Титульник

**Содержание**

1. Цель работы…………………………………………………………………3

2.Задание…………………………………………………………………….…3

3.Ход работы…………………………………………………………………...3

3.1 Создание файла……..………….……………..…………………….3

3.2 Сортировка слиянием. Основные особенности……………….…3

3.2.1 Алгоритм сортировки………………………………………….....3

3.2.2 Достоинства и недостатки………………………………………..4

3.2.3 Функциональность………………………………………………..5

3.3 Тестирование программы и скриншоты выполнения программы…………………………………………..………………………….6

4.Вывод ….……………………………………………………………………..7

5. Список используемой литературы………………………………………....8

6.ПРИЛОЖЕНИЕ А……………………………………………………………9

7.ПРИЛОЖЕНИЕ Б……………………………………………………...……10

**1. Цель работы:**  Изучить распространённые алгоритмы сортировки.

**2. Задание:**

Задача: Сортировка слиянием (восходящая).

**3. Ход работы:**

1. Разобраться с принципом работы алгоритма согласно варианту
2. Реализовать алгоритм
3. Проанализировать алгоритм, выяснить его преимущества и недостатки

**3.1 Создание файлов**

Винтегрированной среде разработки на JavaScript, CSS и HTML JetBrains WebStorm создаем новый файл HTML index.html (рисунок 1), в котором попишем скрипт для подключения файла sorting.js к проекту.

В файле index.html в тегах для языка JavaScript <script></script>создаем массив с данными arr[], далее вызываем функцию Sortingmerge()из файла sorting.js , которая принимает на вход массив arr.

Код программы приведен в приложении А (код файла index.html)

и Б ( код файла sorting.js).

**3.2****Сортировка слиянием.**

**Основные особенности**

**3.2.1 Алгоритм сортировки**

**Сортировка слиянием**—[алгоритм сортировки](https://ru.wikipedia.org/wiki/Алгоритм_сортировки), который упорядочивает списки (или другие с[труктуры данных](https://ru.wikipedia.org/wiki/Структура_данных), доступ к элементам которых можно получать только [последовательно](https://ru.wikipedia.org/wiki/Последовательный_доступ), например — [потоки](https://ru.wikipedia.org/wiki/Поток_данных)) в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

Для решения задачи сортировки эти три этапа выглядят так:

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.

В методе восходящего слияния отсутствует процедура рекурсивного разделения последовательности пополам. Исходная последовательность представляется как последовательный набор отдельных элементов.

**3.2.2 Достоинства и недостатки**

**Достоинства:**

* Работает даже на структурах данных последовательного доступа.
* Хорошо сочетается с подкачкой и кэшированием памяти.
* Не имеет «трудных» входных данных.
* Устойчивая - сохраняет порядок равных элементов (принадлежащих одному классу эквивалентности по сравнению).

**Недостатки:**

* На «почти отсортированных» массивах работает столь же долго, как на хаотичных. Существует вариант сортировки слиянием, который работает быстрее на частично отсортированных данных, но он требует дополнительной памяти, в дополнении ко временному буферу, который используется непосредственно для сортировки.
* Требует дополнительной памяти по размеру исходного массива.

**3.2.3 Функциональность**

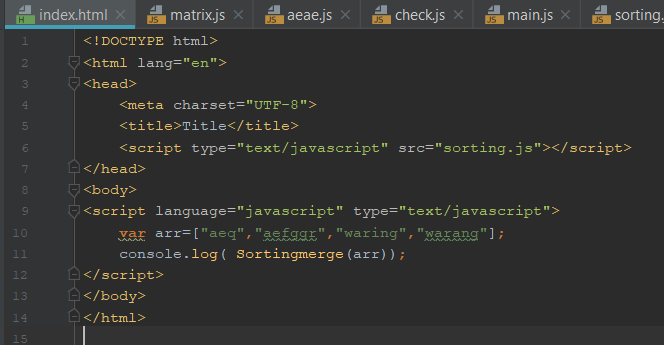
|  |  |
| --- | --- |
| Структура данных | [Массив](https://ru.wikipedia.org/wiki/Массив_(программирование)) |
| [Худшее время](https://ru.wikipedia.org/wiki/Вычислительная_сложность) | O(*n* log2 *n*) |
| [Лучшее время](https://ru.wikipedia.org/wiki/Вычислительная_сложность) | O(*n* log2 *n*) |
| [Среднее время](https://ru.wikipedia.org/wiki/Вычислительная_сложность) | O(*n* log2 *n*) |
| [Затраты памяти](https://ru.wikipedia.org/wiki/Вычислительная_сложность) | O(*n*) вспомогательных |

**3.3 Тестирование программы и скриншоты выполнения программы**

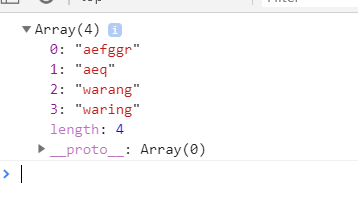
1. Создаем функцию Sortingmerge().
2. В функции создаем локальных две переменных:

var step, var temp [], var left   
(будет означать левую границу сортировки массива), var middle (будет означать среднее значение в сортировке), var right (будет означать правую границу сортировки массива), var index (счетчик).

1. Создаем массив для сортировки, вызываем функцию Sortingmerge().

**

*Рисунок 2 -Ввод начальных данных, вызов методов*

**

*Рисунок 3 -Результат выполнения метода создания структуры*

**4. Вывод**

В данной лабораторной работе познакомились с алгоритмом сортировки слиянием. Данный способ сортировки удобен для структур с последовательным доступом к элементам, таким как связные списки. Недостатком данного алгоритма является то, что он он требует дополнительного выделения памяти под еще один массив, такой же размерности, что и передаваемый.

**5. Список используемой литературы**

1. Сортировка слиянием (binary heap) [Электронный ресурс].

URL:<https://prog-cpp.ru/sort-merge/>(Дата обращения: 15.11.2018).

2. Сортировка слиянием [Электронный ресурс].

URL:<https://ru.wikipedia.org/wiki/Сортировка_слиянием>(Дата обращения: 17.11.2018).

ПРИЛОЖЕНИЕ А

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Title</title>

<script type="text/javascript" src="sorting.js"></script>

</head>

<body>

<script language="javascript" type="text/javascript">

var arr=["aeq","aefggr","waring","warang"];

console.log( Sortingmerge(arr));

</script>

</body>

</html>

ПРИЛОЖЕНИЕ Б

function Sortingmerge(arr) {

var step = 1;

var temp = [arr.length];

var j=0;

while (step < arr.length) {

var left = 0;

var middle = left + step;

var right = left + step \* 2;

var index = 0;

while (left < arr.length) {

middle = middle < arr.length ? middle : arr.length;

right = right < arr.length ? right : arr.length;

var indexleft = left;

var indexmid = middle;

for (; indexleft < middle && indexmid < right;) {

if (Compare(arr[indexleft],arr[indexmid],j)) {

temp[index++] = arr[indexleft++];

}

else {

temp[index++] = arr[indexmid++];

}

}

while (indexleft < middle) temp[index++] = arr[indexleft++];

while (indexmid < right) temp[index++] = arr[indexmid++];

left += step \* 2;

middle += step \* 2;

right += step \* 2;

}

for (var i = 0; i < arr.length; i++) {

arr[i] = temp[i];

}

step \*= 2;

}

return arr;

}

function Compare(str1,str2,j)

{

var answer;

if(str1[j].toLowerCase() <str2[j].toLowerCase())

{

answer = true;

}

if(str1[j].toLowerCase() >str2[j].toLowerCase())

{

answer= false;

}

if(str1[j].toLowerCase() ==str2[j].toLowerCase()){

j++;

Compare(str1,str2,j);

}

return answer;

}