Санкт-Петербургский Политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Кафедра «Мехатроника и роботостроение (при ЦНИИ РТК)»

Курсовая работа
Дисциплина: Объектно-ориентированное программирование
Тема: Сортировка слиянием

Разработал: ст. гр. 3331506/80401 Резец Ю. А.
Преподаватель Кузнецова Е. М.

Санкт-Петербург
2021 г

# Содержание

# Введение

Сортировка слиянием – алгоритм сортировки, который упорядочивает списки (или другие структуры данных, доступ к элементам, которых можно получать только последовательно) в определённом порядке. Алгоритм сортировки слияниями был изобретён венгеро-американским математиком Джоном фон Нейманом в 1945 году.

Данный алгоритм применяется тогда, когда есть возможность использовать для хранения промежуточных результатов память, сравнимую с размером исходного массива. Он построен на принципе декомпозиции. Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Далее их решения комбинируются, и получается решение исходной задачи.

## Описание алгоритма

Сортировка слиянием (mergesort) представляет собой превосходный пример успешного применения метода декомпозиции. Она сортирует заданный массив A[0…n - 1] путем разделения его на две половины, A[0…[n/2] - 1] и A[[n/2] – 1], рекурсивной сортировки каждой половины и слияния двух отсортированных половин в один массив [1, с. 169].

На рисунке 1 представлен псевдокод сортировки слиянием.

```
Алгоритм Mergesort(A[0..n − 1])
  // Рекурсивно сортирует массив A[0..n − 1]
  // Входные данные:  Массив A[0..n − 1] упорядочиваемых
  //                  элементов
  // Выходные данные: Отсортированный в неубывающем порядке
  //                  массив A[0..n − 1]
  if n > 1
    Копировать A[0..⌊n/2⌋ − 1] в B[0..⌊n/2⌋ − 1]
    Копировать A[⌊n/2⌋..n − 1] в C[0..⌊n/2⌋ − 1]
    Mergesort(B[0..⌊n/2⌋ − 1])
    Mergesort(C[0..⌊n/2⌋ − 1])
    Merge(B, C, A)
```

Рисунок 1

Слияние двух отсортированных массивов можно выполнить следующим образом. Два указателя (индекса массивов) после инициализации указывают на первые элементы сливаемых массивов. Затем элементы, на которые указывают указатели, сравниваются, и меньший из них добавляется в новый массив. После этого индекс меньшего элемента увеличивается, и он указывает на элемент, непосредственно следующий за только что скопированным. Эта операция повторяется до тех пор, пока не будет исчерпан один из сливаемых массивов, после чего оставшиеся элементы второго массива просто добавляются в конец нового массива [1, с. 170].

На рисунке 2 представлен псевдокод алгоритма слияния.

Алгоритм $Merge\,(B\,[0..p-1]\,.\,C\,[0..q-1]\,,\,A\,[0..p+q-1])$

// Слияние двух отсортированных массивов в один
// **Входные данные:** Сортированные массивы $B[0..p-1]$
// и $C[0..q-1]$
// **Выходные данные:** Сортированный массив $A[0..p+q-1]$,
// состоящий из элементов $B$ и $C$
$i \leftarrow 0;\ j \leftarrow 0;\ k \leftarrow 0$
**while** $i < p$ **and** $j < q$ **do**
  **if** $B[i] \leqslant C[j]$
    $A[k] \leftarrow B[i];\ i \leftarrow i+1$
  **else**
    $A[k] \leftarrow C[j];\ j \leftarrow j+1$
  $k \leftarrow k+1$
**if** $i = p$
  Копировать $C[j..q-1]$ в $A[k..p+q-1]$
**else**
  Копировать $B[i..p-1]$ в $A[k..p+q-1]$

Рисунок 2

Работа алгоритма со списком 8,3,2,9,7,1,5,4 показана на рисунке 3.

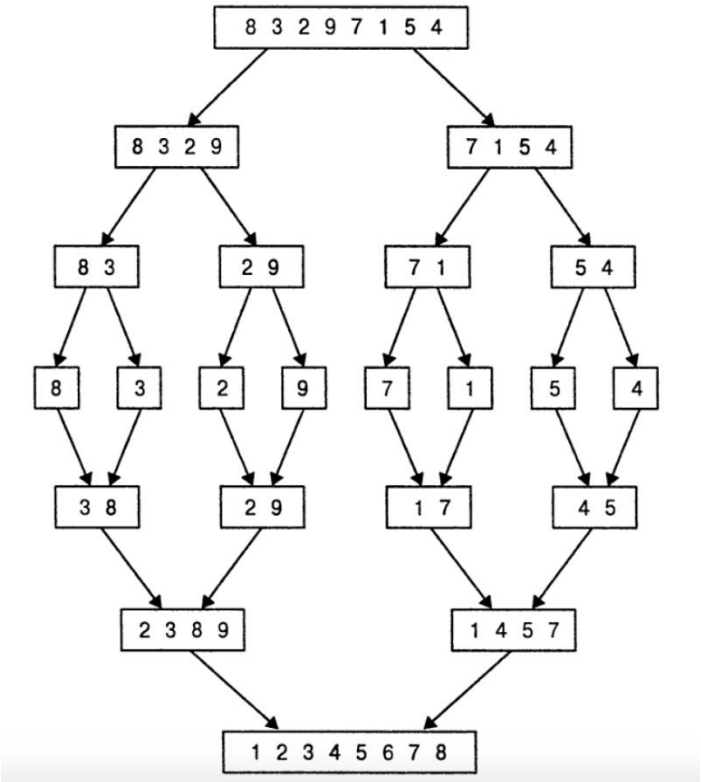


Рисунок 3

Основной недостаток сортировки слиянием - необходимость дополнительной памяти, количество которой линейно пропорционально размеру входных данных. Сортировка слиянием возможна и без привлечения дополнительной памяти, но такая экономия памяти существенно ее усложняет и дает в результате значительно больший постоянный множитель в формуле для времени работы [1, с. 172].

## Скорость работы алгоритма

Количество сравнений ключей, выполняемых сортировкой слиянием, в худшем случае весьма близко к теоретическому минимуму ($log_n n!$) количества сравнений для любого алгоритма сортировки, основанного на сравнениях [1, с. 172].

Скорость работы алгоритма в терминах Big O:

Лучшее время: $O(n\ log(n))$

Худшее время: $O(n\ log(n))$

Среднее время: $O(n\ log(n))$

На рисунке 4 представлен график зависимости времени сортировки массива, от количества элементов в этом массиве.

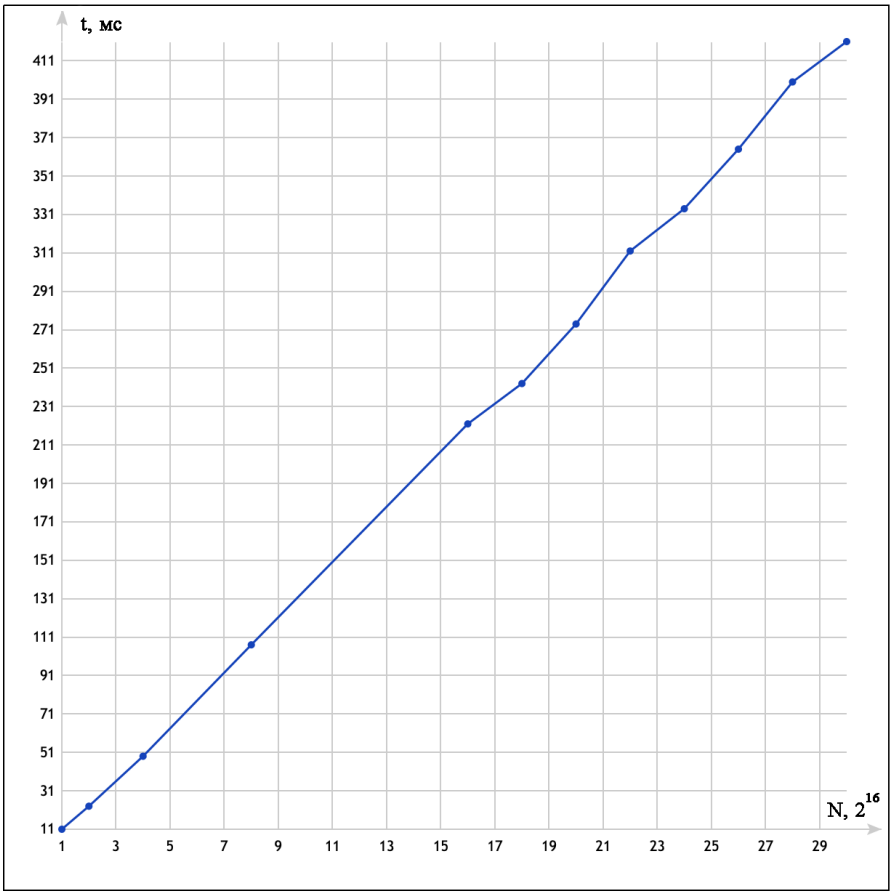


Рисунок 4

## Список литературы

1. Левитин, А. В. Алгоритмы. Введение в разработку и анализ / М. Вильямс, 2006.

# Приложение 1. Код программы

```
#include <iostream>

void merge_concatenated_part_of_array(int *array, int *temp_array, const int start_of_sector,
const int end_of_sector)
{
   const int middle_of_sector = (start_of_sector + end_of_sector)/2;
   int part1_index = start_of_sector;
   int part2_index = middle_of_sector + 1;

   for (int i = start_of_sector; i <= end_of_sector; i++)
   {
      if ((part1_index <= middle_of_sector) && ((part2_index > end_of_sector) ||
(array[part1_index] < array[part2_index])))
      {
         temp_array[i] = array[part1_index];
         part1_index++;
      }
      else
      {
         temp_array[i] = array[part2_index];
         part2_index++;
      }
   }
   for (int i = start_of_sector; i <= end_of_sector; i++)
      array[i] = temp_array[i];
}

void split_and_sort_array(int *array, int *temp_array, const int start_of_sector, const int
end_of_sector)
{
      if (start_of_sector < end_of_sector)
      {
         split_and_sort_array(array, temp_array, start_of_sector, (start_of_sector +
end_of_sector)/2);
         split_and_sort_array(array, temp_array, (start_of_sector + end_of_sector)/2 + 1,
end_of_sector);
         merge_concatenated_part_of_array(array, temp_array, start_of_sector, end_of_sector);
      }
}

void merge_sort(int *array, const int size)
{
   int *temp_array = new int[size];
   split_and_sort_array(array, temp_array, 0, size - 1);
   delete []temp_array;
}

int main()
{
   return 0;
}
```