ФЕДЕРАЛЬН ОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО»

Институт машиностроения, материалов и транспорта

Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: «Объектно-ориентированное программирование»

Вариант: 10

Выполнил студент: гр. 3331506/90401 Шлыков Ф. В.

Преподаватель Ананьевский М. С.

Санкт-Петербург

2022

Оглавление

Введение

Применение

- Сортировка слиянием полезна для сортировки связанных списков.
- Сортировка слиянием может быть реализована без дополнительного места для связанных списков.
- Сортировка слиянием используется для подсчета инверсий в списке.
- Сортировка слиянием используется во внешней сортировке.

Описание алгоритма

1) Массив рекурсивно разбивается пополам, и каждая из половин делиться до тех пор, пока размер очередного подмассива не станет равным единице(рис. 1)

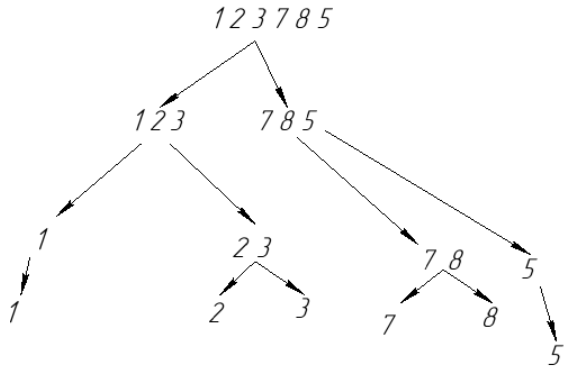

Рисунок 1 – Рекурсивное разбитие массива

2) Далее выполняется операция алгоритма, называемая слиянием(рис. 2). Два единичных массива сливаются в общий результирующий массив, при этом из каждого выбирается меньший элемент (сортировка по возрастанию) и записывается в свободную левую ячейку результирующего массива. После чего из двух результирующих массивов собирается третий общий

отсортированный массив, и так далее. В случае если один из массивов закончиться, элементы другого дописываются в собираемый массив;

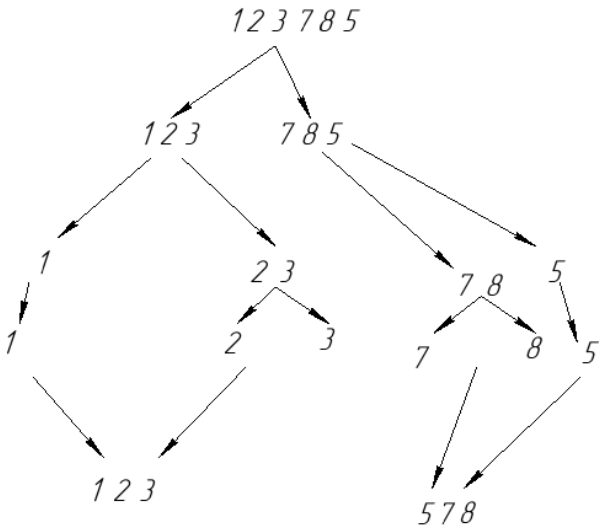

Рисунок – 2 слияние

3) В конце операции слияния, элементы перезаписываются(рис. 3) из результирующего массива в исходный.

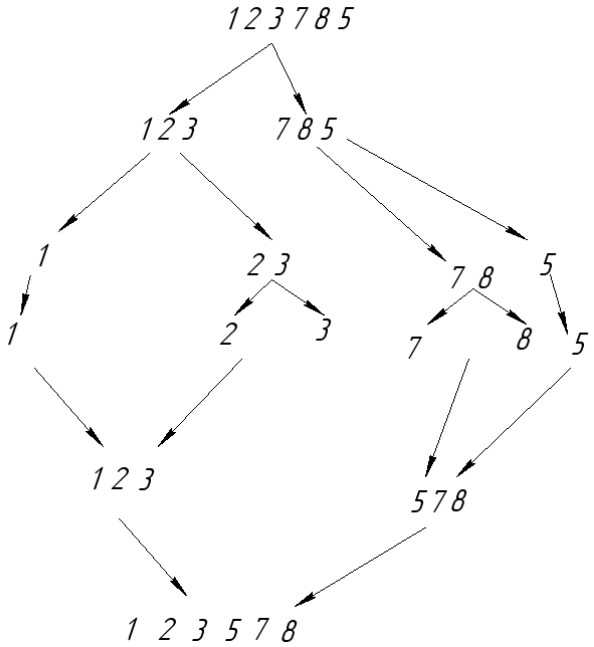

Рисунок – 3 перезапись элементов

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]. Ниже представлен псевдокод сортировки слиянием.

```
Подпрограмма MergeSort(massiv, firstElement, lastElement)
{
//A – массив
//firstElement, lastElement – номера первого и последнего элементов соответственно
Если firstElement<lastElement то
{
Вызов MergeSort(massiv, firstElement, (firstElement+lastElement)/2) //сортировка левой части
Вызов MergeSort(massiv, (firstElement+lastElement)/2+1, last) //сортировка правой части
Вызов Merge(massiv, firstElement, lastElement) //слияние двух частей
}
}

Подпрограмма Merge(A, firstElement, lastElement)
{
//start, final – номера первых элементов левой и правой частей
//massive – массив, middleElement - хранит номер среднего элемента
middleElement=(firstElement+lastElement)/2 //вычисление среднего элемента
start=firstElement //начало левой части
final=middleElement+1 //начало правой части
Цикл i=firstElement до lastElement выполнять //выполнять от начала до конца
Если ((start<=middleElement) и ((final>lastElement) или (A[start]<A[final]))) то
{
mas[i]=A[start]
увеличить start на 1
}
Иначе
{
mas[i]=A[final]
увеличить final на 1
}
```

Цикл i=firstElement до lastElement выполнять //возвращение результата в список
A[i]=mas[i]
}

Исследование алгоритма

Скорость работы алгоритма

Количество сравнений ключей, выполняемых сортировкой слиянием, в худшем случае весьма близко к теоретическому минимуму ($\log_2 n!$) количества сравнений для любого алгоритма сортировки, основанного на сравнениях [1, с. 172]. Скорость работы алгоритма в терминах Big O:

Лучшее время: ($O(n \cdot \log_2 n)$)

Худшее время: ($O(n \cdot \log_2 n)$)

Среднее время: ($O(n \cdot \log_2 n)$)

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

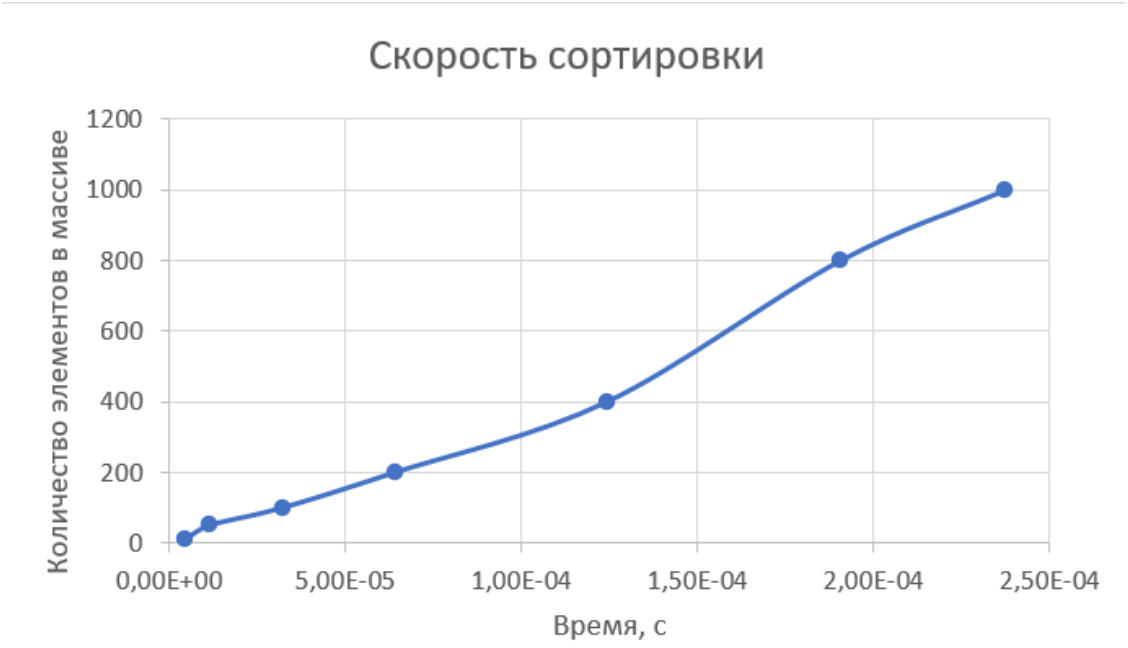

Рисунок 4 – график зависимости времени выполнения от количества элементов

Заключение

В работе был рассмотрен алгоритм сортировки слиянием (Merge sort). В результате работы была полученная экспериментальным способом зависимость времени от количества элементов.

Список литературы

1. Левитин, А. В. Алгоритмы. Введение в разработку и анализ / М. Вильямс, 2006.
2. Analysis of Merge Sort, https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/analysis-of-merge-sort
3. Merge Sort | Brilliant Math and Science Wiki, https://brilliant.org/wiki/merge/#:~:text=Mergesort%20runs%20in%20a%20guaranteed,with%20large%20amounts%20of%20data.

Приложение

```
#include <iostream>
#include <ctime>
#include <chrono>
#define MAXSIZE 1000

using namespace std;

void Merge(int* A, int firstElement, int lastElement) {
//функция, сливающая массивы
    static int middleElement, start, final;
    static int mas[MAXSIZE];
    middleElement = (firstElement + lastElement) / 2;
//вычисление среднего элемента
    start = firstElement;
//начало левой части
    final = middleElement + 1;
//начало правой части
    for (int i = firstElement; i <= lastElement; i++) {
//выполнять от начала до конца
        if ((start <= middleElement) && ((final > lastElement) || (A[start] <
A[final]))) {
            mas[i] = A[start];
            start++;
        } else {
            mas[i] = A[final];
            final++;
        }
    }
    for (int i = firstElement; i <= lastElement; i++){
        A[i] = mas[i];
//возвращение результата в список
    }
};
void MergeSort(int massiv[], int firstElement, int lastElement) {
//рекурсивная процедура сортировки
    if (firstElement < lastElement) {
        MergeSort(massiv, firstElement, (firstElement + lastElement) / 2);
//сортировка левой части
        MergeSort(massiv, (firstElement + lastElement) / 2 + 1, lastElement);
//сортировка правой части
        Merge(massiv, firstElement, lastElement);
//слияние двух частей
    }
};
class Timer {
public:
    Timer() {
        start = std::chrono::high_resolution_clock::now();
    }
    ~Timer() {
        end = std::chrono::high_resolution_clock::now();
        std::chrono::duration < double > duration = end - start;
        std::cout << "DURATION " << duration.count() << " s\n\n";
    }
private:
    std::chrono::time_point < std::chrono::high_resolution_clock > start,
end;
};
int main() {
    srand(time(NULL));
```

```
    int arraSize;
    cout << "Array size > ";
    cin >> arraSize;
    //int testArray[6] = {1,4,5,2,7,8};
    int* sortableArray = new int[arraSize];
    for (int i = 0; i < arraSize; i++) {
        sortableArray[i] = (rand() % 100);
    }
    Timer t;
    //MergeSort(testArray, 0, 6);  // тестирование сортировки
    MergeSort(sortableArray, 0, arraSize-1);    //вызов сортирующей процедуры
    cout << "sorted array: ";   //вывод упорядоченного массива

    for (int i = 1; i <= arraSize; i++) {
        cout << sortableArray[i] << " ";
    }
    return 0;
}
```