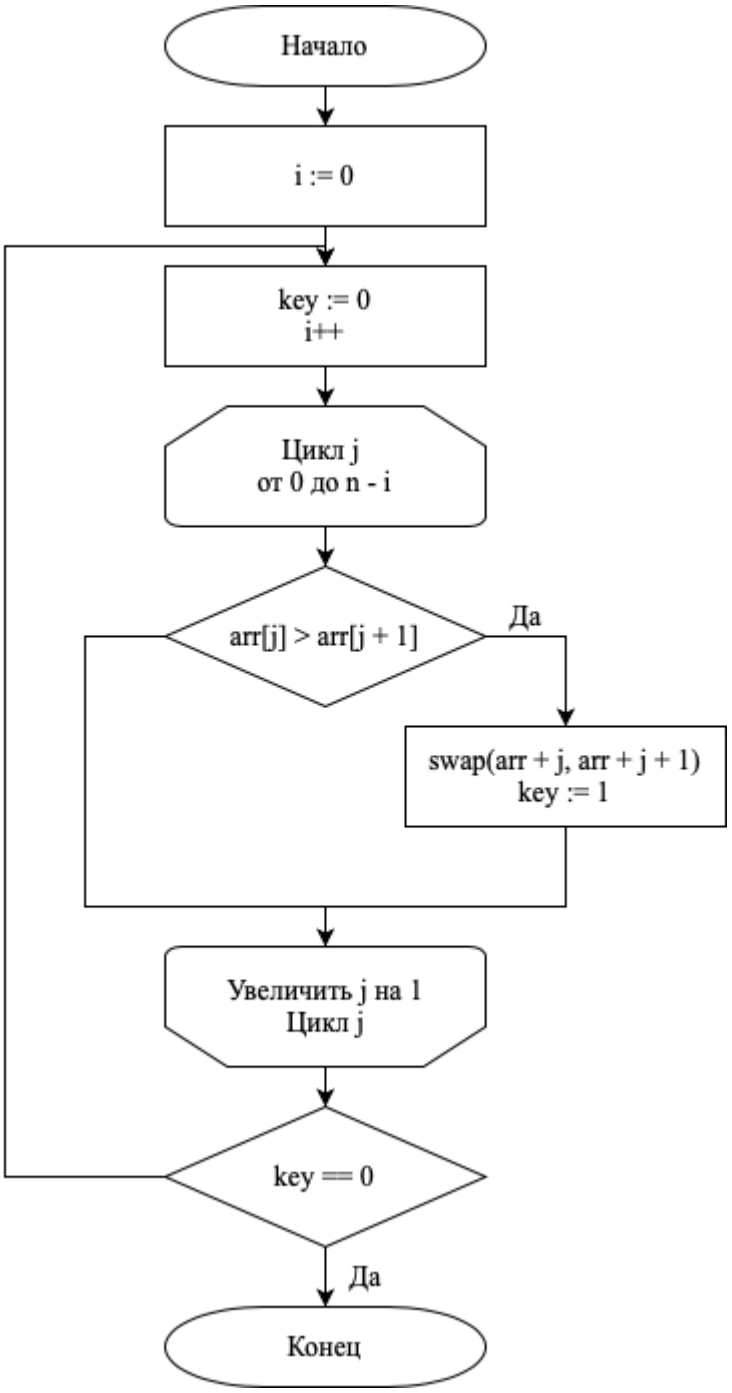

Рисунок 7.3. Схема алгоритма сортировки массива методом пузырька.

7.4 Метод вставки

Сначала второй элемент вставляем в массив до него так, чтобы 2 элемента были упорядочены возрастанию, затем 3-ий и т.д. по одному, так

чтобы каждый новый поступивший элемент размещался в подходящее место среди ранее упорядоченных элементов.

Схема алгоритма сортировки массива методом вставки показана на рисунке 7.4.

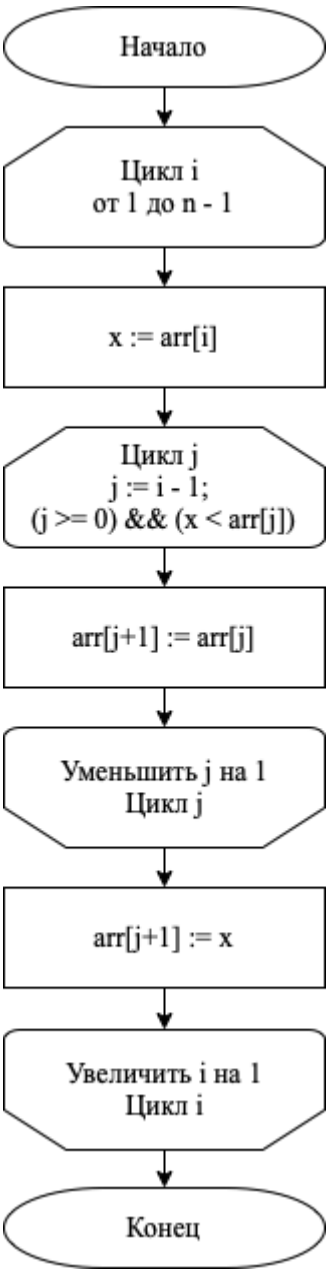

Рисунок 7.4. Схема алгоритма сортировки массива методом вставки.

Лабораторная работа

Лабораторная работа №5