**Минобрнауки России**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«лэти» им. В.и. Ульянова (Ленина)**

**Кафедра сапр**

**ОТчёт**

**по лабораторной работе №3**

**по дисциплине: «Алгоритмы и структуры данных»**

**тема: «Алгоритмы сортировки и поиска»**

Студентка гр. 1302 Королев П.С.

Преподаватель Родионова Е.А.

**Санкт-Петербург**

**2022**

**Цель работы**

Необходимо реализовать следующие алгоритмы сортировки:

1. Сортировка вставками (Insertion sorting)

2. Сортировка выбором (Selection sorting)

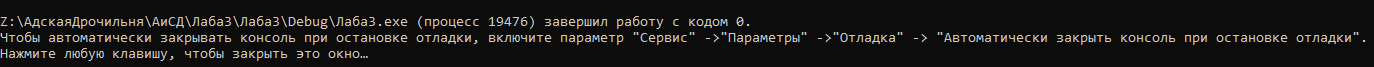
3. Пузырьковая сортировка (Bubble sorting)

4. Сортировка слиянием (Merge sorting)

5. Сортировка Шелла (Shell sorting)

6. Быстрая сортировка (Quick sorting)

**Пример работы программы**



**Оценка временной сложности алгоритмов**

Таблица временных асимптотических сложностей

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Временная асимптотическая сложность | | | Пространственная сложность |
| лучший | худший | средний |
| Сортировка вставками | O(n) | O(n2) | O(n2) | O(1) |
| Сортировка выбором | O(n2) | O(n2) | O(n2) | O(1) |
| Пузырьковая сортировка | O(n) | O(n2) | O(n2) | O(1) |
| Сортировка слиянием | O(nlogn) | O(nlogn) | O(nlogn) | O(n) |
| Сортировка Шелла | O(nlogn) | O(n2) | O(nlogn) | O(1) |
| Быстрая сортировка | O(n) | O(n2) | O(nlogn) | O(n) |

(О — большое) — верхняя граница, в то время как (Омега — большое) — нижняя граница. Тета требует как (О — большое), так и (Омега — большое), поэтому она является точной оценкой (она должна быть ограничена как сверху, так и снизу). К примеру, алгоритм требующий Ω (n logn) требует не менее n logn времени, но верхняя граница не известна. Алгоритм требующий Θ (n logn) предпочтительнее потому, что он требует не менее n logn (Ω (n logn)) и не более чем n logn (O(n logn)).

f(x)=Θ(g(n)) означает, что f растет так же как и g когда n стремится к бесконечности. Другими словами, скорость роста f(x) асимптотически пропорциональна скорости роста g(n).

f(x)=O(g(n)). Здесь темпы роста не быстрее, чем g (n). O большое является наиболее полезной, поскольку представляет наихудший случай.

**Листинг**

#include <iostream>

#include <conio.h>

using namespace std;

const int nmax = 15;

void InsertionSorting(double\* data, int& size)

{

int j = 0;

double curr = 0;

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

{

j = i - 1;

curr = data[i];

while (data[j] > curr and j >= 0)

{

data[j + 1] = data[j];

j--;

}

data[j + 1] = curr;

}

}

void SelectionSorting(double\* data, int& size)

{

double MaxD = 0;

int temp = 0;

double tempV = 0;

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

{

for (int j = 0; j < size - i; j++)

{

if (MaxD < data[j])

{

MaxD = data[j];

temp = j;

}

}

tempV = data[size - i - 1];

data[size - i - 1] = MaxD;

data[temp] = tempV;

MaxD = 0;

temp = 0;

}

}

void BubbleSorting(double\* data, int& size)

{

double temp = 0;

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

{

for (int j = 0; j < size - i - 1; j++)

{

if (data[j] > data[j + 1])

temp = data[j]; data[j] = data[j + 1]; data[j + 1] = temp;

}

}

}

void MergeSorting(double\* arr, int low, int high, int mid)

{

int i, j, k, c[50];

i = low;

k = low;

j = mid + 1;

while (i <= mid && j <= high)

{

if (arr[i] < arr[j])

{

c[k] = arr[i];

k++;

i++;

}

else

{

c[k] = arr[j];

k++;

j++;

}

}

while (i <= mid)

{

c[k] = arr[i];

k++;

i++;

}

while (j <= high)

{

c[k] = arr[j];

k++;

j++;

}

for (i = low; i < k; i++)

arr[i] = c[i];

}

void MergeSorting(double\* arr, int low, int high)

{

int mid;

if (low < high)

{

mid = (low + high) / 2;

MergeSorting(arr, low, mid);

MergeSorting(arr, mid + 1, high);

MergeSorting(arr, low, high, mid);

}

}

void ShellSorting(double\* data, int n)

{

int i, j, d, count;

d = n;

d = d / 2;

while (d > 0)

{

for (i = 0; i < n - d; i++)

{

j = i;

while (j >= 0 && data[j] > data[j + d])

{

count = data[j];

data[j] = data[j + d];

data[j + d] = count;

j--;

}

}

d = d / 2;

}

}

void QSorting(double\* mas, int size)

{

int i = 0;

int j = size - 1;

int mid = mas[size / 2];

do {

while (mas[i] < mid)

i++;

while (mas[j] > mid)

j--;

if (i <= j)

{

int tmp = mas[i];

mas[i] = mas[j];

mas[j] = tmp;

i++;

j--;

}

}

while (i <= j);

if (j > 0)

QSorting(mas, j + 1);

if (i < size)

QSorting(&mas[i], size - i);

}

int main()

{

double data[11] = { 16,2,8,7,3,32,1,2,5,6,7 };

int size = (sizeof(data) / sizeof(\*data));

}

**Вывод**

В процессе написания программы, я познакомился с новыми видами сортировок, вспомнил те, которые уже знал и реализовал их работу.