МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Сортировки

Студент гр. 9381 _____ Авдеев И.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с быстрой сортировкой и создать программу для реализации алгоритма.

Задание.

Варинат 8

Быстрая сортировка, рекурсивная реализация. Во время сортировки массив должен

быть в состоянии:

элементы <x, неотсортированные элементы, элементы >=x.

Описание алгоритма функции

В массиве выбирается элемент, называемый разрешающим, путем вычисления среднего значения между крайними элементами массива. Алгоритм использует два индекса, один - в начале массива, другой - в конце, которые приближаются друг к другу, пока не найдётся пара элементов, где один больше разрешающего и расположен перед ним, а второй меньше и расположен после. Эти элементы меняются местами. Обмен происходит до тех пор, пока индексы не пересекутся.

Тем самым массив разбивается на две части:

1. элементы слева от разрешающего элемента меньшие
2. элементы справа от разрешающего элемента большие

Чтобы отсортировать эти два меньших подмассива, алгоритм рекурсивно вызывает сам себя.

Оценка сложности алгоритма

Операция разделения массива на две части относительно опорного элемента занимает время $O(\log_2 n)$. Поскольку все операции разделения, проделываемые на одной глубине рекурсии, обрабатывают разные части исходного массива, размер которого постоянен, суммарно на каждом уровне рекурсии потребуется также операций $O(n)$. Следовательно, общая сложность алгоритма определяется лишь количеством разделений, то есть глубиной рекурсии. Глубина рекурсии, в свою очередь, зависит от сочетания входных данных и способа определения опорного элемента.

Лучший случай

В наиболее сбалансированном варианте при каждой операции разделения массив делится на две одинаковые (плюс-минус один элемент) части, следовательно, максимальная глубина рекурсии, при которой размеры обрабатываемых подмассивов достигнут 1, составит $O(\log_2 n)$. В результате общая сложность алгоритма $O(n*\log_2 n)$.

Худший случай

В самом несбалансированном варианте каждое разделение даёт два подмассива размерами 1 и n - 1, то есть при каждом рекурсивном вызове больший массив будет на 1 короче, чем в предыдущий раз. Такое может произойти, если в качестве опорного на каждом этапе будет выбран элемент либо наименьший, либо наибольший из всех обрабатываемых. При простейшем выборе опорного элемента — первого или последнего в массиве, — такой эффект даст уже отсортированный (в прямом или обратном порядке) массив, для среднего или любого другого фиксированного элемента «массив худшего случая» также может быть специально подобран. В этом случае потребуется n – 1 операций разделения, а общее время работы составит $O(n^2)$ операций, то есть сортировка будет выполняться за квадратичное время. Но количество обменов и, соответственно, время работы — это не самый большой его недостаток. Хуже то, что в таком случае глубина рекурсии при выполнении алгоритма достигнет n, что будет означать n-кратное сохранение адреса возврата и локальных переменных процедуры разделения

массивов. Для больших значений n худший случай может привести к исчерпанию памяти во время работы программы.

Плюсы

1. Один из самых быстродействующих (на практике) из алгоритмов внутренней сортировки общего назначения.
2. Работает на связных списках и других структурах с последовательным доступом, допускающих эффективный проход как от начала к концу, так и от конца к началу.

Минусы

1. Сильно деградирует по скорости в худшем или близком к нему случае, что может случиться при неудачных входных данных.
2. Прямая реализация в виде функции с двумя рекурсивными вызовами может привести к ошибке переполнения стэка, так как в худшем случае ей может потребоваться сделать O(n) вложенных рекурсивных вызовов.

Тестирование функции

Входные данные	Ожидаемый результат	Результат
10 2.1 3.4 8.5 5.1 4.6 9 4.3 7.1 10 9.4	2.1 3.4 4.3 4.6 5.1 7.1 8.5 9 9.4 10	2.1 3.4 4.3 4.6 5.1 7.1 8.5 9 9.4 10
f 1 f 1	1	1

0	0	
9 5 9 1 8 0 7 3 5 5	0 1 3 5 5 5 7 8 9	0 1 3 5 5 5 7 8 9

Выводы

Был освоен алгоритм быстрой сортировки. Изучены различные виды их реализации, такие как разбиение Ломуто и разбиение Хоара. Было проведено исследование сложности алгоритма быстрой сортировки. В ходе выполнения работы, был реализован рекурсивный алгоритм быстрой сортировки на языке c++.

ПРИЛОЖЕНИЕ А

СОДЕРЖАНИЕ ФАЙЛА Source.cpp

```
#include <iostream>
#include <ctime>
#include <conio.h>
#include <fstream>

int fnum = 0;

template<class T>
void qsortRecursive(T* mas, int size)
{
    //Указатели в начало и в конец массива
    int i = 0;
    int j = size - 1;

    //Разрешающий элемент массива
    T mid = (mas[i] + mas[j]) / 2;

    //Делим массив
    do {
        //ищем то, что нужно перекинуть в другую часть
             //В левой части массива пропустить элементы, которые меньше
центрального
        while (mas[i] < mid)
        {
            i++;
        }
        //В правой части пропустить элементы, которые больше центрального
        while (mas[j] > mid)
        {
            j--;
        }

        //Поменять элементы местами
        if (i <= j)
        {
            T tmp = mas[i];
```

```
            mas[i] = mas[j];
            mas[j] = tmp;

            i++;
            j--;
        }
    } while (i <= j);

    if (j > 0)
    {
        //Левая часть
        qsortRecursive(mas, j + 1);
    }
    if (i < size)
    {
        //Правая часть
        qsortRecursive(&mas[i], size - i);
    }
}

template<class T>
void qsortRecursivelog(T* mas, int size)
{
    //Указатели в начало и в конец массива
    int i = 0;
    int j = size - 1;


    int f = ++fnum;
    int q = size - 1;

    std::cout << f << "-й запуск функции\n массив:\n";
    for (; i <= j; i++)
        std::cout << mas[i] << "   ";
    i = 0;

    std::cout << "\n";

    //Разрешающий элемент массива
```

```
    T mid = (mas[i] + mas[j]) / 2;

    std::cout << "разрешающий элемент - " << mid << "\n";

    int l = 0;
    _getch();
    //Делим массив
    do
    {
        std::cout << ++l << "-ая итерация: \n";
        //ищем то, что нужно перекинуть в другую часть
             //В левой части массива пропустить элементы, которые меньше
центрального
        while (mas[i] < mid)
        {
            i++;
        }
        std::cout << "i = " << i << "\n";
        //В правой части пропустить элементы, которые больше центрального
        while (mas[j] > mid)
        {
            j--;
        }
        std::cout << "j = " << j << "\n";
        //Поменять элементы местами
        if (i <= j)
        {
            T tmp = mas[i];
            mas[i] = mas[j];
            mas[j] = tmp;

            for (int p = 0; p <= q; p++)
                std::cout << mas[p] << "   ";

            std::cout << "\n";
            _getch();

            i++;
```

```
            j--;
        }
    } while (i <= j);

    std::cout << size << "\n";
    //Рекурсивные вызовы
    if (j > 0)
    {
        //Левая часть
        std::cout << "Запуск левой части \n";
        _getch();
        qsortRecursivelog(mas, j + 1);
        std::cout << "Вернулся в " << f << "-й запуск функции\n";
    }
    if (i < size)
    {
        //Првая часть
        std::cout << "Запуск правой части \n";
        _getch();
        qsortRecursivelog(&mas[i], size - i);
        std::cout << "Вернулся в " << f << "-й запуск функции\n";
    }
    std::cout << f << "-й запуск функции завершился\n";
    _getch();
}

int main()
{
    setlocale(LC_ALL, "ru");
    int j;
    char jin[10];
    std::cout << "введите размер массива\n";
    while (1)
    {
        std::cin >> jin;
        if (isdigit(*jin))
        {
            j = atoi(jin);
            break;
```

```
        }
        else
        {
            std::cout << "нужно ввести цифру, попробуйте еще\n";
        }
    }
    float* mas = new float[j];

     std::cout << "Ввод из консоли на кнопку 1, из файла на 2, заполнить раномными числами на любую другую\n";
    char k;
    k = _getch();
    srand(time(0));

    std::ifstream lin("lin.txt");
    if (!lin.is_open())
        return 1;
    switch (k)
    {
    case '1':
        for (int i = 0; i < j; i++)
        {
            std::cin >> mas[i];
        }
        break;
    case '2':
        for (int i = 0; i < j; i++)
        {
            lin >> mas[i];
        }
        break;
                                                                    default:
        for (int i = 0; i < j; i++)
        {
            mas[i] = rand() % 10;
                                                                        }
        break;
    }
```

```
    lin.close();
    qsortRecursivelog(mas, j);

    for (int i = 0; i < j; i++)
        std::cout << mas[i] << "   ";


    delete[] mas;

    return 0;
}
```