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 3

Исследование алгоритма 6

Заключение 7

Список литературы 8

Приложение 9

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)

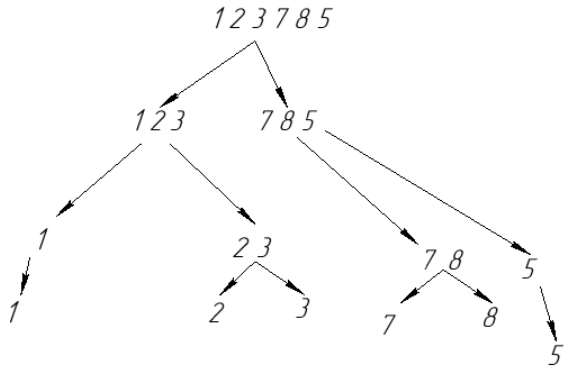

Рисунок 1 – Рекурсивное разбитие массива

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;

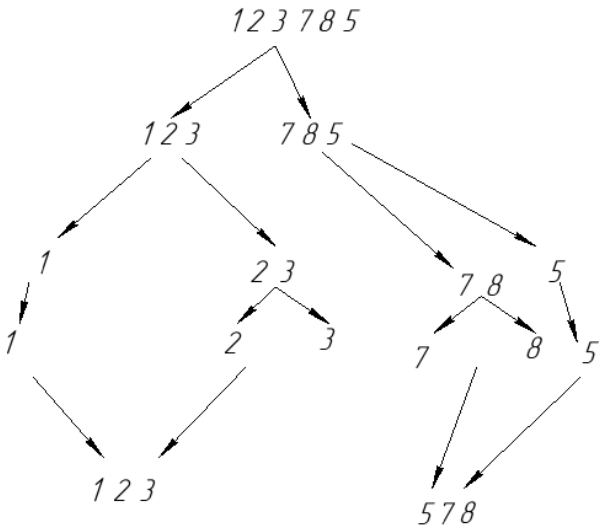

Рисунок – 2 слияние

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

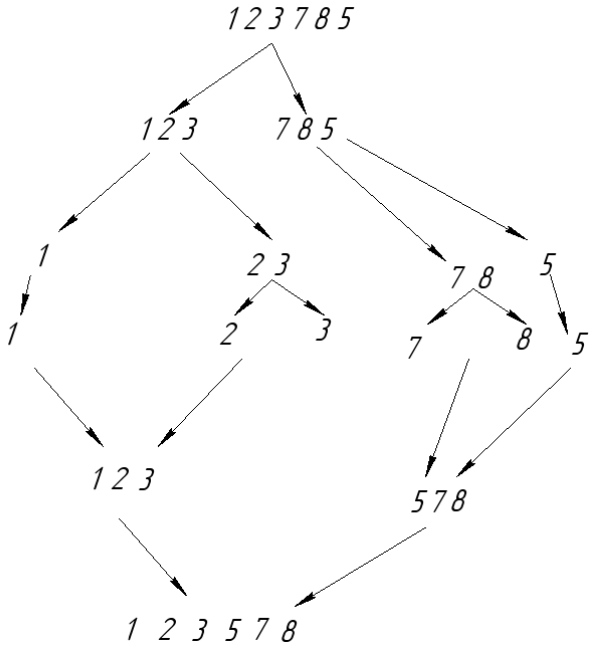

Рисунок – 3 перезапись элементов

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(A, first, last)
{
//A – массив
//first, last – номера первого и последнего элементов соответственно
Если first<last то
{
Вызов MergeSort(A, first, (first+last)/2) //сортировка левой части
Вызов MergeSort(A, (first+last)/2+1, last) //сортировка правой части
Вызов Merge(A, first, last) //слияние двух частей
}
}
Подпрограмма Merge(A, first, last)
{
//start, final – номера первых элементов левой и правой частей
//mas – массив, middle - хранит номер среднего элемента
middle=(first+last)/2 //вычисление среднего элемента
start=first //начало левой части
final=middle+1 //начало правой части
Цикл j=first до last выполнять //выполнять от начала до конца
Если ((start<=middle) и ((final>last) или (A[start]<A[final]))) то
{
mas[j]=A[start]
увеличить start на 1
}
Иначе
{
mas[j]=A[final]
увеличить final на 1
}
Цикл j=first до last выполнять //возвращение результата в список
A[j]=mas[j]
}
```

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.

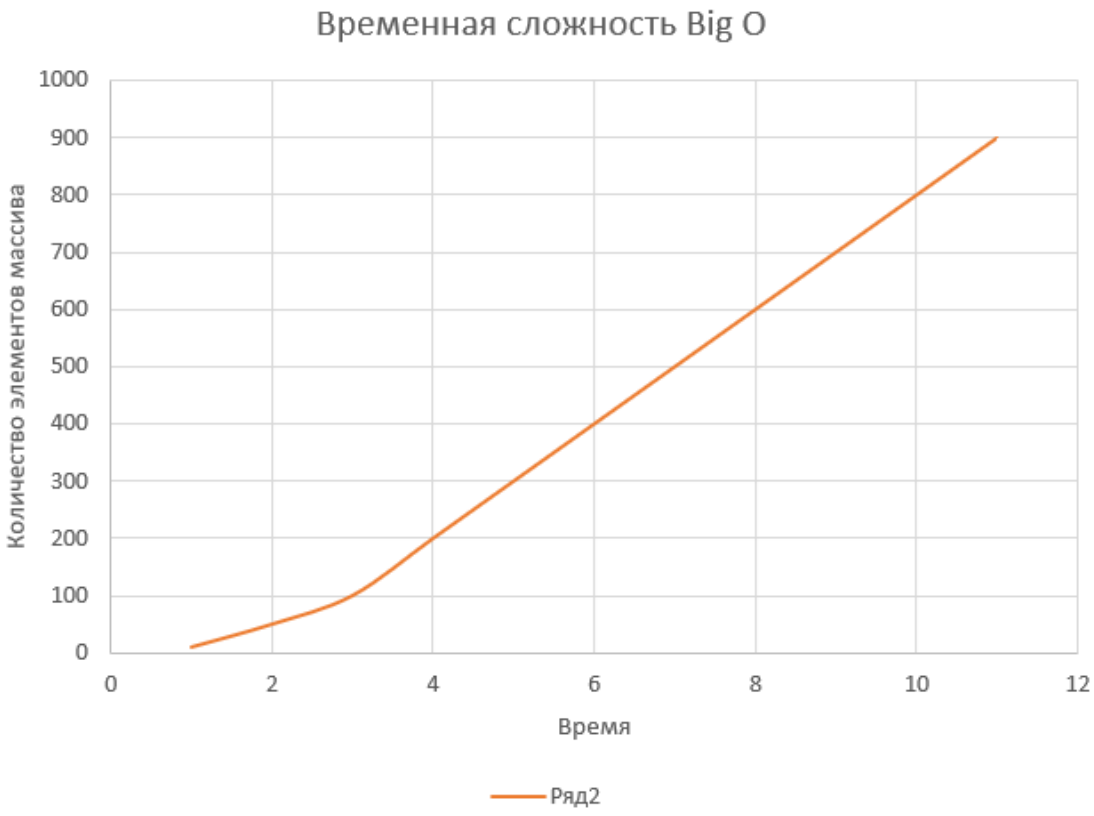

Рисунок 4 – график зависимости от времени

Заключение

В работе был рассмотрен алгоритм сортировки слиянием (Merge sort). Например, такая сортировка используется для связных списков.

Анализ алгоритма показал, что скорость работы почти линейно растет с увеличением размера массива, и не имеет разницы по значению между худшим и лучшим.

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>

using namespace std;
//функция, сливающая массивы
void Merge(int * A, int first, int last) {
    int middle, start, final, j;
    int * mas = new int[1000];
    middle = (first + last) / 2; //вычисление среднего элемента
    start = first; //начало левой части
    final = middle + 1; //начало правой части
    for (j = first; j <= last; j++) //выполнять от начала до конца
        if ((start <= middle) && ((final > last) || (A[start] < A[final]))) {
            mas[j] = A[start];
            start++;
        }
        else {
            mas[j] = A[final];
            final++;
        }
    //возвращение результата в список
    for (j = first; j <= last; j++) A[j] = mas[j];
    delete[] mas;
};
//рекурсивная процедура сортировки
void MergeSort(int * A, int first, int last) {
    {
        if (first < last) {
            MergeSort(A, first, (first + last) / 2); //сортировка левой части
            MergeSort(A, (first + last) / 2 + 1, last); //сортировка правой
части
            Merge(A, first, last); //слияние двух частей
        }
    }
};
//главная функция
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(0));
    int i, n;
    int * A = new int[1000];
    cout << "Array size > ";
    cin >> n;
    for (i = 0; i < n; i++) {
        A[i] = (rand() % 1000);
```

```
    } {
        Timer t;
        MergeSort(A, 1, n); //вызов сортирующей процедуры
    }
    cout << "sorted array: "; //вывод упорядоченного массива
    for (i = 1; i <= n; i++) cout << A[i] << " ";
    delete[] A;
    return 0;
}
```