**МИНОБРНАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ   
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ТУЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Институт прикладной математики и компьютерных наук

Кафедра информационной безопасности

**ЯЗЫКИ ПРОГРАММИРОВАНИЯ**

Отчет по выполнению лабораторной работы № 12  
Вариант №9

Выполнила \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ст. гр.230711 Павлова Виктория Сергеевна

Проверила \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

доц. каф. ИБ Басалова Галина Валерьевна

Тула 2022

# Лабораторная работа №12. Алгоритмы сортировки

## Цель работы:

Изучение и сравнение алгоритмов сортировки данных.

## Задание на работу:

1. Реализовать заданный алгоритм сортировки. Проверить его работу на разных наборах входных данных.

2. Реализовать сортировку данных с помощью функции *sort*.

3. Создать программу по алгоритму:

— Заполняем массив из N целых чисел случайными значениями в диапазоне от 0 до 1000. Выводим.

— Засекаем первый момент времени Т1.

— Упорядочиваем массив алгоритмом, заданным по варианту.

— Засекаем второй момент времени.

— Выводим (для контроля), можно в файл. Для больших значений можно не выводить.

4. Заполняем таблицу.   
*Исследование алгоритмов сортировки*

|  |  |  |
| --- | --- | --- |
| Количество чисел N | Алгоритм по варианту | Стандартная (из библиотеки алгоритмов) |
| 100 | *0,0001 c* |  |
| 1000 |  |  |
| 10 000 |  |  |
| 100 000 |  |  |
| 1 000 000 |  |  |

## Ход работы:

**Вариант сортировки №9. Быстрая сортировка**

Этот алгоритм состоит из трёх шагов. Сначала из массива нужно выбрать один элемент — его обычно называют опорным. Затем другие элементы в массиве перераспределяют так, чтобы элементы меньше опорного оказались до него, а большие или равные — после. А дальше рекурсивно применяют первые два шага к подмассивам справа и слева от опорного значения.

В качестве опорного элемента возьмём средний (middle), его индекс будет целым от деления размера массива пополам. Далее пройдёмся по индексам левее, пока не будет найден неподходящий (больший опорного) элемент. Аналогично для правого, только логическое ограничение – пока не будет встречен меньший элемент. Теперь для упорядочивания достаточно поменять местами элементы на полученных индексах, на каждой итерации уменьшая правый и увеличивая левый соответственно. На случай, если останутся неупорядоченные элементы, необходимо вызвать функцию сортировки рекурсивно для оставшихся индексов.

Код функции для сортировки QuickSort:

int\* QuickSort(int\* a, int size)

{

int i = 0, j = size - 1;

int middleIndex = size / 2, middle = a[middleIndex];

do {

while (a[i] < middle) {

i++;

}

while (a[j] > middle) {

j--;

}

if (i <= j) {

swap(a[i], a[j]);

i++;

j--;

}

} while (i <= j);

if (j > 0) {

QuickSort(a, j + 1);

}

if (i < size) {

QuickSort(&a[i], size - i);

}

return a;

}

*Таблица 1. Исследование алгоритмов сортировки*

|  |  |  |
| --- | --- | --- |
| Количество чисел N | Алгоритм по варианту  (QuickSort) | Стандартная (из библиотеки алгоритмов) |
| 100 | *0 c* | *0 с* |
| 1000 | *0 c* | *0.001 c* |
| 10 000 | *0.002 c* | *0.003 c* |
| 100 000 | *0.016 c* | *0.024 c* |
| 1 000 000 | *0.206 c* | *0.247 c* |

## Выводы:

Алгоритм сортировки QuickSort, основанный на использовании опорного элемента, в среднем работает незначительно быстрее, чем стандартная функция sort() из библиотеки STL. Сложность данного алгоритма сортировки по времени в среднем равна O(, в худшем – О(). Согласно документации стандартной библиотеки C++, средняя сложность по памяти функции sort() также примерно равна O(, что делает данные сортировки приблизительно одинаковыми по размеру используемой памяти и времени, а значит применение любой из них целесообразно.