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»

Лабораторная работа №3
по дисциплине "Анализ Алгоритмов"

Тема Алгоритмы сортировки

Студент Светличная А.А.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л., Строганов Ю.В.

Москва — 2022 г.

Оглавление

Введение

Сортировка – процесс перестановки объектов данного множества в определенном порядке. Сортировка служит для последующего облегчения поиска элементов в отсортированном множестве. При этом соритровка является примером огромного разообразия алгоритмов, выполняющих одну и ту же задачу. Несмотря на большое количество алгоритмов сортировки, любой из них можно разбить на три основные части:

- сравнение, определяющее порядок элементов в паре;
- перестановка, меняющая элементы в паре местами;
- сам сортирующий алгоритм, который выполняет два предыдущих шага до полного упорядочивания.

Ввиду большого количества алгоритмов сортировки одни из них имеют преимущества над другими, что приводит к необходимости их сравнительного анализа [1].

1 Аналитическая часть

1.1 Цель и задачи

Целью данной работы является исследование алгоритмов сортировок – блочной, пирамидальной, бусинами.

Для этого в ходе исследования алгоритмов требуется решить следующие **задачи**:

1) изучить и рассмотреть выбранные алгоритмы сортировок;
2) построить блок-схемы выбранных алгоритмов;
3) реализовать каждый из алгоритмов;
4) рассчитать их трудоемкость;
5) экспериментально оценить временные характеристики алгоритмов;
6) сделать вывод на основании проделанной работы.

1.2 Пирамидальная сортировка

Пирамидальная сортировка (или сортировка кучей, Heap sort) – это метод сортировки сравнением, основанный на такой структуре данных как двоичная куча.

Двоичная куча – это законченное двоичное дерево, в котором элементы хранятся в особом порядке: значение в родительском узле больше (или меньше) значений в его двух дочерних узлах. Первый вариант называется max-heap, а второй — min-heap. Куча может быть представлена двоичным деревом или массивом.

Суть пирамидальной сортировки:

1) построить max-heap из входных данных;
2) заменинть самый большой элемент из корня кучи на последний элемент кучи;

3) уменьшить размер кучи на 1;

4) преобразовать полученное дерево в max-heap с новым корнем;

5) повторить вышеуказанные действий, пока размер кучи больше 1.

1.3 Сортировка бусинами

Сортировка бусинами (или бисерная сортировка, Bead sort) – основана на внешнем виде и принципе действия античного «калькулятора».

Суть сортировки бусинами для набора натуральных чисел:

1) выложить друг под другом каждое число в виде горизонтального ряда из соответствующего числа шариков;

2) сдвинуть шарики вниз до упора;

3) переключиться на горизонтали и пересчитать бусинки в каждом ряду;

4) получить упорядоченный набор чисел.

Ограничения применимости:

- для натруальных чисел – метод работает без дополнительных действий;
- для целых чисел – обрабатывать отрицательные числа отдельно;
- для дробных чисел – привести к целым, умножив на 10k;
- для строк – представить в виде целого положительного числа.

1.4 Блочная сортировка

Блочная сортировка (или карманная сортировка, корзинная сортировка, Bucket sort) — алгоритм сортировки, в котором сортируемые элементы распределяются между конечным числом отдельных блоков так,

чтобы все элементы в каждом следующем по порядку блоке были всегда больше (или меньше), чем в предыдущем. Каждый блок затем сортируется отдельно, либо рекурсивно тем же методом, либо другим. Затем элементы помещаются обратно в массив.

Вывод

В данном разделе были изучены и рассмотрены алгоритмы сортировок по варианту.

2 Конструкторская часть

2.1 Описания алгоритмов

На рисунках 2.2, 2.2, 2.4 представлены схемы алгоритмов пирамидальной, бисерной и блочной сортировки соответсвенно.

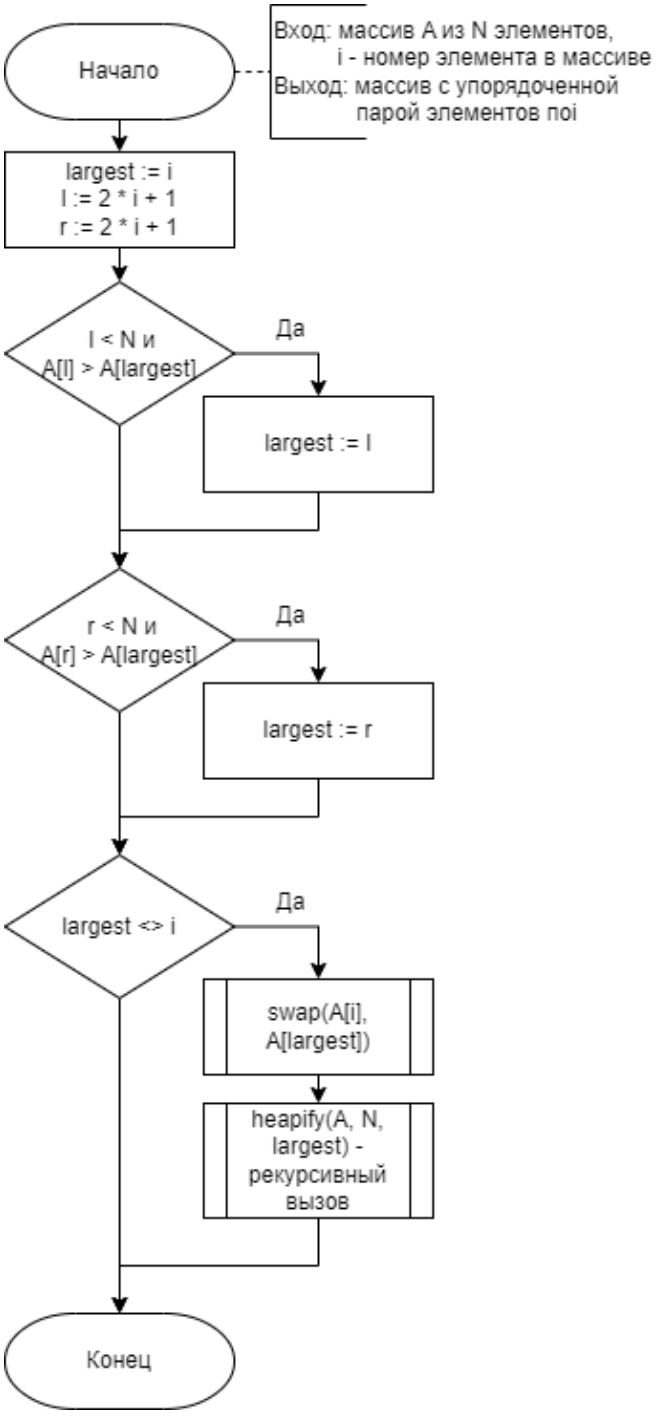

Рисунок 2.1 – Схема алгоритма рекурсивной подгромма пирамидальной сортировки

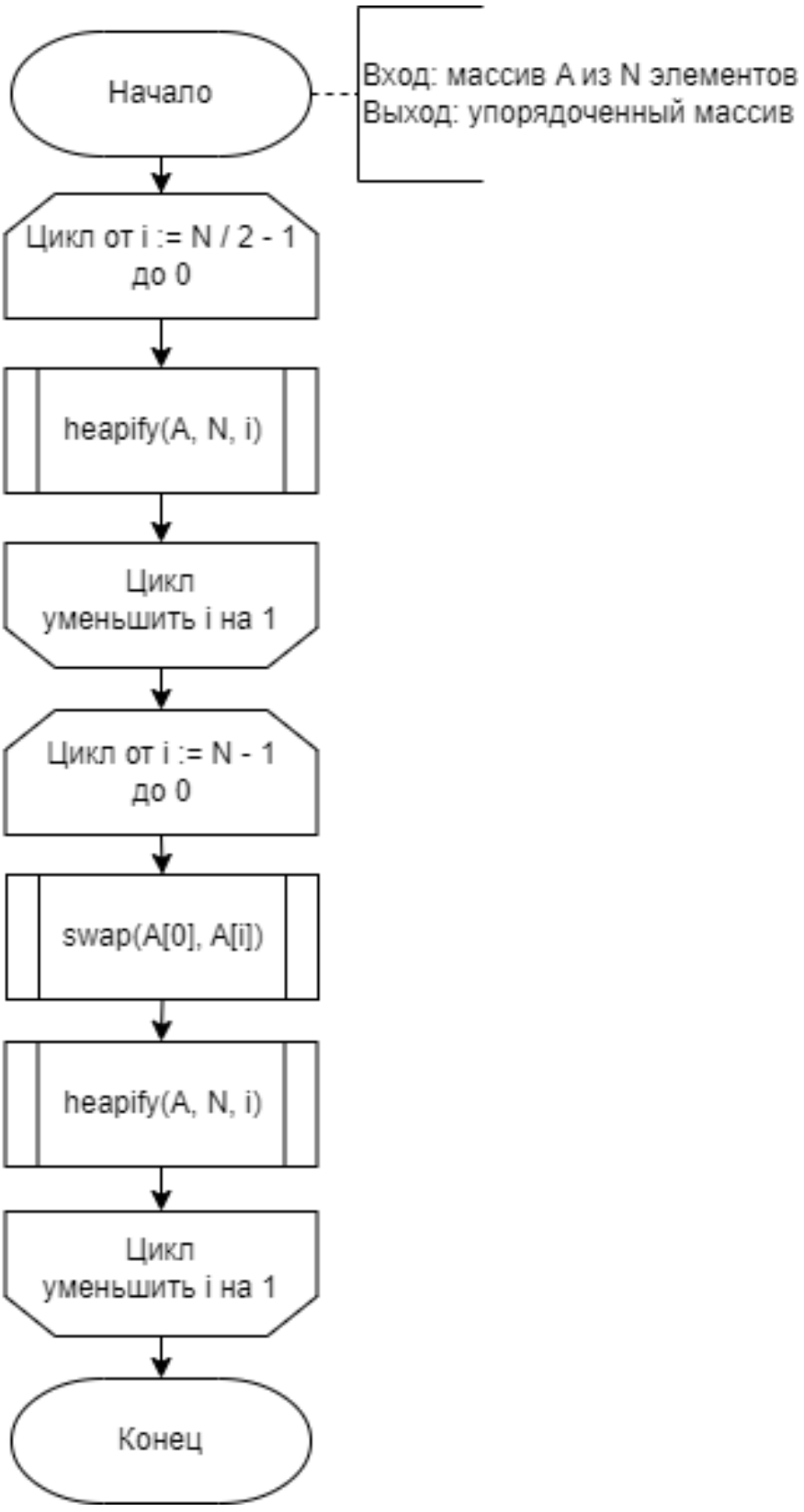

Рисунок 2.2 – Схема алгоритма пирамидальной сортировки

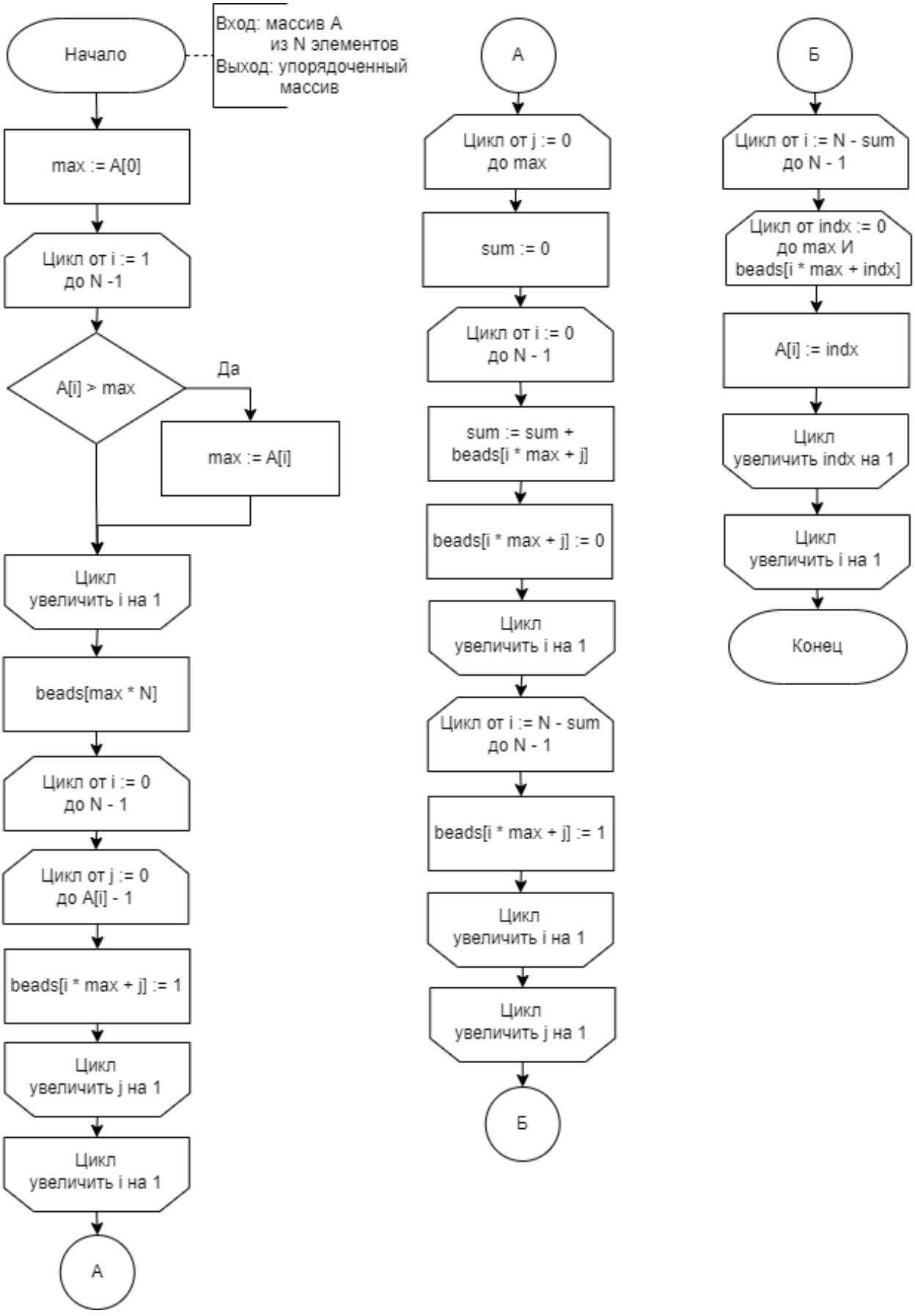

Рисунок 2.3 – Схема алгоритма бисерной сортировки

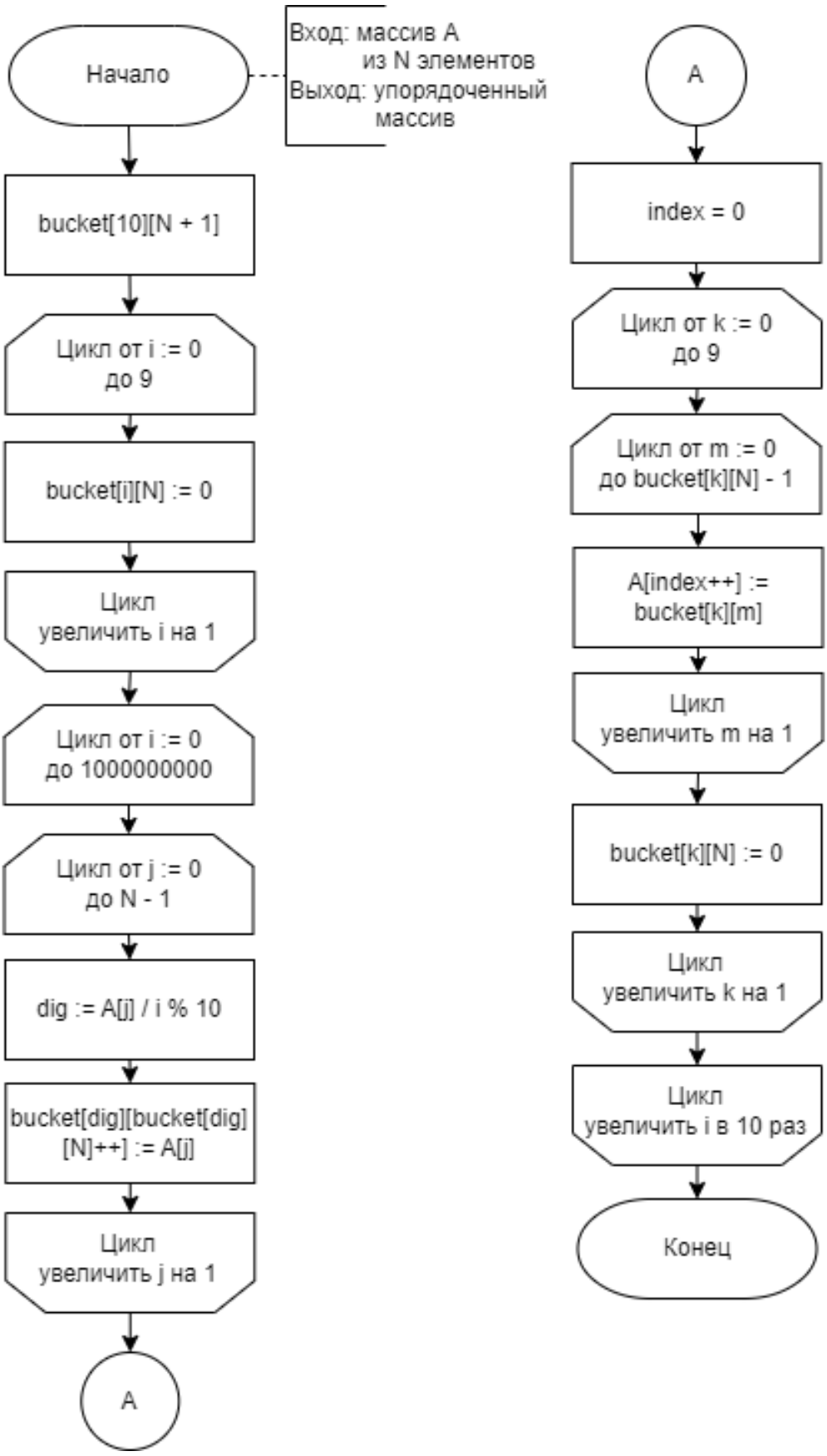

Рисунок 2.4 – Схема алгоритма блочной сортировки

2.2 Модель вычислений трудоемкости

1) Операции, имеющие трудоемкость 1:

$$+, -, \%, ==, !=, <, >, <=, >=, [], ++, --, << , >> .$$

2) Операции, имеющие трудоемкость 2:

$$*, /.$$

3) Трудоемкость оператора выбора if условие then A else B рассчитывается как:

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{условие выполняется,} \\ f_B, & \text{иначе.} \end{cases}$$

4) Трудоемкость вызова функции равна 0.

5) Трудоемкость цикла рассчитывается как:

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}).$$

2.3 Оценка сложности алгоритмов

1. **Блочная сортировка:** $f = 108N + 630K + 570$, где K — количество блоков.

   Однако сложность сортировки зависит и от сложности сортировки, выбранной для сортировки элементов в блоках. Таким образом, для данной сортировки определен только лучший случай — $O(N)$.

2. **Пирамидальная сортировка.**

   Функция сортировки состоит из двух модулей: процедура просеивания элемента через пирамиду и основная функция сортировки.

Для начала рассмотрим процедуру просеивания элемента через пирамиду.

Введем $T(N)$ — трудоемкость процедуры просеивания элемента через пирамиду, состоящую из N элементов. Это время для левой части пирамиды определяется по следующему далее соотношению.

$$T(N) = Q(1) + T(\frac{2}{3}N), \tag{2.1}$$

где Q(1) – трудоемкость, которая определяется процедурой обмена элемента с одним из своих «потомков».

Для просеивания элемента, расположенного на высоте h требуется $Q(h)$ операций. Число вершин на высоте h меньше или равно величине — $2^{\frac{N}{h}} - 1$ Трудоемкость построения пирамиды определяется по соотношению:

$$T(N) = \sum_{h=1}^{\log_2 N} (2^{\frac{n}{h}} - 1) * Q(h). \tag{2.2}$$

Таким образом, трудоемкость процедуры просеивания элемента через пирамиду — $f_1 = O(\log N)$.

Основная функциця сортировки — $f = 1.5N * f_1 + 8N + 4$.

Общая сложность сортировки — $O(N * \log N)$.

3. **Бисерная сортировка.**

   В теории при использовании более сложных алгоритмов сиспользованием параллельных вычислений можно добить сложности $O(N)$, однако большинство реализация расчитаны лишь на $O(N^2 * Max)$ [2].

Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых алгоритмов,е была проведена оценка сложности этих алгоритмов.

3 Технологическая часть

3.1 Требования к программному обеспечению

В программе должна быть возможность:

1) генирировать входные данные;

2) сортировать входных данные одинм указанных алгоритмов;

3) замерять процессорное время выполнения реализаций алгоритмов.

3.2 Выбор языка программирования

Для реализации алгоритмов поиска редакционных расстояний был выбран язык программирования C в силу наличия точных библиотек и быстродейственности языка.

3.3 Выбор библиотеки и способа для замера времени

Для замера процессорного времени выполнения реализаций агоритмов была выбрана не стандартная функция библиотеки <time.h> языка C — clock(), которая недостаточно четко работает при замерах небольших промежутков времени, а QueryPerformanceCounter - API-интерфейс, использующийся для получения меток времени с высоким разрешением или измерения интервалов времени.

Для облегчения работы с данным инструментом были самостоятельно написаны обертки-функции, представленные на листинге 3.1.

Листинг 3.1 – Функции замеров процессорного времени

```
1     LARGE_INTEGER frequency;
2     LARGE_INTEGER t1,t2;
3     double elapsedTime;
4
5     void TIMER_INIT()
6     {
7         QueryPerformanceFrequency(&frequency);
8     }
9
10    void TIMER_START()
11    {
12      QueryPerformanceCounter(&t1);
13    }
14
15    double TIMER_STOP()
16    {
17        QueryPerformanceCounter(&t2);
18        elapsedTime=(float)(t2.QuadPart-t1.QuadPart)/frequency.
              QuadPart/COUNT*MICRO;
19
20      return elapsedTime;
21    }
```

В силу существования явления вытеснения процессов из ядра, квантования процессорного времени все процессорное время не отдается какой-либо одной задаче, поэтому для получения точных результатов необходимо усреднить результаты вычислений: замерить совокупное время выполнения реализации алгоритма N раз и вычислить среднее время выполнения.

3.4 Реализации алгоритмов

В листингах 3.2, 3.3, 3.4 приведены реализации алгоритмов блочной, пирамидальной и бисерной сортировок соответсвенно.

Листинг 3.2 – Алгоритм блочной сортировки

```
1    void bucket_sort(int *array, int array_size)
2    {
3        int bucket[10][array_size + 1];
4        for(int x = 0; x < 10; x++)
5            bucket[x][array_size] = 0;
6
7        for(int digit = 1; digit <= 1000000000; digit *= 10) {
8            for(int i = 0; i < array_size; i++) {
9                int dig = (array[i] / digit) % 10;
10               bucket[dig][bucket[dig][array_size]++] = array[i];
11           }
12
13           int idx = 0;
14           for(int x = 0; x < 10; x++) {
15               for(int y = 0; y < bucket[x][array_size]; y++)
16                   array[idx++] = bucket[x][y];
17               bucket[x][array_size] = 0;
18           }
19       }
20   }
```

Листинг 3.3 – Алгоритм пирамидальной сортировки

```
1   void swap(int *a, int *b)
2   {
3       int tmp;
4
5       tmp = *a;
6       *a = *b;
7       *b = tmp;
8   }
9
10  void heapify(int *arr, int array_size, int i)
11  {
12      int largest = i;
13
14      int l = 2 * i + 1;
15      int r = 2 * i + 2;
16
17      if (l < array_size && arr[l] > arr[largest])
18          largest = l;
19
20      if (r < array_size && arr[r] > arr[largest])
21          largest = r;
22
23      if (largest != i)
24      {
25          swap(&(arr[i]), &(arr[largest]));
26          heapify(arr, array_size, largest);
27      }
28  }
29
30  void heap_sort(int *arr, int array_size)
31  {
32      for (int i = array_size / 2 - 1; i >= 0; i--)
33          heapify(arr, array_size, i);
34
35      for (int i = array_size - 1; i >= 0; i--)
36      {
37          swap(&(arr[0]), &(arr[i]));
38          heapify(arr, i, 0);
39      }
40  }
```

Листинг 3.4 – Алгоритм бисерной сортировки

```
1   void bead_sort(int *array, int array_size)
2   {
3       int max = array[0];
4       for (int i = 1; i < array_size; i++)
5           if (array[i] > max)
6               max = array[i];
7
8       unsigned char *beads = calloc(1, max * array_size);
9       for (int i = 0; i < array_size; i++)
10          for (int j = 0; j < array[i]; j++)
11              beads[i * max + j] = 1;
12
13      for (int j = 0; j < max; j++) {
14          int sum = 0;
15          for (int i = 0; i < array_size; i++) {
16              sum += beads[i * max + j];
17              beads[i * max + j] = 0;
18          }
19
20          for (int i = array_size - sum; i < array_size; i++)
21              beads[i * max + j] = 1;
22      }
23
24      int indx;
25      for (int i = 0; i < array_size; i++) {
26          for (indx = 0; indx < max && beads[i * max + indx];
               indx++);
27          array[i] = indx;
28      }
29      free(beads);
30  }
```

3.5 Тестирование алгоритмов

В таблице 3.1 приведены проведенные тесты для алгоритмов блочной, пирамидальной и бисерной сортировок.

Таблица 3.1 – Функциональные тесты

Входной массив	Результат	Правильность
[1 2 3 4 5 6]	[1 2 3 4 5 6]	Верно
[6 5 4 3 2 1]	[1 2 3 4 5 6]	Верно
[7 3 8 5 0]	[0 3 5 7 8]	Верно
[5 5 5]	[5 5 5]	Верно
[9]	[9]	Верно
[]	[]	Верно

Вывод

В данном разделе были разработаны алгоритмы блочной, пирамидальной и бисерной сортировок, а также проведено тестирование.

4 Экспериментальная часть

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование программного обеспечения:

1) операционная система Windows-10, 64-bit;
2) оперативная память 8 ГБ;
3) процессор Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz, 2304 МГц, ядер 2, логических процессоров 4.

4.2 Замеры времени

4.2.1 Произвольно сгенерированный массив

В таблице 4.1 приведены результаты замеров в миллисекундах времени работы алгоритмов для входных произвольных массивов разной длины.

Таблица 4.1 – Замеры времени выполнения алгоритмов на произвольных массивах разной длины

Длина	BlockSort	HeapSort	BeadSort
100	0.03	0.05	1.06
200	1.09	1.14	3.15
300	3.21	3.27	6.15
400	6.22	6.30	10.08
500	10.17	10.27	15.41
600	15.52	15.65	22.04
700	22.17	22.33	29.29
800	29.44	29.62	37.87
900	38.03	38.24	47.17
1000	47.34	47.57	57.78

Из таблицы 4.1 видно, что блочная и пирамидальная сортировка работают за приблизительно равное время, по этой причине строить отдельно график каждой из данных сортировок нецелесообразно.

Зависимость времени работы алгоритмов блочной и бисерной сортировок от длины входных массивов представлена на рисунке 4.1.

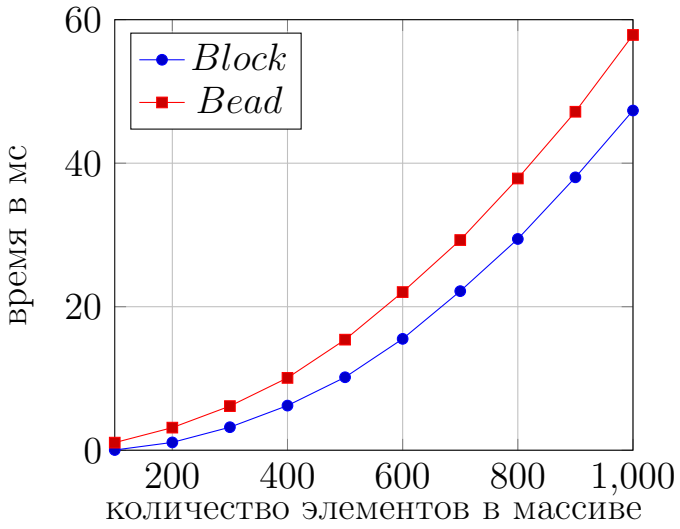

Рисунок 4.1 – Зависимость времени работы алгоритмов от длины входных массивов

4.2.2 Отсортированный массив

В таблице 4.2 приведены результаты замеров в миллисекундах времени работы алгоритмов для входных отсортированных массивов разной длины.

Таблица 4.2 – Замеры времени выполнения алгоритмов на отсортированных массивах разной длины

Длина	BlockSort	HeapSort	BeadSort
100	0.02	0.04	0.50
200	0.56	0.59	2.39
300	2.45	2.50	6.45
400	6.52	6.59	14.20
500	14.29	14.39	27.21
600	27.33	27.45	46.19
700	46.31	46.44	73.02
800	73.17	73.33	106.81
900	106.98	107.16	148.99
1000	149.17	149.37	201.06

Из таблицы 4.2 видно, что блочная и пирамидальная сортировка работают за приблизительно равное время, по этой причине строить отдельно график каждой из данных сортировок нецелесообразно.

Зависимость времени работы алгоритмов блочной и бисерной сортировок от длины входных массивов представлена на рисунке 4.2.

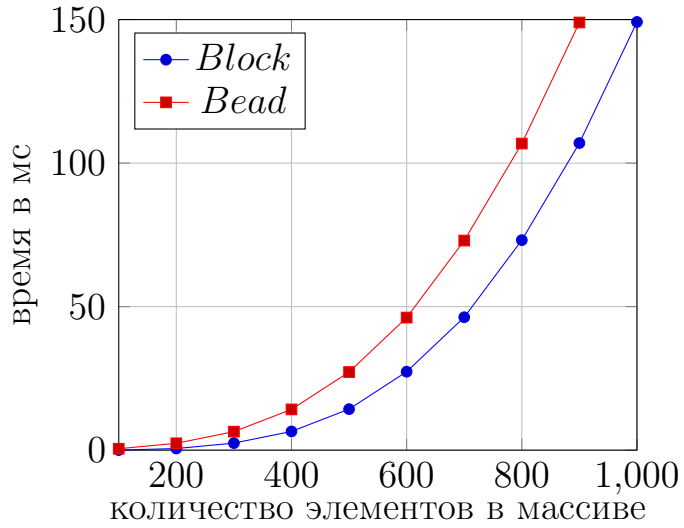

Рисунок 4.2 – Зависимость времени работы алгоритмов от длины входных массивов

4.2.3 Обратно отсортированный массив

В таблице 4.3 приведены результаты замеров в миллисекундах времени работы алгоритмов для входных обратно отсортированных массивов разной длины.

Таблица 4.3 – Замеры времени выполнения алгоритмов на обратно отсортированных массивах разной длины

Длина	BlockSort	HeapSort	BeadSort
100	0.02	0.03	8.26
200	8.30	8.34	22.75
300	22.81	22.86	43.35
400	43.42	43.49	68.40
500	68.49	68.58	102.30
600	102.40	102.52	142.06
700	142.22	142.39	184.00
800	184.14	184.30	237.57
900	237.75	237.95	286.56
1000	286.74	286.94	354.96

Из таблицы 4.3 видно, что блочная и пирамидальная сортировка работают за приблизительно равное время, по этой причине строить отдельно график каждой из данных сортировок нецелесообразно.

Зависимость времени работы алгоритмов блочной и бисерной сортировок от длины входных массивов представлена на рисунке 4.3.

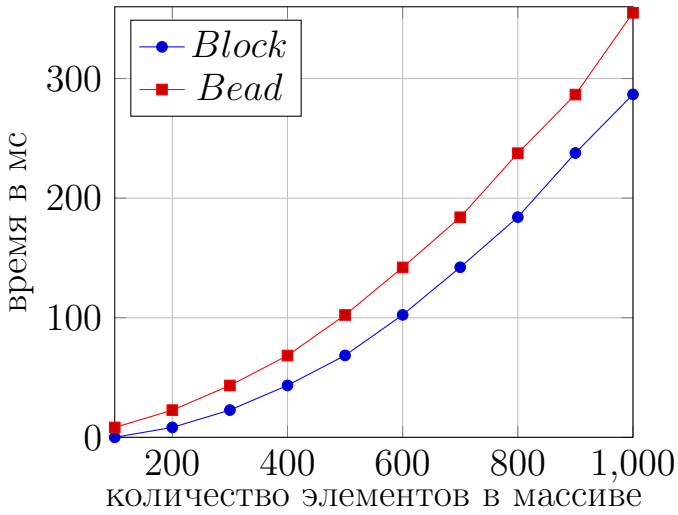

Рисунок 4.3 – Зависимость времени работы алгоритмов от длины входных массивов

Вывод

Результаты замеров показали, что бисерная сортировка во всех случаях работает дольше всего. А блочная и пирамидальная сортировки показали почти одинаковый результат, однако во всех случаях пирамидальная сортировка незначительно медленнее. Замеры подтверждают теоретическую оценку трудоемкостей данных алгоритмов.

Заключение

Цель лабораторной работы достигнута – исследованы алгоритмы сортировок – блочной, пирамидальной, бусинами.

Все задачи решены:

1) изучены и рассмотрены выбранные алгоритмы сортировок;
2) построены блок-схемы выбранных алгоритмов;
3) реализован каждый из алгоритмов;
4) рассчитана их трудоемкость;
5) экспериментально оценены временные характеристики алгоритмов;
6) сделан вывод на основании проделанной работы.

На основании проведенных экспериментов было определено, что время работы алгоритмов с увелечением длины входных массивов увеличивается в геометрической прогрессии, причем блочная и пирамидальная сортировки работают также, как бисерная с количеством элементов на 100 меньше, таким образом данная сортировка самая медленная. А результаты двух других отличаются незначительно.

Список использованных источников

[1] Вирт Н. Алгоритмы и структуры данных. М.: Мир, 1989. с. 360.

[2] Bead–Sort: A Natural Sorting Algorithm. URL: https://www.researchgate.net/publication/37987842_Bead–Sort_A_Natural_Sorting_Algorithm (дата обращения: 25.10.2022).