Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

Выполнил студент группы .....КС-36 Алёшин Михаил Алексеевич
Ссылка на репозиторий: ..... https://github.com/MUCTR-IKT-CPP/MAAleshin_36_algo

Приняли: ..... Пысин Максим Дмитриевич
.....Краснов Дмитрий Олегович

Дата сдачи: 14.02.2022

Оглавление

Описание задачи.

Необходимо реализовать сортировку слиянием для разного количества элементов. Подсчитать количество дополнительной потребляемой памяти, подсчитать количество вызовов рекурсивной функции, и высоту рекурсивного стека. Построить график потребления памяти и сравнить его с функцией c * n, построить график худшего, лучшего, и среднего случая для каждой серии тестов для количества вызовов рекурсивных функций. Подобрать константы c1 и c2 для ограничения графика сверху и снизу. Проанализировать полученные графики.

Описание метода/модели.

Сортировка слиянием - один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем $O(n \log n)$ обменов при упорядочении n элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

Алгоритм:

1. Сортируемый массив разбивается на две части примерно одинакового размера
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

У нас есть два массива a и b (фактически это будут две части одного массива, но для удобства будем писать, что у нас просто два массива). Нам надо получить массив с размером |a|+|b|. Для этого можно применить процедуру слияния. Эта процедура заключается в том, что мы сравниваем элементы массивов (начиная с начала) и меньший из них записываем в финальный. И затем, в массиве у которого оказался меньший элемент, переходим к следующему элементу и сравниваем теперь его. В конце, если один из массивов закончился, мы просто дописываем в финальный другой массив. После мы наш финальный массив записываем вместо двух исходных и получаем отсортированный участок.

Достоинства метода:

- Устойчива
- Многопоточна
- Распределена

Выполнение задачи.

Данная сортировка была реализована на языке python. Реализованы функции mergeSortRecursive и merge соответственно. При вызове функции генерируется новый массив от -1 до 1 (числа с плавающей точкой) и вызывается метод анализа сортировки, в котором 20 раз прогоняется каждое

количество элементов начиная с 1000 и умножая на 2, заканчивая 128000 элементами. В конце получаем данный результат:

```
При 1000 элементах: min: 0.00794672966003418 sec | max: 0.010968446731567383 sec | avg:0.008277416229248047 sec | mem: 148360.0 bytes | count_functions: 1999.0
При 2000 элементах: min: 0.016956567764282227 sec | max: 0.01894831657409668 sec | avg:0.017951977252960206 sec | mem: 313272.0 bytes | count_functions: 3999.0
При 4000 элементах: min: 0.03789687156677246 sec | max: 0.0528640747070312 5 sec | avg:0.04135282039642334 sec | mem: 660464.0 bytes | count_functions: 7999.0
При 8000 элементах: min: 0.08184933662414551 sec | max: 0.08605670928955078 sec | avg:0.08347105979919434 sec | mem: 1390080.0 bytes | count_functions: 15999.0
При 16000 элементах: min: 0.17852306365966797 sec | max: 0.22241806983947754 sec | avg:0.20034050941467285 sec | mem: 2920736.0 bytes | count_functions: 31999.0
При 32000 элементах: min: 0.3789784908294678 sec | max: 0.4807701110839844 sec | avg:0.42041443586349486 sec | mem: 6126848.0 bytes | count_functions: 63999.0
При 64000 элементах: min: 0.8018472194671631 sec | max: 1.0865089893341064 sec | avg:0.911845326423645 sec | mem: 12768256.0 bytes | count_functions: 127999.0
При 128000 элементах: min: 1.7553086280822754 sec | max: 2.3910789489746094 sec | avg:1.960184347629547 sec | mem: 26580064.0 bytes | count_functions: 255999.0
```

Рис. 1. Результаты анализа.

Далее посмотрим результаты потребления памяти на графике:

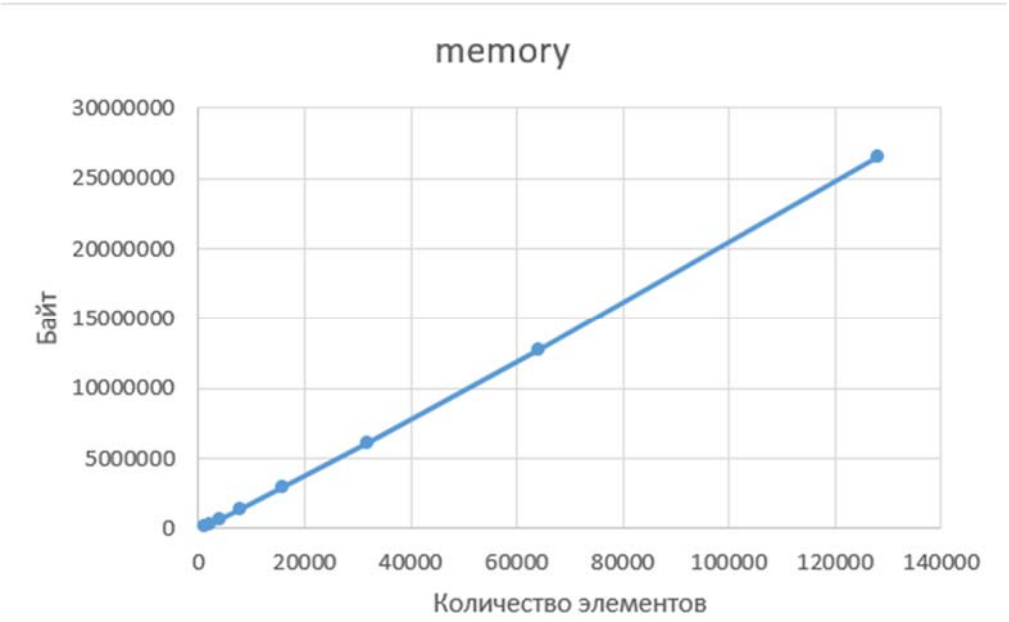

Рис. 2. График зависимости памяти от количества элементов массива.

Если сравнить данный график с графиком c*n, где константой c является отношение y/x последней точки графика. Получим следующий результат:

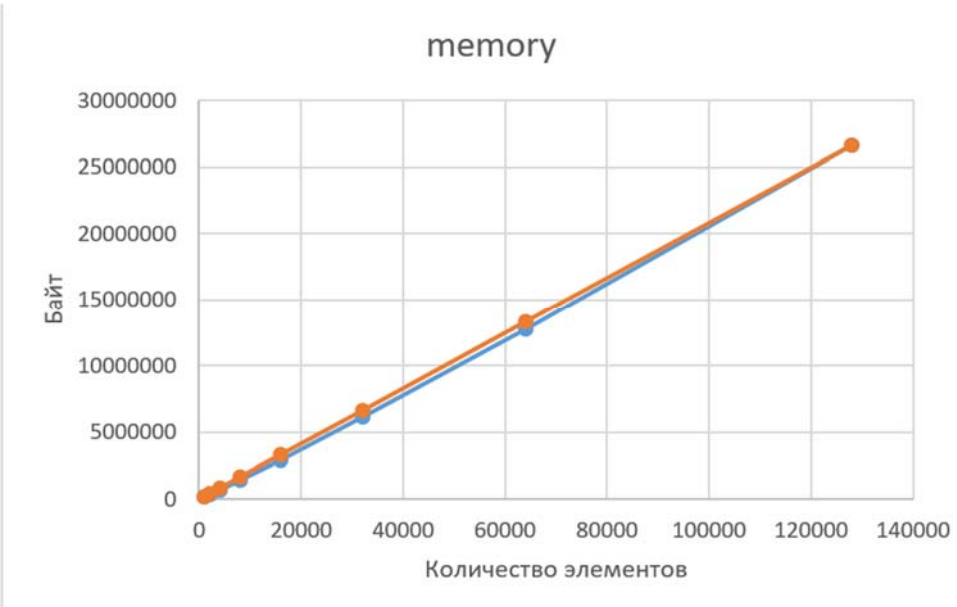

Рис. 3. Функция c*n.

Из графика видно, что наш график имеет более параболический вид, чем график c*n, следовательно, при увеличении количества элементов потребуется выделение большего количества памяти.

Количество вызовов функции будет всегда константным и равняться N*2-1. Далее на графике можно увидеть график, который будет очень схож с графиком 2*n.

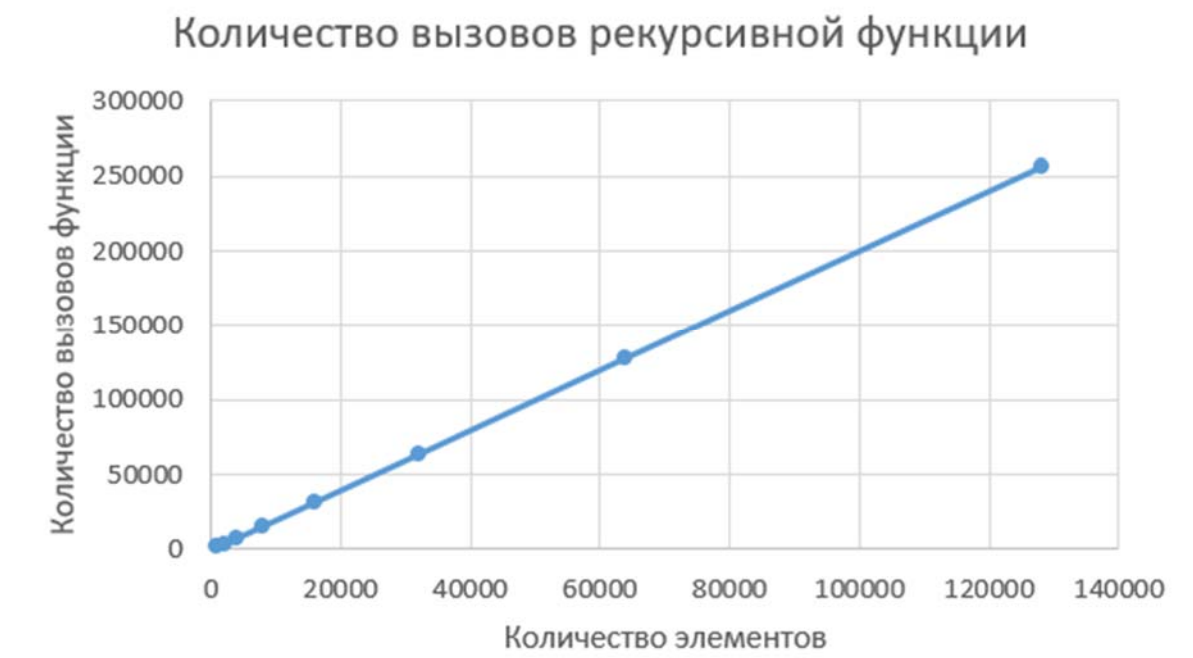

Рис. 4. Количество вызовов рекурсивной функции.

Далее посмотрим на результат времени работы сортировки относительно количества элементов:

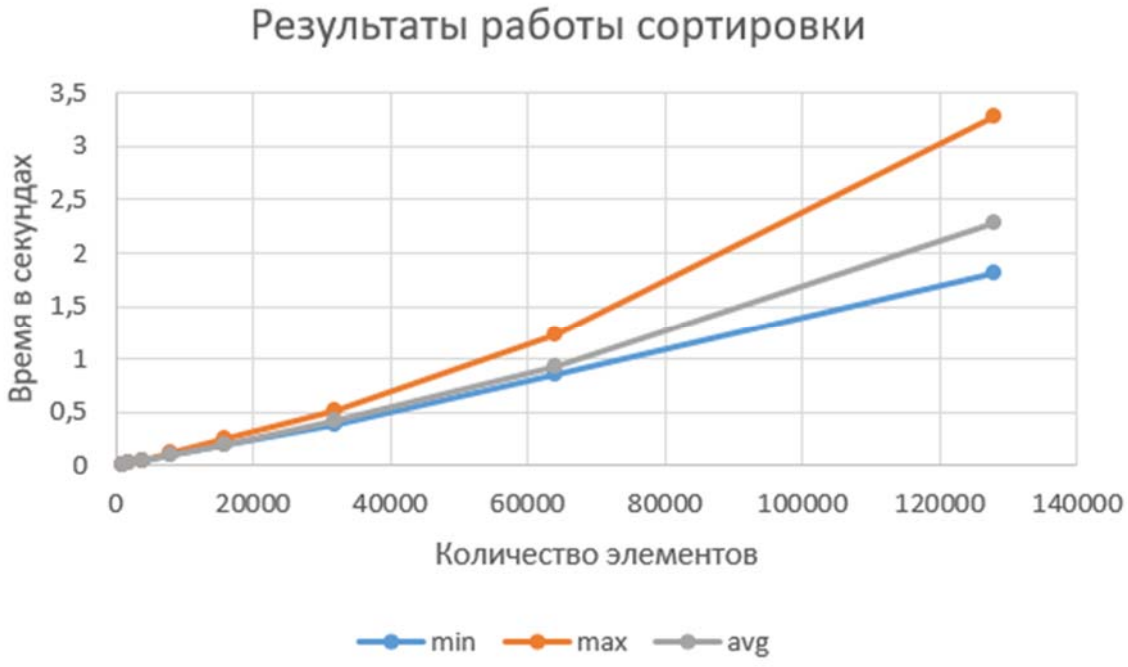

Рис. 5. Время работы сортировки.

Далее подберем константы c1 и c2 для графика c1*N*log(N) и c2*N*log(N) соответственно, чтобы первый график ограничивал наш график сверху, а график с константой c2 ограничивал наш график снизу. Подобрав значения, получим c1 = 5,04801E-06 и c2 = 2,33333E-06. Далее посмотрим результаты на графике:

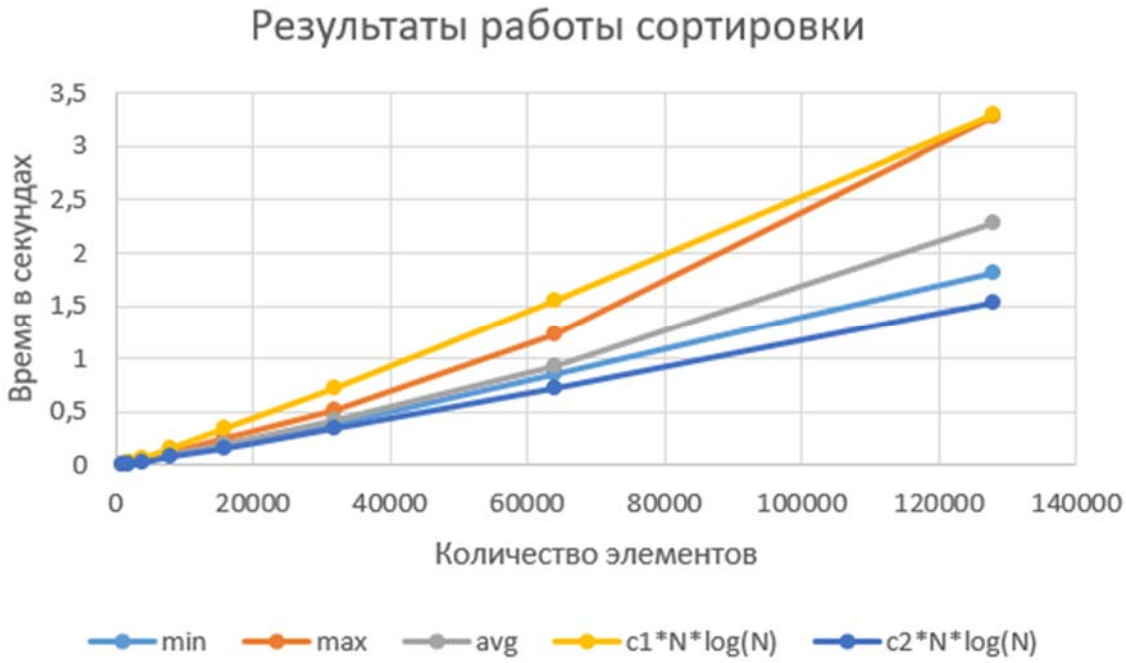

Рис. 6. Константы c1 и c2.

Заключение.

Из полученных нами результатов видно, что данный метод сортировки является довольно эффективным, так как имеет скорость O(n*log(n)). На рис.1 наглядно видно, что при увеличении элементов в 2 раза скорость работы сортировки увеличивается чуть более, чем в 2 раза, что доказывает сложность O(n*log(n)). Недостатком данного метода является то, что он может занимать большое количество памяти для хранения промежуточных результатов. Однако метод является стабильным, несмотря на данный недостаток, связанный с памятью. Данный метод сортировки является рекурсивным, однако количество рекурсий является константным (в моём случае) и равняется 2*n-1. Деградация метода отсутствует.