Московский технический университет связи и информатики

Кафедра «Информатика»

Лабораторная работа по дисциплине «Теория и методы программирования» № 3

Тема: «*РАЗРАБОТКА И ИССЛЕДОВАНИЕ АЛГОРИТМОВ СЛОЖНОЙ РЕКУРСИИ НА ПРИМЕРЕ СОРТИРОВКИ ДАННЫХ»*

Выполнил:

Блёсткин Михаил

Группа БИБ2203

Вариант 6

Москва 2023 г.

**Цель:**

Изучить рекурсивный метод Хоара на примере быстрой сортировки массивов данных. Написать программу и исследовать работу алгоритма на различных наборах данных.

**Постановка задачи:**

1. Разработать программу быстрой сортировки массива определенного типа данных short int с рекурсией и без нее.
2. Отладить правильность работы сортировок на массивах с количеством элементов N=50 сгенерированные датчиком случайных чисел в диапазоне [10060, 30706].
3. Для сравнения двух методов сортировки составить таблицу (см. табл.3) следующего вида (данные получить экспериментально) для N=1000, 10000, 100000, 500000, 1000000. (N – количество элементов в массиве). Для этого в программе обязательно вставляется точки для замера времени для выдачи времени выполнения алгоритма.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Количество элементов | 1000 | 10000 | 100000 | 500000 | 1000000 |
| Время выполнения алгоритма сортировки (в миллисекундах) | | | | |
| Без рекурсии | 0.873 | 4.577 | 21.78 | 103.63 | 210.74 |
| С рекурсией | 0.849 | 2.844 | 18.91 | 102.15 | 210.68 |

1. Работу оформить в виде отчета. Отчет по работе должен соответствовать следующей структуре.

Титульный лист.

Постановка задачи.

Алгоритм решения задачи.

Листинг программы.

Контрольный тест. Результаты работы программы заполненной таблицей.

Выводы по работе.

**Алгоритм:**

### Быстрая сортировка

Быстрая сортировка — это алгоритм типа “разделяй и властвуй”.

**Общая идея алгоритма:**

* Выбрать из массива элемент, который обычно называют опорным. Это может быть любой элемент из массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.
* Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».
* Рекурсивно применить первые два шага к отрезкам, содержащим «меньшие» и «большие» значения. Не применять к массиву, в котором только один элемент или отсутствуют элементы.

**Теперь рассмотрим метод Хоара.** Метод Хоара — это метод признан одним из лучших методов сортировки, которые когда-либо придумали. В методе Хоара первоначально выделяют базовый элемент, относительно которого ключи с большим весом перебрасываются вправо, а с меньшим влево. Базовый элемент сравнивается с противоположным элементом. В качестве базового элемента очень удобно брать крайние элементы.

В качестве примера возьмем массив исходных данных:

6 3 5 1 9 8 7 10

Дерево данной процедуры на конкретном примере выглядит следующим образом:

6

3 5 1 9 8 7 10

1 5 7 8 10

8

Можно выбирать разделяющий элемент на каждом шаге случайным образом, тогда задача будет немного сложнее, алгоритм изменится:

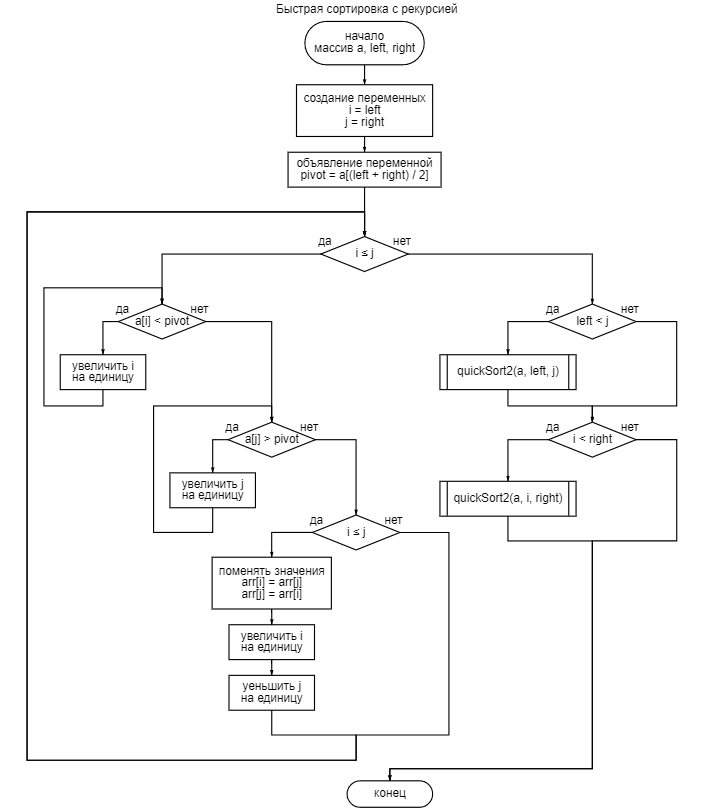
Элементы массива набрать случайным образом. Выбрать, так же случайным образом, некоторый элемент массива x, после чего массив просматривается слева, пока не встретится элемент a[i] такой, что a[i] > x, а затем массив просматривается справа, пока не встретится элемент a[j] такой, что a[j] < x. Эти два элемента меняются местами, и процесс просмотра, сравнения и обмена продолжается, пока мы не дойдем до элемента x. Массив окажется разбитым на две части - левую, в которой значения элементов будут меньше x, и правую со значениями элементов, большими x. Далее процесс рекурсивно продолжается для левой и правой частей массива до тех пор, пока каждая часть не будет содержать в точности один элемент. Пример сортировки стандартного массива приведен в таблице:

|  |  |
| --- | --- |
| Начальное состояние массива | 8 23 5 65 |44| 33 1 6 |
| Шаг 1 (в качестве x выбирается a[5]) | |--------|  8 23 5 6 44 33 1 65  |---|  8 23 5 6 1 33 44 65 |
| Шаг 2 (в подмассиве a[1], a[5] в качестве x выбирается a[3]) | 8 23 |5| 6 1 33 44 65  |--------|  1 23 5 6 8 33 44 65  |--|  1 5 23 6 8 33 44 65 |
| Шаг 3 (в подмассиве a[3], a[5] в качестве x выбирается a[4]) | 1 5 23 |6| 8 33 44 65  |----|  1 5 8 6 23 33 44 65 |
| Шаг 4 (в подмассиве a[3], a[4] выбирается a[4]) | 1 5 8 |6| 23 33 44 65  |--|  1 5 6 8 23 33 44 65 |

**Быстрая сортировка без рекурсии:**

Вместо того, чтобы вызывать функцию, будем сохранять в стек крайнее левое и правое значения. Можно сохранять сразу пару значений, но мы вместо этого сделаем два параллельных стека. В первый будем класть крайнее левое значение для следующего вызова, а во второй - крайнее правое. Цикл заканчивается, когда стеки становятся пустыми.

Для выполнения быстрой сортировки воспользуемся стеком, в котором в виде сортируемых подмассивов содержится перечень действий, которые предстоит выполнить. Каждый раз, когда возникает необходимость в обработке подмассива, он выталкивается из стека. После разделения массива получаются два подмассива, требующих дальнейшей обработки, которые и заталкиваются в стек. Нерекурсивная реализация использует стек, заменяя рекурсивные вызовы помещением в стек параметров функции, а вызовы процедур и выходы из них — циклом, который осуществляет выборку параметров из стека и их обработку, пока стек не пуст.



**Листинг программы:**

Sort.h:

#pragma once

#include <iostream>

#include <vector>

using namespace std;

const int MIN\_RANGE = 1120120, MAX\_RANGE = 3127012;

// Быстрая сортировка без рекурсии

void quickSort1(vector<short int>& arr, int size) {

int i, j;

int lb, ub; // границы сортируемого в цикле фрагмента

int lbstack[2048], ubstack[2048];

int stackpos = 1; // текущая позиция стека

int ppos; // середина массива

int pivot; // опорный элемент

lbstack[1] = 0;

ubstack[1] = size;

do {

lb = lbstack[stackpos];

ub = ubstack[stackpos];

stackpos--;

do {

ppos = (lb + ub) >> 1;

i = lb; j = ub; pivot = arr[ppos];

while (i <= j) {

while (arr[i] < pivot) i++;

while (arr[j] > pivot) j--;

if (i <= j)

{

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

i++;

j--;

}

}

if (i < ppos)

{

if (i < ub)

{

stackpos++;

lbstack[stackpos] = i;

ubstack[stackpos] = ub;

}

ub = j;

}

else

{

if (j > lb) {

stackpos++;

lbstack[stackpos] = lb;

ubstack[stackpos] = j;

}

lb = i;

}

} while (lb < ub);

} while (stackpos != 0);

}

// Быстрая сортировка с рекурсией

void quickSort2(vector<short int>& arr, int left, int right)

{

int i = left, j = right;

int pivot = arr[(left + right) / 2];

while (i <= j)

{

while (arr[i] < pivot) i++;

while (arr[j] > pivot) j--;

if (i <= j)

{

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

i++;

j--;

}

}

if (left < j)

quickSort2(arr, left, j);

if (i < right)

quickSort2(arr, i, right);

}

// генерация рандомного числа в заданном диапозоне чисел

int rand\_number()

{

return rand() % (MAX\_RANGE - MIN\_RANGE + 1) + MIN\_RANGE;

}

//ввод массива

vector<short int> input\_arr(int n)

{

vector<short int> arr(n);

for (int i = 0; i < n; i++)

arr[i] = rand\_number();

return arr;

}

// вывод массива

void output\_arr(vector<short int> arr, int n)

{

for (int i = 0; i < n; i++)

{

cout << arr[i] << " ";

}

cout << "\n\n";

}

Time.h:

#pragma once

#pragma once

#pragma once

#include <iostream>

#include <algorithm>

#include <iostream>

#include <chrono>

#include <windows.h>

using namespace std;

using cl = std::chrono::steady\_clock::time\_point;

struct chronos

{

std::chrono::steady\_clock::time\_point starthere()

{

auto start = chrono::steady\_clock::now();

return start;

}

std::chrono::steady\_clock::time\_point endhere()

{

auto end = chrono::steady\_clock::now();

return end;

}

void gettimemyfunction(cl start, cl end)

{

setlocale(LC\_ALL, "russion");

cout << "Прошедшее время в наносекундах: "

<< chrono::duration\_cast<chrono::nanoseconds>(end - start).count()

<< " ns" << endl;

cout << "Прошедшее время в микросекундах: "

<< chrono::duration\_cast<chrono::microseconds>(end - start).count()

<< " micros" << endl;

cout << "Прошедшее время в миллисекундах: "

<< chrono::duration\_cast<chrono::milliseconds>(end - start).count()

<< " ms" << endl;

cout << "Прошедшее время в секундах: "

<< chrono::duration\_cast<chrono::seconds>(end - start).count()

<< " sec\n\n";

}

};

main.ccp:

#include "Sort.h"

#include "Time.h"

#include <iostream>

#include <vector>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

system("chcp 1251");

int povt, n, choice;

do

{

cout << "Введите кол-во элементов в массиве: ";

cin >> n;

cout << "\n";

vector<short int> arr;

arr = input\_arr(n);

cout << "Cгенерированный массив: \n\n";

output\_arr(arr, n);

cout << "\n\n";

cout << "Метод сортировки:";

cout << "\n 1.Быстрая сортировка без рекурсии\n 2.Быстрая сортировка с рекурсией\n 3.Обе сортировки\n";

cin >> choice;

while (choice < 1 && choice > 3)

{

cout << "Ошибка, повторите \n Ввод: ";

cin >> choice;

}

if (choice == 1)

{

chronos times;

auto start = times.starthere();

quickSort1(arr, n - 1);

cout << "Отсортированный массив: \n\n";

output\_arr(arr, n);

auto end = times.endhere();

times.gettimemyfunction(start, end);

arr.clear();

}

else if (choice == 2)

{

chronos times;

auto start = times.starthere();

quickSort2(arr, 0, n - 1);

cout << "Отсортированный массив: \n\n";

output\_arr(arr, n);

auto end = times.endhere();

times.gettimemyfunction(start, end);

arr.clear();

}

else if (choice == 3)

{

vector<short int> arr2 = arr;

chronos times;

auto start = times.starthere();

quickSort1(arr, n - 1);

cout << "Отсортированный массив: \n\n";

output\_arr(arr, n);

auto end = times.endhere();

times.gettimemyfunction(start, end);

arr.clear();

start = times.starthere();

quickSort2(arr2, 0, n - 1);

cout << "Отсортированный массив: \n\n";

output\_arr(arr2, n);

end = times.endhere();

times.gettimemyfunction(start, end);

arr2.clear();

}

else

{

cout << "Ошибка!\n";

}

cout << "Повторить?(1/0) ";

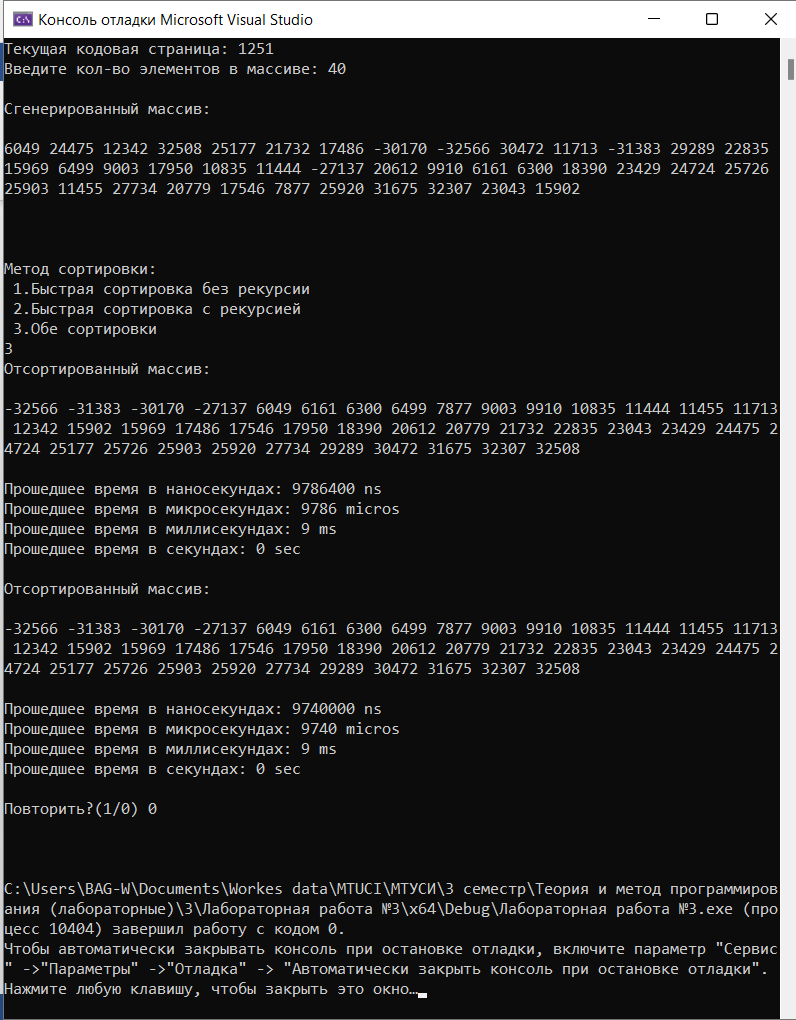
cin >> povt;

cout << "\n\n";

} while (povt == 1);

}

**Контрольный тест**

****

**Вывод**

Я изучил рекурсивный метод Хоара на примере быстрой сортировки массивов данных, написал программу и исследовал работу алгоритма на различных наборах данных.