

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

|  |
| --- |
|  |

Институт информационных технологий

Кафедра математического обеспечения и стандартизации

информационных технологий (МОСИТ)

**ОТЧЕТ**

**ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3**

«Асимптотический анализ эффективности на примерах алгоритмов

сортировки»

**по дисциплине**

«Структуры и алгоритмы обработки данных»

Выполнил студент группы ИКБО-01-21 Хитров Н. С.

Принял преподаватель кафедры МОСИТ Рысин М. Л.

Практическая работа выполнена «\_\_»\_\_\_\_\_\_\_2022 г.

«Зачтено» «\_\_»\_\_\_\_\_\_\_2022 г.

Москва 2022

# ЦЕЛЬ

Получить навыки по анализу вычислительной сложности усовершенствованных и быстрых алгоритмов сортировки и определение наиболее эффективного алгоритма.

# ЗАДАНИЕ 1

**Формулировка задания**:

Разработать алгоритмы сортировки, определенные вариантом. Провести анализ вычислительной и емкостной сложности алгоритма на массивах, заполненных случайно. Определить наиболее эффективный алгоритм.

## Описание математической модели алгоритма ускоренной сортировки

Данный массив перестраивается в бинарную кучу при помощи функции heapify. Куча устроена таким образом, что в каждый родительский элемент больше каждого из двух своих дочерних элементов. (Индекс дочерних элементов можно узнать по формулам child1/2 = parent \* 2 + 1/2, а индекс родительского элемента можно узнать по формуле parent = child1/2 div 2). Добавление дочерних элементов в кучу идет с левого края кучи. Когда куча сформирована начинается сортировка. В цикле: наибольший элемент/корень меняется местами с элементом в конце массива, далее, рекурсивно вызывается метод heapify для построения уменьшенной кучи (длиной на 1 меньше чем предыдущая куча), пока массив не будет отсортирован.

### Блок-схема алгоритма

|  |  |
| --- | --- |
|  |  |
| Рисунок 1 – блок-схема алгоритма преобразования массива в бинарную кучу | Рисунок 2 – блок-схема алгоритма сортировки |

### Реализация алгоритма

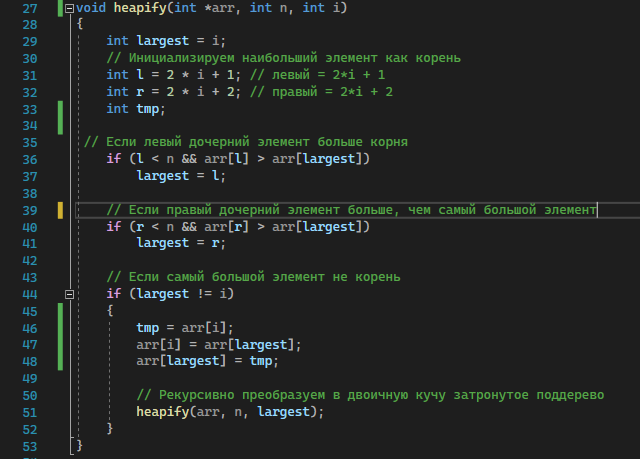


Рисунок 2 Алгоритм превращения массива в двоичную кучу

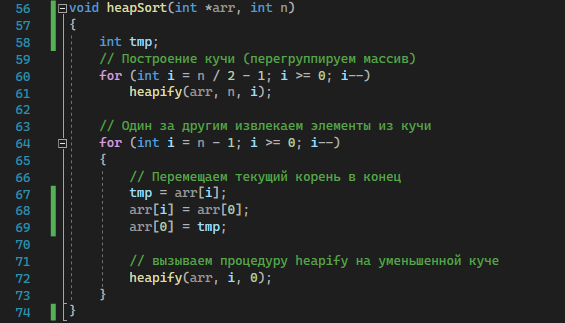


Рисунок 3 Алгоритм сортировки

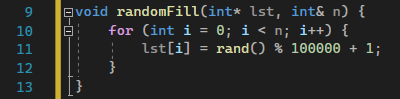


Рисунок 4 – Функция заполнения массива

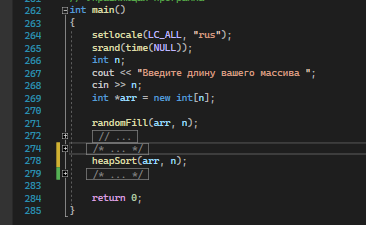


Рисунок 5 – Функция main

### Тестирование

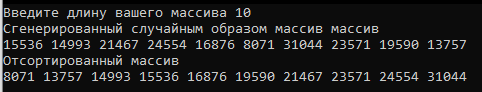


Рисунок 6 - Тестирование и отладка при 10 элементах

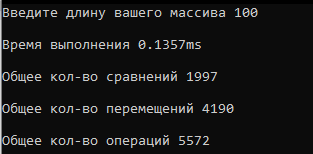


Рисунок 7 – Средний случай. 100 элементов

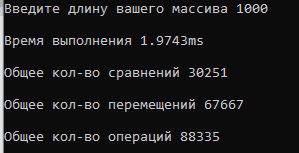


Рисунок 8 – Средний случай. 1000 элементов

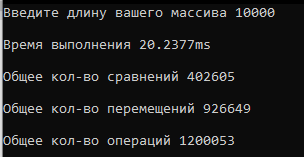


Рисунок 9 – Средний случай. 10 000 элементов

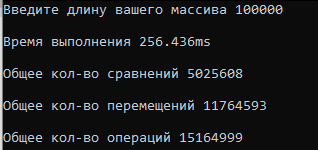


Рисунок 10 – Средний случай. 100 000 элементов

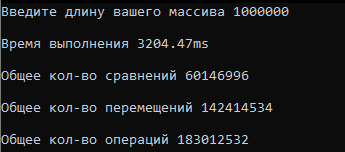


Рисунок 11 – Средний случай. 1 000 000 элементов

Таблица 1.1 – Эмпирическая оценка ускоренной сортировки

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Tn(n) = Сф + Мф** |
| 100 | 0,357 | 5572 |
| 1000 | 1,9743 | 88335 |
| 10000 | 20,2377 | 1 200 053 |
| 100000 | 256,436 | 15 164 999 |
| 1000000 | 3204,47 | 183 012 532 |

### Оценка емкостной сложности

Используется массив размером n для хранения элементов, а также 5 вспомогательная переменных. Емкостная сложность: 5

## Описание математической модели алгоритма быстрой сортировки

Алгоритм рекурсивно разбивает исходный массив на подмассивы половинной длины до тех пор, пока не будет получен единичный массив (считается упорядоченным). Далее происходит слияние двух уже упорядоченных подмассивов путем выбора наименьшего элемента среди первых элементов каждого подмассива (являются наименьшими для каждого из них из-за упорядоченности) и добавления его в отсортированную область текущего массива.

### Блок-схема алгоритма, доказательство корректности цикла

|  |  |
| --- | --- |
|  | Алгоритм разбиения массива на минимальные подмассивы |
| Алгоритм сортировки при помощи слияния |  |

Рисунок 12 – Блок-схема сортировки слиянием

Инвариант цикла

### Реализация алгоритма

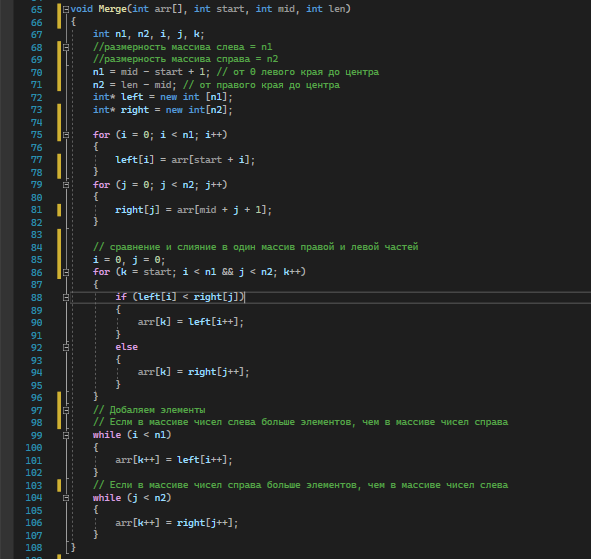


Рисунок 13 – Алгоритм сортировки при помощи слияния

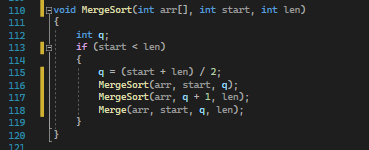


Рисунок 14 – Алгоритм разбиения массива на минимальные подмассивы

### Тестирование

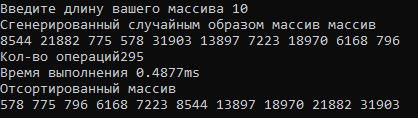


Рисунок 15 - Тестирование и отладка при 10 элементах.



Рисунок 16 – Средний случай. 100 элементов



Рисунок 17 – Средний случай. 1000 элементов



Рисунок 18 – Средний случай. 10 000 элементов



Рисунок 19 – Средний случай. 100 000 элементов



Рисунок 20 – Средний случай. 1 000 000 элементов

Таблица 2.1 – Эмпирическая оценка быстрой сортировки

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Tn(n) = Сф + Мф** |
| 100 | 0,4877 | 295 |
| 1000 | 1 | 4 881 |
| 10000 | 6 | 66 900 |
| 100000 | 46 | 10 371 909 |
| 1000000 | 468 | 121 760 237 |

### Оценка емкостной сложности

Используются подмассивы суммарной длины n, а также 6 дополнительных переменных для вспомогательных целей. Емкостная сложность – линейная:

## График зависимости сложности от n

График 1

График 2

## Вывод по заданию 1

На основе данных, полученных в результате тестирования, можно сделать следующий вывод: в среднем случае алгоритм быстрой сортировки существенно лучше алгоритма ускоренной сортировки по времени и количеству операций. Но проигрывает по используемой памяти.

# ЗАДАНИЕ 2

## Формулировка задания

Оценить эффективность алгоритмы ускоренной и быстрой сортировок в случаях строгой упорядоченности по возрастанию и убыванию (условно худший и лучший случаи).

## Тестирование ускоренной сортировки

Рисунок 21 – По возрастанию. 100. Рисунок 22 – По убыванию. 100.

Рисунок 23 – По возрастанию. 1000. Рисунок 24 – По убыванию. 1000.

Рисунок 25 – По возрастанию. 10000. Рисунок 26 – По убыванию. 10000.

Рисунок 27 – По возрастанию. 100000. Рисунок 28 – По убыванию. 100000.

Рисунок 29 – По возрастанию. 1000000. Рисунок 30 – По убыванию. 1000000.

Таблица 1.2 – Эмпирический анализ ускоренной сортировки на массиве, упорядоченном по возрастанию

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Tn(n) = Сф + Мф** |
| 100 | 0,1732 | 6 325 |
| 1000 | 1,7985 | 94 637 |
| 10000 | 22,4444 | 1 276 242 |
| 100000 | 259,683 | 15 910 019 |
| 1000000 | 3065,55 | 190 392 830 |

Таблица 1.3 – Эмпирический анализ ускоренной сортировки на массиве, упорядоченном по убыванию

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Tn(n) = Сф + Мф** |
| 100 | 0,1173 | 5099 |
| 1000 | 1,5613 | 80878 |
| 10000 | 21,2854 | 1127197 |
| 100000 | 238,491 | 14404292 |
| 1000000 | 2924,34 | 175900898 |

Полученные данные позволяют сделать вывод, что алгоритм не зависит от входного массива.

## Тестирование быстрой сортировки

Рисунок 12 – По возрастанию. 100. Рисунок 13 – По убыванию. 100.

Рисунок 12 – По возрастанию. 1000. Рисунок 13 – По убыванию. 1000.

Рисунок 12 – По возрастанию. 10000. Рисунок 13 – По убыванию. 10000.

Рисунок 12 – По возрастанию. 100000. Рисунок 13 – По убыванию. 100000.

Рисунок 12 – По возрастанию. 1000000. Рисунок 13 – По убыванию. 1000000.

Таблица 1.2 – Эмпирический анализ быстрой сортировки на массиве, упорядоченном по возрастанию

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Tn(n) = Сф + Мф** |
| 100 | 1,0477 | 4 431 |
| 1000 | 1,3805 | 58 935 |
| 10000 | 5,243 | 743 459 |
| 100000 | 40,1149 | 8 929 603 |
| 1000000 | 411,92 | 103 872 115 |

Таблица 1.3 – Эмпирический анализ быстрой сортировки на массиве, упорядоченном по убыванию

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **Tn(n) = Сф + Мф** |
| 100 | 0,8552 | 4707 |
| 1000 | 1,3927 | 63755 |
| 10000 | 5,1665 | 803 667 |
| 100000 | 40,464 | 9705747 |
| 1000000 | 404,072 | 113575667 |

Полученные данные позволяют сделать вывод, что алгоритм не зависит от входного массива.

## Вычислительная сложность алгоритмов

**Пирамидальная сортировка:**

**Сортировка вставками:**

В предыдущем варианте наиболее эффективным алгоритмом оказался алгоритм сортировки вставками. Теоретически были выведены формулы роста для него. В худшем случае сложность определяется формулой , а в лучшем . По определению если существует такое c, что .

если существует такое с, что Рассмотрим , тогда при . Так как , с всегда > 0, значит левая часть неравенства всегда выполнена. Следовательно, при с равном, например, 15 неравенство будет всегда выполняться, значит для худшего случая

2. Рассмотрим , тогда при . Значит, например, для с = 3 искомое неравенство всегда выполняться, а поскольку такое с существует, значит для лучшего случая Т(n)= Ω (n).

**Сортировка слиянием:** Ранее было доказано, что время работы алгоритма не зависит от состояния входных данных, так что лучший и худший случаи совпадают. 𝑇(𝑛) = 𝑂(𝑛𝑙𝑜𝑔(𝑛))

**Пирамидальная сортировка:** Ранее было доказано, что время работы алгоритма не зависит от состояния входных данных, так что лучший и худший случаи совпадают. 𝑇(𝑛) = 𝑂(𝑛𝑙𝑜𝑔(𝑛))

## Сводная таблица результатов

Асимптотическая сложность простого алгоритма сортировки отличается в условно лучшем и худшем случае, поэтому невозможно дать точную оценку асимптотической сложности в среднем случае.

Таблица 3 – Сводная таблица результатов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Алгоритм** | **Асимптотическая сложность алгоритма** | | | |
| Наихудший случай (сверху) | Наилучший случай (снизу) | Средний случай (точная оценка) | Ёмкостная сложность, V(n) |
| Простой | О. |  | - | O(n) |
| Усовершен-ствованный |  | ()) | ʘ | O(n) |
| Быстрый |  | ()) | ʘ | O(n) |

## 

## Вывод по заданию 2

Исходя из полученных на практике данных можно утверждать, что алгоритм быстрой сортировки является наиболее эффективным.

# ВЫВОД

В результате выполнения работы были получены навыки по анализу вычислительной сложности усовершенствованных и быстрых алгоритмов сортировки и определение наиболее эффективного алгоритма. Был разработан каждый из названных алгоритмов. Произведен анализ вычислительной и емкостной сложности алгоритма на массивах, заполненных случайно или упорядоченных по возрастанию или убыванию. Наиболее эффективный алгоритм по времени – быстрая сортировка со сложностью O(nlog(n)), по памяти – все сортировки проявляют линейную сложность.