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» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

О Т Ч Е Т

по лабораторной работе № 3

Название: Алгоритмы сортировки

Дисциплина: Анализ алгоритмов

Студент ИУ7-52Б _____ В.А. Иванов
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ _____
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Оглавление

Введение

В данной лабораторной изучается и оценивается три алгоритма сортировки:

- сортировка пузырьком;
- поразрядная сортировка;
- сортировка слиянием.

Сортировка - это процесс упорядочения некоторого множества элементов, на котором определены отношения порядка.

Задача сортировки множества данных является одной из самых часто встречающихся при разработке программ. Также существует и достаточное количество алгоритмов, решающих данную задачу. Однако, "лучшего"алгоритма подходящего для решения любых задач сортировки наиболее оптимальным образом не существует. Поэтому задача изучения алгоритмов упорядочения до сих пор остаётся актуальной.

1. Аналитическая часть

1.1. Цели и задачи работы

Целью лабораторной работы является оценка трудоёмкости алгоритмов сортировки.

Выделены следующие задачи лабораторной работы:

- описание операции сортировки;
- описание и реализация алгоритмов сортировки;
- проведение замеров процессорного времени работы алгоритмов при различных размерах массивов;
- оценка трудоёмкости алгоритов;
- проведение сравнительного анализа алгоритмов на основании экспериментов.

1.2. Описание операции сортировки

Сортировка массива по неубыванию - операция над массивом $arr[N]$, в результате которой в нём начинает выполняется условие[1]:

$$arr[i+1] >= arr[i], i \in [0, N-2] \tag{1.1}$$

Аналогично формулируется определение для сортировки по невозрастанию. В случае, если в массиве нет равных элементов, также возможно применить операции сортироки по возрастанию и убыванию.

2. Конструкторская часть

Рассмотрим вышеупомянутые алгоритмы сортировки. Для удобства изложения сути алгоритмов, будем рассматривать сортировку по неубыванию. Алгоритмы сортировки других порядков могут быть получены заменой условия сравнения.

2.1. Сортировка пузырьком

Алгоритм сортировки пузырьком основывается на следующем действии. Массив прссматривается от 0 до N-2 элемента, и в случае, если текущий элемент массива больше следующего, они меняются местами. Таким образом, после первого прохода в конце массива окажется максимальный элемент, после второго - два максимальных, и так далее до полного упорядочивания массива.

Схема алгоритма приведена на рисунке 2.1.

2.2. Поразрядная сортировка

Цель алгоритма заключается в реализации трудоёмкости, линейно зависящей от размера массива. Упорядоченнсть достигается последовательной сортировкой по значению разрядов (в порядке от меньшего разряда к большему). Алгоритм применим к массиву, состоящему из целых положительных чисел или иных значений, которые м.б. спроецированы на множетво положительных чисел [3].

Будем считать, что максимальное число в массиве состоит из K разрядов. В случае, если массив содержит целыйе отрицательные числа, то перед сортировкой все числа должны быть увеличены на модуль минимального числа, а после сортировки уменьшены.

Схема алгоритма приведена на рисунках 2.2. и 2.3.

2.3. Сортировка слиянием

Идея алгоритма заключается в том, что достаточно эффективно производится операция слияния (т.е. создания из двух массивов одного) при условии, что оба сливаемых массива уже упорядочены. Алгоритм разбивает массив на два подмассива, после чего применяет к ним ту же операцию, до тех пор, пока длина массива не станет меньше 3. После завершения сортировки подмассивов производится построение упорядоченного массива их слиянием.

Схема алгоритма приведена на рисунке 2.4.

2.4. Требования к программному обеспечению

Для полноценной проверки и оценки алгоритмов необходимо выполнить следующее.

1. Обеспечить возможность консольного ввода массива и выбора алгоритма сортировки. Программа должна вывести отсортированный массив.

2. Реализовать функцию замера процессорного времени, затраченного функцией. Для этого также создать возможность ввода размера массива, на котором будет выполнен замер.

2.5. Заготовки тестов

При проверке алгоритмов необходимо будет использовать следующие классы тестов:

- массив размером 1;
- массив одинаковых элементов;
- упорядоченный и обратно упорядоченный массив.

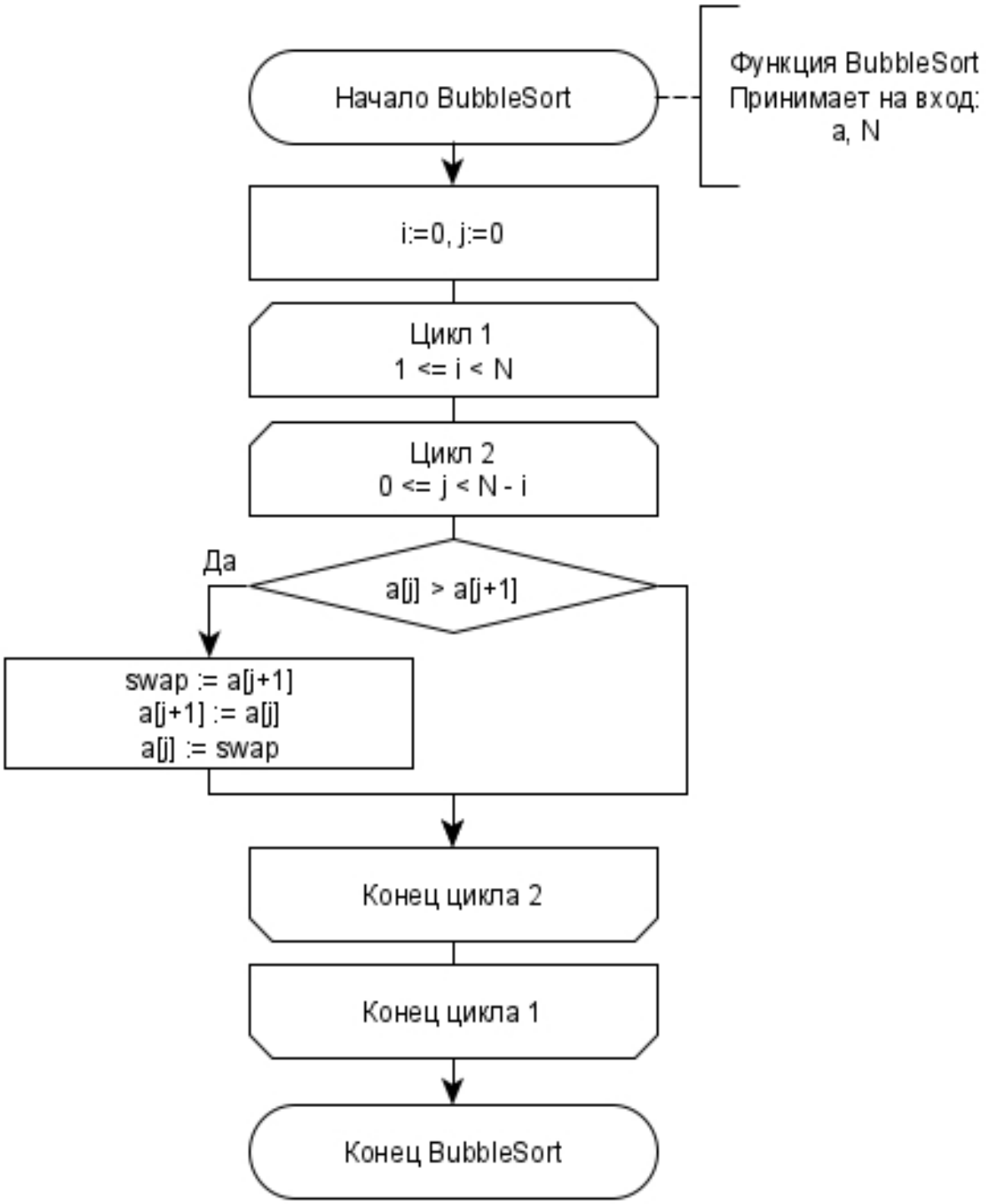

Рис. 2.1: Сортировка пузырьком

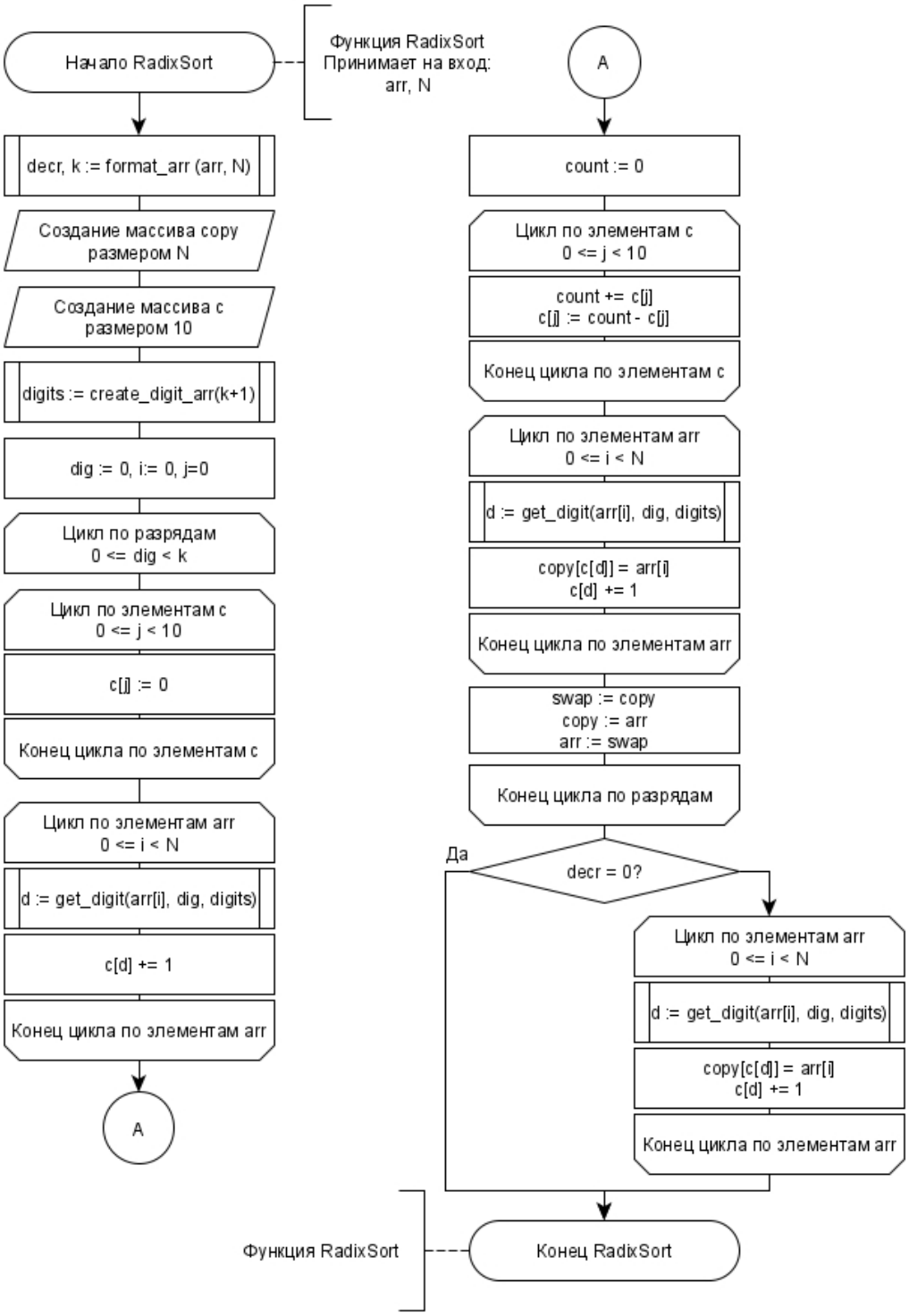

Рис. 2.2: Поразрядная сортировка (часть 1)

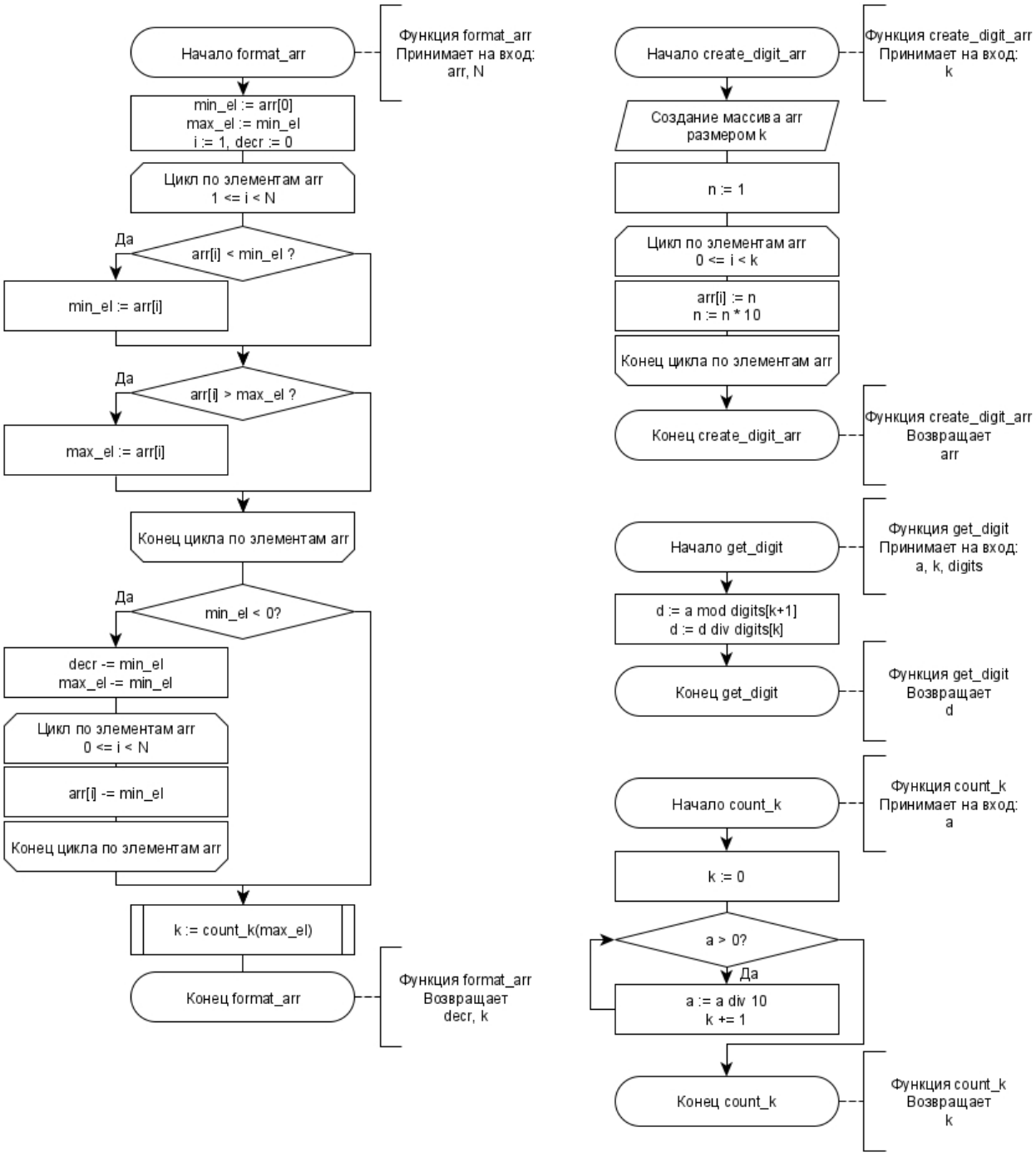

Рис. 2.3: Поразрядная сортировка (часть 2)

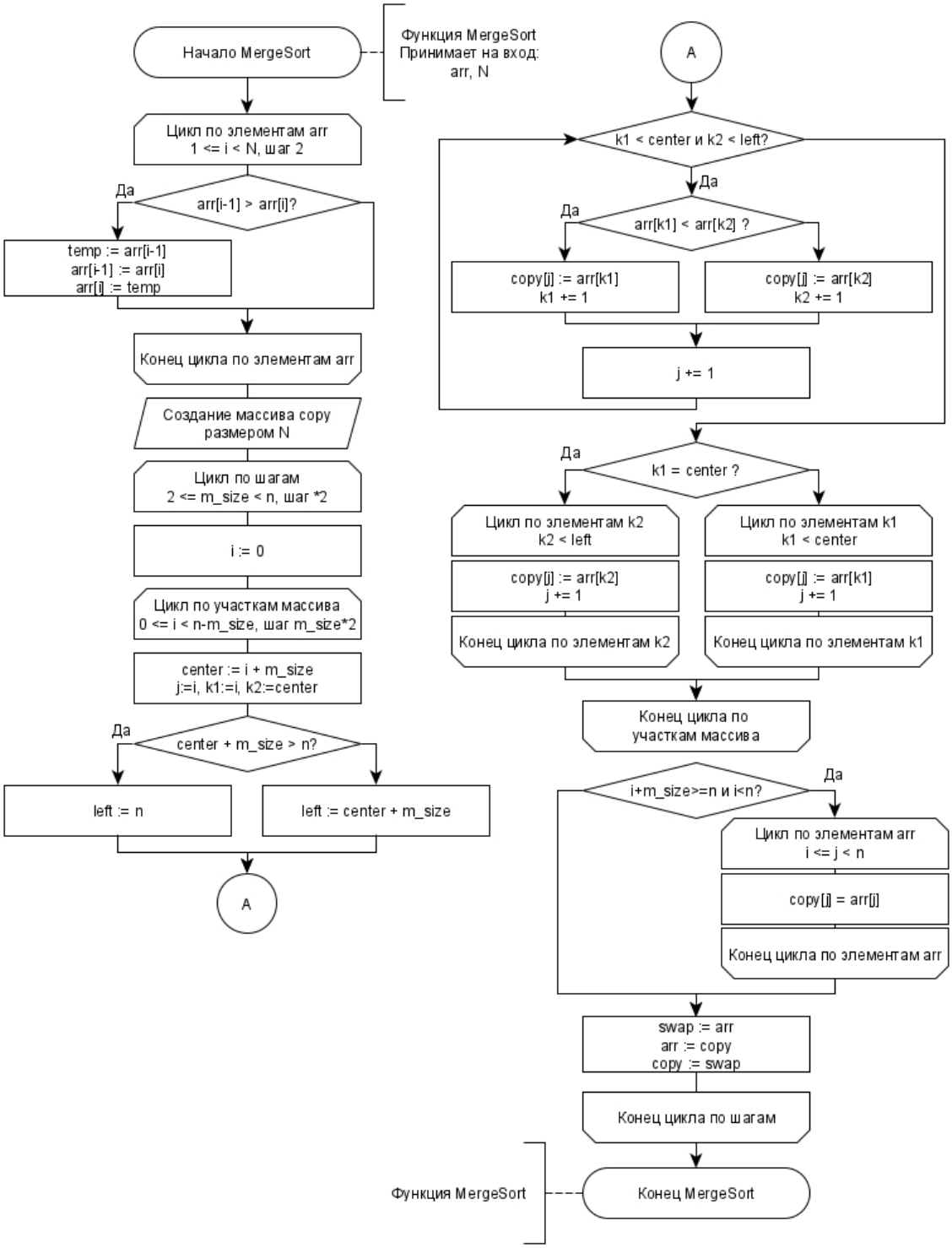

Рис. 2.4: Сортировка слиянием

3. Технологическая часть

3.1. Выбор языка программирования

В качестве языка программирования был выбран C++, так как имеется опыт работы с ним, и с библиотеками, позволяющими провести исследование и тестирование программы. Также в языке имеются средства для отключения оптимизации компилятора.

3.2. Листинг кода

Реализация алгоритмов умножения матриц представлена на листингах 3.1-3.3.

Листинг 3.1: Функция сортировки пузырьком.

```
#include "bubble.h"

#pragma optimize( "", off )
void bubble_sort(arr_t& arr, int n)
{
  content_t temp;
  for (int i=1; i<n; i++)
    for (int j=0; j<n-i; j++)
      if (arr[j] > arr[j + 1])
      {
        temp = arr[j + 1];
        arr[j + 1] = arr[j];
        arr[j] = temp;
      }
}
#pragma optimize( "", on )
```

Листинг 3.2: Функция поразрядной сортировки.

```
#include "radix.h"
#pragma optimize( "", off )

```

```
4  int* create_digit_arr(int k)
5  {
6    int* arr = create_arr(k);
7    int n = 1;
8    for (int i = 0; i < k; i++, n *= 10)
9      arr[i] = n;
10   return arr;
11 }
12
13 inline int count_k(int a)
14 {
15   int k = 0;
16   while (a > 0)
17   {
18     a /= 10;
19     k++;
20   }
21   return k;
22 }
23
24 inline int get_digit(int a, int k, int* digits)
25 {
26   a %= digits[k+1];
27   a /= digits[k];
28   return a;
29 }
30
31 void format_arr(arr_t arr, int n, int& decr, int& k)
32 {
33   int min_el = arr[0];
34   int max_el = min_el;
35   for (int i = 1; i < n; i++)
36   {
37     int el = arr[i];
```

```
38      if (el < min_el)
39        min_el = el;
40      if (el > max_el)
41        max_el = el;
42    }
43
44    if (min_el < 0)
45    {
46      decr = -min_el;
47      max_el -= min_el;
48      for (int i = 0; i < n; i++)
49        arr[i] -= min_el;
50    }
51    else
52    {
53      decr = 0;
54    }
55
56    k = count_k(max_el);
57  }
58
59  void radix_sort(arr_t& arr, int n)
60  {
61    int k, decr;
62    format_arr(arr, n, decr, k);
63    int c[10]; // = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
64    int* digits = create_digit_arr(k+1);
65    arr_t copy = create_arr(n);
66    arr_t swap_arr;
67
68    for (int dig = 0; dig < k; dig++)
69    {
70      for (int i = 0; i < 10; i++)
71        c[i] = 0;
```

```
72
73      for (int i = 0; i < n; i++)
74        c[get_digit(arr[i], dig, digits)]++;
75
76      int count = 0;
77      for (int i = 0; i < 10; i++)
78      {
79        count += c[i];
80        c[i] = count - c[i];
81      }
82
83      for (int i = 0; i < n; i++)
84        copy[c[get_digit(arr[i], dig, digits)]++] = arr[i];
85
86      swap_arr = arr;
87      arr = copy;
88      copy = swap_arr;
89    }
90
91    if (decr)
92    for (int i = 0; i < n; i++)
93      arr[i] -= decr;
94
95    free_arr(digits);
96    free_arr(copy);
97  }
98  #pragma optimize( "", on )
```

Листинг 3.3: Функция сортировки слиянием.

```
1  #include "merge.h"
2  #pragma optimize( "", off )
3
4  void merge_sort(arr_t& arr, int n)
```

```
5 {
6   content_t temp;
7   for (int i = 1; i < n; i += 2)
8     if (arr[i - 1] > arr[i])
9     {
10      temp = arr[i - 1];
11      arr[i - 1] = arr[i];
12      arr[i] = temp;
13    }
14
15  arr_t copy = create_arr(n);
16  arr_t swap_arr;
17  for (int merge_size = 2; merge_size < n; merge_size *= 2)
18  {
19    int i = 0;
20    for (; i+merge_size < n; i += merge_size * 2)
21    {
22      int center = i + merge_size;
23      int left;
24      if (center + merge_size > n)
25      left = n;
26      else
27      left = center + merge_size;
28
29      int j = i;
30      int k1 = i;
31      int k2 = center;
32      while (k1 < center && k2 < left)
33      {
34        if (arr[k1] < arr[k2])
35        {
36          copy[j] = arr[k1];
37          k1++;
38        }
```

```
39          else
40          {
41            copy[j] = arr[k2];
42            k2++;
43          }
44          j++;
45        }
46
47        if (k1 == center)
48          for (; k2 < left; k2++, j++)
49            copy[j] = arr[k2];
50        else
51          for (; k1 < center; k1++, j++)
52            copy[j] = arr[k1];
53      }
54
55      if (i + merge_size >= n && i < n)
56      {
57        for (int j = i; j < n; j++)
58          copy[j] = arr[j];
59      }
60
61      swap_arr = arr;
62      arr = copy;
63      copy = swap_arr;
64    }
65
66    free_arr(copy);
67  }
68
69  #pragma optimize( "", on )
```

3.3. Результаты тестирования

Для тестирования написанных функций был создан отдельный файл с ранее описанными классами тестов. Тестирование функций проводилось за счёт сравнения результов функций с резельтатом функции из стандартной библиотеки algorithm.

Состав тестов приведён в листинге 3.4.

Листинг 3.4: Модульные тесты

```
#include "tests.h"
// Сравнение работы функций со стандартной
bool _cmp_funcs(arr_t arr, int n)
{
  bool flag = true;
  sort_func f_arr[] = { bubble_sort, merge_sort, radix_sort
   };
  arr_t std_arr = copy_arr(arr, n);
  std::sort(std_arr, std_arr + n);

  for (int i = 0; i < 3 && flag; i++)
  {
    arr_t my_arr = copy_arr(arr, n);
    f_arr[i](my_arr, n);
    flag = is_equal_arr(std_arr, my_arr, n);
    free_arr(my_arr);
  }

  free_arr(std_arr);
  return flag;
}

// Массив размером 1
void _size_one_test()
```

```
24 {
25   std::string msg;
26   msg = __FUNCTION__;    msg += " - OK";
27
28   arr_t arr = create_arr(1);
29   arr[0] = 42;
30
31   if (!_cmp_funcs(arr, 1))
32   {
33     msg = __FUNCTION__;    msg += " - FAILED";
34   }
35
36   std::cout << msg << std::endl;
37   free_arr(arr);
38 }
39 // Массив одинаковых элементов
40 void _same_test()
41 {
42   std::string msg;
43   msg = __FUNCTION__;    msg += " - OK";
44
45   int n = 10;
46   arr_t arr = create_arr(n);
47   for (int i = 0; i < n; i++)
48   arr[i] = 42;
49
50   if (!_cmp_funcs(arr, n))
51   {
52     msg = __FUNCTION__;    msg += " - FAILED";
53   }
54
55   std::cout << msg << std::endl;
56   free_arr(arr);
57 }
```

```
58 // Отсортированный массив
59 void _sorted_test()
60 {
61   std::string msg;
62   msg = __FUNCTION__;    msg += " - OK";
63
64   int n = 100;
65   arr_t arr = random_arr(n);
66   std::sort(arr, arr + n);
67
68   if (!_cmp_funcs(arr, n))
69   {
70     msg = __FUNCTION__;    msg += " - FAILED";
71   }
72
73   std::cout << msg << std::endl;
74   free_arr(arr);
75 }
76 //  Отсортированный в обратном порядке массив
77 void _reverce_sorted_test()
78 {
79   std::string msg;
80   msg = __FUNCTION__;    msg += " - OK";
81
82   int n = 100;
83   arr_t arr = random_arr(n);
84   std::sort(arr, arr + n);
85   std::reverse(arr, arr + n);
86
87   if (!_cmp_funcs(arr, n))
88   {
89     msg = __FUNCTION__;    msg += " - FAILED";
90   }
91
```

```
92    std::cout << msg << std::endl;
93    free_arr(arr);
94 }
95
96 void run_tests()
97 {
98    _size_one_test();
99    _same_test();
100   _sorted_test();
101   _reverce_sorted_test();
102 }
```

Все тесты пройдены успешно.

3.4. Оценка трудоёмкости

Произведём оценку трудоёмкости алгоритов. Будем считать, что сортируется массив A$[N]$, максимальная разница двух элементов состоит из K элементов.

3.4.1. Алгоритм сортировки пузырьком

$$f_{bub} = 2 + (N-1) \cdot (2 + 3 + \tfrac{N-1+1}{2} \cdot [3 + 4 + \begin{cases} 0, & \text{л.с.} \\ 9, & \text{х.с.} \end{cases}])$$

$$f_{bub} = 2 + (N-1) \cdot (5 + \tfrac{N}{2} \cdot [7 + \begin{cases} 0, & \text{л.с.} \\ 9, & \text{х.с.} \end{cases}])$$

Лучший случай (отсортированный массив):

$f_{bub} = 2 + (N-1) \cdot (5 + \frac{N}{2} \cdot 7) = \frac{7}{2}N^2 + \frac{3}{2}N - 3$

Худший случай (массив в обратном порядке):

$f_{bub} = 2 + (N-1) \cdot (5 + 9N) = 9N^2 - 4N - 3$

3.4.2. Алгоритм поразрядной сортировки

Функция нормализации массива:

$$f_{format} = 2+2+N\cdot[2+4+\begin{cases} 0, & \text{л.с.} \\ 1, & \text{х.с.} \end{cases}]+2+\begin{cases} 0, & \text{л.с.} \\ 1+2+N\cdot(2+2), & \text{х.с.} \end{cases}+$$

$$1+(2+3K)$$

$$f_{format} = 9+3K+7N+\begin{cases} 0, & \text{л.с.} \\ 3+4N, & \text{х.с.} \end{cases}$$

Функция получения значения разряда:

$$f_{dig} = 3+2 = 5$$

Функция поразрядной сортировки:

$$f_{rad} = f_{format}+1+2+K\cdot(2+2+10(2+1)+2+N\cdot(2+f_{dig}+2)+$$

$$1+2+10(2+6)+2+N\cdot(2+f_{dig}+5)+3)+1+\begin{cases} 0, & \text{л.с.} \\ 2+N\cdot(2+2), & \text{х.с.} \end{cases}$$

$$f_{rad} = 24KN+125K+7N+12+\begin{cases} 0, & \text{л.с.} \\ 5+8N, & \text{х.с.} \end{cases}$$

Лучший случай (без отрицательных чисел):

$$f_{rad} = 24KN+125K+7N+12$$

Худший случай (с отрицательными числами):

$$f_{rad} = 24KN+125K+15N+17$$

3.4.3. Алгоритм сортировки слиянием

$$f_{mer} = 2+\tfrac{N}{2}\cdot(2+4+\begin{cases} 0, & \text{л.с.} \\ 10, & \text{х.с.} \end{cases})+1+2+\log_2(N)\cdot(2+$$

$$2+\frac{2(1-2^{\log_2(N)})}{(1-2)\cdot\log_2(N)}\cdot[4+2+1+\begin{cases} 1, & \text{л.с.} \\ 2, & \text{х.с.} \end{cases}+3+2+1+\frac{log_2(N)*N}{\frac{2(1-2^{\log_2(N)})}{(1-2)}}\cdot$$

$$\left\{ \begin{array}{ll} (3+4+1)/2+(3+3)/2, & \text{л.с.} \\ 3+4+1, & \text{х.с.} \end{array} \right.]+5+3+\left\{ \begin{array}{ll} 0, & \text{л.с.} \\ 2+\dfrac{2(1-2^{\log_2(N)})}{(1-2)\cdot\log_2(N)}\cdot(2+3), & \text{х.с.} \end{array} \right.)$$

$$f_{mer} = -21+29N+\left\{ \begin{array}{ll} 0, & \text{л.с.} \\ 5N, & \text{х.с.} \end{array} \right. +12\log_2(N)+\left\{ \begin{array}{ll} 2N-2, & \text{л.с.} \\ 4N-4, & \text{х.с.} \end{array} \right. +$$

$$(log_2(N)*N)\cdot\left\{ \begin{array}{ll} 7, & \text{л.с.} \\ 8, & \text{х.с.} \end{array} \right. +\left\{ \begin{array}{ll} 0, & \text{л.с.} \\ 2\log_2(N)+10(N-1), & \text{х.с.} \end{array} \right.$$

Лучший случай (N является степенью 2, отсортированный массив):

$$f_{mer} = 7N\log_2(N)+31N+12\log_2(N)-23$$

Худший случай (N является степенью 2 - 1, случайный масив):

$$f_{mer} = 8N\log_2(N)+48N+14\log_2(N)-35$$

3.5. Оценка времени

Для замера процессорного времени исполнения функции используется функция QueryPerformanceCounter библиотеки windows.h[2]. Проведение измерений производится в функции, приведённой в листинге 3.5.

Листинг 3.5: Функция замера процессорного времени работы функции

```
1 void test_time(sort_func f, int n)
2 {
3   arr_t arr = random_arr(n);
4
5   int count = 0;
6   start_counter();
7   while (get_counter() < 3.0 * 1000)
```

```
8   {
9     fill_random_arr(arr, n, -10000, 10000);
10    f(arr, n);
11    count++;
12  }
13  double t = get_counter() / 1000;
14
15  start_counter();
16  for (int i = 0; i < count; i++)
17  fill_random_arr(arr, n, -10000, 10000);
18  t -= get_counter() / 1000;
19
20  cout << "Выполнено " << count << " операций за " << t << "
    секунд" << endl;
21  cout << "Время: " << t / count << endl;
22
23  free_arr(arr);
24 }
```

4. Исследовательская часть

4.1. План экспериментов

Измерения процессорного времени проводятся на массивах, заполненых целыми числами от -10^4 до 10^4. Содержание массивов сгенерировано случайным образом. Изучается серия экспериментов с размерностями массива: $20, 10^2, 10^3, 10^4, 10^5$

Для повышения точности, каждый замер производится пять раз, за результат берётся среднее арифметическое.

4.2. Результат экспериментов

По результатам измерений процессорного времени можно составить таблицу 4.1

Таблица 4.1: Результат измерений процессорного времени (в секундах)

Размерность	20	10^2	10^3	10^4	10^5
Пузырёк	$1.3 \cdot 10^{-6}$	$2.7 \cdot 10^{-5}$	$2.0 \cdot 10^{-3}$	0.25	30.7
Поразрядно	$3.2 \cdot 10^{-6}$	$1.1 \cdot 10^{-5}$	$1.2 \cdot 10^{-4}$	$1.1 \cdot 10^{-3}$	0.011
Слиянием	$8.1 \cdot 10^{-7}$	$6.8 \cdot 10^{-6}$	$8.0 \cdot 10^{-5}$	$1.2 \cdot 10^{-3}$	0.014

Эксперименты проводились на компьютере с характеристиками:

- ОС - Windows 10, 64 бит;
- Процессор - Intel Core i7 8550U (1800 МГц, 4 ядра, 8 логических процессоров);
- Объем ОЗУ: 8 ГБ.

4.3. Вывод

По результатам экспериментов можно заключить следующее.

- Сортировка пузырьком показывает самый большой прирост по процессорному времени по сравнению с другими алгоритмами при увеличении размерности массивов.
- Поразрядная сортировка уступает в скорости сортирвке слиянием при относительно небольших размерах массивов. При больших размерах поразрядная сортировка показывет себя лучше, но крайне незначительно. При этом, сортировка слиянием способна работать с большим типом данных.
- При увеличении размера массивов в 10 раз, наблюдается рост затраченного процессорного времени для сортировки пузырьком примерно в 100 раз, для поразрядной в 10 раз и для сортировки слиянием в 12 раз, что в целом соответсвует расчётам их трудоёмкости.

Заключение

В ходе лабораторной работы достигнута поставленная цель: оценка трудоёмкости алгоритмов сортировки массиов. Решены все задачи работы.

Были изучены и описаны понятия трудоёмкости и операции сортировки. Также были описаны, реализованы и оценены по трудоёмкости алгоритмы сортировки массивов. Проведены замеры процессорного времени работы каждого алгоритма при различных размерах массивов. На основании оценок и экспериментов проведён сравнительный анализ.

Из расчётов трудоёмкости и проведённых экспериментов было выявлено, что наибольшим приростом процессорного времени на случайно сгенерированном массиве целых чисел обладает сортировка пузырьком. Также было выявлено, что поразрядная сортировка обладает меньшим приростом в процессорном времени при увеличении размерности массива по сравнению с сортировкой слиянием даже при наличии в массиве отрицательных чисел, что является худшим случаем для поразрядной сортировки.

Список литературы

1. Методы сортировки: метод. указания к лаб. работе по дисциплине «Информационные технологии» для студентов направления подготовки бакалавра 210400 «Радиотехника» дневной формы обучения / НГТУ; Сост.: Е.Н.Приблудова, С.Б.Сидоров. Н.Новгород, 2012, 11 с. (дата обращения: 30.09.2020).

2. QueryPerformanceCounter function [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter, свободный (дата обращения: 28.09.2020).

3. Кормен, Томас Х. и др., Алгоритмы: построение и анализ, 3-е изд. : Пер. с англ. — М. : ООО "И.Д.Вильямс 2018. — 1328 с. : ил. — Парал. тит. англ. IBSN 978-5-8459-2016-4 (рус.)