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 курсу «Анализ алгоритмов»

Тема Трудоемкость сортировок ‘

Студент Пискунов П.

Группа ИУ7-56Б

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

Москва — 2023 г.

Содержание

Введение

В данной лабораторной работе будут расмотрены сортировки. Сортировка – преобразование последовательности элементов в неубывающую (или невозрастающую) последовательность[1]. В частности, одной из целей сортировки является облегчение последующего поиска элементов в отсортированном множестве. Любой алгоритм сортировки можно разбить на три основные части:

- сравнение элементов для определения их упорядоченности;
- перестановка элементов;
- сортирующий алгоритм, который осуществляет сравнение и перестановку элементов до тех пор, пока все элементы не будут упорядочены.

Самой важной характеристикой алгоритма сортировки является устойчивость. Она заключается в сохранении относительного порядка объектов с одинаковыми значениями ключа.

Целью данной лабораторной работы является исследование трудоемкости алгоритмов сортировки. Для успешного выполнения лабораторной работы необходимо выполнить следующие задачи:

1) изучить и реализовать 3 алгоритма сортировки: слиянием, бынарным деревом, расчесткой;
2) создать программный продукт, позволяющий протестировать реализованные алгоритмы;
3) выбрать инструменты для замера процессорного времени выполнения реализаций алгоритмов;
4) провести анализ затрат работы алгоритмов по времени и по памяти;
5) дать оценку трудоемкости алгоритмов;
6) подготовить отчет по лабораторной работе.

1 Аналитическая часть

1.1 Сортировка слиянием

Алгоритм сортировки слиянием в большой степени соответствует парадигме метода разбиения[2]. Сортировка слиянием применяется для структур данных, доступ к элементам которых можно получать только последовательно, например, списки или массивы.

Алгоритм сортировки для массива:

1) сортируемый массив разбивается на две части примерно одинакового размера;

2) каждая из получившихся частей сортируется отдельно тем же самым алгоритмом;

3) два упорядоченных массива половинного размера соединяются в один.

Процесс разбиения задачи на более мелкие выполняется рекурсивно до тех пор, пока размер массива не уменьшится до одного элемента.

1.2 Сортировка бинарным деревом

Из элементов массива формируется бинарное дерево поиска. Первый элемент – корень дерева, остальные добавляются по следующему методу: начиная с корня дерева, элемент сравнивается с узлами. Если элемент меньше чем узел, тогда спускаемся по левой ветке, иначе по правой. Спустившись до конца, сам элемент становится узлом. Построенное таким образом дерево можно обойти так, чтобы двигаться от узлов с меньшими значениями к узлам с большими значениями. При этом получаем все элементы в возрастающем порядке[3].

1.3 Сортировка расческой

Основная идея «расчёстки» в том, чтобы первоначально брать достаточно большое расстояние между сравниваемыми элементами и по мере упорядочивания массива сужать это расстояние вплоть до минимального. Таким образом, мы как бы причёсываем массив, постепенно разглаживая на всё более аккуратные пряди.

Для сравнения двух элементов рекомендуется учитывать специальную величину, известную как фактор уменьшения, оптимальное значение которой приблизительно равно 1,247. Сначала расстояние между элементами максимально, то есть равно размеру массива минус один. Затем, пройдя массив с этим шагом, необходимо поделить шаг на фактор уменьшения и пройти по списку вновь. Так продолжается до тех пор, пока разность индексов не достигнет единицы. В этом случае сравниваются соседние элементы как и в сортировке пузырьком, но такая итерация одна.

2 Конструкторская часть

В этом разделе будут представлены требования к программному обеспечению (ПО), схема и трудоемкость алгоритмов.

2.1 Требования к программному обеспечению

Программе передается массив в качестве входных данных, а на выход получается отсортированный массив. Кроме того, необходимо сообщить пользователю затраченное каждым алгоритмом процессорное время и используемая память.

В создаваемом приложении пользователю должны быть доступны функции ввода элементов массива и выбора желаемого алгоритма.

2.2 Разработка алгоритма соритовки слиянием

На рисунке 2.1 приведена схема рассматриваемого алгоритма.

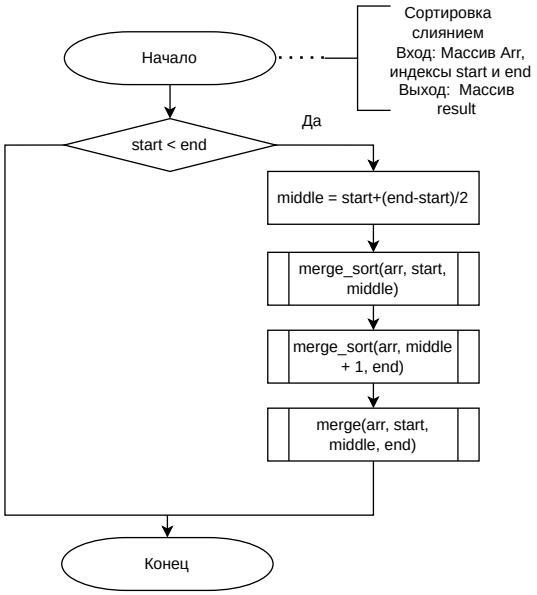

Рисунок 2.1 – Схема алгоритма сортировки слиянием

2.3 Разработка алгоритма сортировки бинарным деревом

На рисунке 2.2 приведена схема рассматриваемого алгоритма.

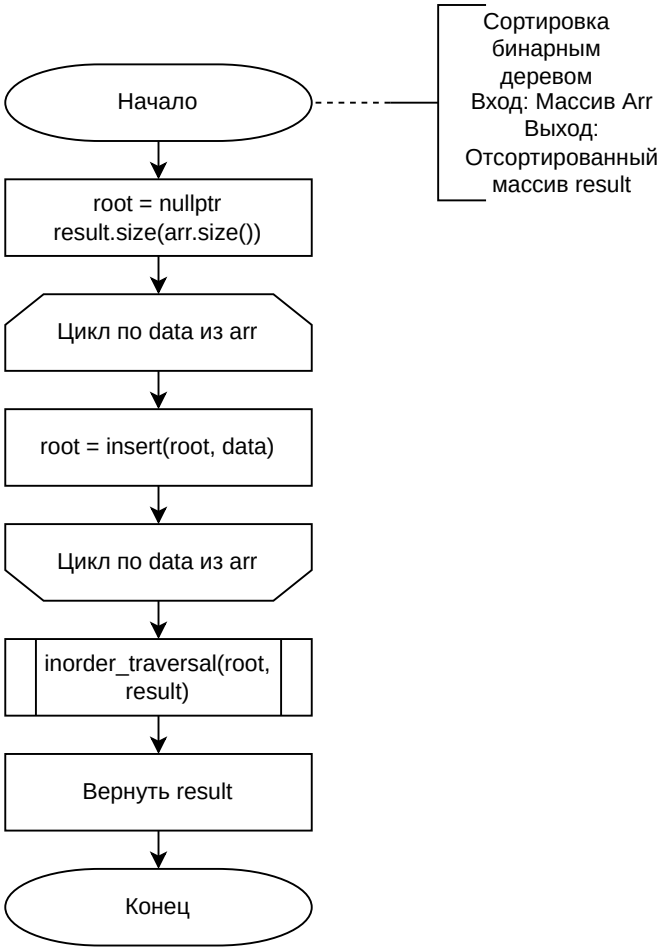

Рисунок 2.2 – Схема алгоритма сортировки бинарным деревом

2.4 Разработка алгоритма сортировки расческой

На рисунке 2.3 приведена схема рассматриваемого алгоритма.

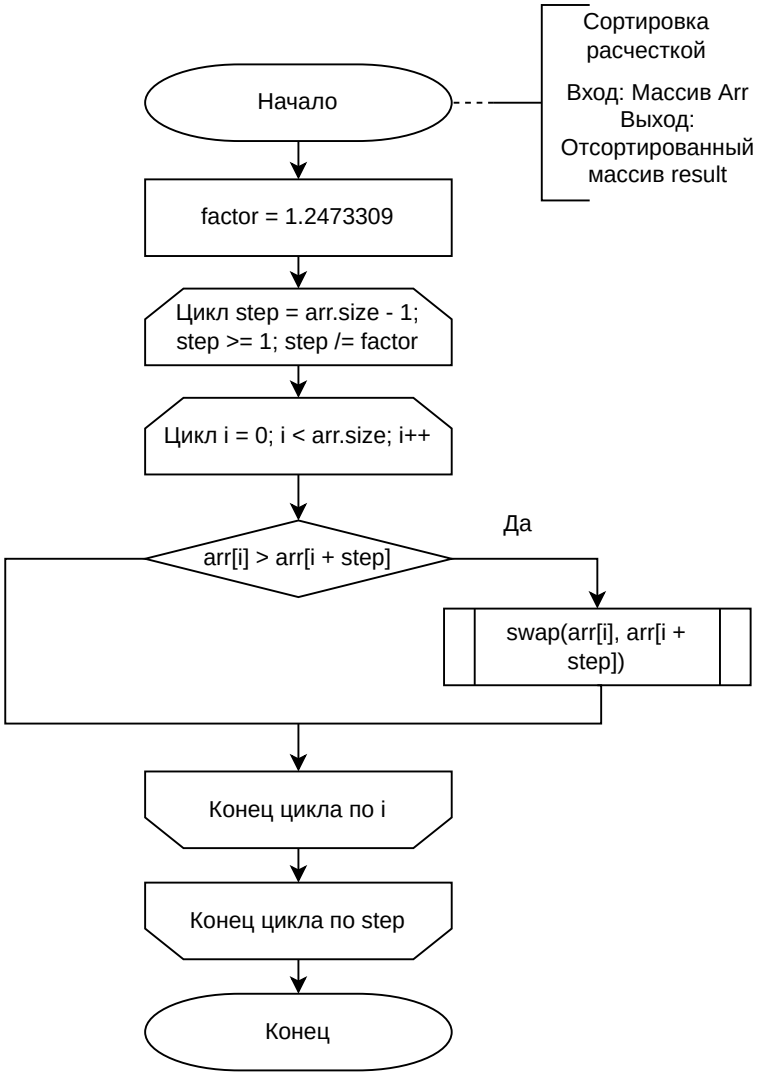

Рисунок 2.3 – Схема алгоритма сортировки расческой

2.5 Модель вычислений

Для дальнейшего анализа трудоемкости необходимо ввести модель вычислений.

Операции из списка (2.1) имеют трудоемкость 1.

$$=, +=, -=, +, -, ==, !=, <, >, <=, >=, [], ++, -- \tag{2.1}$$

Операции из списка (2.2) имеют трудоемкость 2.

$$*, /, \% \tag{2.2}$$

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

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

Трудоемкость цикла рассчитывается как:

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

Трудоемкость вызова функции равна 0.

2.6 Трудоемкость алгоритмов

Далее размер массива обозначаетсй как size.

2.6.1 Алгоритм сортировки слиянием

Трудоемкость алгоритма сортировки слиянием состоит из:

- рекурсивное разбиение массива на массивы половинной длины:

$$f_{\text{разбиение}} = log_2(size) \tag{2.5}$$

- трудоемкость соединения двух массивов половинной длины:

$$f_{\text{соединение}} = 2 + size \cdot (3 + \begin{cases} 4, & \text{в лучшем случае,} \\ 10, & \text{в худшем случае.} \end{cases} \tag{2.6}$$

2.6.2 Алгоритм сортировки бинарным деревом

Трудоемкость алгоритма сортировки бинарным деревом состоит из:

- Трудоемкости построения бинарного дерева, которая равна (2.7).

$$f_{make\_tree} = (5 \cdot \log(N) + 3) * N = N \cdot \log(N) \tag{2.7}$$

- Трудоемкости восстановления порядка элементов массива, которая равна (2.8):

$$f_{main\_loop} = 7N \tag{2.8}$$

Таким образом общая трудоемкость алгоритма выражается как (2.9).

$$f_{total} = f_{make\_tree} + f_{main\_loop} \tag{2.9}$$

Трудоемкость алгоритма в лучшем случае (2.10).

$$f_{best} = O(N \log(N)) \tag{2.10}$$

Трудоемкость алгоритма в худшем случае (2.11).

$$f_{worst} = O(N \log(N)) \tag{2.11}$$

Трудоёмкость в лучшем случае (массив отсортирован):

$$f_{\text{лучший}} = log_2(size) \cdot (2 + size \cdot 7) \approx size \cdot log_2(size) = O(size \cdot log_2(size)) \tag{2.12}$$

Трудоёмкость в худшем случае (массив отсортирован в обратном порядке):

$$f_{\text{худший}} = log_2(size) \cdot (2 + size \cdot 13) \approx size \cdot log_2(size) = O(size \cdot log_2(size)) \tag{2.13}$$

Трудоемкость в худшем и лучшем случае равна $O(size \cdot log_2(size))$.

2.6.3 Алгоритм сортировки расческой

Трудоемкость алгоритма сортировки расчёсткой может быть оценена следующим образом:

Трудоемкость алгоритма в лучшем случае (2.14).

$$f_{best} = O(size \log(size)) \tag{2.14}$$

Трудоемкость алгоритма в худшем случае (2.15).

$$f_{worst} = O(size^2) \tag{2.15}$$

Вывод

На основе теоретических знаний, полученных в аналитическом разделе, были разработаны схемы алгоритмов, благодаря которым может быть упорядочено множество объектов разными способами. Также для каждого из них были рассчитаны и оценены лучшие и худшие случаи.

3 Технологическая часть

В этом разделе предоставляются листинги выполненных алгоритмов и осуществляется выбор средств реализации.

3.1 Выбор средств реализации

Для выполнения данной лабораторной работы был выбран язык программирования C++. Время измерялось с помощью функции *clock()* из библиотеки *time.h* [4].

3.2 Реализация алгоритмов

В листингах 3.1 – 3.3 представлены реализации рассматриваемых алгоритмов.

В Листинге 3.1 показана реализация алгоритма сортировки слиянием.

```
1  void merge_sort(vector<int> &array, size_t start, size_t end)
2  {
3    if (start < end)
4    {
5      size_t middle = start + (end - start) / 2;
6      merge_sort(array, start, middle);
7      merge_sort(array, middle + 1, end);
8      merge(array, start, middle, end);
9    }
10 }
```

Листинг 3.1 – Алгоритм сортировки слиянием

В Листинге 3.2 показана реализация алгоритма сортировки бинарным деревом.

```
vector<int> bin_tree_sort(vector<int> &array)
{
  node_t *root = nullptr;

  for (int data : array)
  root = insert(root, data);

  vector<int> result;
  inorder_traversal(root, result);
  return result;
}
```

Листинг 3.2 – Алгоритм сортировки бинарным деревом

В Листинге 3.3 показана реализация алгоритма сортировки расческой.

```
void coctail_sort(vector<int> &array)
{
  double factor = 1.2473309;

  for (int step = array.size() - 1; step >= 1; step /= factor)
      for (int index = 0; index + step < (int)array.size(); index++)
          if (array[index] > array[index + step])
              swap(array[index], array[index + step]);
}
```

Листинг 3.3 – Алгоритм сортировки расческой

Вывод

Были выбраны инструменты для реализации и разработаны алгоритмы упорядочивания объектов массива: сортировки слиянием, бинарным деревом и расческой. Также предоставлены листинги кода на выбранном языке программирования.

4 Исследовательская часть

4.1 Интерфейс приложения

На рисунках 4.1 – 4.2 приведено изображение интерфейса главного экрана приложения.

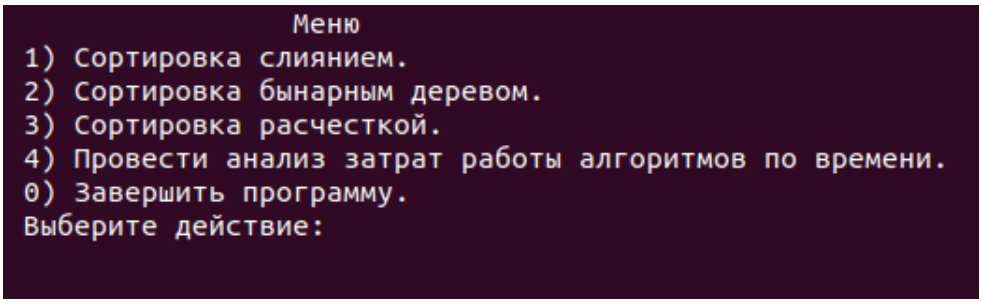

Рисунок 4.1 – Интерфейс

Пользователь может ввести элементов массива вручную или автоматически, для которого необходимо упорядочить элементы по возростанию, а также выбрать метод, который будет использован для этого действия. В результате выводится отсортированный массив.

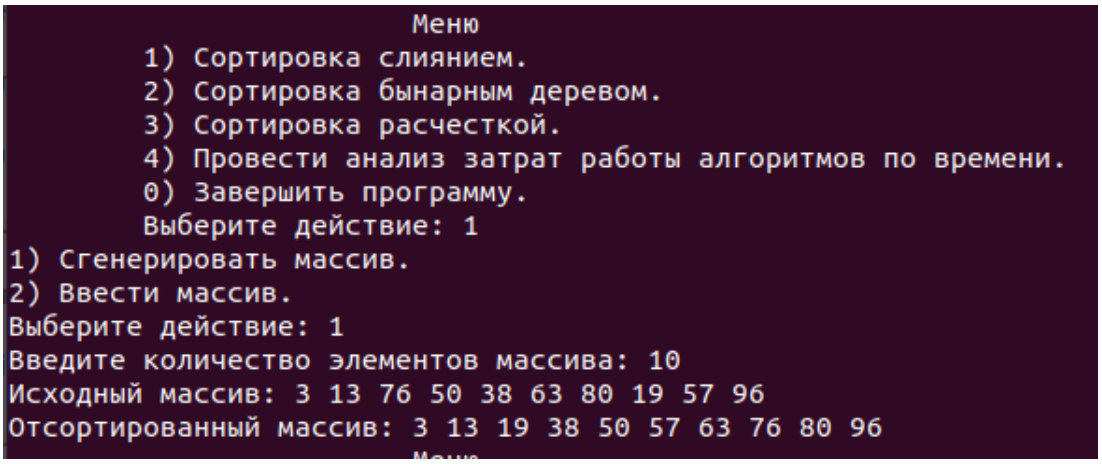

Рисунок 4.2 – Экран с результатом

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система: Ubuntu 22.04.3 LTS;
- оперативная память: 12 Гб;
- процессор: AMD® Athlon silver 3050u with radeon graphics × 2;

4.3 Время выполнения реализаций алгоритмов

На рисунке 4.3 приведено сравнение реализации всех алгоритмов сортировки в среднем случае, которые содержат 10, 20, 30, 40, 50, 100, 200, 300, 500, 1000 элементов.

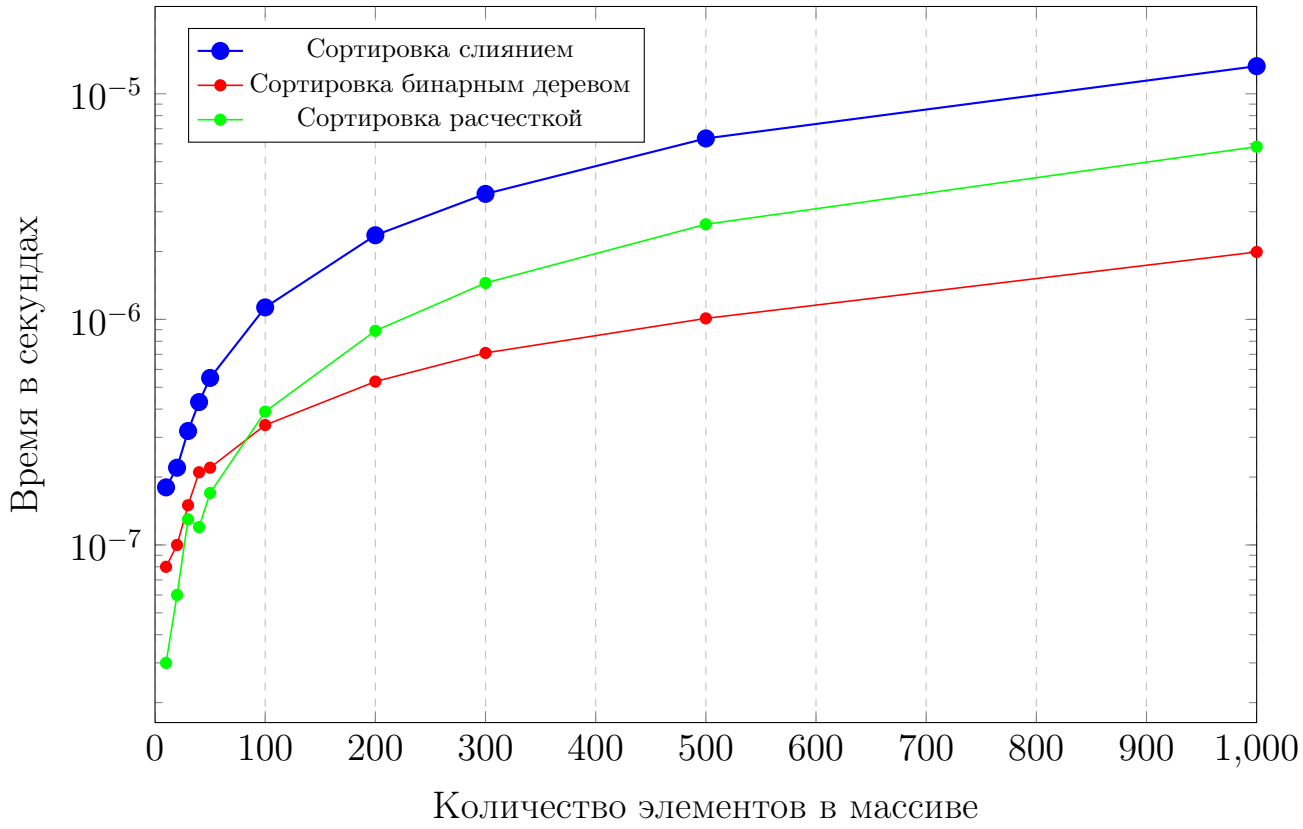

Рисунок 4.3 – Сравнение времени работы реализаций алгоритмов

На рисунке 4.4 приведено сравнение реализации всех алгоритмов сортировки в лудшем случае, которые содержат 10, 20, 30, 40, 50, 100, 200, 300, 500, 1000 элементов.

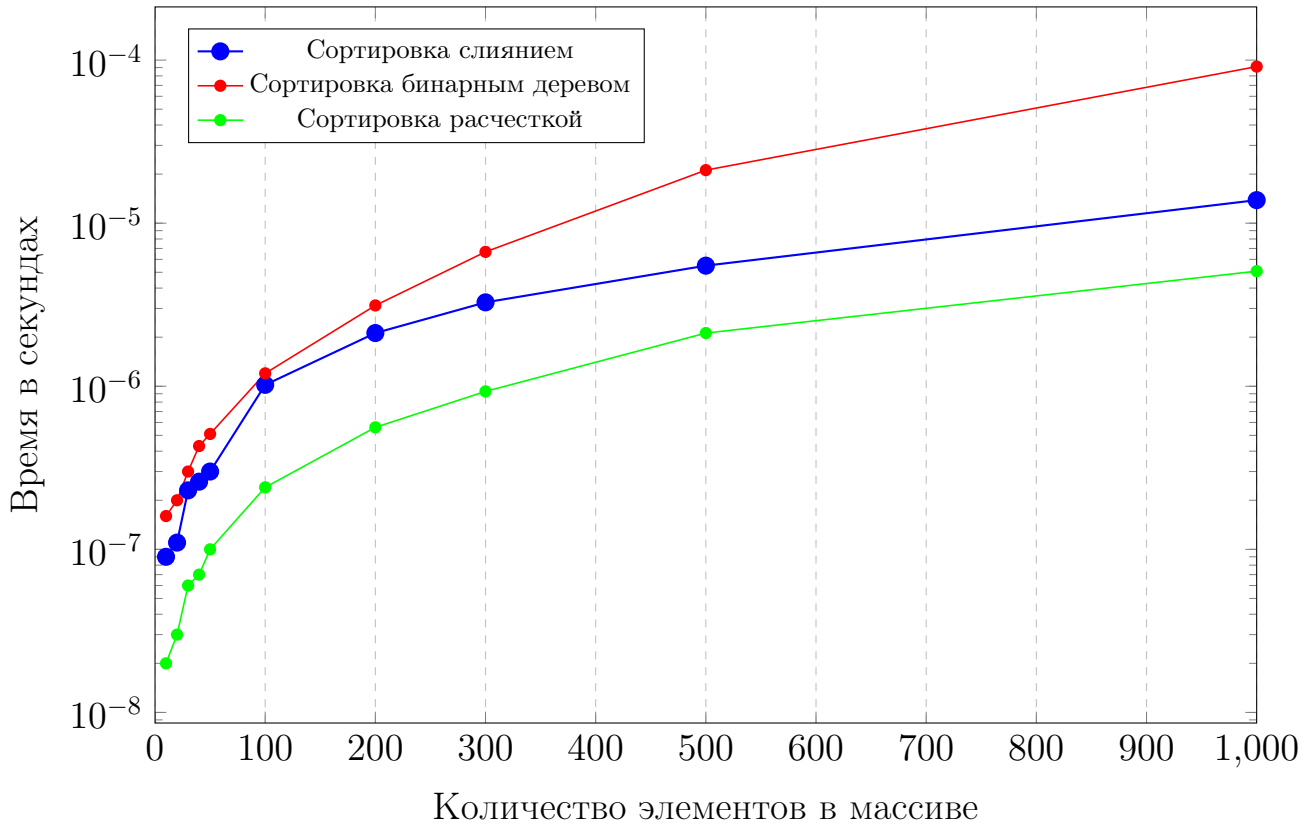

Рисунок 4.4 – Сравнение времени работы реализаций алгоритмов

На рисунке 4.5 приведено сравнение реализации всех алгоритмов сортировки в худшем случае, которые содержат 10, 20, 30, 40, 50, 100, 200, 300, 500, 1000 элементов.

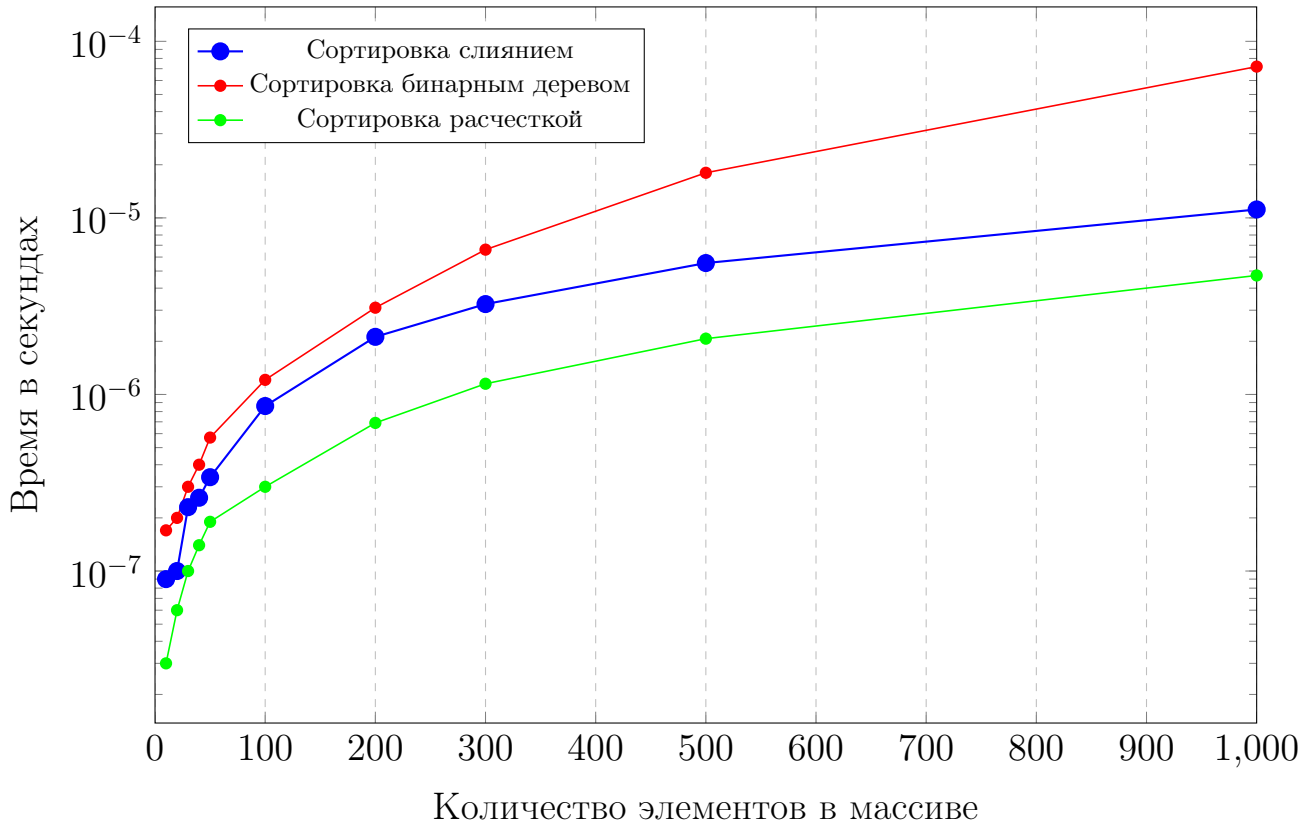

Рисунок 4.5 – Сравнение времени работы реализаций алгоритмов

4.4 Используемая память

На рисунке 4.6 приведено сравнение реализации всех алгоритмов сортировки по памяти, которые содержат 10, 20, 30, 40, 50, 100, 200, 300, 500, 1000 элементов.

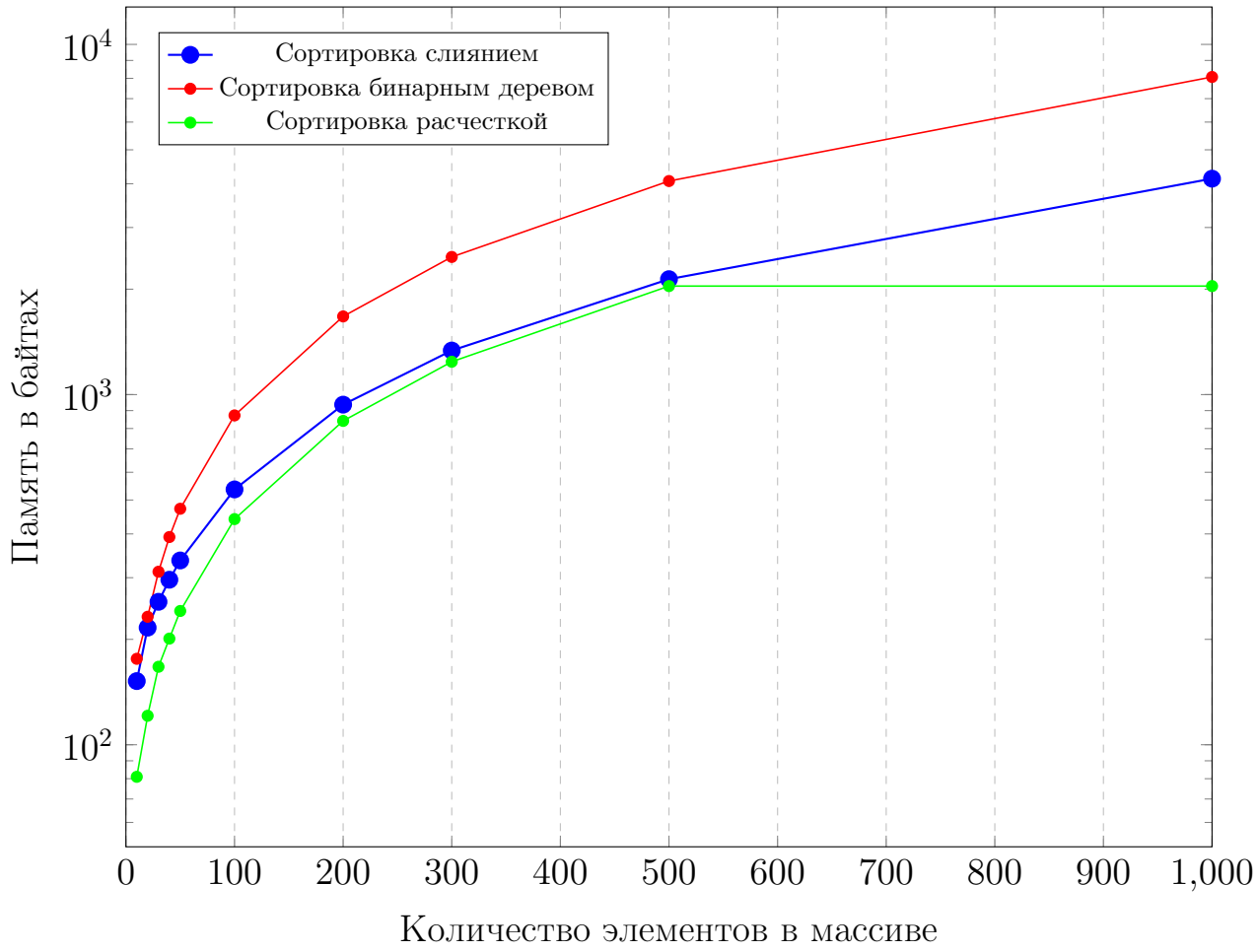

Рисунок 4.6 – Сравнение размеров реализаций алгоритмов в байтах

Вывод

В данном разделе проведено сравнение времени выполнения и использования памяти алгоритмов сортировки для массивов различного размера. Помимо оценки скорости выполнения, изучили, как каждый из алгоритмов влияет на использование памяти. В массивах до 50 элементов из графика 4.3 видно, что алгоритм сортировки бинарным деревом в среднем случае медленее работает чем алгоритм соритовки расчесткой. Такой результат получисля потому, что бинарное дерево не находится ближе к сбалансированнному состоянию, что ухудшает его производительность.

Наименее затратным по памяти оказался алгоритм сортировки расческой из-за того, что он использует только структуру массива, пока остальные алгоритмы используют дополнительные структуры.

Заключение

В результате исследования трудоемкости алгоритмов было установлено, что наименее затратным по времени в среднем случае оказался алгоритм сортировки бинарным деревом. Этот результат объясняется тем, что в среднем случае, когда массив заполняется произвольными значениями, бинарое дерево ближе к сбалансированному состоянию, что улучшает его производительность по сравнению с другими алгоритмами. С другой стороны, в лучшем и худшем случае данный алгоритм проявляет неэффективность, поскольку находится в несбалансированном состоянии. В этих случаях предпочтение отдается алгоритму сортировки расческой, который в подобных сценариях продемонстрировал более высокую производительность.

Наименее затратным по памяти оказался алгоритм сортировки расческой из-за того, что он использует только структуру массива, пока остальные алгоритмы используют дополнительные структуры. Таким образом, выбор алгоритма сортировки зависит от размера массива и характера данных, которые требуется упорядочить, а также от трудоемкости алгоритма.

В результате выполнения лабораторной работы были выполнены следующие задачи:

1) разработаны и реализованы алгоритмы сортировки массивов;
2) создан программный продукт, позволяющий протестировать реализованные алгоритмы;
3) проведен сравнительный анализ процессорного времени выполнения реализаций данных алгоритмов;
4) проведен сравнительный анализ затрачеваемой алгоритмами памяти;
5) дана оценка трудоемкости алгоритмов;
6) был подготовлен отчет по лабораторной работе.

Список использованных источников

[1] Сортировки [Электронный ресурс]. — Режим доступа: https://acmp.ru/article.asp?id_text=519 (дата обращения: 6.11.2023).

[2] Кормен Т. Лейзерсон Ч. Ривест Р. Штайн К. Алгоритмы: построение и анализ. — М.: Издательский дом «Вильямс», 2011. с. 1296.

[3] А. Ахо Дж. Хопкрофт Дж. Ульман. Построение и анализ вычислительных алгоритмов. Мир, 1979. с. 533.

[4] C library function clock() [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/c/chrono/clock_t (дата обращения: 26.10.2023).