*Федеральное государственное бюджетное образовательное учреждение* *высшего профессионального образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана (национальный исследовательский институт)»***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА ИУ7

**Отчёт**

**по лабораторной работе №3**

**Дисциплина: Анализ алгоритмов**

**Тема лабораторной работы: Сортировка массива**

Студент гр. ИУ7-51Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Громова В.П.** (Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Волкова Л.Л.**

(Подпись, дата) (И.О. Фамилия)

Москва, 2019г.

[**Введение**](#_k5dnresp2ym9) **3**

[**Аналитическая часть**](#_8y2m4p612xwl) **4**

[Описание алгоритмов](#_hxixtc4k4y3r) 4

[Описание модели вычислений](#_l0mwksllceyh) 5

[**Конструкторская часть**](#_e605ypxc4hq) **7**

[Схемы алгоритмов](#_5u6pxkyjbb59) 7

[Трудоемкость алгоритмов](#_hqa7ecywmdob) 10

[**Технологическая часть**](#_te5b7ppyrcfk) **12**

[Требования к ПО](#_dflahe5brn4y) 12

[Средства реализации](#_tzfvlwapbdm8) 12

[Листинги кода](#_a7ffra2y4hml) 12

[**Экспериментальная часть**](#_hafwjwrs4nle) **14**

[Примеры работы программы](#_8dhvbm9mn6x2) 14

[Постановка эксперимента по замеру времени](#_qskq9s8tr9ha) 15

[Сравнительный анализ на материале экспериментальных данных](#_osqjva9o5i2k) 15

[**Заключение**](#_g6gpwoin67m5) **20**

[**Список литературы**](#_5xxiijderauq) **21**

### Введение

Алгоритм сортировки — это алгоритм для упорядочения элементов в массиве. Сортировка остается по-прежнему самой распространенной практической алгоритмической задачей[1, с. 123]. Сортировка также зачастую занимает больше процессорного времени, чем остальные задачи программы. Поэтому изучение алгоритмов сортировки актуально и важно, поскольку помогает в выборе наиболее подходящего для конкретной задачи алгоритма.

Цель данной лабораторной работы: изучить алгоритмы сортировки массива и исследовать их трудоемкость.

Задачи лабораторной работы следующие.

1. Реализовать 3 алгоритма сортировки.
2. Рассчитать трудоемкость одного из алгоритмов.
3. Найти в литературе трудоемкость двух других алгоритмов.
4. Сравнить реализованные алгоритмы по времени.
5. Описать проделанную работу и обосновать получившиеся результаты.

### Аналитическая часть

#### Описание алгоритмов

* + 1. Сортировка выбором

Основная идея алгоритма: проход по массиву в поисках минимального элемента. Найденный минимум меняется местами с последним элементом. Неотсортированная часть массива уменьшается на один элемент (не включает первый элемент, куда был переставлен найденный минимум). К этой неотсортированной части применяются те же действия — находится минимум и ставится на первое место в неотсортированной части массива. И так далее до тех пор, пока неотсортированная часть массива не уменьшится до одного элемента.

* + 1. Пирамидальная сортировка

Пирамидальная сортировка (или сортировка кучей, HeapSort) — 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. Рассмотрим представление кучи на основе массива, так как оно является эффективным с точки зрения расхода памяти. Если родительский узел хранится в индексе I, левый дочерний элемент может быть вычислен как 2 I + 1, а правый дочерний элемент — как 2 I + 2 (при условии, что индексирование начинается с 0). Пирамидальная сортировка в действительности есть не что иное, как реализация сортировки выбором с применением удачной структуры данных [1, с. 129]. Таким образом, шаги алгоритма такие же, как и в сортировке выбором. Основные отличия заключаются в том, что необходимо построить двоичную кучу и найти в ней минимум.

Основная идея алгоритма пирамидальной сортировки в порядке по возрастанию.

* Построить max-heap из входных данных.
* На данном этапе самый большой элемент хранится в корне кучи. Заменить его на последний элемент кучи, а затем уменьшить ее размер на 1. Наконец, преобразовать полученное дерево в max-heap с новым корнем.
* Повторять вышеуказанные шаги, пока размер кучи больше 1.
  + 1. Сортировка слиянием

Сортировка слиянием предполагает рекурсивную реализацию. При рекурсивном подходе сортируемые элементы разбиваются на две группы, каждая их этих меньших групп сортируется рекурсивно, после чего два отсортированных массива “сливаются” в один, их элементы чередуются, образуя полностью отсортированный общий массив. Базовый случай такой сортировки имеет место, когда исходный массив состоит из одного элемента, вследствие чего перестановки в нем невозможны. Алгоритм сортировки слиянием хорошо подходит для сортировки связных списков, так как он, в отличие от алгоритма пирамидальной сортировки, не основывается на произвольном доступе к элементам. Его основным недостатком является необходимость во вспомогательном буфере при сортировке массивов. Отсортированные связные списки можно легко соединить, не используя дополнительную память.

#### Описание модели вычислений

Для того, чтобы рассчитать трудоемкость алгоритмов, будет использована следующая модель вычислений:

* трудоемкость базовых операций +, -,, =, <, >, <=, >=, ==, !=, [], +=, -=, \*, /, %, //(целочисленное деление) будет оцениваться в единицу;
* трудоемкость условного перехода будет оцениваться в ноль;
* трудоемкость цикла будет оцениваться по следующей формуле:

fцикла = fинициализации + fсравнения + N\*(f тела цикла + f инкремента + f сравнения), где N - количество повторений тела цикла.

### Конструкторская часть

#### Схемы алгоритмов

На рисунках 2.1 - 2.3 представлены схемы алгоритмов сортировки входного массива arr размером n по возрастанию его элементов. Процедура swap(x, y) меняет местами значения, хранящиеся в x и y соответственно.

|  |  |
| --- | --- |
|  |  |

*Рис. 2.1 - алгоритм сортировки выбором*

|  |  |
| --- | --- |
|  |  |

*Рис. 2.2 - алгоритм пирамидальной сортировки*

|  |
| --- |
|  |
| *Рис. 2.3 - алгоритм сортировки слиянием* |

#### Трудоемкость алгоритмов

Проведем оценку трудоемкости трех алгоритмов сортировки.

* + 1. Алгоритм сортировки выбором

Вычисление трудоемкости алгоритма сортировки выбором для массива из n элементов:

Лучший случай (массив отсортирован, не было ни одного обмена):

fa = 2 + n \* (2 + 3 + ((n - 1)/2) \* (3 + 3 ) + 1) = 2 + n \* (6 + 3 \* (n - 1)) = 3 \* n \* n + 3 \* n + 2 ≈ **О(n2)**

Худший случай (массив отсортирован в обратном нужному порядке, на каждой итерации был обмен):

fa = 2 + n \* (2 + 3 + ((n - 1)/2) \* (3 + 3 + 2 ) + 1 + 6) = 2 + n \* (12 + 4 \* (n - 1)) = 4 \* n \* n + 8 \* n + 2 ≈ **О(n2)**

Как видно из получившихся расчетов трудоемкость алгоритма одинакова как для лучшего, так и для худшего случая.

* + 1. Алгоритм пирамидальной сортировки

При создании пирамиды очередной новый (добавляемый) элемент должен быть большим, чем его предшественник в невозрастающей (max-heap) двоичной пирамиде. Эта задача решается за счет обмена местами такого элемента с родительским узлом. После этого потребуется дальнейшая аналогичная корректировка пирамиды на более высоком уровне (реализовано рекурсивно). Таким образом, новый элемент “пузырьковым” методом перемещается на должное место. На каждом уровне процесс обмена занимает постоянное время, так как высота пирамиды из n элементов равна log(n), то каждая вставка занимает O(logn) времени. Тогда для построения всей пирамиды необходимо O(n\*logn) времени. При этом осталось рассмотреть временные затраты на операцию удаления корневого элемента, операция поиска занимает O(1), так как верхний элемент пирамиды находится в первой ячейке массива. Удаление корня занимает O(logn). Обмен местами наибольшего элемента с последним и многократный вызов процедуры восстановления пирамиды дает нам алгоритм пирамидальной сортировки с временем исполнения O(n \* logn) [1, с. 134]. При этом трудоемкость алгоритма остается неизменной и для лучшего, и для худшего случаев.

* + 1. Алгоритм сортировки слиянием

Слияние всех элементов на каждом уровне рекурсии выполняется за линейное время. На каждом уровне количество элементов делится пополам. Количество делений пополам числа n до тех пор, пока не будет получена единица, равно log2n. Так как глубина рекурсии составляет logn уровней, а каждый уровень обрабатывается за линейное время, то в наихудшем случае время исполнения алгоритма O(n \* logn) [1, c. 142]. Трудоемкость данного алгоритма имеет один и тот же порядок при лучшем и при худшем случаях.

Таким образом, трудоемкость алгоритмов сортировки слиянием и пирамидальной одинаковы.

### Технологическая часть

#### Требования к ПО

Программа на вход получает последовательность чисел: элементы массива. Результат работы программы: отсортированный по возрастанию исходный массив.

#### Средства реализации

В данной работе использовался язык программирования Python. Для замера времени использовалась дополнительный модуль time.

#### Листинги кода

На вход каждой функции подавался массив arr.

В листинге 1 представлена реализация алгоритма сортировки выбором.

Листинг 1:

1. def selection\_sort(arr):
2. n = len(arr)
3. for i in range(n):
4. flag = 0
5. min\_index = i
6. for j in range(i + 1, n):
7. if arr[min\_index] > arr[j]:
8. min\_index = j
9. flag = 1
10. if flag:
11. tmp = arr[min\_index]
12. arr[min\_index] = arr[i]
13. arr[i] = tmp

В листинге 2 представлена реализация алгоритма пирамидальной сортировки.

Листинг 2:

1. def heapify(arr, n, index):
2. largest = index
3. left = 2 \* index + 1
4. right = 2 \* index + 2
5. if left < n and arr[index] < arr[left]:
6. largest = left
7. if right < n and arr[largest] < arr[right]:
8. largest = right
9. if largest != index:
10. arr[index], arr[largest] = arr[largest], arr[index]
11. heapify(arr, n, largest)
12. def heap\_sort(arr):
13. n = len(arr)
14. for i in range(n - 1, -1, -1):
15. heapify(arr, n, i)
16. for i in range(n - 1, 0, -1):
17. arr[i], arr[0] = arr[0], arr[i]
18. heapify(arr, i, 0)

В листинге 3 представлена реализация алгоритма сортировки слиянием.

Листинг 3:

1. def merge\_sort(arr):
2. n = len(arr)
3. if n > 1:
4. mid = n // 2
5. left\_arr = arr[:mid]
6. right\_arr = arr[mid:]
7. merge\_sort(left\_arr)
8. merge\_sort(right\_arr)
9. i = j = k = 0
10. nl = len(left\_arr)
11. nr = len(right\_arr)
12. while i < nl and j < nr:
13. if left\_arr[i] < right\_arr[j]:
14. arr[k] = left\_arr[i]
15. i += 1
16. else:
17. arr[k] = right\_arr[j]
18. j += 1
19. k += 1
20. while i < nl:
21. arr[k] = left\_arr[i]
22. i += 1
23. k += 1
24. while j < nr:
25. arr[k] = right\_arr[j]
26. j += 1
27. k += 1

### Экспериментальная часть

Далее будут приведены примеры работы программы и результаты проведенных экспериментов.

#### Примеры работы программы

Далее на рисунках 4.1 - 4.5 будут представлены примеры работы программы на различных входных данных.

|  |
| --- |
|  |
| *Рис. 4.1 - пример работы программы для пустого массива* |

|  |
| --- |
|  |
| *Рис. 4.2 - пример работы программы для отсортированного по возрастанию массива* |

|  |
| --- |
|  |
| *Рис. 4.3 - пример работы программы для отсортированного по убыванию массива* |

|  |
| --- |
|  |
| *Рис. 4.4 - пример работы программы для массива, заполненного случайными числами* |

|  |
| --- |
|  |
| *Рис. 4.4 - пример работы программы для массива, заполненного одинаковыми числами* |

При функциональном тестировании все тесты были пройдены и результаты совпали с ожидаемыми.

#### Постановка эксперимента по замеру времени

При сравнении быстродействия алгоритмов были использованы массивы размерами в диапазоне от 100 до 1000 с шагом 100. Результат одного эксперимента рассчитывался как средний из результатов проведенных испытаний с одинаковыми входными данными. Количество повторов каждого эксперимента = 100. Результат одного эксперимента рассчитывается как средний из результатов проведенных испытаний с одинаковыми входными данными.

#### Сравнительный анализ на материале экспериментальных данных

Ниже на рис. 4.6 - 4.8 приведены графики зависимости временных затрат работы алгоритмов (в секундах) от размеров входного массива.

|  |
| --- |
|  |
| *Рис. 4.6 - график зависимости времени работы программы от длины входного массива при отсортированном входном массиве* |

|  |
| --- |
|  |
| *Рис. 4.7 - график зависимости времени работы программы от длины входного массива при заполнении массива случайными числами* |

|  |
| --- |
|  |
| *Рис. 4.8 - график зависимости времени работы программы от длины входного массива при отсортированном в обратном порядке массиве* |

Как видно из графиков, время работы сортировки выбором гораздо больше, чем время работы остальных двух алгоритмов. При этом линия графика сортировки выбора есть ветвь параболы, что подтверждает проведенный ранее расчет трудоемкости. Поскольку было сказано, что каждый из трех алгоритмов имеет одинаковый характер трудоемкость в худшем и в лучшем случаях, то картина не меняется критически для лучшего и произвольного случаев. Однако стоит заметить, что алгоритм сортировки слиянием работает быстрее пирамидальной сортировки, хотя ранее отмечалось, что  порядок сложности алгоритмов одинаковый.

### Заключение

Цель данной лабораторной работы была достигнута. В ходе работы были изучены алгоритмы сортировки: выбором, слиянием и пирамидальная сортировка. Данные алгоритмы были реализованы и были оценены их трудоемкости. Был проведен сравнительный анализ перечисленных выше алгоритмов, а также были экспериментально получены зависимости времени выполнения алгоритмов от размеров входных данных. В результате: сортировка выбором оказалась наиболее трудоемкой и самой медленной из рассматриваемых алгоритмов. Однако при использовании более подходящей для этого алгоритма структуры данных (двоичной пирамиды) трудоемкость снижается с квадратичной до O(n \* logn). При этом стоит отметить, что пирамидальную сортировку считают не устойчивой. В то же время сортировка слиянием, оцениваемая в ту же временную сложность, считается устойчивой и работает немного быстрее пирамидальной.

### Список литературы

1. Скиена, С. Алгоритмы. Руководство по разработке. [Текст] / С. Скиена. – 2-е изд.; Пер. с англ. – СПб.: БХВ-Петербург, 2011. – 720 с.: ил.