*Государственное образовательное учреждение высшего профессионального образования*

*«Московский Государственный Технический Университет имени Н. Э. Баумана»*

**ОТЧЕТ**

По лабораторной работе №2

По курсу «Анализ алгоритмов»

Тема: «Сложность нерекурсивных сортировок»

Студент: Иванов И. В.

Группа: ИУ7-53

Преподаватель: Волкова Л. Л.

Москва, 2014

**Алгоритм сортировки пузырьком**

Код функции:

void SortBubble(int\* original, int arrsize, int \*arr)

{

memcpy( arr, original, sizeof(int)\*arrsize );

int t;

for(int i = 1; i < arrsize; i++) // F: 2 + (N-1)\*(2 + F{})

for(int j = 0; j < arrsize - i; j++) // F: 3 + m\*(3 + F{})

if (arr[j] > arr[j + 1]) // F: 4

{

t = arr[j+1]; // F: 3

arr[j+1] = arr[j]; // F: 4

arr[j] = t; // F: 2

}

}

Пусть массив содержит N элементов. В цикле по j F = **3** + m\*(**3** + F), поскольку происходит сравнение < N-i вместо < N – дополнительная операция вычитания.

Без учета затрат на копирование элементов, трудоёмкость сортировки пузырьком:  
 .

**Алгоритм сортировки вставками**

Код функции:

void SortInsert(int\* original, int arrsize, int \*arr)

{

if ( arrsize < 1 )

throw "Array size < 1!\n";

memcpy( arr, original, sizeof(int)\*arrsize );

int t;

for (int i = 1; i < arrsize; i++) // F: 2 + (N-1)(2 + 2/\*2 for\*/)

for (int j = i; j > 0; j--) // F: **m**(2 + F{}) //m = 1+2+…+N-1=N^2/2

if ( arr[j] < arr[j-1] ) // F: 4

{

t = arr[j-1]; // F: 3

arr[j-1] = arr[j]; // F: 4

arr[j] = t; // F: 2

}

}

Без учета затрат на копирование элементов, трудоёмкость сортировки вставками:

*.*

**Алгоритм сортировки индексами**

Код функции:

void SortIndex(int\* original, int arrsize, int \*arr)

{

if ( arrsize < 1 )

throw "Array size < 1!\n";

int max = Index\_FindMax(original, arrsize); // 2 + (N-1)(5 + [0 1])

int min = Index\_FindMin(original, arrsize);

int diff = max - min + 1;

int \*helper = new int [diff];

memset( helper, 0, sizeof(int) \* diff );

//заполняем дополнительный массив по индексам

for (int i = 0; i < arrsize; i++) // F: 2 + N(2 + F{})

{

int index = original[i] - min; // F: 3

++helper[index]; // F: 2

}

//вытаскиваем значения из допмассива в результирующий

int k = 0;

int cur;

for (int i = 0; i < diff; i++) // F: 2 + diff(2 + F{})

{

cur = helper[i]; // F: 2

while (cur > 0) // F: 1

{

arr[k] = i + min; // F: 3

cur--; k++; // F: 2

}

}

delete [] helper;

}

Без учета затрат на выделение\удаление вспомогательного массива, трудоёмкость сортировки индексами имеет значение: ;

Таким образом, трудоемкость сортировки вставками зависит не только от количества элементов в массиве, но и от самих элементов; конкретно – от разницы между максимальным и минимальным элементами массива. Кроме того, сортировка применима только к целым числам.

Несмотря на это, в случае, когда алгоритм применим – он даёт наилучшую производительность, поскольку трудоёмкость оценивается как O(N) вместо .

**Тестирование алгоритмов**

Были проведены исследования зависимости времени работы алгоритмов от размеров сортируемых массивов и их содержания. Исследовались четыре типа массивов:

а) отсортированные по возрастанию (1 2 … N);

б) отсортированные по убыванию (N N-1 … 2 1);

в) заполненные случайными числами от 1 до 1000;

г) заполненные случайными числами от 1 до 30000;

д) заполненные одинаковыми числами (1 … 1).

Замеры времени проводились с использованием функций QueryPerformanceFrequency и QueryPerformanceCounter, входящих в состав Windows API.

Кроме того, для сравнения были проведены замеры для функций сортировки qsort() и std::sort. Исследование показало, что сортировка индексами становится неэффективной только в случае массива, заполненного одинаковыми числами.

Отсортированные по возрастанию массивы:

Отсортированные по убыванию массивы:

Массив случайных чисел 1 … 1000:

Массив случайных чисел 1 … 30000:

Массив единиц: