МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

КАФЕДРА ИИТ

ОТЧЁТ

по лабораторной работе №2

**«Алгоритмы сортировки»**

Выполнил:

Студент 1 курса

группы ПО-9

Мисиюк Алексей Сергеевич

Проверила:

Войцехович О. Ю.

Брест 2022

**Порядок выполнения работы**

**Вариант 15**

*Задание*

Реализовать 3 алгоритма сортировки и провести экспериментальное исследование их эффективности:

* cортировка подсчетом
* cортировка вставками
* cортировка Хоара

Необходимо измерить время работы каждого алгоритма при различном количестве элементов в массиве и заполнить таблицу для каждого алгоритма. По заполненной таблице построить для каждого алгоритма график зависимости времени его выполнения от числа элементов в массиве.

По результатам экспериментов определить какой алгоритм работает быстрее и почему.

*Описание алгоритмов:*

/\*\* Variant 15

\* Sort by inserts

\*

\* for i = 2 to n do

\* x = A[i]

\* j = i

\* while (int j > 1 and A[j-1] > x) do

\* A[j] = A[j-1]

\* j = j - 1

\* end while

\* A[j] = x

\* end for

\*/

void sortByInserts(int \*A, int n) {

for (int i=1; i<n; i++) {

int x = A[i];

int j = i;

while (j > 0 && A[j-1] > x) {

A[j] = A[j-1];

j--;

}

A[j] = x;

}

}

Алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

Скорость: O(n2)

(лучший случай – отсортированный массив, O(n) )

Память: O(n)

/\*\* Variant 15

\* Sort by count

\*

\* SortByCount:

\* for i = 0 to k

\* C[i] = 0;

\* for i = 0 to n - 1

\* C[A[i]] = C[A[i]] + 1;

\* b = 0;

\* for j = 0 to k

\* for i = 0 to C[j] - 1

\* A[b] = j;

\* b = b + 1;

\*/

void sortByCount(int \*A, int n, int k) {

int C[k];

for (int i=0; i<k; i++) C[i] = 0;

for (int i=0; i<n; i++) C[A[i]]++;

int b=0;

for (int j=0; j<k; j++) {

for (int i=0; i<C[j]; i++) {

A[b] = j;

b++;

}

}

}

Применение сортировки подсчётом целесообразно лишь тогда, когда сортируемые числа имеют (или их можно отобразить в) диапазон возможных значений, который достаточно мал по сравнению с сортируемым множеством

Скорость: O(n+k)

Память: O(k),

где 1..k – диапазон чисел массива

Пример реализации:

/\*\* Variant 15

\* Sort Xoara

\*

\* quicksort(A, low, high) is

\* if low < high then

\* p := pivot(A, low, high)

\* left, right := partition(A, p, low, high)

\* quicksort(A, low, left)

\* quicksort(A, right, high)

\*/

void XoarSort(int\* a, int first, int last) {

int i = first, j = last;

int tmp, x = a[(first + last) / 2];

do {

while (a[i] < x) i++;

while (a[j] > x) j--;

if (i <= j) {

if (i < j) {

tmp=a[i];

a[i]=a[j];

a[j]=tmp;

}

i++; j--;

}

} while (i <= j);

if (i < last)

XoarSort(a, i, last);

if (first < j)

XoarSort(a, first,j);

}

Один из самых быстрых известных универсальных алгоритмов сортировки массивов. Обычно к нему применяются модификации.

Скорость: O(n2) (худший случай, при неудачных данных)

(средний случай O( n*log*n ) )

Память: O(n)

#include <iostream>

#include <ctime>

using namespace std;

void sortByInserts(int \*A, int n);

int main()

{

int n;

cin >> n;

int \*ArrayOfRandomInts = new int[n];

// pceudo random ints

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

ArrayOfRandomInts[i] = rand() \*

(rand() % 3 + 1);

}

unsigned int time = clock();

//для остальных сортировок аналогично

sortByInserts(ArrayOfRandomInts, n);

time = clock() - time;

cout << "Time for sort: " << time/1000.

<< "s" << endl;

delete ArrayOfRandomInts;

return 0;

}

void sortByInserts(int \*A, int n) {

for (int i=1; i<n; i++) {

int x = A[i];

int j = i;

while (j > 0 && A[j-1] > x) {

A[j] = A[j-1];

j--;

}

A[j] = x;

}

}

*Организация эксперементов*

* Процессор: AMD Ryzen 5 2600 Six-Core Processor 3.40 GHz
* Объем памяти: 8,00 ГБ
* Версия ОС: Windows 10 (64-разрядная операционная система, процессор x64)

*Результаты экспериментов*

*Таблица 1*

|  |  |  |  |
| --- | --- | --- | --- |
| n | Подсчётом, мс | Вставками, мс | Хоара, мс |
| 50000 | 1 | 2066 | 6 |
| 100000 | 2 | 8156 | 13 |
| 150000 | 3 | 23583 | 25 |
| 200000 | 3 | 51736 | 34 |
| 250000 | 4 |  | 36 |
| 300000 | 3 |  | 51 |
| 350000 | 4 |  | 46 |
| 400000 | 4 |  | 60 |
| 450000 | 5 |  | 71 |
| 500000 | 6 |  | 75 |
| 550000 | 7 |  | 77 |
| 600000 | 6 |  | 94 |
| 650000 | 6 |  | 95 |
| 700000 | 5 |  | 119 |
| 750000 | 7 |  | 140 |
| 800000 | 7 |  | 110 |
| 850000 | 8 |  | 140 |
| 900000 | 8 |  | 145 |
| 950000 | 10 |  | 155 |
| 1000000 | 9 |  | 126 |

*График 1 – зависимость времени выполнения алгоритмов от объема исходных данных.*

**Вывод:** реализовали алгоритмы сортировок подсчетом, вставками и Хоара. Проведена экспериментальная оценка, в следствии которой видно, что сортировка подсчетом наиболее эффективная по скорости, хотя и имеет ограниченый спектр применений.