Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа**

**«Сортировки»**

Выполнил:

студент группы РИС-23-3б

Коротаев Александр Дмитриевич

Проверила:

доцент кафедры ИТАС

О.А. Полякова

2024 г.

**Разработка алгоритма**

**Постановка задачи:**

Реализовать сортировки: блочную, подсчетом, Хоара, слиянием, натуральным слиянием, Шелла, многофазную, быструю.

**Анализ задачи:**

1. Создадим главный файл для основного кода**.**
2. Для удобства все сортировки будут находиться в отдельных файлах
3. Для использования сортировок в главном файле будем подключать файлы с кодом сортировок для дальнейшего использования

Сортировка слиянием

void Merge(int\* mas, int startIndex, int endIndex)

{

vector<int> temp;

int start\_first\_part = startIndex;

int end\_first\_part = (startIndex + endIndex) / 2;

int start\_second\_part = end\_first\_part;

while (int(temp.size()) < endIndex - startIndex)

{

if (start\_first\_part >= end\_first\_part

|| (start\_second\_part < endIndex

&& mas[start\_second\_part] <= mas[start\_first\_part]))

{

temp.push\_back(mas[start\_second\_part]);

start\_second\_part++;

}

else

{

temp.push\_back(mas[start\_first\_part]);

start\_first\_part++;

}

}

for (int i = startIndex; i < endIndex; ++i)

mas[i] = temp[i - startIndex];

}

void MergeSort(int\* mas, int startIndex, int endIndex)

{

if ((endIndex - startIndex) < 2)

{

return;

}

if ((endIndex - startIndex) == 2)

{

if (mas[startIndex] > mas[startIndex + 1])

{

swap(mas[startIndex], mas[startIndex + 1]);

return;

}

}

MergeSort(mas, startIndex, (startIndex + endIndex) / 2);

MergeSort(mas, (startIndex + endIndex) / 2, endIndex);

Merge(mas, startIndex, endIndex);

# }

**Сортировка подсчетом**

void CountingSort(int\* mas)

{

int max\_element = mas[0];

int min\_element = mas[0];

for (int i = 0; i < int(lenght\_mas); i++)

{

if (max\_element < mas[i]) max\_element = mas[i];

else if (min\_element > mas[i]) min\_element = mas[i];

}

vector<int> temp(max\_element - min\_element + 1);

for (int i = 0; i < int(lenght\_mas); i++)

{

temp[mas[i] - min\_element]++;

}

int index = 0;

for (int i = 0; i < max\_element - min\_element + 1; i++)

{

for (int step = 0; step < temp[i]; step++)

{

mas[index] = i + min\_element;

index++;

}

}

}

**Блочная сортировка**

void BucketSort(int\* mas) {

int max\_element = mas[0];

int min\_element = mas[0];

for (int i = 1; i < lenght\_mas; i++)

{

if (mas[i] > max\_element)

{

max\_element = mas[i];

}

else if (mas[i] < min\_element)

{

min\_element = mas[i];

}

}

int count\_buckets = ((max\_element - min\_element) / 10);

int\*\* buckets = new int\* [count\_buckets];

for (int i = 0; i < count\_buckets; i++)

{

buckets[i] = new int[lenght\_mas];

}

for (int i = 0; i < lenght\_mas; i++)

{

int bucket\_index = ((mas[i] - min\_element) / count\_buckets);

buckets[bucket\_index][i] = mas[i];

}

for (int i = 0; i < count\_buckets; i++)

{

cout << i + 1 << "-й блок: ";

for (int j = 0; j < lenght\_mas; j++)

{

if (buckets[i][j] > -1)

cout << buckets[i][j] << ' ';

}

cout << endl;

}

for (int i = 0; i < count\_buckets; i++)

{

for (int j = 0; j < lenght\_mas; j++)

{

for (int k = j + 1; k < lenght\_mas; k++)

{

if (buckets[i][j] > buckets[i][k])

{

swap(buckets[i][j], buckets[i][k]);

}

}

}

}

int counter = 0;

for (int i = 0; i < count\_buckets; i++)

{

for (int j = 0; j < lenght\_mas; j++)

{

if (buckets[i][j] > -100) {

mas[counter++] = buckets[i][j];

}

}

}

for (int i = 0; i < count\_buckets; i++)

{

delete[] buckets[i];

}

delete[] buckets;

}

**Быстрая сортировка**

void FastSort(int\* mas, int size)

{

int i = 0;

int j = size - 1;

int pivot = mas[size / 2];

do

{

while (mas[i] < pivot)

{

i++;

}

while (mas[j] > pivot)

{

j--;

}

if (i <= j)

{

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

i++;

j--;

}

} while (i <= j);

if (j > 0)

{

FastSort(mas, j + 1);

}

if (i < size)

{

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

}

}

**Сортировка Шелла**

void ShellSort(int\* mas)

{

int j, tmp;

bool flag;

for (int step = 5; step > 0; step /= 2)

{

flag = true;

for (int i = step; i < lenght\_mas; i++)

{

tmp = mas[i];

for (j = i; j >= step && flag; j -= step)

{

if (tmp < mas[j - step])

{

mas[j] = mas[j - step];

}

else

{

break;

}

}

mas[j] = tmp;

}

}

}

**Сортировка Хоара**

void KhoarSort(int\* mas, int left, int right)

{

int i, j, pivot;

i = left;

j = right;

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

do

{

while ((mas[i] < pivot) && (i < right))

{

i++;

}

while ((pivot < mas[j]) && (j > left))

{

j--;

}

if (i <= j)

{

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

i++;

j--;

}

} while (i <= j);

if (left < j)

{

KhoarSort(mas, left, j);

}

if (i < right)

{

KhoarSort(mas, i, right);

}

}

**Сортировка естественным слиянием**

void natural\_merge(vector<int>& vect1, vector<int>& vect2)

{

for (int i = 0; i < vect2.size(); i++)

{

vect1.push\_back(vect2[i]);

for (int j = vect1.size() - 1; j > 0; j--)

{

if (vect1[j] < vect1[j - 1])

{

swap(vect1[j], vect1[j - 1]);

}

else

{

break;

}

}

}

}

void natural\_merge\_sort(int\* mas)

{

vector <vector<int>> series;

vector<int> new\_series;

for (int i = 0; i <= lenght\_mas - 1; i++)

{

if (i == lenght\_mas - 1)

{

new\_series.push\_back(mas[i]);

series.push\_back(new\_series);

new\_series.clear();

}

else

{

new\_series.push\_back(mas[i]);

if (mas[i] > mas[i + 1])

{

series.push\_back(new\_series);

new\_series.clear();

}

}

}

while (series.size() != 1)

{

natural\_merge(series[0], series[series.size() - 1]);

series.pop\_back();

}

for (int i = 0; i < lenght\_mas; i++)

{

mas[i] = series[0][i];

}

}

**Сортировка двухфазным слиянием**

void natural\_merge(vector<int>& vect1, vector<int>& vect2)

{

for (int i = 0; i < vect2.size(); i++)

{

vect1.push\_back(vect2[i]);

for (int j = vect1.size() - 1; j > 0; j--)

{

if (vect1[j] < vect1[j - 1])

{

swap(vect1[j], vect1[j - 1]);

}

else

{

break;

}

}

}

}

pair<int, int> ideal\_series(pair<int, int> last\_count\_series, int count\_all\_series)

{

int current\_num = last\_count\_series.first + last\_count\_series.second;

if (count\_all\_series > current\_num)

{

ideal\_series({ last\_count\_series.second, current\_num }, count\_all\_series);

}

else

{

return last\_count\_series;

}

}

void series\_merge(vector <vector<int>>& series\_1, vector <vector<int>>& series\_2, vector <vector<int>>& series\_3)

{

if (series\_1.size() == 1 && series\_2.size() == 0 && series\_3.size() == 0)

{

return;

}

else if (series\_2.size() == 1 && series\_1.size() == 0 && series\_3.size() == 0)

{

return;

}

else if (series\_3.size() == 1 && series\_1.size() == 0 && series\_2.size() == 0)

{

return;

}

int i = series\_1.size()-1;

int j = series\_2.size()-1;

while (j != -1) //пока series\_2 не пуста

{

natural\_merge(series\_1[i], series\_2[j]);

series\_3.push\_back(series\_1[i]);

series\_1.pop\_back();

series\_2.pop\_back();

i--;

j--;

}

series\_merge(series\_3, series\_1, series\_2); //1-ый, там, где больше серий, 2-ой, там где меньше серий (потом пустой), 3-ий, куда сливаются

}

void multiphase\_merge\_sort(int\* mas)

{

pair <int, int> ideal\_pair;

vector <int> new\_series;

vector <vector<int>> series, series\_1, series\_2, series\_3;

for (int i = 0; i <= lenght\_mas - 1; i++)

{

if (i == lenght\_mas - 1)

{

new\_series.push\_back(mas[i]);

series.push\_back(new\_series);

new\_series.clear();

}

else

{

new\_series.push\_back(mas[i]);

if (mas[i] > mas[i + 1])

{

series.push\_back(new\_series);

new\_series.clear();

}

}

}

ideal\_pair = ideal\_series({ 0, 1 }, series.size());

for (int i = 0; i < ideal\_pair.second; i++)

{

series\_1.push\_back(series[i]);

}

for (int i = 0; i < ideal\_pair.first; i++)

{

if (series\_1.size() + i + 1 > series.size())

{

series\_2.push\_back({});

}

else

{

series\_2.push\_back(series[series\_1.size() + i]);

}

}

series\_merge(series\_1, series\_2, series\_3);

for (int i = 0; i < lenght\_mas; i++)

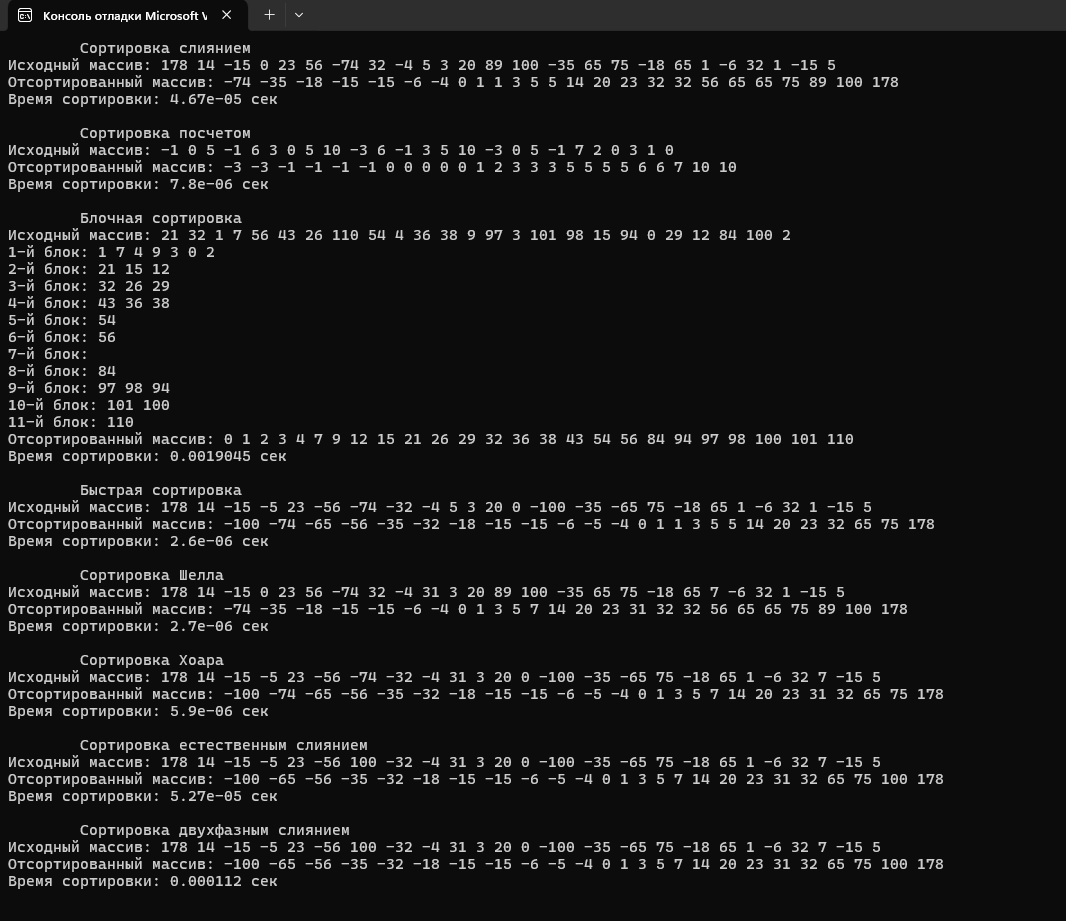
{

mas[i] = series\_1[0][i];

}

}

**Пример решения**

****

**GitHub:** <https://github.com/Korovay4ik/Laboratory-works>