**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**“Московский государственный университет геодезии и картографии” (МИИГАИК)**

**Факультет геоинформатики и информационной безопасности**

**Кафедра геоинформационных систем и технологий**

**Лабораторная работа №3**

**"Алгоритмы сортировки"**

**Преподаватель: Лебедев Евгений Денисович**

**Работу выполнил: Зазыкин Иван Дмитриевич**

**Группа: 2024-ФГИИБ-ПИ-1б**

**Вариант: 11**

**Москва 2025**

**1. Формулировка задания**

Реализовать три алгоритма сортировки: сортировку пузырьком, сортировку слиянием и быструю сортировку. Выполнить асимптотическую оценку сложности каждого алгоритма для лучшего, худшего и среднего случаев. Произвести замеры времени выполнения алгоритмов на массивах различного размера и представить результаты в виде таблицы и графика.

**2. Ссылка на GitHub-репозиторий**

https://github.com/re-side/Inf\_university/tree/main/ALG/lab3

**3.1. Сортировка пузырьком**

// Сортировка пузырьком

void bubbleSort(vector<int>& arr) {

int n = arr.size();

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - 1 - i; j++) {

if (arr[j] > arr[j + 1]) {

swap(arr[j], arr[j + 1]);

}

}

}

}

**Описание алгоритма:**  
Сортировка пузырьком — это простой алгоритм сортировки, который многократно проходит по массиву, сравнивая соседние элементы и меняя их местами, если они находятся в неправильном порядке. На каждом проходе самый большой неотсортированный элемент "всплывает" в конец массива, как пузырьки в воде, отсюда и название. Процесс повторяется, пока массив не будет полностью отсортирован. Алгоритм имеет квадратичную сложность O(n²) в среднем и в худшем случае, но O(n) в лучшем случае (почти отсортированный массив). (Блок схему см. Рис. 1)

**3.2. Сортировка слиянием**

// Сортировка слиянием

void merge(vector<int>& arr, int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

vector<int> L(n1), R(n2);

for (int i = 0; i < n1; i++)

L[i] = arr[left + i];

for (int j = 0; j < n2; j++)

R[j] = arr[mid + 1 + j];

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {

if (L[i] < R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(vector<int>& arr, int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

**Описание алгоритма:**  
Сортировка слиянием использует метод "разделяй и властвуй". Массив рекурсивно делится на две половины, пока каждая часть не станет содержать один элемент (уже отсортированный). Затем подмассивы сливаются в отсортированном порядке, сравнивая элементы и размещая их в правильной последовательности. Алгоритм имеет стабильную сложность O(n log n) во всех случаях благодаря предсказуемому делению и слиянию. (Блок схему см. Рис. 2)

**3.3. Быстрая сортировка**

// Быстрая сортировка

int partition(vector<int>& arr, int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] <= pivot) {

i++;

swap(arr[i], arr[j]);

}

}

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

return i + 1;

}

void quickSort(vector<int>& arr, int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

**Описание алгоритма:**  
Быстрая сортировка также использует метод "разделяй и властвуй". Алгоритм выбирает опорный элемент (в нашей реализации — последний элемент массива), разделяет массив на две части: элементы, меньшие или равные опорному, и элементы, большие опорного. Затем рекурсивно сортируются обе части. Алгоритм имеет среднюю сложность O(n log n), но в худшем случае (например, для уже отсортированного массива) — O(n²). (Блок схему см. Рис. 3)

**4.Таблица примеров входных и выходных данных**

| **Алгоритм** | **Входные данные** | **Выходные данные** |
| --- | --- | --- |
| Сортировка пузырьком | [5, 2, 8, 1, 9] | [1, 2, 5, 8, 9] |
| Сортировка слиянием | [5, 2, 8, 1, 9] | [1, 2, 5, 8, 9] |
| Быстрая сортировка | [5, 2, 8, 1, 9] | [1, 2, 5, 8, 9] |

Таблица показывает работу алгоритмов на массиве [5, 2, 8, 1, 9]. Все алгоритмы возвращают отсортированный массив [1, 2, 5, 8, 9].

**5. Асимптотическая оценка**

Оценим временную сложность каждого алгоритма для лучшего, худшего и среднего случаев. Сложность выражена в нотации O, где n - размер входного массива.

**Таблица асимптотической оценки**

| **Алгоритм** | **Лучший случай** | **Худший случай** | **Средний случай** |
| --- | --- | --- | --- |
| Сортировка пузырьком | O(n) | O(n²) | O(n²) |
| Сортировка слиянием | O(n log n) | O(n log n) | O(n log n) |
| Быстрая сортировка | O(n log n) | O(n²) | O(n log n) |

**Применение мастер-теоремы к быстрой сортировке:**

Мастер-теорема помогает найти сложность рекурсивных алгоритмов вида T(n) = a\*T(n/b) + f(n), где:

a: число рекурсивных вызовов,

b: во сколько раз уменьшается размер задачи,

f(n): время работы вне рекурсии.

Для быстрой сортировки в среднем:

Массив делится на две части по n/2.

Функция partition занимает O(n).

Уравнение: T(n) = 2\*T(n/2) + O(n).

По мастер-теореме: a = 2, b = 2, f(n) = O(n). Сравниваем f(n) = O(n) с n^(log b a) = n^(log 2 2) = n. Так как f(n) равно n, получаем O(n\*log(n)).

В худшем случае (отсортированный массив):

Массив делится на n-1 и 0.

Уравнение: T(n) = T(n-1) + O(n).

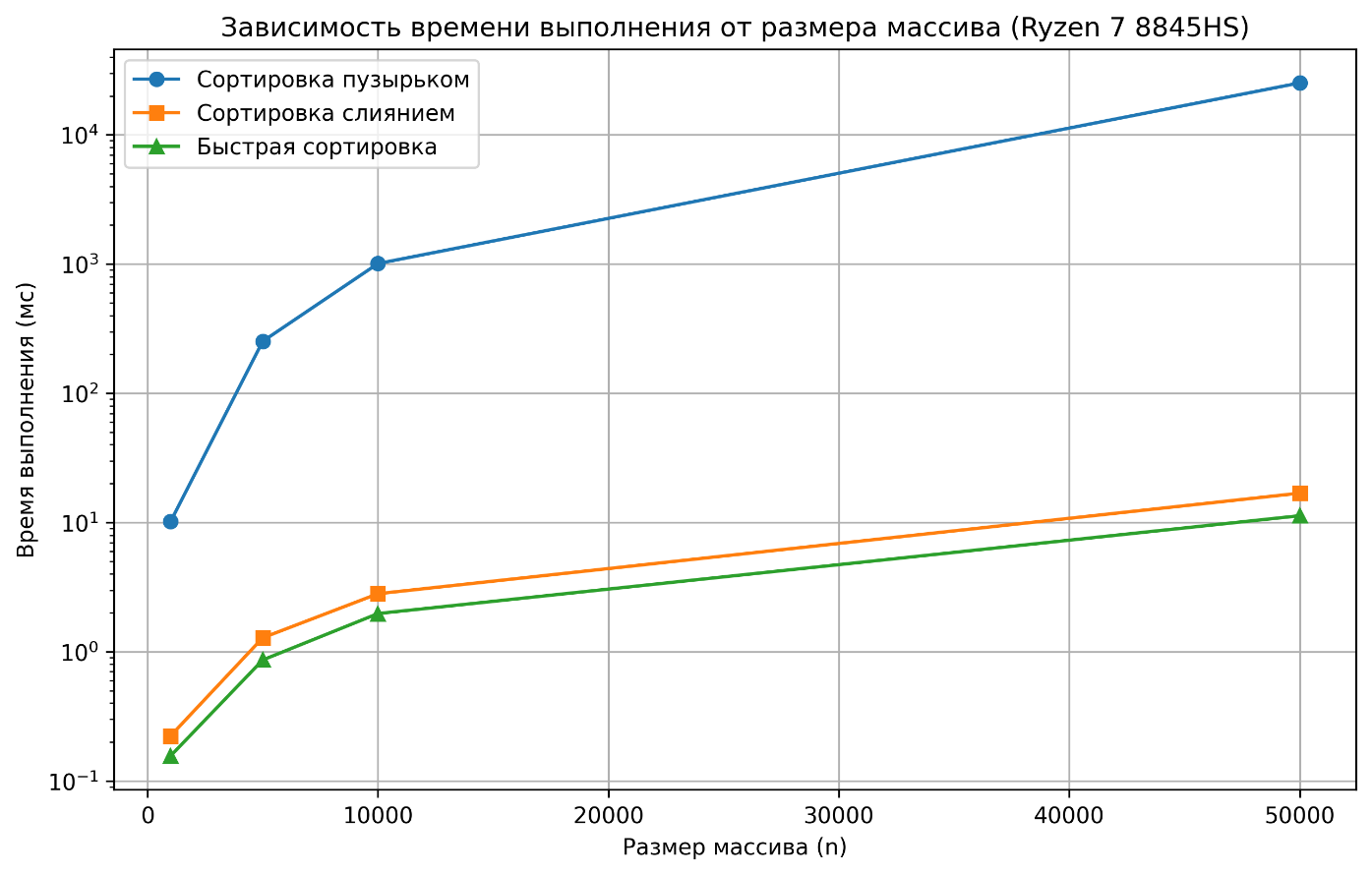
Мастер-теорема не работает, но сумма O(n + (n-1) + ... + 1) дает O(n^2).

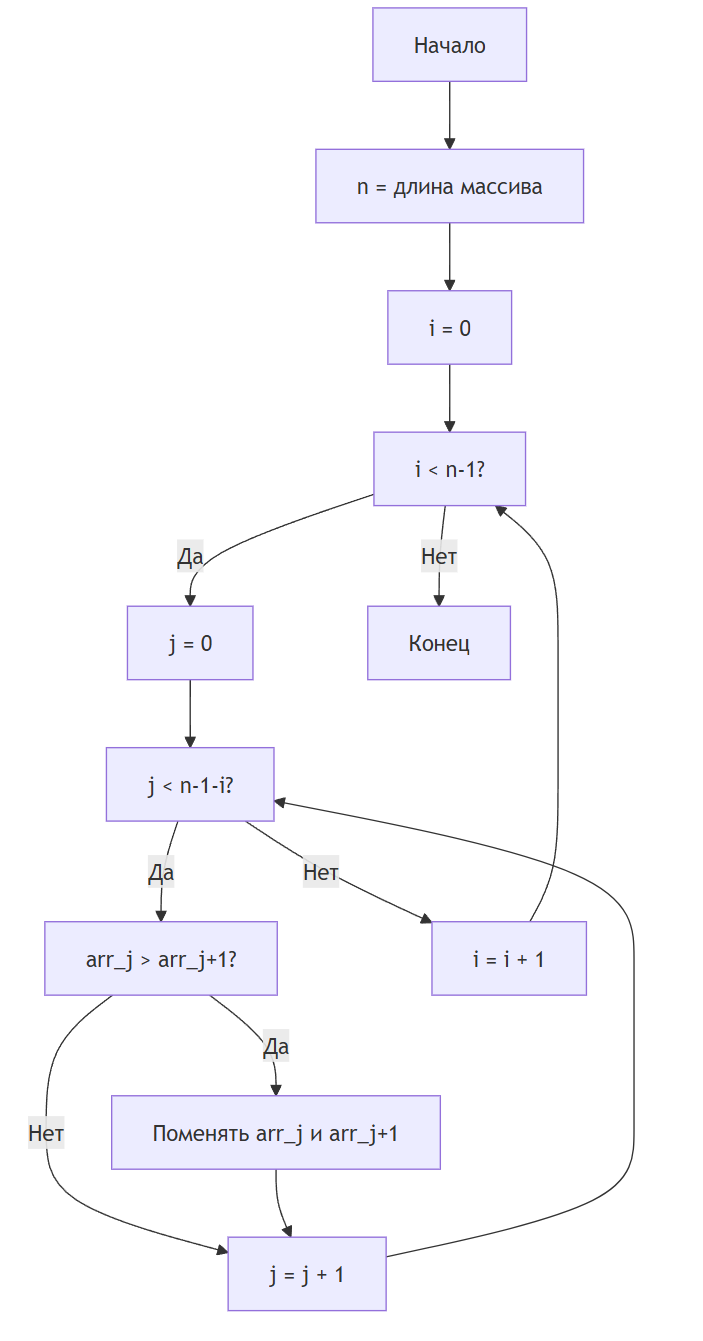
Вывод: В среднем и лучшем случае быстрая сортировка — O(n\*log(n)), в худшем — O(n^2).

**6.Таблица замеров времени**

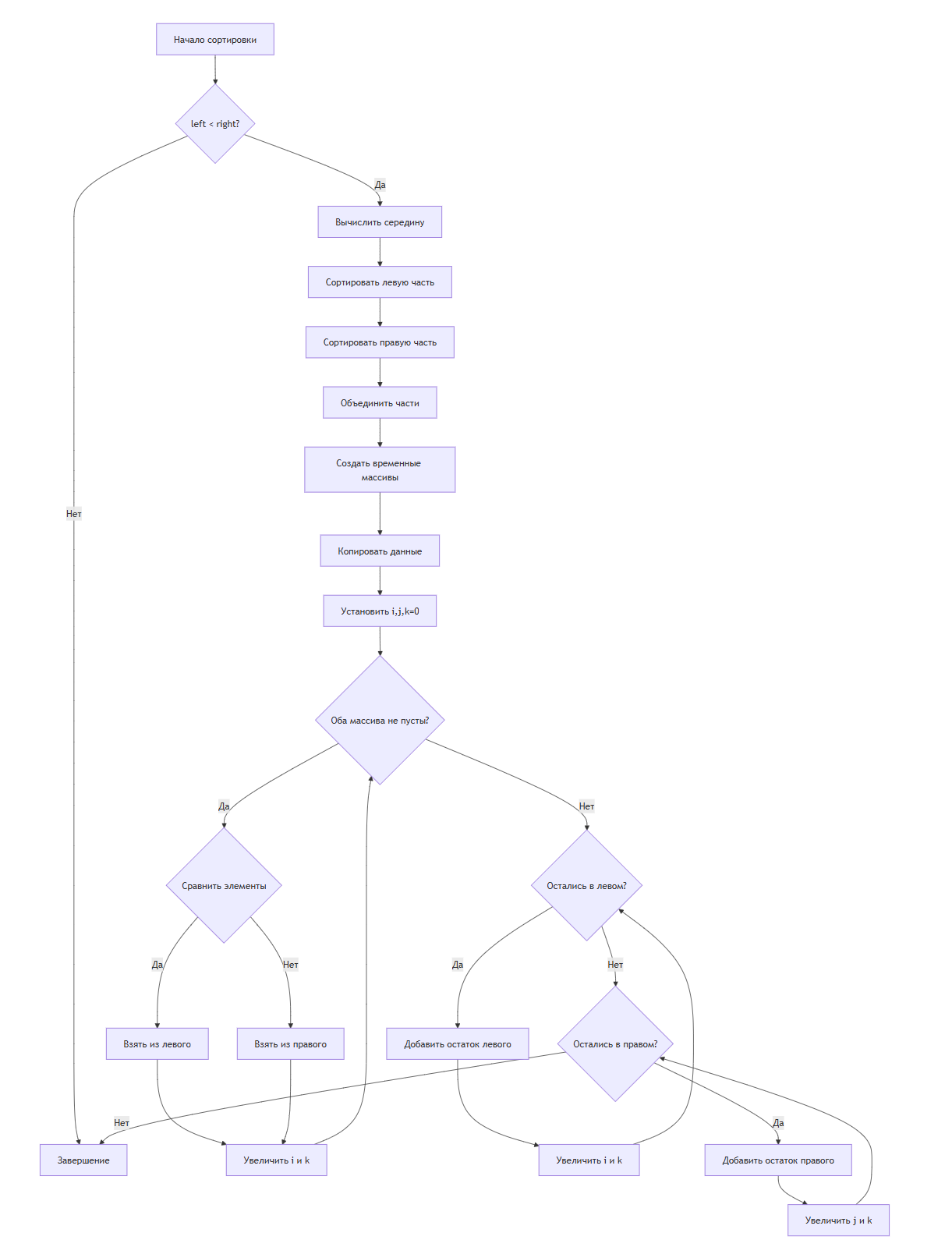
Замеры времени выполнения алгоритмов сортировки на AMD Ryzen 7 8845HS

| **Размер массива (n)** | **Сортировка пузырьком (мс)** | **Сортировка слиянием (мс)** | **Быстрая сортировка (мс)** |
| --- | --- | --- | --- |
| 1000 | 10.152 | 0.223 | 0.156 |
| 5000 | 252.349 | 1.281 | 0.863 |
| 10000 | 1008.582 | 2.810 | 1.973 |
| 50000 | 25228.104 | 16.863 | 11.281 |

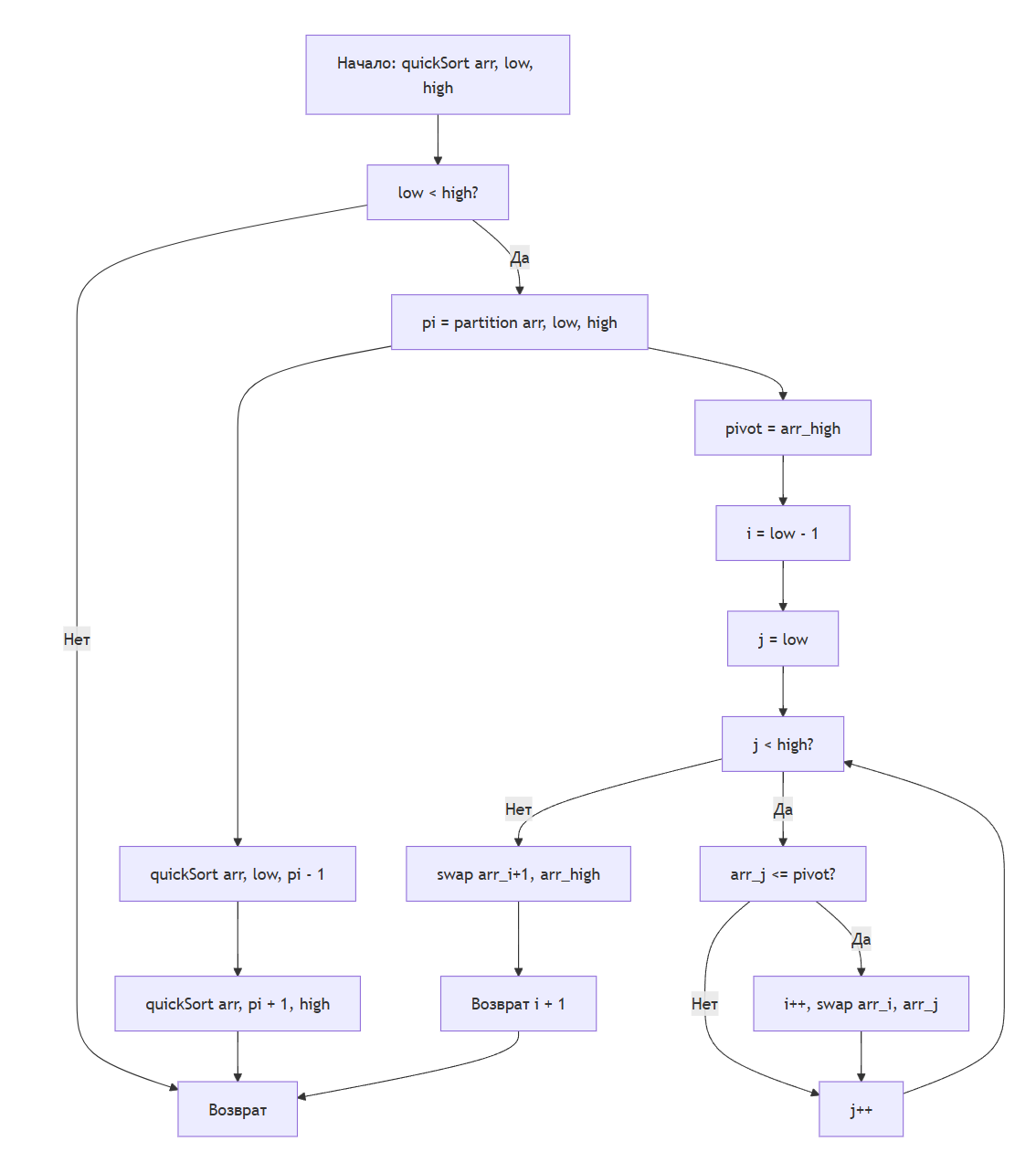


****

**Рис. 1**

****

**Рис. 2**

**Рис. 3**