Федеральное государственное бюджетное образовательное учреждение высшего образования Московский государственный технологический университет «СТАНКИН»

Кафедра «Управление и информатика в технических системах

Отчёт по лабораторным работам по дисциплине "Алгоритмы и структуры данных""

Выполнил:
Студент группы ИДБ-22-10
Инкин Д.В.
Преподаватель:
доцент Евдокимов С.А.

Москва, 2023

Содержание

1 Сортировка подсчетом **3**

1.1 Постановка задачи 3

1.2 Описание алгоритма 3

1.3 Блок-схема алгоритма 4

1.4 Код алгоритма на языке Python 5

1.5 График времени сортировки 8

1.6 Выводы 9

1.7 Библиография 10

1 Сортировка подсчетом

1.1 Постановка задачи

Написать программу сортировки простыми вставками и сравнить ее время выполнения со стандартной функцией qsort. Для сравнения вычисляем время выполнения функции сортировки на массиве целых чисел следующих размеров: 16, 100, 500, 1000, 5000. Для работы программы сортировки генерируем случайные массивы целых чисел:

Для проверки программы сравниваем результат работы с коректно отсортированным массивом.

Время работы функции сортировки рассчитываем с точностью до тысячных долей миллисекунд. Например, 0.002, 0.018, 0.377, 1.380, 35.806 мсек.

Для оценки поведения функции сортировки создаем графики в зависимости от числа элементов (N) в массиве:N) в массиве:

- график функции N^2;
- график функции $Nlog_2N$;
- график времени работы стандартной функции qsort;
- график времени работы функции сортировки подсчетом;

1.2 Описание алгоритма

Этот алгоритм сортирует записи R_1,...,R_n по ключам a_1,...,a_n. После завершения алгоритма величина COUNT[j] определяет положение в записиR_j

Основные шаги алгоритма:

1 Установить в счетчиках COUNT[0]-COUNT[N-1] нули.

2 Выполнить шаг 3 при i = 0, 1,...,n-1; затем завершить выполнение процедуры.

3 Выполнить шаг 4 при j = i+1, i+2,...,n-1.

4 Если $a_i < a_j$, то увеличить COUNT[j] на 1; в противном случае увеличить COUNT[i] на 1.

Формальная запись алгоритма посредством блок-схемы показана ниже.

Результат работы программы на массиве из 16 чисел показан ниже:

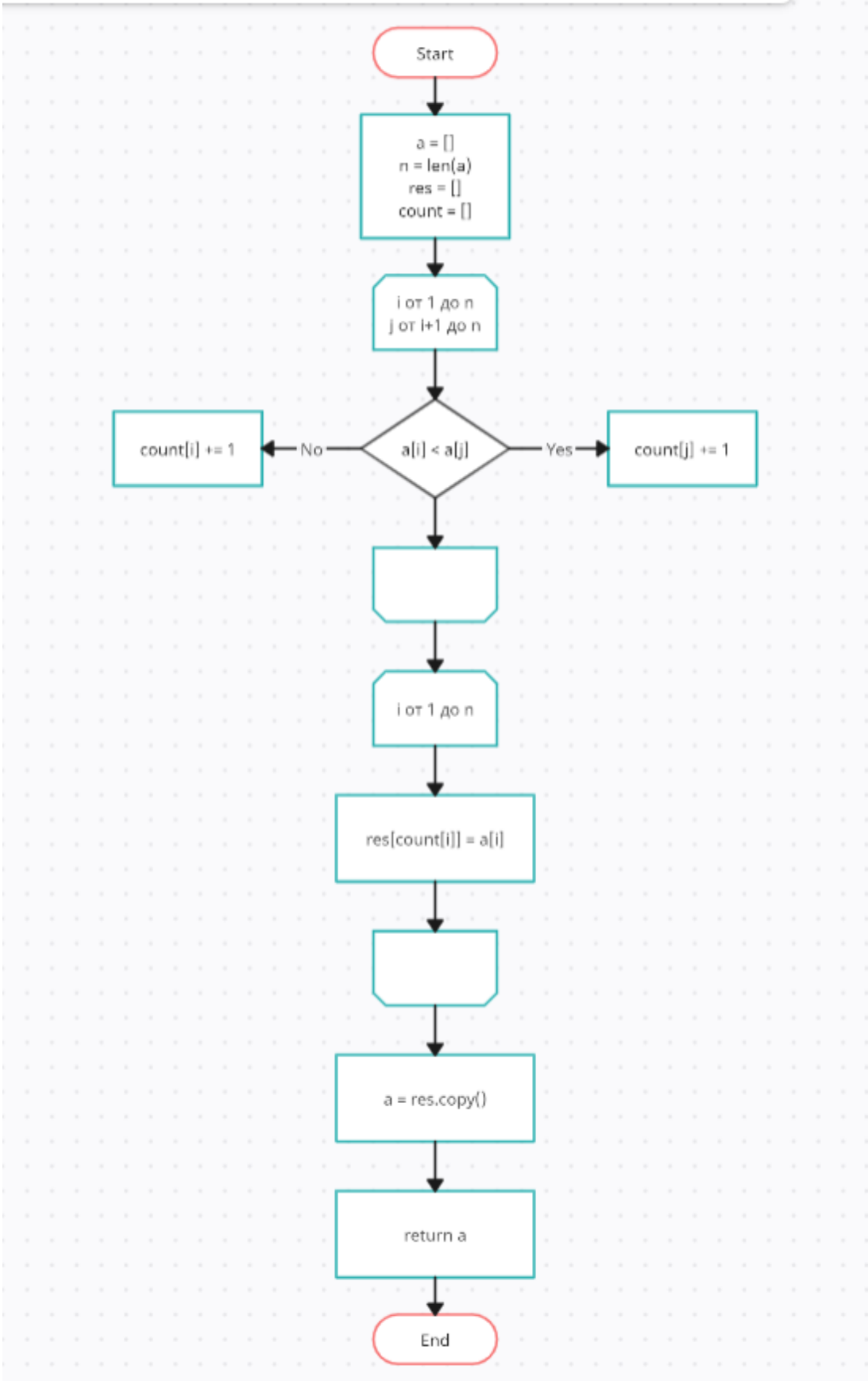
Результат работы программы: Сортируется массив из 16 чисел

Исходный массив: 643, 723, 209, 763, 827, 596, 338, 263, 734, 589, 773, 491, 207, 973, 168, 838

Отсортированный массив: 168, 207, 209, 263, 338, 491, 589, 596, 643, 723, 734, 763, 773, 827, 838, 973

Время сортировки: 0.0362 мсек.

1.3 Блок-схема алгоритма



1.4 Код алгоритма на языке Python

```
import random
import time
N = [16,100,500,1000,5000]

def merge(list1, list2): # Функция слияния
    i = 0
    j = 0
    res = []
    while i < len(list1) and j < len(list2):
        if list1[i] <= list2[j]:
            res.append(list1[i])
            i += 1
        else:
            res.append(list2[j])
            j += 1

    while i < len(list1):
        res.append(list1[i])
        i += 1

    while j < len(list2):
        res.append(list2[j])
        j += 1

    return res


def merge_sort(a): # Сортировка Merge Sort
    mid = len(a) // 2
    L = a[:mid]
    R = a[mid:]
    if len(a) == 1:
        return a
    return merge(merge_sort(L), merge_sort(R))


def count_sort(a):
    n = len(a) # длина массива a
    res = [0 for i in range(n)] # Массив, заполненный нулями
```

```
    count = [0 for i in range(n)] # Массив, заполненный нулями
    for i in range(n):
        for j in range(i + 1, n):
            if a[i] < a[j]:
                count[j] += 1
            else:
                count[i] += 1

    for i in range(n):
        res[count[i]] = a[i]
    a = res.copy() # Копирование из вспомогательного массива в основной
    return a

def is_correction(a): # Функция проверки на корректность сортировки
    n = len(a)
    for i in range(n-1):
        if a[i] > a[i+1]:
            print(a[i], a[i+1])
            return False
    return True

for n in N:
    a = []
    with open('test.txt', 'r') as f:
        for i in range(n):
            a.append(int(f.readline()))

    #замер времени работы функций
    start_count_sort = time.time()
    count_sort(a)
    finish_count_sort = time.time()

    start_merge_sort = time.time()
    merge_sort(a)
    finish_merge_sort = time.time()

    start_quick_sort = time.time()
    sorted(a) # quick sort
    finish_quick_sort = time.time()
```

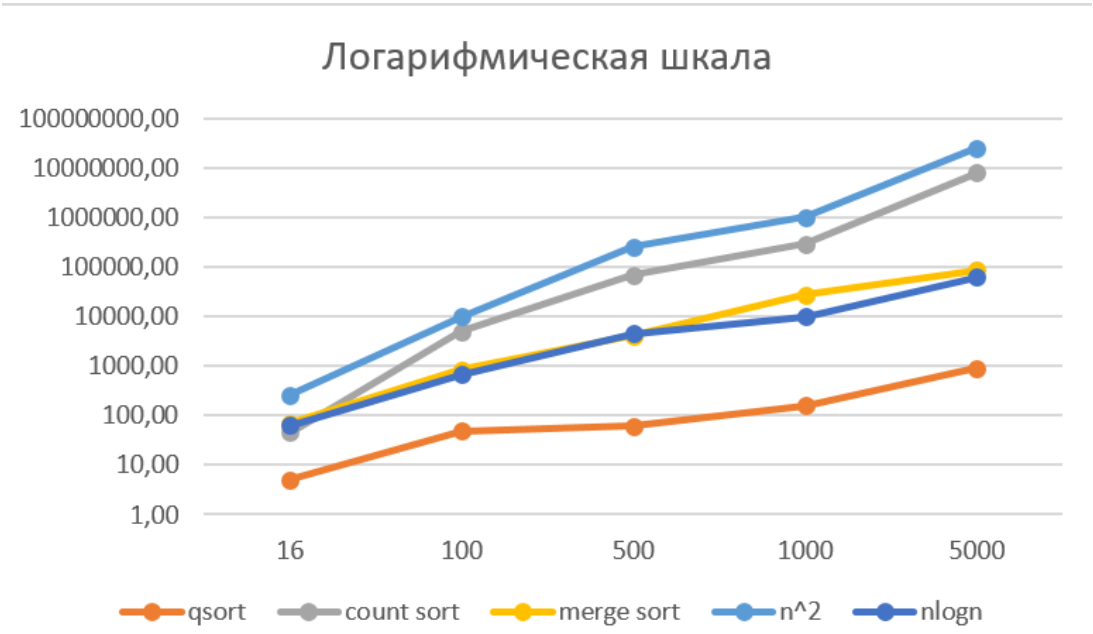
```
print(f'Время работы функций для массива размером {n}:')
print(f'Count sort = {(finish_count_sort-start_count_sort)*1000000} мксек')
print(f'Merge sort = {(finish_merge_sort - start_merge_sort) * 1000000} мксек
print(f'Quick sort = {(finish_quick_sort - start_quick_sort) * 1000000} мксек
print()
a = count_sort(a)

if not is_correction(a):
    print('Ошибка при сортировке!')
```

1.5 График времени сортировки

Для оценки поведения функции сортировки создаем графики в зависимости от числа элементов (N) в массиве:

- график функции N^2;
- график функции $Nlog_2N$;
- график времени работы стандартной функции qsort;
- график времени работы функции merge sort;
- график времени работы функции сортировки подсчетом;

			мксек		
N	N^2	N*LOGN	qsort	count_sort	merge_sort
16	256	64	5,01	45,54	68,90
100	10000	664	48,40	5022,05	836,13
500	250000	4483	61,99	67947,63	4039,29
1000	1000000	9966	156,40	290851,12	27182,58
5000	25000000	61439	890,97	8068194,63	84904,43



1.6 Выводы

Из анализа графиков следует:

- Сортировка простыми вставками имеет временную сложность $O(N^2)$
- На малом числе элементов в массиве время сортировки простыми вставками практически такое же как и у функции $Nlog_2N$;

1.7 Библиография

1. Д.Кнут. Искуство программирования для ЭВМ. Сортировка и поиск. Том 3. М.:Мир, Москва, 1978. — 844 с.