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. 9303 _____ Лойконен М.Р.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Изучить различные алгоритмы сортировки данных и реализовать один из них. Дать теоретическую оценку сложности алгоритма, указать его недостатки и достоинства.

Задание.

Вариант 11.

Быстрая сортировка, рекурсивная реализация – отсечение малых подмассивов.

Быстрая сортировка выполняется не до конца. Когда сегменты становятся достаточно маленькими, они окончательно сортируются другим методом. Параметр, определяющий размер малого подмассива, задаётся пользователем.

Описание алгоритма.

Сначала функция выбирает опорный элемент (в нашем случае элемент, находящийся в середине массива), и сортирует остальные элементы относительного опорного — левее опорного находятся элементы, меньшие его, правее — бОльшие. У нас есть два индекса — `left` и `right`, которые соответственно указывают на начало и конец рассматриваемого массива. `left` увеличивается, пока элемент с этим индексом меньше опорного, `right` уменьшается, пока элемент с этим индексом больше опорного. Если `left` меньше `right`, то мы меняем местами с индексами `left` и `right` местами, увеличиваем `left` и уменьшаем `right`. Если `left` стал больше `right`, то цикл прерывается. Далее идет 2 проверки, если значение `right` больше изначального значения `left`, то алгоритм рекурсивно вызывает себя для левого подмассива, если значение `left` меньше изначального значения `right`, то алгоритм рекурсивно вызывает себя для правого подмассива.

Для данного алгоритма добавлена модификация, если размер передаваемого массива становится достаточно малым (данный параметр задает пользователь), то данный подмассив сортируется окончательно алгоритмом сортировки пузырьком.

Дадим оценку сложности алгоритма. В среднем алгоритм быстрой сортировки имеет сложность O(n*logn), алгоритм сортировки пузырьком имеет сложность O(n^2). В нашем случае быстрая сортировка выполняется не до конца, а сортировка пузырьком применяется не для всего массива. Предположим что достаточно малым считается массив, размер которого в m раз меньше исходного, n — размер исходного массива. Алгоритм сортировки пузырьком в среднем выполнится m раз для массивов размера m, и будет иметь сложность n/m*O((n/m)^2). Быстрая сортировка будет иметь сложность O(n*log(n/m - 1)). Таким образом можно сказать, что в общем случае сложность данного алгоритма это наиболее быстро растущее слагаемое из полученных двух, то есть мы получаем сложность O(n^2).

В недостаткам данного алгоритма можно отнести рекурсивность, так как существует вероятность того, что на стеке закончится память, а также произвольный выбор размера, который будет считаться достаточно малым, чтобы началась сортировка другим методом, т.к. пользователь может задать размер достаточно большим, и тогда сортировка будет долгой. К достоинствам можно отнести быстроту алгоритма при выборе достаточно малого размера m для подмассива, а так же саму модификацию алгоритма в виде дополнительного вида сортировки, благодаря которому мы можем избежать большого количества рекурсивных вызовов.

Выполнение работы.

Для реализации алгоритма было создано 2 функции - `void quickSort(T* arr, int left, int right, int size, int par, ofstream& fout)` и `void bubbleSort(T* arr, int left,`

`int right)`, которые являются соответственно алгоритмами быстрой сортировки и сортировки пузырьком.

Исходные данные программа считывает из файла, результат так же записывается в файл.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

В ходе выполнения работы был изучен и реализован рекурсивный алгоритм быстрой сортировки с отсечением малых подмассивов. Были изучены его недостатки и достоинства, а также определена сложность алгоритма в среднем случае.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>

using namespace std;

template <typename T>
T* readFromFile(ifstream& fin, int size){
     T* arr = new T[size];
     for (int i = 0; i<size; i++)
          fin >> arr[i];
     return arr;
}

template <typename T>
void printArr(T* arr, ofstream& fout, int size){
     fout << "[ ";
     for (int i = 0; i<size; i++){
          fout << arr[i] << " ";
     }
     fout << "]\n";
}

template<typename T>
void bubbleSort(T* arr, int left, int right){
     for (int i = left; i<=right; i++){
          for (int j = left; j<=right-1; j++){
               if (arr[j] > arr[j+1]){
                    T tmp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = tmp;
               }
          }
     }
}

template <typename T>
void quickSort(T* arr, int left, int right, int size, int
par, ofstream& fout){
     if ((right - left + 1) <= par){
          bubbleSort(arr, left, right);
          printArr<T>(arr, fout, size);
          return;
     }
     int mid = arr[(left + right)/2];
     int l = left;
     int r = right;
```

```
        while (left <= right){
            while (arr[left] < mid)
                left++;
            while (arr[right] > mid)
                right--;
            if (left <= right){
                T tmp = arr[right];
                arr[right] = arr[left];
                arr[left] = tmp;
                left++;
                right--;
            }
        }
        printArr<T>(arr, fout, size);
        if (l < right)
            quickSort<T>(arr, l, right, size, par, fout);
        if (r > left)
            quickSort<T>(arr, left, r, size, par, fout);
}

int main(){
        ifstream fin("test.txt");
        ofstream fout("result.txt");
        while (!fin.eof()){
            int size, par;
            fin >> size;
            fin >> par;
            int* arr = readFromFile<int>(fin, size);
            fout << "Исходный массив: ";
            printArr<int>(arr, fout, size);
            quickSort<int>(arr, 0, size-1, size, par, fout);
            fout << "Отсортированный массив: ";
            printArr<int>(arr, fout, size);
            fout << '\n';
            delete [] arr;
        }
        fin.close();
        fout.close();
        return 0;
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные
1.	10 2 32 5 0 2 5 -12 78 1 76 -52	Исходный массив: [32 5 0 2 5 -12 78 1 76 -52] [-52 1 0 2 -12 5 78 5 76 32] [-52 -12 0 2 1 5 78 5 76 32] [-52 -12 0 2 1 5 78 5 76 32] [-52 -12 0 1 2 5 78 5 76 32] [-52 -12 0 1 2 5 78 5 76 32] [-52 -12 0 1 2 5 5 78 76 32] [-52 -12 0 1 2 5 5 32 76 78] Отсортированный массив: [-52 -12 0 1 2 5 5 32 76 78]
2.	10 2 -78 2 841 6 -4 -43 -1 0 3 9	Исходный массив: [-78 2 841 6 -4 -43 -1 0 3 9] [-78 -43 -4 6 841 2 -1 0 3 9] [-78 -43 -4 6 841 2 -1 0 3 9] [-78 -43 -4 -1 841 2 6 0 3 9] [-78 -43 -4 -1 3 2 0 6 841 9] [-78 -43 -4 -1 0 2 3 6 841 9] [-78 -43 -4 -1 0 2 3 6 9 841] [-78 -43 -4 -1 0 2 3 6 9 841] Отсортированный массив: [-78 -43 -4 -1 0 2 3 6 9 841]
3.	10 2 7 3 -52 -35 89 19 4 -6 -98 -999	Исходный массив: [7 3 -52 -35 89 19 4 -6 -98 -999] [7 3 -52 -35 -999 19 4 -6 -98 89] [-999 3 -52 -35 7 19 4 -6 -98 89] [-999 3 -52 -35 -98 -6 4 19 7 89] [-999 -98 -52 -35 3 -6 4 19 7 89] [-999 -98 -52 -35 3 -6 4 19 7 89] [-999 -98 -52 -35 -6 3 4 19 7 89] [-999 -98 -52 -35 -6 3 4 19 7 89] [-999 -98 -52 -35 -6 3 4 7 19 89]

		Отсортированный массив: [-999 -98 -52 -35 -6 3 4 7 19 89]