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»

Отчет по лабораторной работе № 3 по дисциплине “Анализ алгоритмов”

Тема Трудоемкость сортировок

Студент Ву Хай Данг

Группа ИУ7и-52Б

Оценка (баллы)

Преподаватель Строганов Д.В., Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение

Сортировка — процесс перегруппировки заданной последовательности объектов в определенном порядке. Это одна из главных процедур обработки структурированной информации [1].

Алгоритмы сортировки имеют большое значение, так как позволяют эффективнее проводить работу с последовательностью данных. Например, возьмем задачу поиска элемента в последовательности — при работе с отсортированным набором данных время, которое нужно на нахождение элемента, пропорционально логарифму количества элементов. Последовательность, данные которой расположены в хаотичном порядке, занимает время, которое пропорционально количеству элементов, что куда больше логарифма.

Целью данной лабораторной работы является изучение алгоритмов сортировки. Для достижения поставленной цели требуется выполнить следующие задачи:

1) Описание трех алгоритмов сортировки:
    * Битонная сортировка;
    * Поразрядная сортировка;
    * Сортировка пузырьком.
2) Построение схемы рассматриваемых алгоритмов;
3) Создание программного обеспечения, реализующего перечисленные алгоритмы;
4) Проведение сравнительного анализа реализаций алгоритмов по затраченному процессорному времени и памяти;
5) Описание и обоснование полученных результатов в отчете о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе будут рассмотрены три алгоритма сортировок: битонная сортировка, поразрядная сортировка, сортировка пузырьком.

1.1 Битонная сортировка

Битонная сортировка (англ. Bitonic sorter) — параллельный алгоритм сортировки данных, метод для создания сортировочной сети [2]. Разработан американским информатиком Кеннетом Бэтчером в 1968 году. В основе алгоритма лежит понятие «битонной последовательности». Название было выбрано по аналогии с монотонной последовательностью.

Алгоритм основан на сортировке битонных последовательностей. Такой последовательностью называется последовательность, которая сначала монотонно не убывает, а затем монотонно не возрастает, либо приводится к такому виду в результате циклического сдвига.

Процесс битонного слияния преобразует битонную последовательность в полностью отсортированную последовательность. Алгоритм битонной сортировки состоит из применения битонных преобразований до тех пор, пока множество не будет полностью отсортировано.

1.2 Поразрядная сортировка

Люди изобрели много сортировок для разных данных и под разные задачи. **Поразрядная сортировка** — это почти как сортировка по алфавиту, но для данных. В английском языке это называется Radix sort — сортировка по основанию системы счисления [3].

Алгоритм поразрядной сортировки гениален в том, что сортирует не числа целиком, а значения разрядов. Получается, что он как бы разбирается с числами на уровне единиц, десятков, сотен и т. д. и только потом он делает общую сортировку. Это позволяет ему не бегать по всем

сравниваемым числам и не делать миллион сравнений. Отсюда и экономия времени.

1.3 Сортировка пузырьком

Сортировка пузырьком — один из самых известных алгоритмов сортировки. Здесь нужно последовательно сравнивать значения соседних элементов и менять числа местами, если предыдущее оказывается больше последующего [4]. Таким образом элементы с большими значениями оказываются в конце списка, а с меньшими остаются в начале.

1.4 Вывод

В данной работе стоит задача реализации 3 алгоритмов сортировки, а именно: битонная сортировка, поразрядная сортировка и сортировка пузырьком.

2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов сортировки: алгоритма битонной сортировки, алгоритма поразрядной сортировки и алгоритма сортировки пузырьком. .

2.1 Разработка алгоритмов

На рисунке 2.1 – 2.2 приведена схема алгоритма битонной сортировки. Схема алгоритма поразрядной сортировки приведена на рисунках 2.3 – 2.5, схема алгоритма сортировки пузырьком приведена на рисунках 2.6.

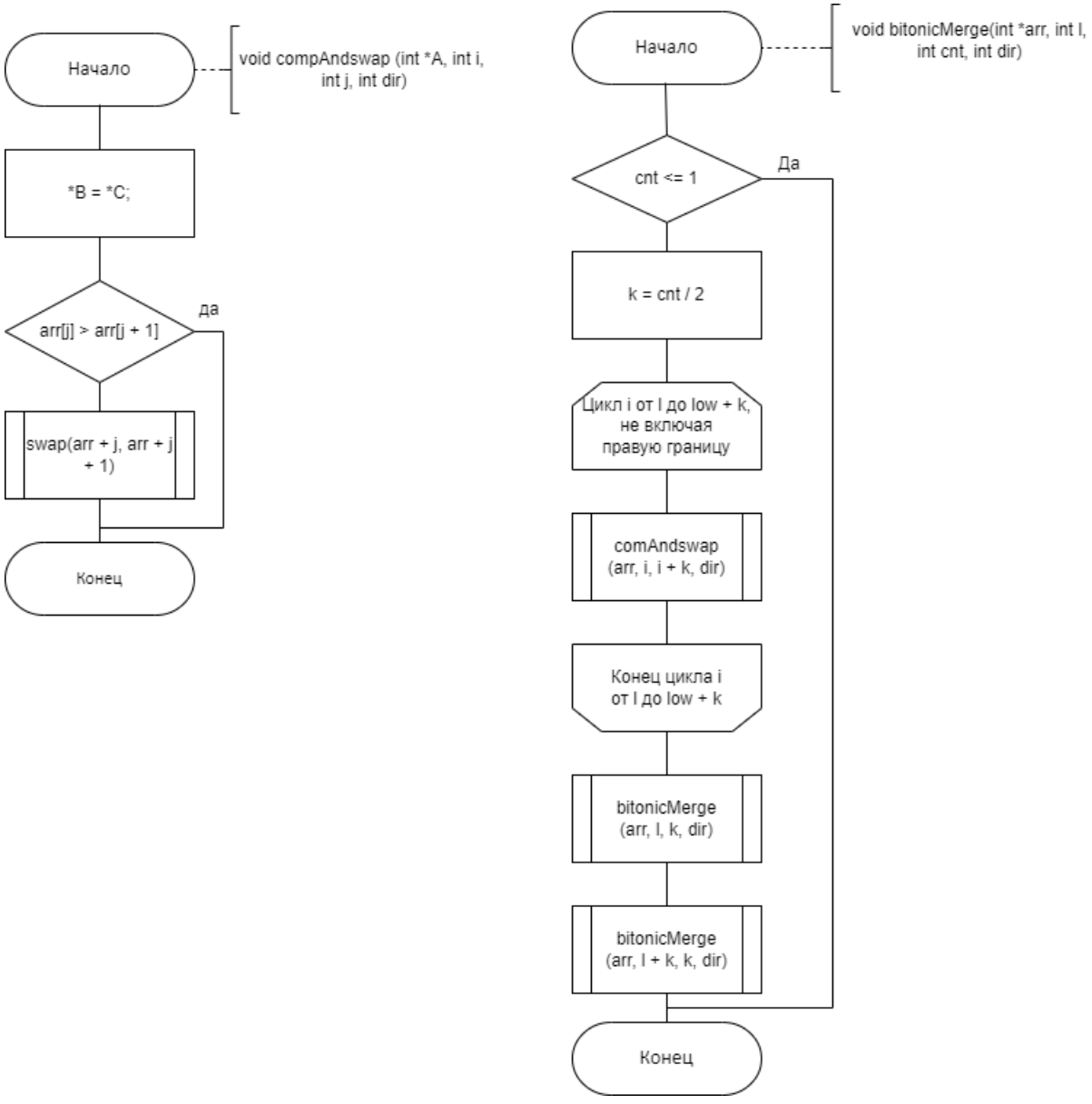

Рисунок 2.1 – Алгоритм битонной сортировки

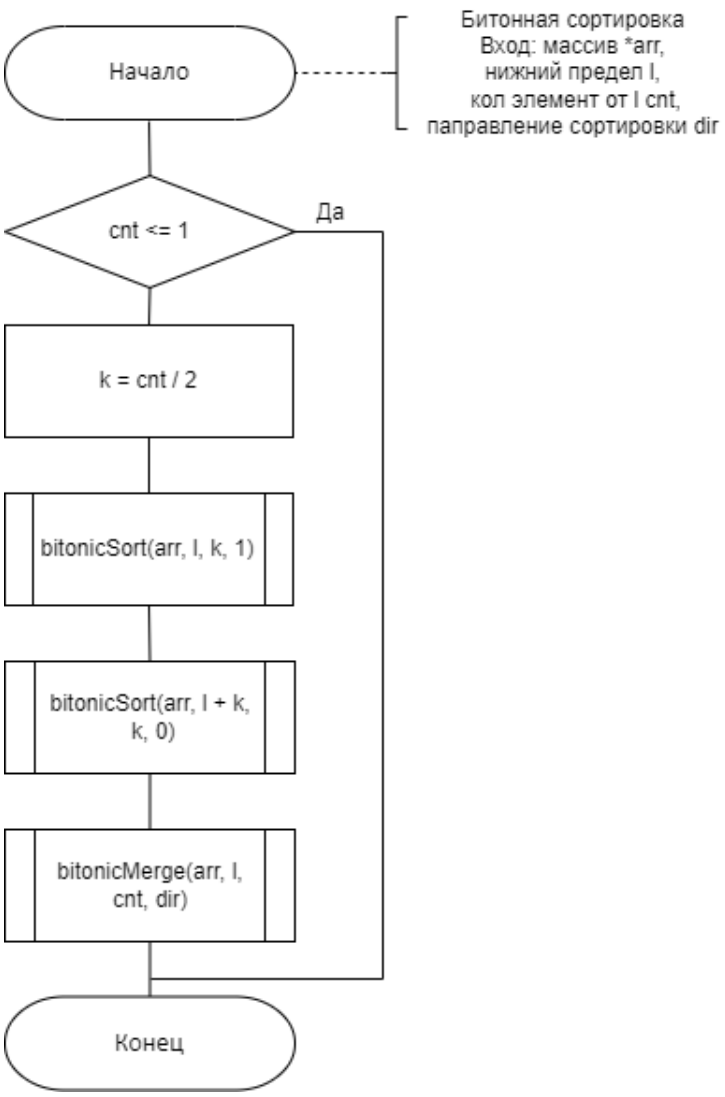

Рисунок 2.2 – Алгоритм битонной сортировки (продолжение)

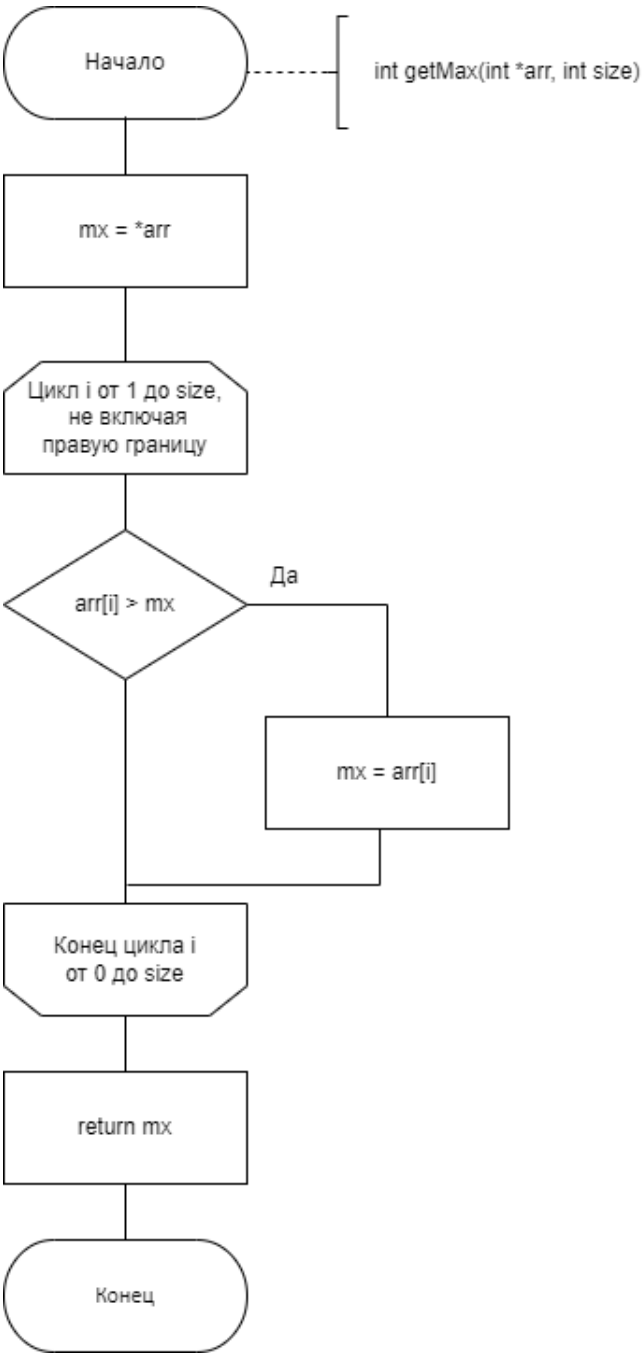

Рисунок 2.3 – Алгоритм поразрядной сортировки

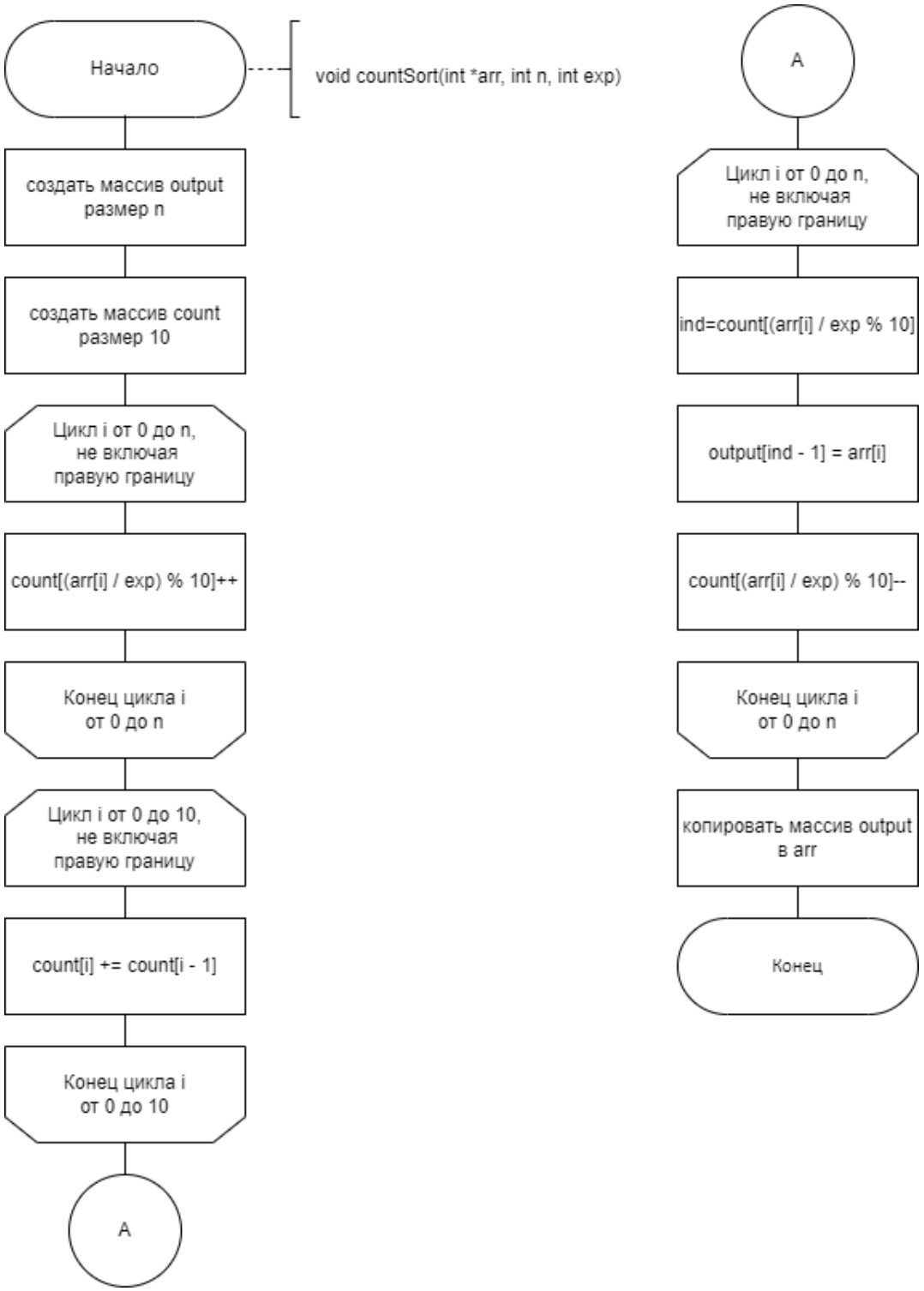

Рисунок 2.4 – Алгоритм поразрядной сортировки (продолжение)

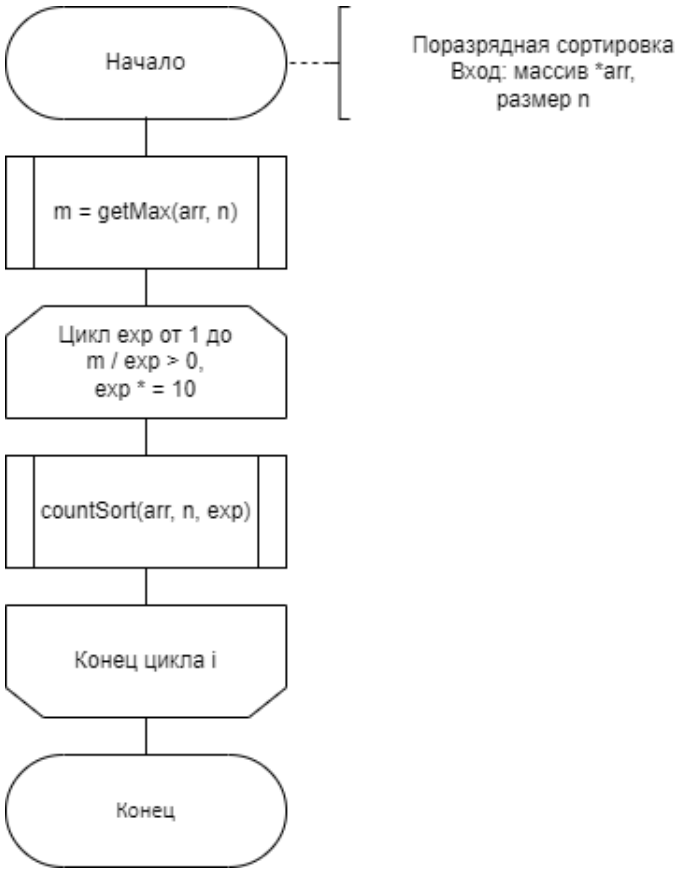

Рисунок 2.5 – Алгоритм поразрядной сортировки (продолжение)

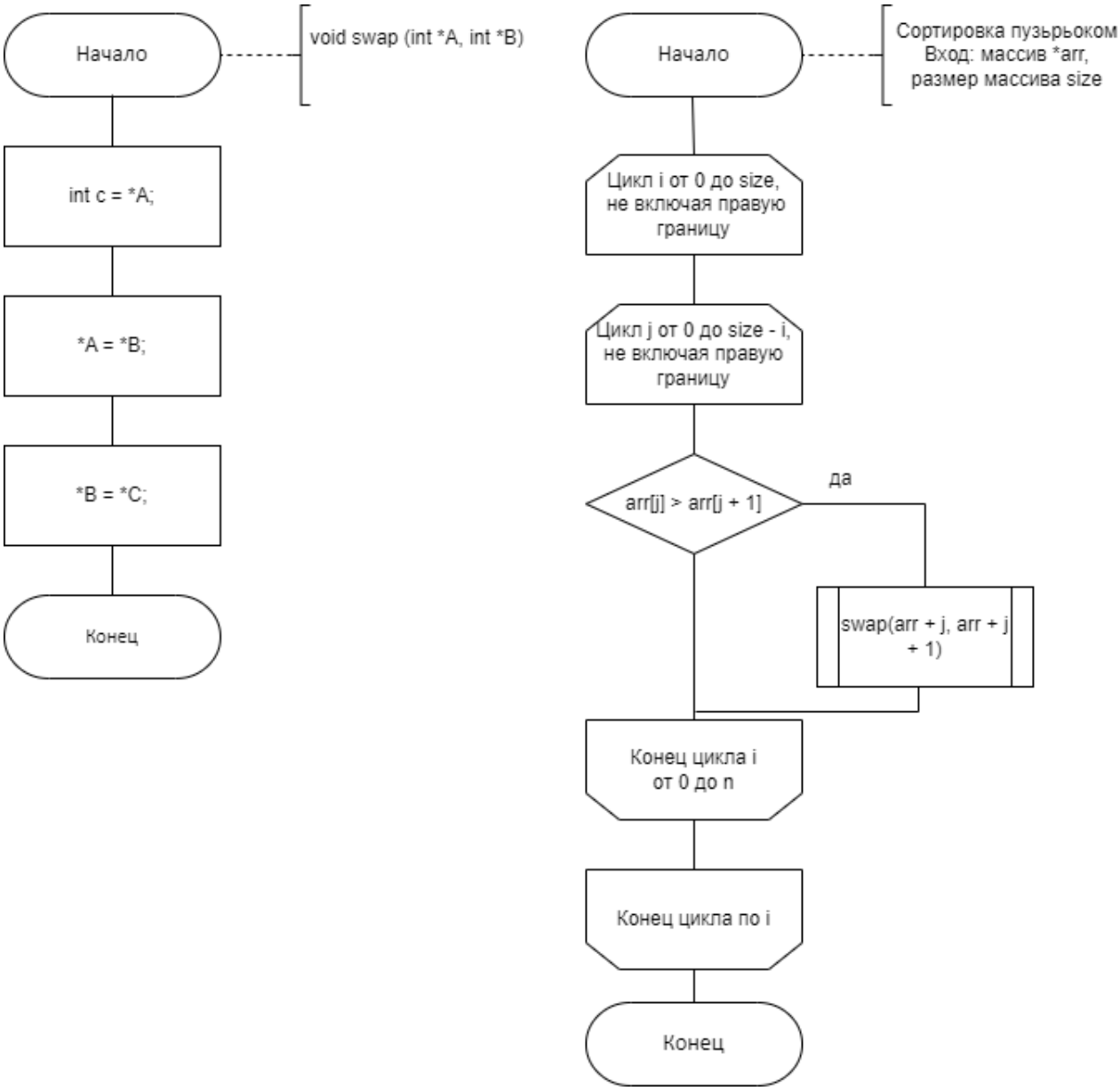

Рисунок 2.6 – Алгоритма сортировки пузырьком

2.2 Модель вычислений для оценки трудоемкости алгоритмов

Для определения трудоемкости алгоритмов необходимо ввести модель вычислений [5]:

1) операции из списка (2.1) имеют трудоемкость равную 2;

$$*, /, \%, * =, / =, \% = \tag{2.1}$$

2) операции из списка (2.2) имеют трудоемкость равную 1;

$$+, -, + =, - =, =, ==, ! =, <, >, <=, >=, [], ++, -- \tag{2.2}$$

3) трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.3);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.3}$$

4) трудоемкость цикла рассчитывается, как (2.4);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.4}$$

5) трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов

Проведем сравнительный анализ реализованных алгоритмов по трудоемкости для сортировки массивы arr[N].

2.3.1 Алгоритм битонной сортировки

Трудоемкость данного алгоритма будет складываться из:

* трудоемкости процесса создания битонной последовательности равной $\log_2 N \cdot 3$;
* трудоекмости процесса сравнения элементы 2 последовательности равной $\log_2 N \cdot (3 + 2)$;

Таким образом, трудоемкость стандартного алгоритма умножения равна:

$$f = \log_2 N \cdot 3 * (\log_2 N \cdot (3 + 2)) = O(log^2 N) \tag{2.5}$$

2.3.2 Алгоритм поразрядной сортировки

Трудоемкость данного алгоритма будет складываться из:

* трудоемкости определения максимальное значение массива равной $2 + 3 \cdot N$;
* трудоемкости определения сортированный массив по каждому разряду равной $2 + N \cdot 8 + 2 + 10 \cdot 4 + 3 + N \cdot 17 = 47 + 25N$;
* трудоемкости главного цикла равной $2 + log_{10} M \cdot (47 + 25N)$;

Трудоемкость алгоритма поразрядной сортировки, где M — максимальное значение массива, равна:

$$f = 2+3 \cdot N+2+log_{10} M \cdot (47+25N) \approx 25N \cdot \log_{10} M = O(N \log M) \tag{2.6}$$

2.3.3 Алгоритма сортировки пузырьком

Трудоемкость в лучшем случае при отсортированном массиве, когда ничего не обменивается, но все же данные рассматриваются. Трудоемкость алгоритма в лучшем случае равна:

$$f_{best} = 2 + N \cdot (2 + 4 \cdot N) \approx O(N^2) \tag{2.7}$$

Трудоемкость в худшем случае при отсортированном массиве, когда массиве в обратном порядке. Трудоемкость алгоритма в худшем случае равна:

$$f_{worst} = 2 + N \cdot (2 + 14 \cdot N) \approx O(N^2) \tag{2.8}$$

2.4 Вывод

Были представлены схемы алгоритмов умножения матриц. Проведённая теоретическая оценка трудоемкости алгоритмов показала, что в трудоёмкость выполнения алгоритма поразрядной сортировки зависит от количество элементов и максимального значения массива. Тажке трудоёмкость выполнения алгоритма сортировки пузырьком $O(N^2)$ больше чем трудоёмкость выполнения алгоритма битонной сортировки $O(log^2 N)$.

3 Технологическая часть

В данном разделе будут указаны требования к программному обеспечению, средства реализации, будут представлены реализация алгоритмов, а также функциональные тесты.

3.1 Требования к программному обеспечению

К программе предъявляются следующие требования.

* Программа должна предоставлять 2 режима работы: режим сортировки массива между введёнными пользователем и режим замера процессорного времени выполнения сортировки массива.

* В начале работы программы пользователю нужно ввести целое число — это выбор пункта меню.

К первому режиму работы программы предъявляются следующие требования:

* если пункт меню — число от 1 до 3, то сортировать массив, для этого надо ввести размер массива и его элементы;

* программа должна вывести результирующий массив.

Ко второму режиму работы программы предъявляются следующие требования:

* если пункт меню — число от 4 до 5, то провести замеры времени и памяти сортировки каждого алгоритма;

* массив генерируются автоматически.

3.2 Средства реализации

Реализация данной лабораторной работы выполнялась при помощи языка программирования C. Данный выбор обусловлен наличием у языка функции *clock*() измерения процессорного времени.

Визуализация графиков с помощью библиотеки *Matplolib*.

3.3 Сведения о модулях программы

Программа состоит из следующих модулей:

* main.c — главный файл программы, предоставляющий пользователю меню для выполнения основных функций;
* sort.c — файл, содержащий реализации этих функций;
* sort.h — заголовочный файл, содержащий объявлении функций, реализуюших рассматриваемых алгоритмови;
* time.c — файл, содержащий функции замеров времени работы указанных алгоритмов;
* time.h — заголовочный файл, содержащий объявлении функций замеров времени работы указанных алгоритмов;
* graph_result.py — файл, содержащий функции визуализации временных характеристик описанных алгоритмов;
* memory.c — файл, содержащий функции замеров памяти для реализации алгоритмов;
* memory.h — заголовочный файл, содержащий объявлении функций замеров памяти для реализации.

3.4 Реализация алгоритмов

Алгоритм битонной сортировки, алгоритм поразрядной сортировки и алгоритм сортировки пузырьком приведены в листингах 3.1 – 3.3.

```
void compAndSwap(int *arr, int i, int j, int dir)
{
  if (dir==(arr[i]>arr[j]))
    swap(arr + i, arr + j);
}

void bitonicMerge(int *arr, int low, int cnt, int dir)
{
  if (cnt>1)
  {
    int k = cnt/2;
    for (int i=low; i<low+k; i++)
      compAndSwap(arr, i, i+k, dir);
    bitonicMerge(arr, low, k, dir);
    bitonicMerge(arr, low+k, k, dir);
  }
}

void bitonicSort(int *arr,int low, int cnt, int dir)
{
  if (cnt>1)
  {
    int k = cnt/2;

    bitonicSort(arr, low, k, 1);

    bitonicSort(arr, low+k, k, 0);

    bitonicMerge(arr, low, cnt, dir);
  }
}

void bitonic_sort(int *arr, int size)
{
  bitonicSort(arr, 0, size, 1);
}
```

Листинг 3.1 – Алгоритм битонной сортировки

```
int getMax(int *arr, int n)
{
  int mx = *arr;
  for (int i = 1; i < n; i++)
    if (arr[i] > mx)
      mx = arr[i];
  return mx;
}

void countSort(int *arr, int n, int exp)
{
  int *output = allocate_arr(n);
  int i, count[10] = { 0 };

  for (i = 0; i < n; i++)
    count[(arr[i] / exp) % 10]++;

  for (i = 1; i < 10; i++)
    count[i] += count[i - 1];

  for (i = n - 1; i >= 0; i--) {
    output[count[(arr[i] / exp) % 10] - 1] = arr[i];
    count[(arr[i] / exp) % 10]--;
  }

  for (i = 0; i < n; i++)
    arr[i] = output[i];
  deallocate_arr(output);
}

void radix_sort(int *arr, int n)
{
  int m = getMax(arr, n);

  for (int exp = 1; m / exp > 0; exp *= 10)
    countSort(arr, n, exp);
}
```

Листинг 3.2 – Алгоритм поразрядной сортировки

```
1     void bubble_sort(int *arr, int size)
2     {
3       for (int i = 0; i < size; i++)
4         for (int j = 0; j < size - i - 1; j++)
5           if (arr[j] > arr[j + 1])
6             swap(arr + j, arr + j + 1);
7     }
8
```

Листинг 3.3 – Алгоритма сортировки пузырьком

3.5 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для функций, реализующих алгоритмы сортировки. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

	Входные данные	Ожидаемый результат		
№	Массив	Битонная	Поразрядрая	Пузырьком
1	[ ]	[ ]	[ ]	[ ]
2	[1]	[1]	[1]	[1]
3	[3, 3, 3, 3]	[3, 3, 3, 3]	[3, 3, 3, 3]	[3, 3, 3, 3]
4	[1, 2, 3, 4]	[1, 2, 3, 4]	[1, 2, 3, 4]	[1, 2, 3, 4]
5	[9, 5, 3, 1]	[1, 3, 5, 9]	[1, 3, 5, 9]	[1, 3, 5, 9]
6	[9, 5, 6, 1]	[1, 5, 6, 9]	[1, 5, 6, 9]	[1, 5, 6, 9]

3.6 Вывод

Были реализованы функции всех алгоритмов сортировок — битонной, поразрядной и пузырьком. Было проведено функциональное тестирование указанных функций.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, и проведены замеры процессорного времени и предоставлена информация о технических характеристиках устройства.

4.1 Технические характеристики

Ниже представлены характеристики компьютера, на котором проводились замеры времени работы реализации алгоритмов:

* операционная система Windows 10 Домашняя 21H2;
* оперативная память 12 Гб;
* процессор Intel(R) Core(TM) i7-9750H CPU @ 2.6ГГц.

Загруженность компонентов:

* процессор — 10%;
* оперативная память — 53%

4.2 Демонстрация работы программы

На рисунке 4.1 – 4.3 приведены примеры работы программы.

```
МЕНЮ:
        1 - Битонная сортировка;
        2 - Поразрядная сортировка;
        3 - Сортировка пузырьком;
        4 - Замерить время;
        5 - Замерить память;
        0 - Выход.
      Выбор:
Ввести выбор из меню: 2
Ввести размер массивы: 5
Ввести элементы массивы: 10 3 5 1 6
Результат: 1 3 5 6 10
```

Рисунок 4.1 – Пример работы алгоритма битонной сортировки

```
МЕНЮ:
         1 - Битонная сортировка;
         2 - Поразрядная сортировка;
         3 - Сортировка пузырьком;
         4 - Замерить время;
         5 - Замерить память;
         0 - Выход.
       Выбор:
Ввести выбор из меню:
2
Ввести размер массивы:
5
Ввести элементы массивы: 10 3 5 8 1
1 3 5 8 10 Ввести выбор из меню:
```

Рисунок 4.2 – Пример работы алгоритма поразрядной сортировки

```
МЕНЮ:
         1 - Битонная сортировка;
         2 - Поразрядная сортировка;
         3 - Сортировка пузырьком;
         4 - Замерить время;
         5 - Замерить память;
         0 - Выход.
       Выбор:
Ввести выбор из меню:
3
Ввести размер массивы:
5
Ввести элементы массивы: 4 5 6 3 2
2 3 4 5 6 Ввести выбор из меню:
```

Рисунок 4.3 – Пример работы алгоритма сортировки пузырьком

4.3 Временные характеристики

Функция clock() из библиотеки time.h языка программирования C возвращает процессорное время в секундах - значение типа clock_t.

Для замера времени:

* получить значение времени до начала выполнения алгоритма, затем после её окончания. Чтобы получить результат, необходимо вычесть из второго значения первое;

* первый шаг необходимо повторить iters раз (в программе iters равно 10), суммируя полученные значения, а затем усреднить результат.

Результаты эксперимента замеров по времени приведены в таблицах 4.1 – 4.3.

В таблице 4.1 приведены результаты замеров по времени алгоритмов сортировок неотсортированных массивов, размером которых от 2 до 512.

На рисунке 4.4 приведены графические результаты сравнения временных характеристик для неотсортированных массивов.

Таблица 4.1 – Результаты замеров времени (неотсортированный массив).

Размер	**Выходные данные по алгоритму, с**		
	Битонная	Поразрядная	Пузырьком
2	0.001	0.002	0.001
4	0.001	0.003	0
8	0.02	0.001	0.001
16	0.011	0.001	0.001
32	0.018	0.016	0.003
64	0.029	0.004	0.025
128	0.042	0.007	0.076
256	0.102	0.014	0.187
512	0.162	0.028	0.606

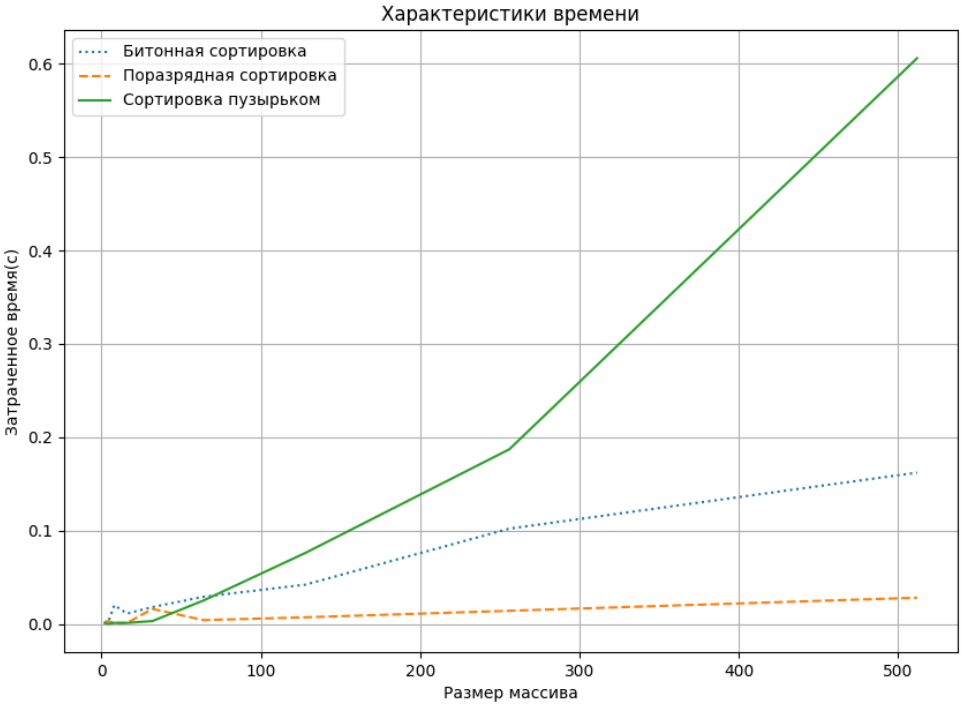

Рисунок 4.4 – Сравнение по времени алгоритмов сортировок неотсортированных массивов

В таблице 4.2 приведены результаты замеров по времени алгоритмов сортировок отсортированных массивов по возрастанию, размером которых от 2 до 512.

Таблица 4.2 – Результаты замеров времени (неотсортированный массив по возрастанию).

Размер	**Выходные данные по алгоритму, с**		
	Битонная	Поразрядная	Пузырьком
2	0.002	0.002	0.002
4	0.001	0.002	0.001
8	0.001	0.002	0.001
16	0.002	0.002	0.001
32	0.027	0.002	0.003
64	0.009	0.004	0.01
128	0.044	0.008	0.073
256	0.098	0.032	0.232
512	0.215	0.036	0.533

На рисунке 4.5 приведены графические результаты сравнения временных характеристик для отсортированных массивов по возрастанию.

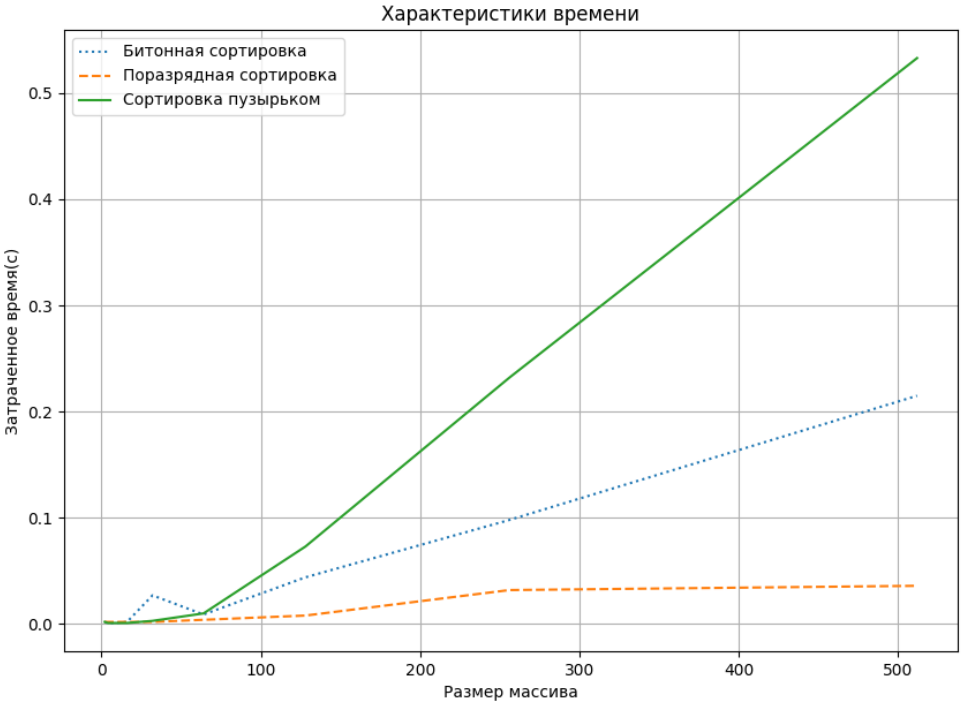

Рисунок 4.5 – Сравнение по времени алгоритмов сортировок отсортированных массивов по возрастанию

В таблице 4.3 приведены результаты замеров по времени алгоритмов сортировок отсортированных массивов по убыванию, размером которых от 2 до 512.

Таблица 4.3 – Результаты замеров времени (неотсортированный массив по убыванию).

Размер	**Выходные данные по алгоритму, с**		
	Битонная	Поразрядная	Пузырьком
2	0.001	0.002	0.002
4	0.001	0.001	0.001
8	0.001	0.002	0.001
16	0.002	0.002	0.002
32	0.004	0.002	0.004
64	0.03	0.003	0.011
128	0.036	0.018	0.103
256	0.118	0.026	0.209
512	0.217	0.02	0.647

На рисунке 4.6 приведены графические результаты сравнения временных характеристик для отсортированных массивов по возрастанию.

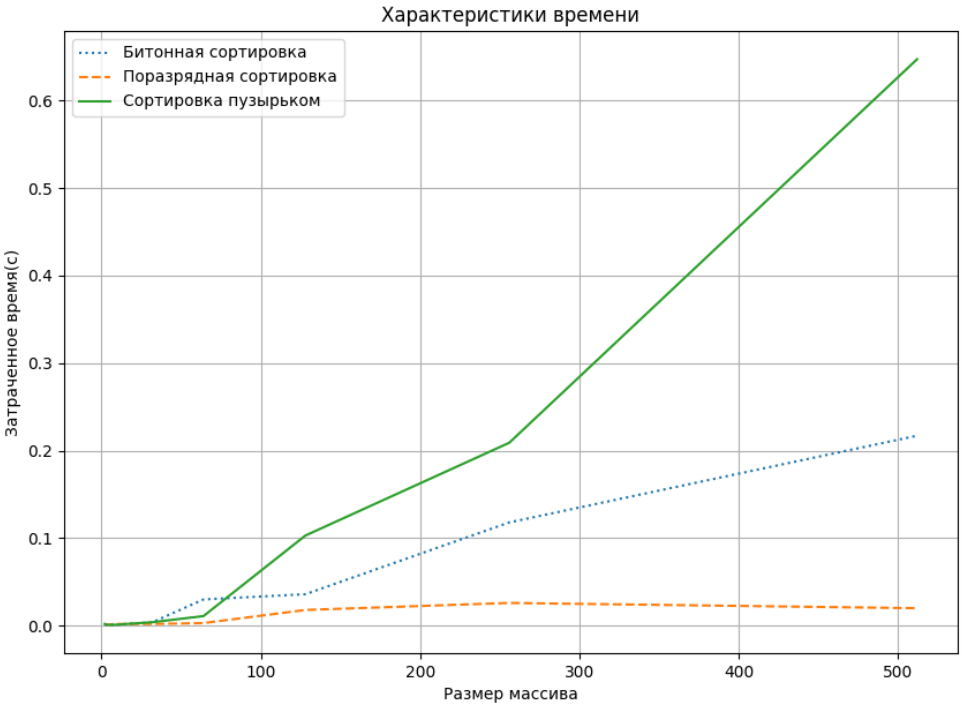

Рисунок 4.6 – Сравнение по времени алгоритмов сортировок отсортированных массивов по убыванию

4.4 Характеристики по памяти

Пусть N — количество элементов массива, M — максимальное значение массива, тогда затраты памяти на рассматриваемые алгоритмы будут следующими.

Затраты по памяти для реализации алгоритма битонной сортировки:

* массив: $N \cdot \text{sizeof(int)}$;
* размер массива B: sizeof(int);
* дополнительные переменные (k, cnt, i, low): $4 \cdot \text{sizeof(int)}$;

Итого:

$$N \cdot \text{sizeof(int)} + \text{sizeof(int)} + 4 \cdot \text{sizeof(int)} = (N + 5) \cdot \text{sizeof(int)} \quad (4.1)$$

Затраты по памяти для реализации алгоритма поразрядная сортировка:

* массив: $N \cdot \text{sizeof(int)}$;
* размер массива B: sizeof(int);
* дополнительные переменные: $8 \cdot \text{sizeof(int)}$;
* дополнительные массивы count, output: $(N + 10) \cdot \text{sizeof(int)}$;
* адрес возврата.

Итого:

$$N \cdot \text{sizeof(int)} + \text{sizeof(int)} + 8 \cdot \text{sizeof(int)} + (N + 10) \cdot \text{sizeof(int)} = (2 \cdot N + 19) \cdot \text{sizeof(int)} \quad (4.2)$$

Затраты по памяти для реализации алгоритма сортировки пузырьком:

* массив: $N \cdot \text{sizeof(int)}$;
* размер массива B: sizeof(int);

* дополнительные переменные (i, j, c): $3 \cdot \text{sizeof(int)}$;

Итого:

$$N \cdot \text{sizeof(int)} + \text{sizeof(int)} + 3 \cdot \text{sizeof(int)} = (N + 4) \cdot \text{sizeof(int)} \quad (4.3)$$

На рисунке 4.7 также приведены результаты замеров памяти.

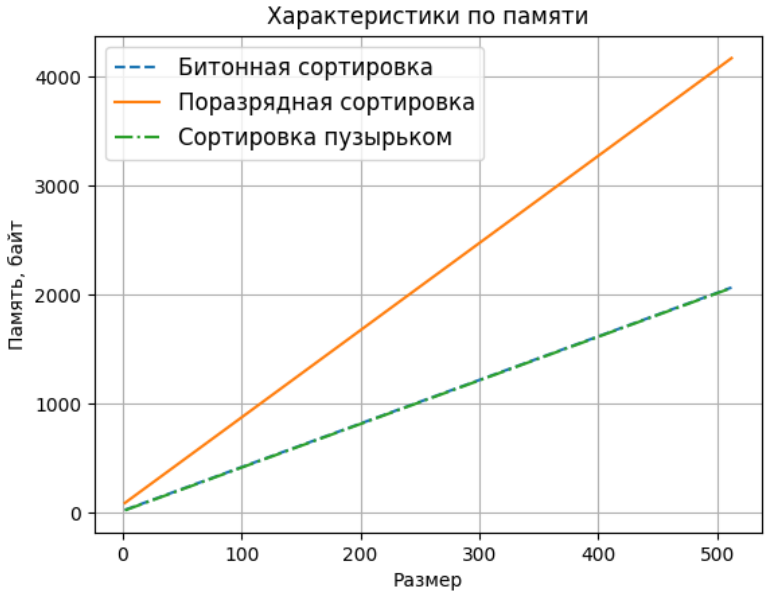

Рисунок 4.7 – Сравнение по памяти алгоритмов сортировок

4.5 Вывод

В результате эксперимента было получено, что при размере массива больше 128, алгоритм сортировки пузырьком работает медленее алгоритма битонной сортировки в 1.8 раза и алгоритма поразрядной сортировки в 10.8 раза. В случае, если массив уже отсортирован, время выполнения сортировки пузырьком в 1.2 раза убывается, так как не нужно выполнять обмен элементы.

Проведя анализ оценки затрат реализаций алгоритмов по памяти, можно сказать, что поразрядная сортировка больше затратны, так как для них использовать дополнительные массивы. Затраты по памяти для

алгоритма битонной сортировки и сортировки пузырьком не отличаются друг от друга.

Заключение

В результате исследования было получено, что при размере массива больше 128, алгоритм сортировки пузырьком работает медленее алгоритма битонной сортировки в 1.8 раза и алгоритма поразрядной сортировки в 10.8 раза.

Проведя анализ оценки затрат реализаций алгоритмов по памяти, можно сказать, что поразрядная сортировка больше затратны, так как для них использовать дополнительные массивы. Затраты по памяти для алгоритма битонной сортировки и сортировки пузырьком не отличаются друг от друга.

Решены все поставленные задачи:

1) Описаны трех алгоритмов сортировки:
   * Битонная сортировка;
   * Поразрядная сортировка;
   * Сортировка пузырьком.
2) Построены схемы рассматриваемых алгоритмов;
3) Создано программного обеспечения, реализующего перечисленные алгоритмы;
4) Проведен сравнительного анализа реализаций алгоритмов по затраченному процессорному времени и памяти.
5) Подготовлен отчете о выполненной лабораторной работе.

Список использованных источников

1. Липачев. Е.К. Технология программирования. Методы сортировки данных: учебное пособие. — Казань: Казанский университет, 2017. — С. 59.

2. Битонная сортировка [Электронный ресурс]. Режим доступа: https://agro-archive.ru/stati/16144-bitonnaya-sortirovka.html (дата обращения: 17.11.2023).

3. Radix Sort — самая быстрая сортировка для чисел и строк [Электронный ресурс]. Режим доступа: https://thecode.media/radix/ (дата обращения: 17.11.2023).

4. Radix Sort — самая быстрая сортировка для чисел и строк [Электронный ресурс]. Режим доступа: https://education.yandex.ru/journal/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii (дата обращения: 17.11.2023).

5. М. В. Ульянов Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. -М.: Изд. «Физматлит», 2007, 304 стр.