Практическая работа №2

Тема: «Алгоритмы сортировки».

Цель работы: изучить алгоритмы сортировки.

Пузырьковая сортировка

Сортировка пузырьком в основном применяется в учебных проектах. В реальной практике её заменяют более эффективные алгоритмы, однако сортировка пузырьком лежит в основе некоторых из них.

В общем случае алгоритм сортировки пузырьком следующий:

1. Сравнить текущий элемент со следующим.
2. Если следующий элемент меньше/больше текущего, поменять их местами.
3. Если массив отсортирован, закончить алгоритм, иначе перейти на шаг 1.

В алгоритме используется два цикла: основной и вложенный. В результате одного прохода вложенного цикла наибольший элемент помещается в конец массива, а наименьший смещается на одну позицию ближе к началу.

Внешний цикл в худшем случае совершает N (кол-во элементов) – 1 проходов, то есть внутренний цикл выполняется N-1 раз.

Таким образом, в каждом проходе совершается серия обменов элементов так, что наибольший элемент передвигается в конец массив перед элементом, который переместился туда в прошлой итерации. Процесс происходит до тех пор, пока массив не будет отсортирован.

Если рассмотреть реализацию алгоритма, то можно легко заметить, что время его работы (количество операций) значительно возрастает с увеличением количества элементов сортируемой последовательности.

Сложность алгоритма

Сложность алгоритма позволяет дать ему оценку по времени выполнения, то есть определяет его эффективность. Можно выражать сложность по-разному, но чаще всего используется асимптотическая сложность, которая определяет его

эффективность при стремлении входных данных к бесконечности.

Точное время выполнения алгоритма не рассматривается, потому что оно зависит слишком от многих факторов: мощность процессора, тип данных массива, используемый язык программирования.

Алгоритм сортировки пузырьком имеет сложность $O(n^2)$, где n – количество элементов массива. Из формулы видно, что сложность сортировки пузырьком квадратично зависит от количества сортируемых элементов. Это значит, что он неэффективен при работе с большими массивами данных.

Следует понимать, что с помощью асимптотической функции нельзя точно вычислить время работы алгоритма. Например, дана последовательность “6 5 4 3 2 1”, для её сортировки придется сделать максимальное количество проходов. Такой случай называют наихудшим. Если дана последовательность “3 1 2 4 5”, то количество проходов будет минимально, соответственно сортировка пройдет гораздо быстрее. Если же дан уже отсортированный массив, то алгоритму сортировки и вовсе не нужно совершать проходов. Это называется наилучшим случаем.

```
from random2 import randint

N = 10
a = []
for i in range(N):
    a.append(randint(1, 99))
print(a)

i = 0
while i < N - 1:
    j = 0
    while j < N - 1 - i:
        if a[j] > a[j+1]:
            a[j], a[j+1] = a[j+1], a[j]
        j += 1
    i += 1

print(a)
```

Рис. 1 Пузырьковая сортировка.

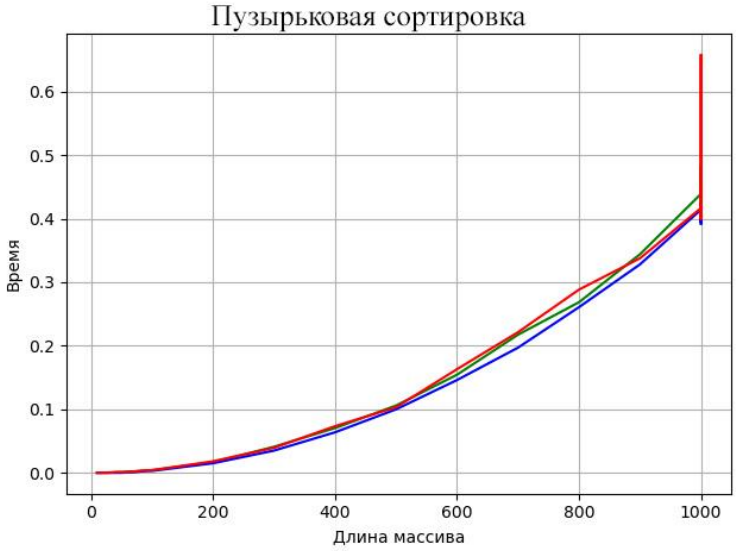

Рис. 2 График пузырьковой сортировки

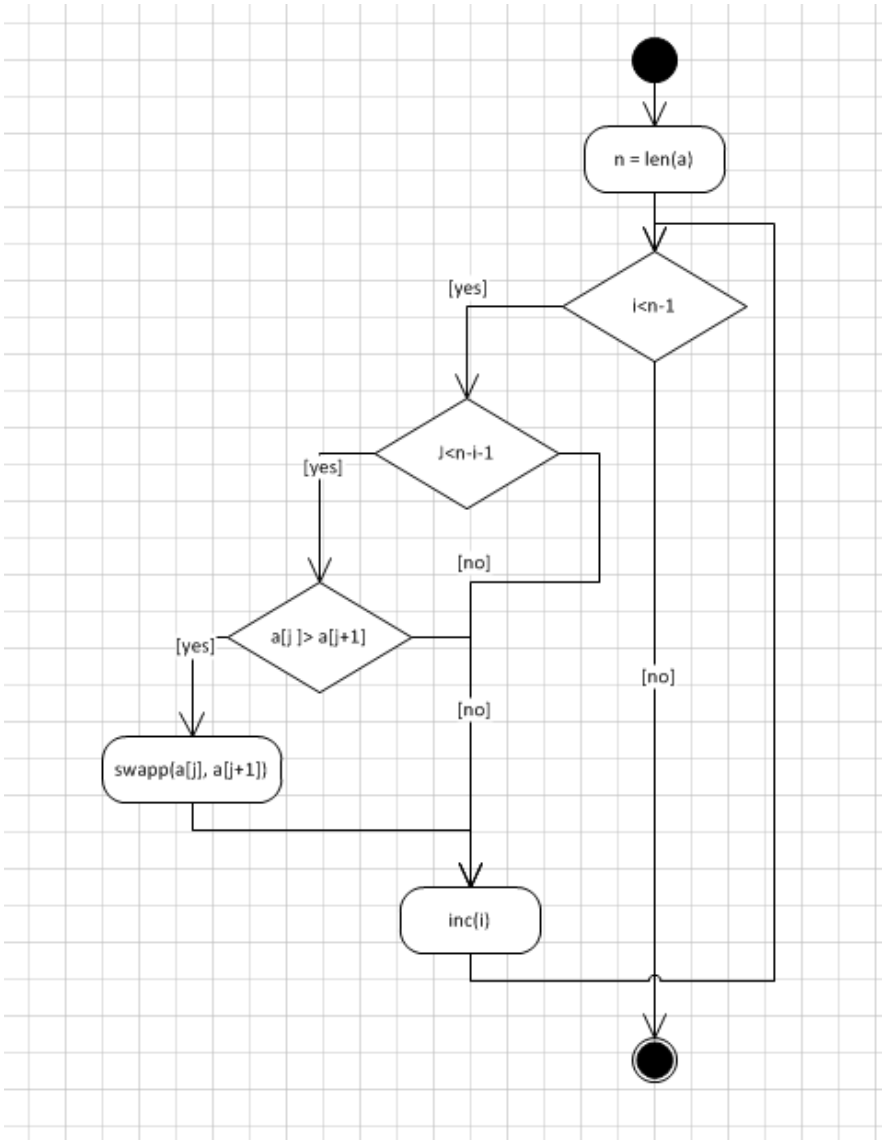

Рис. 3 Диаграмма деятельности пузырьковой сортировки

Сортировка вставками

Сортировка вставками – это алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. Вычислительная сложность - $O(n^2)$.

Сортировки вставками всегда делят массив на 2 части — отсортированную и неотсортированную. Из неотсортированной части извлекается любой элемент. Поскольку другая часть массива отсортирована, то в ней достаточно быстро можно найти своё место для этого извлечённого элемента. Элемент вставляется куда нужно, в результате чего отсортированная часть массива увеличивается, а неотсортированная уменьшается. По такому принципу работают все сортировки вставками. Самое слабое место в этом подходе — вставка элемента в отсортированную часть массива.

```
from random2 import randint

def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >=0 and key < arr[j] :
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key

N = 10
arr = []
for i in range(N):
    arr.append(randint(1, 99))

print (arr)

insertion_sort(arr)
print ("Sorted array is:")
for i in range(len(arr)):
    print (arr[i])

input()
```

Рис. 4 Сортировка вставками

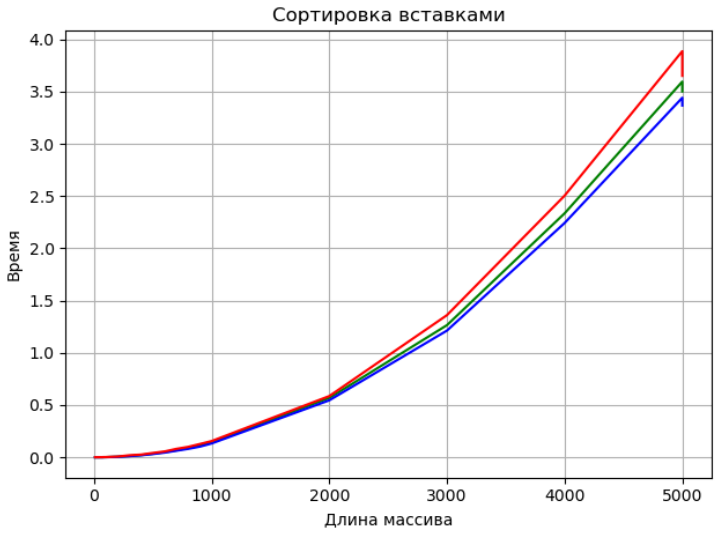

Рис. 5 График сортировки вставками

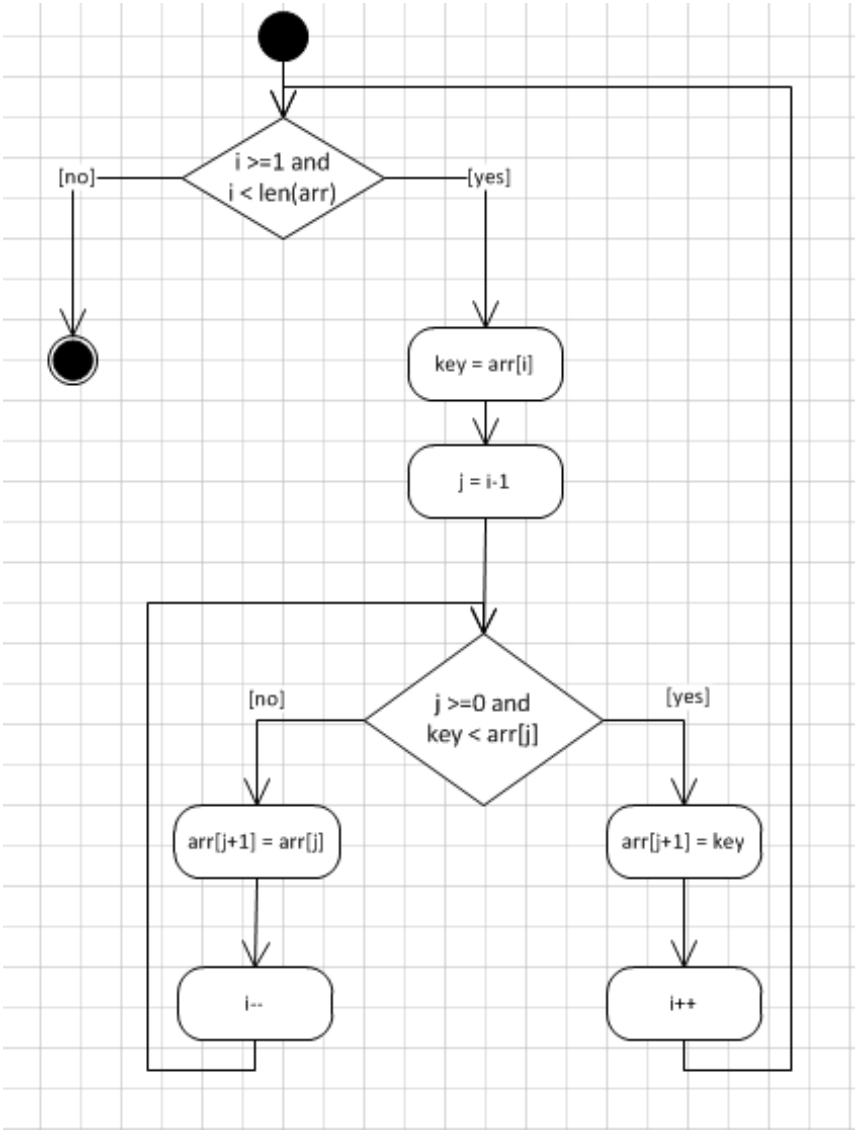

Рис. 6 Диаграмма деятельности сортировки вставками

Шейкерная сортировка

Сортировка перемешиванием, или Шейкерная сортировка - разновидность пузырьковой сортировки. Анализируя метод пузырьковой сортировки, можно отметить два обстоятельства.

Во-первых, если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, её можно исключить из рассмотрения.

Во-вторых, при движении от конца массива к началу минимальный элемент «всплывает» на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо.

Эти две идеи приводят к следующим модификациям в методе пузырьковой сортировки. Границы рабочей части массива (то есть части массива, где происходит движение) устанавливаются в месте последнего обмена на каждой итерации. Массив просматривается поочередно справа налево и слева направо.

```
from random2 import randint

N = 10
arr = []
for i in range(N):
    arr.append(randint(1, 99))
print(arr)

left = 0
right = len(arr) - 1

while left <= right:
    for i in range(left, right, +1):
        if arr[i] > arr[i + 1]:
            arr[i], arr[i + 1] = arr[i + 1], arr[i]
    right -= 1

    for i in range(right, left, -1):
        if arr[i - 1] > arr[i]:
            arr[i], arr[i - 1] = arr[i - 1], arr[i]
    left += 1

print(arr)
```

Рис. 5 Шейкерная сортировка

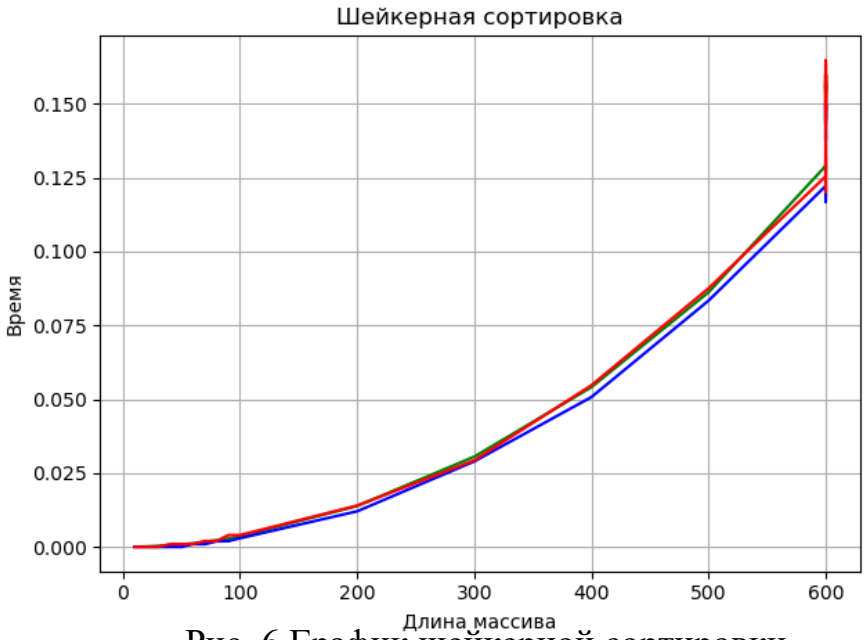

Рис. 6 График шейкерной сортировки

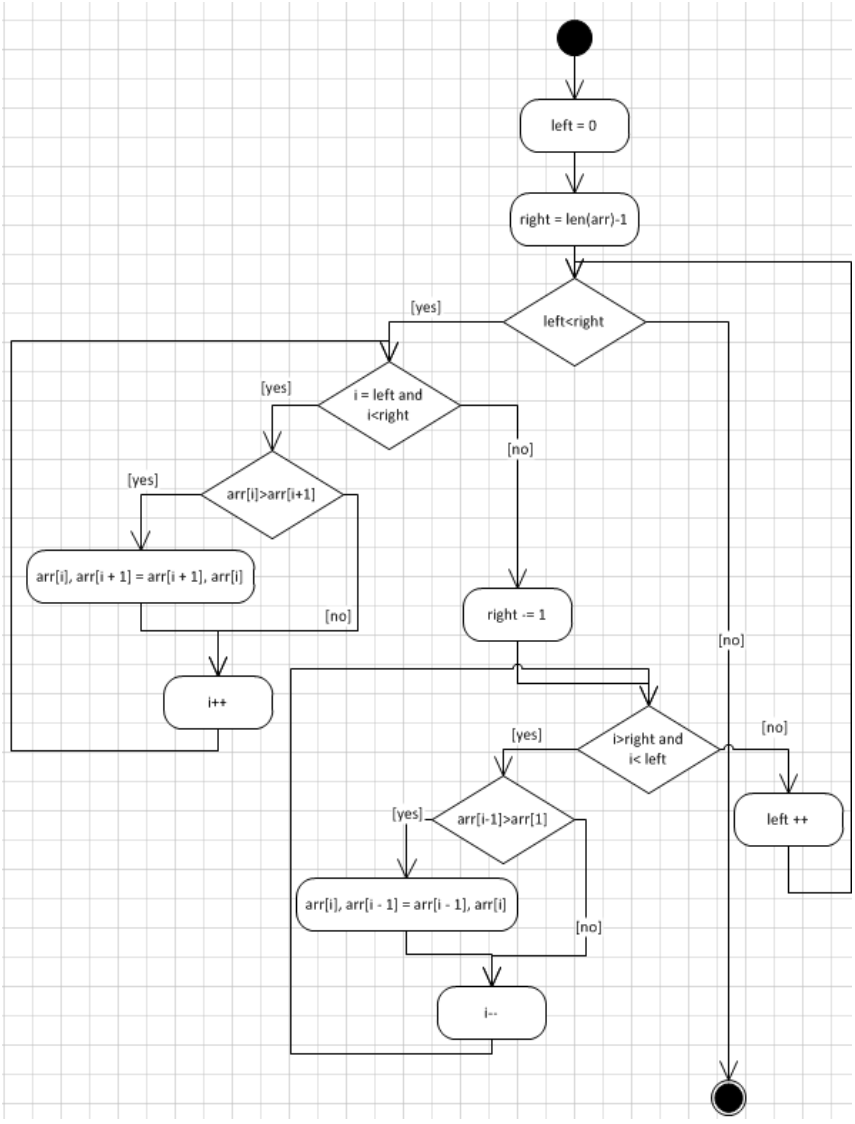

Рис. 7 Диаграмма деятельности Шейкерной сортировки

Сортировка выбором

Этот алгоритм сегментирует список на две части: отсортированные и несортированные. Он постоянно удаляет наименьший элемент из несортированного сегмента списка и добавляет его в отсортированный сегмент.

На практике нам не нужно создавать новый список для отсортированных элементов, мы будет обрабатывать крайнюю левую часть списка как отсортированный сегмент. Затем мы ищем во всем списке наименьший элемент и меняем его на первый элемент.

Теперь мы знаем, что первый элемент списка отсортирован, мы получаем наименьший элемент из оставшихся элементов и заменяем его вторым элементом. Это повторяется до тех пор, пока последний элемент списка не станет оставшимся элементом для изучения.

```
from random2 import randint
N = 10
arr = []
for i in range(N):
    arr.append(randint(1, 99))
print(arr)
i = 0
while i < N - 1:
    m = i
    j = i + 1
    while j < N:
        if arr[j] < arr[m]:
            m = j
        j += 1
    arr[i], arr[m] = arr[m], arr[i]
    i += 1
print(arr)
```

Рис. 8 Сортировка выбором

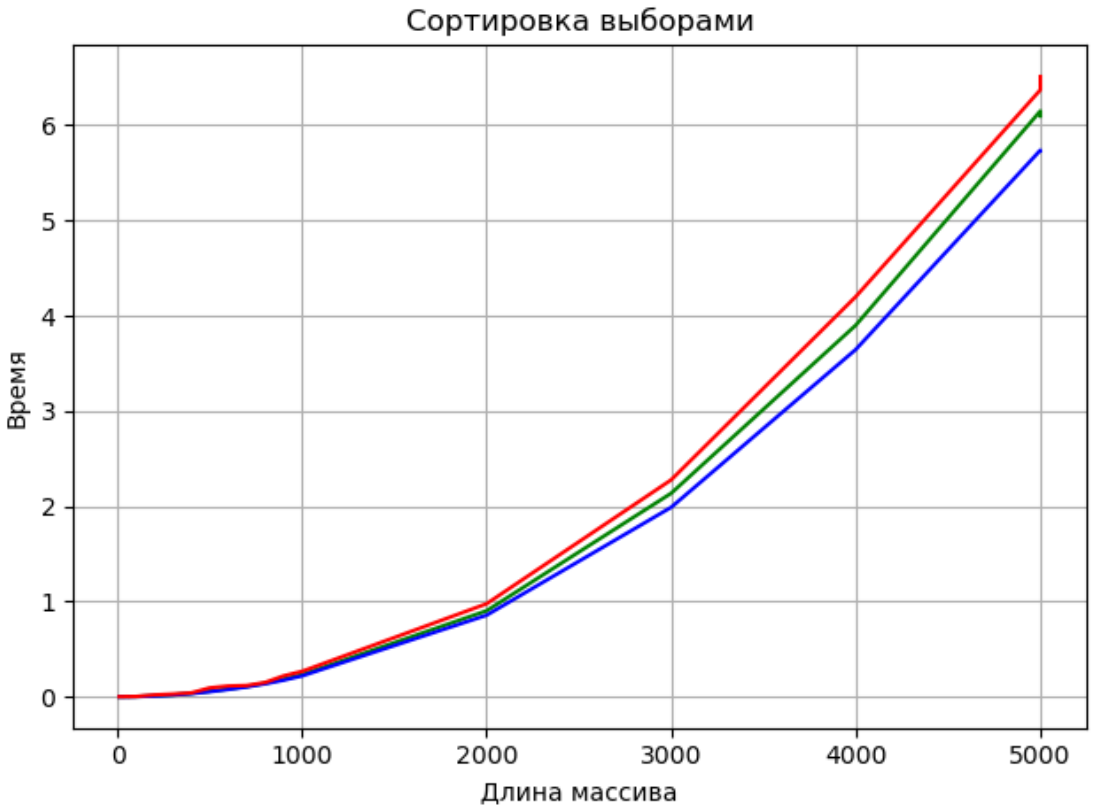

Рис. 9 График сортировки выбором

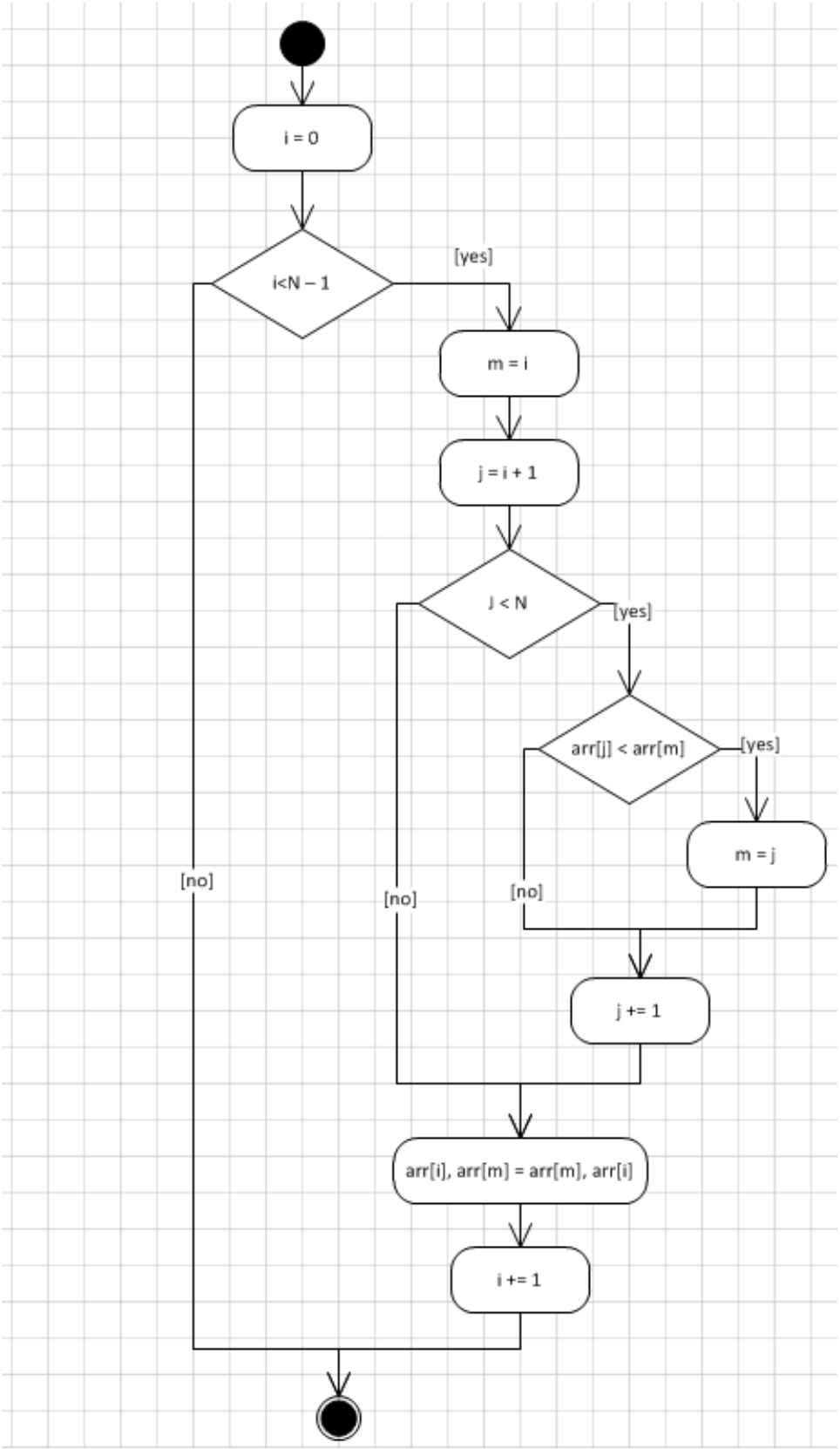

Рис. 10 Диаграмма деятельности сортировки выбором