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. 9382 _____ Рыжих Р.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

На практике изучить быструю сортировку, а также сортировку вставками, реализовав программу при помощи языка C++.

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.

Основные теоретические положения.

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке

Быстрая сортировка - один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O(nlogn) обменов при упорядочении элементов.

Сортировка вставками - алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов

Описание алгоритма быстрой сортировки (рекурсивная реализация):

Быстрая сортировка функционирует по принципу "разделяй и властвуй". Она разбивает сортируемый массив на две части, затем сортирует эти части независимо друг от друга.

1) Выбрать средний (центральный) элемент переданного массива.

2) Остальные элементы переупорядочиваются таким образом, что меньшие элементы находятся слева от центрального элемента, а большие - справа.

3) Выполнить рекурсивную сортировку левой и правой частей массива.

4) Отсечь рекурсии для небольших подмассивов (размер малого подмассива вводится пользователем).

5) В завершении использовать другой метод – сортировка вставками.

Выполнение работы:

В данной задаче требуется считать массив с элементами, который далее требуется отсортировать.

Функция read(Item arr[], int n)

Данная функция считывает массив n элементов типа Item.

Функция printArray(Item arr[],int l, int r)

Данная функция печатает на экран массив n элементов от l до r.

Функция print(Item arr[], int n)

Данная функция печатает на экран массив n элементов.

Функция exch(Item& a, Item& b)

Функция меняет местами два элемента a и b.

Функция insertionSort()

Функция поочередно меняет текущий элемент с предыдущим в переданном ей подмассиве, пока они стоят в неправильном порядке.

на случай, если центральный элемент окажется наименьшим.

Функция quickSort()

Выберем некоторый опорный элемент. После этого перекинем все элементы, меньшие его, налево, а большие – направо. Рекурсивно вызовемся от каждой из частей. В итоге получим отсортированный массив, так как каждый элемент меньше опорного стоял раньше каждого большего опорного.

Функция main()

В данной функции происходит считывание количества элементов массива, размер малого подмассива, и сам массив элементов. Считывание может происходить как с консоли, так и из файла, в зависимости от того, сколько у программы аргументов (0 или 1 (название файла) соответственно) Происходит динамическое выделение памяти, а далее вызывается функция *quickSort()*. Во время работы программы на экран выводятся промежуточные результаты. В завершении печатается отсортированный массив.

Пример работы программы.

Результаты тестирования представлены в табл. 1.

Таблица 1 — Результаты тестирования

No	Входные данные	Выходные данные	Комментарии
1	5 1 5 1 2 -3 4	Entered array: 5 1 2 -3 4 Middle element is 2 Element 4 stays in place, because it bigger than middle element 2 Changing elements: 5, -3 Intermediate value(quick sort): -3 1 2 5 4 Element 1 stays in place, because it less than middle element 2 Middle element is 1 Element -3 stays in place, because it less than middle element 1 Element 2 stays in place, because it bigger	

		than middle element 1 Beginning of the insertion sorting: Unsorted subarray: 5 4 Changing elements: 5, 4 Intermediate value (insertion): -3 1 2 4 5 Sorted array: -3 1 2 4 5	
2	-5 -10	Enter array: Entered array: Beginning of the insertion sorting: Unsorted subarray: No need to use insertion sorting; Array is already sorted Sorted array:	При вводе отрицательных обязательных значений (или равных нулю) программа просто не считывает массива и ничего не сортирует.
3	10 3 5 -1 5 -2 7 -3 8 -4 9 -5	Entered array: 5 -1 5 -2 7 -3 8 -4 9 -5 Middle element is 7 Element 5 stays in place, because it less than middle element 7 Element -1 stays in place, because it less than middle element 7 Element 5 stays in place, because it less than middle element 7 Element -2 stays in place, because it less than middle element 7 Changing elements: 7, -5 Intermediate value(quick sort): 5 -1 5 -2 -5 -3 8 -4 9 7 Element -3 stays in place, because it less than middle element 7 Element 9 stays in place, because it bigger than middle element 7 Changing elements: 8, -4 Intermediate value(quick sort): 5 -1 5 -2 -5 -	Вывод довольно подробный, выводится центральный элемент, относительно которого всё сортируется, и выводится подробная информация о том, почему тот или иной элемент сортируется или нет. Также, в промежуточном выводе указывается, какой алгоритм сортировки в данный момент используется (в скобках)

		3 -4 8 9 7 Middle element is -2 Element 8 stays in place, because it bigger than middle element -2 Changing elements: 5, -4 Intermediate value(quick sort): -4 -1 5 -2 -5 -3 5 8 9 7 Changing elements: -1, -3 Intermediate value(quick sort): -4 -3 5 -2 -5 -1 5 8 9 7 Changing elements: 5, -5 Intermediate value(quick sort): -4 -3 -5 -2 5 -1 5 8 9 7 Middle element is -5 Element 5 stays in place, because it bigger than middle element -5 Element -2 stays in place, because it bigger than middle element -5 Changing elements: -4, -5 Intermediate value(quick sort): -5 -3 -4 -2 5 -1 5 8 9 7 Beginning of the insertion sorting: Unsorted subarray: -5 -3 -4 Changing elements: -3, -4 Intermediate value (insertion): -5 -4 -3 -2 5 -1 5 8 9 7 Beginning of the insertion sorting: Unsorted subarray: -4 -3 -2 5 No need to use insertion sorting; Array is already sorted Middle element is -1 Element -2 stays in place, because it less than middle element -1 Element 8 stays in place, because it bigger than middle element -1	

		Element 5 stays in place, because it bigger than middle element -1 Changing elements: 5, -1 Intermediate value(quick sort): -5 -4 -3 -2 -1 5 5 8 9 7 Beginning of the insertion sorting: Unsorted subarray: -2 -1 5 No need to use insertion sorting; Array is already sorted Beginning of the insertion sorting: Unsorted subarray: 5 5 8 No need to use insertion sorting; Array is already sorted Beginning of the insertion sorting: Unsorted subarray: 8 9 7 Changing elements: 9, 7 Intermediate value (insertion): -5 -4 -3 -2 -1 5 5 8 7 9 Changing elements: 8, 7 Intermediate value (insertion): -5 -4 -3 -2 -1 5 5 7 8 9 Sorted array: -5 -4 -3 -2 -1 5 5 7 8 9	
4	4 4 -4 3 2 8	Entered array: -4 3 2 8 Beginning of the insertion sorting: Unsorted subarray: -4 3 2 8 Changing elements: 3, 2 Intermediate value (insertion): -4 2 3 8 Sorted array: -4 2 3 8	При использовании только сортировки вставками (малый подмассив = длинна строки)
5	5 0 5 1 3 -8 16	Entered array: 5 1 3 -8 16 Middle element is 3 Element 16 stays in place, because it bigger than middle element 3	При использовании только быстрой сортировки (малый подмассив = 0)

		Changing elements: 5, -8 Intermediate value(quick sort): -8 1 3 5 16 Element 1 stays in place, because it less than middle element 3 Middle element is 1 Element -8 stays in place, because it less than middle element 1 Element 3 stays in place, because it bigger than middle element 1 Middle element is 5 Element 16 stays in place, because it bigger than middle element 5 Sorted array: -8 1 3 5 16	

Разработанный программный код см. в приложении А.

Выводы.

Были изучены быстрая сортировка и сортировка вставками, а так же реализована программа при помощи языка C++.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл qsort.cpp

```
#include <iostream>
#include <fstream>
using namespace std;


template <class Item>
void read(Item arr[], int n) {    //считывание массива
	for (int i = 0; i < n; i++)
		cin >> arr[i];
}


template <class Item>
void print(Item arr[], int n) {    //печать массива
	for (int i = 0; i < n; i++)
		cout << arr[i] << ' ';
	cout << "\n";
}


template <class Item>
void printArray(Item arr[], int l, int r) {
	for (int i = l; i <= r; i++)
		cout << arr[i] << ' ';
	cout << "\n\n";
}


template <class Item>
void exch(Item& a, Item& b) {   //перестановка элементов
	int tmp;
	tmp = a;
	a = b;
	b = tmp;
}


template <class Item>
void insertionSort(Item arr[], int l, int r, int lenght) {
	bool check = false;
	for (int i = l + 1; i < r + 1; i++) {
		int j = i;
		while (j > l && arr[j - 1] > arr[j]) {
			cout << "Changing elements: " << arr[j - 1] << ", " << arr[j] << endl;
			exch(arr[j - 1], arr[j]);
			j--;
			cout << "Intermediate value (insertion): ";
			print(arr, lenght + 1);
			cout << "\n";
			check = true;
		}
	}
	if (!check)
		cout << "No need to use insertion sorting; Array is already sorted\n\n";
}
```

```
template <class Item>
void quickSort(Item arr[], int l, int r, int M, int lenght) {
	if (r - l <= M) {
		cout << "Beginning of the insertion sorting:\n";
		cout << "Unsorted subarray: ";
		printArray(arr, l, r);
		insertionSort(arr, l, r, lenght);
		return;
	}
	int z = arr[l + (r - l) / 2];  //средний элемент
	cout << "Middle element is " << z << "\n\n";
	int ll = l, rr = r; //первый и последний элементы
	while (ll < rr) {
		while (arr[ll] < z) {
			cout << "Element " << arr[ll] << " stays in place, because it less than
middle element " << z << "\n\n";
			ll++;
		}
		while (arr[rr] > z) {
			cout << "Element " << arr[rr] << " stays in place, because it bigger
than middle element " << z << "\n\n";
			rr--;
		}
		if (ll < rr) {
			cout << "Changing elements: " << arr[ll] << ", " << arr[rr] << endl;
			exch(arr[ll], arr[rr]);
			ll++;
			rr--;
			cout << "Intermediate value(quick sort): ";
			print(arr, lenght + 1);
			cout << "\n";
		}
		else if (ll == rr)
		{
			ll++;
			rr--;
		}
	}
	if (l < rr) quickSort(arr, l, rr + 1, M, lenght);
	if (ll < r) quickSort(arr, ll, r, M, lenght);
}

int main(int argc, char** argv) {
	int n, M;
	int* arr = nullptr;
	if (argc == 2)
	{
		char* file = argv[1];
		ifstream fin(file);// окрываем файл для чтения
		if (fin.is_open())
		{
			fin >> n;
			if (n < 0)
				n = 0;
			fin >> M;
			if (M < 0)
				M = 0;
			arr = new int[n];
			for (int i = 0; i < n; i++)
				fin >> arr[i];
		}
		else
		{
```

```
                std::cout << "Can`t open file" << std::endl;
                return 0;
        }
        fin.close();
    }
    else
    {
        cout << "Enter capacity of array:\n";
        cin >> n;
        if (n < 0)
            n = 0;
        cout << "Enter capacity of small subarray:\n";
        cin >> M;

        if (M < 0)
            M = 0;
        cout << "Enter array:\n";
        arr = new int[n];
        read(arr, n);
    }
    cout << "Entered array:\n";
    print(arr, n);
    cout << "\n";
    quickSort(arr, 0, n - 1, M, n - 1);
    cout << "\nSorted array:\n";
    print(arr, n);
    delete[] arr;
    return 0;
}
```