**Оценка эффективности сортировок C++**

Довгополый Е.А. РЛ6-21 МГТУ им. Н.Э.Баумана

Основная функция main состоит из цикла for от 1 000 (минимальный размер сортируемого массива) до 8 800 000 (максимальный размер сортируемого массива).

Внутри цикла создается массив нужного размера, далее он рандомно заполняется при помощи функции randomFill.

Текущее время при помощи встроенной функции GetTickCount() записывается в переменную time, проводится сортировка, время сортировки time1 = текущее время – время начала сортировки. Если время последнего выполнения этой сортировки превысило #define maxTime 10000 (10 000 мс = 10 с), то сортировка не проводится. Эта часть кода внутри цикла for повторяется для каждой сортировки.

Далее выполняется запись полученных данных в файл при помощи функции durSortToFile и переход к следующему размеру массива при помощи функции nextSize. Использованный массив удаляется.

*for* (int size = 1000; size < 10000000;) {

my\_array = *new* int [size];

*if* (time1<maxTime)

{

randomFill(*my\_array*, size);

time = GetTickCount();

bubbleSort(*my\_array*, size);

time1 = GetTickCount() - time;

}

…

durSortToFile(time1, time2, time3, time4, time5, time6, time7, size);

nextSize(*size*);

*delete* [] my\_array;

}

Функция рандомного заполнения массива randomFill(int\* a, int \_size) принимает на вход массив в виде указателя и его размер. В цикле for i-ый член массива принимает рандомное значение.

void **randomFill**(int\* a, int \_size) {

*for* (int i=0; i<\_size; i++)

{

a[i]=rand();

}

}

Функция записи в файл durSortToFile(int time1…, int \_size) принимает на вход размер массива и время всех сортировок. Файл открывается (создаётся) по заданному адресу, в него дописываются размер массива, время сортировок (разделитель - табуляция) и осуществляется переход на следующую строку. Далее файл закрывается.

void **durSortToFile**(int time1, int time2, int time3, int time4, int time5, int time6, int time7, int \_size) {

FILE \*sort1 = fopen ("C:\\МГТУ\\2 семестр\\Основы программирования на C++\\SortTime\\SortTime.txt", "a");

fprintf(*sort1*, " %d\t%d\t%d\t%d\t%d\t%d\t%d\t%d\n", \_size, time1, time2, time3, time4, time5, time6, time7);

fclose(*sort1*);

}

Для автоматического удаления устаревшего файла с данными в начало main был добавлен следующий код. Файл открывается на запись, полностью очищается и сразу закрывается.

FILE \*sort = fopen ("C:\\МГТУ\\2 семестр\\Основы программирования на C++\\SortTime\\SortTime.txt", "w");

fclose(*sort*);

Пример строки, записанной в файл:

3100 63 0 15 32 0 78 31

Функция перехода к следующему размеру массива nextSize(int& \_size) принимает на вход адрес переменной, в которой хранится размер массива и увеличивает её значение в зависимости от текущего размера.

void **nextSize**(int& \_size) {

*if* (1000<=\_size && \_size<10000)

\_size+=100;

*if* (10000<=\_size && \_size<100000)

\_size+=1000;

*if* (100000<=\_size && \_size<1000000)

\_size+=10000;

*if* (1000000<=\_size)

\_size+=100000;

}

**Виды сортировок:**

1. Пузырьковая сортировка (bubbleSort)

Будем идти по массиву слева направо. Если текущий элемент больше следующего, меняем их местами. Делаем так, пока массив не будет отсортирован. Заметим, что после первой итерации самый большой элемент будет находиться в конце массива, на правильном месте. После двух итераций на правильном месте будут стоять два наибольших элемента, и так далее.

void **bubbleSort**(int data[], int lenD)

{

int tmp = 0;

*for*(int i = 0;i<lenD;i++)

{

*for*(int j = (lenD-1);j>=(i+1);j--)

{

*if*(data[j]<data[j-1])

{

tmp = data[j];

data[j]=data[j-1];

data[j-1]=tmp;

}

}

}

}

1. Быстрая сортировка (quickSort)

Выберем некоторый опорный элемент. После этого перекинем все элементы, меньшие его, налево, а большие – направо. Рекурсивно вызовемся от каждой из частей. В итоге получим отсортированный массив, так как каждый элемент меньше опорного стоял раньше каждого большего опорного.

void **quickSort**(int \*mas, int size) {

*//Указатели* *в* *начало* *и* *в* *конец* *массива*

int i = 0;

int j = size - 1;

*//Центральный* *элемент* *массива*

int mid = mas[size / 2];

*//Делим* *массив*

*do* {

*//Пробегаем* *элементы,* *ищем* *те,* *которые* *нужно* *перекинуть* *в* *другую* *часть*

*//В* *левой* *части* *массива* *пропускаем(оставляем* *на* *месте)* *элементы,* *которые* *меньше* *центрального*

*while*(mas[i] < mid) {

i++;

}

*//В* *правой* *части* *пропускаем* *элементы,* *которые* *больше* *центрального*

*while*(mas[j] > mid) {

j--;

}

*//Меняем* *элементы* *местами*

*if* (i <= j) {

int tmp = mas[i];

mas[i] = mas[j];

mas[j] = tmp;

i++;

j--;

}

} *while* (i <= j);

*//Рекурсивные* *вызовы,* *если* *осталось,* *что* *сортировать*

*if*(j > 0) {

*//"Левый* *кусок"*

quickSort(*mas*, j + 1);

}

*if* (i < size) {

*//"Првый* *кусок"*

quickSort(*&mas[i]*, size - i);

}

}

1. Сортировка выбором (selectionSort)

На очередной итерации будем находить минимум в массиве после текущего элемента и менять его с ним, если надо. Таким образом, после i-ой итерации первые i элементов будут стоять на своих местах.

void **selectionSort**(int\* a, int size) *//выбором*

{

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

{

int min\_ind=i;

*for*(int j=i;j<size;j++)

{

*if*(a[j]<a[min\_ind])

{

min\_ind=j;

}

}

*if*(i!=min\_ind)

{

int temp=0;

temp=a[i];

a[i]=a[min\_ind];

a[min\_ind]=temp;

}

}

}

1. Сортировка вставками (insertSort)

При сортировке вставками массив разбивается на две области: упорядоченную и неупорядоченную. Изначально весь массив является неупорядоченной областью. При первом проходе первый элемент из неупорядоченной области изымается и помещается в правильное положении в упорядоченной области. На каждом проходе размер упорядоченной области возрастает на 1, а размер неупорядоченной области сокращается на 1.

Основной цикл работает в интервале от 1 до N-1. На j-й итерации элемент [i] вставлен в правильное положение в упорядоченной области. Это сделано путем сдвига всех элементов упорядоченной области, которые больше, чем [i], на одну позицию вправо. [i] вставляется в интервал между теми элементами, которые меньше [i], и теми, которые больше [i].

void **insertSort**(int\* a, int size) *//* *вставками*

{

*for*(int i=1;i<size;i++)

{

int key=a[i];

int j= i-1;

*while*(j>=0 && a[j]>key)

{

a[j+1]=a[j];

a[j]=key;

j--;

}

}

}

1. Сортировка слиянием (mergeSort)

Массив сначала разбивается на мелкие кусочки - на первом этапе - на состоящие из одного элемента. Затем эти кусочки объединяются в более крупные кусочки - по два элемента и элементы при этом сравниваются, а в результате в новом кусочке меньший элемент занимает место слева, а больший - справа. Далее происходит слияние в ещё более крупные кусочки и так до конца алгоритма, когда все кусочки будут объединены в один, уже отсортированный массив.

void **mergeSort**(int data[], int lenD) *//слиянием*

{

*if*(lenD>1){

int middle = lenD/2;

int rem = lenD-middle;

int\* L = *new* int[middle];

int\* R = *new* int[rem];

*for*(int i=0;i<lenD;i++){

*if*(i<middle){

L[i] = data[i];

}

*else*{

R[i-middle] = data[i];

}

}

mergeSort(L,middle);

mergeSort(R,rem);

merge(data, lenD, L, middle, R, rem);

}

}

void **merge**(int merged[], int lenD, int L[], int lenL, int R[], int lenR){

int i = 0;

int j = 0;

*while*(i<lenL||j<lenR){

*if* (i<lenL & j<lenR){

*if*(L[i]<=R[j]){

merged[i+j] = L[i];

i++;

}

*else*{

merged[i+j] = R[j];

j++;

}

}

*else* *if*(i<lenL){

merged[i+j] = L[i];

i++;

}

*else* *if*(j<lenR){

merged[i+j] = R[j];

j++;

}

}

}

1. Шейкерная сортировка (shakerSort)

Начинается процесс как в пузырьковой: выдавливаем максимум на самые задворки. После этого разворачиваемся на 180о и идём в обратную сторону, при этом уже перекатывая в начало не максимум, а минимум. Отсортировав в массиве первый и последний элементы, снова делаем разворот. Сделав это несколько раз, в итоге заканчиваем процесс, оказавшись в середине списка.

void **shakerSort**(int \*arr, int size)

{

int leftMark = 1;

int rightMark = size - 1;

*while* (leftMark <= rightMark)

{

*for* (int i = rightMark; i >= leftMark; i--)

*if* (arr[i - 1] > arr[i]){

int temp=0;

temp=arr[i];

arr[i]=arr[i-1];

arr[i-1]=temp;

}

leftMark++;

*for* (int i = leftMark; i <= rightMark; i++)

*if* (arr[i - 1] > arr[i]){

int temp=0;

temp=arr[i];

arr[i]=arr[i-1];

arr[i-1]=temp;

}

rightMark--;

}

}

1. Гномья сортировка (gnomeSort)

Гномья сортировка основана на технике, используемой обыкновенным голландским садовым гномом. Вот как садовый гном сортирует ряд цветочных горшков. По существу, он смотрит на два соседних цветочных горшка, если они находятся в правильном порядке, то он переходит на один горшок вперед, иначе он меняет их местами и возвращается на один горшок назад. Граничные условия: если позади нет ни одного горшка – он шагает вперед, а если нет следующего горшка, тогда он закончил.

void **gnomeSort**(int\* A, int N) {

int i = 0; int tmp = 0;

*while* (i < N) {

*if* (i == 0 || A[i - 1] <= A[i])

++i;

*else* {

tmp = A[i];

A[i] = A[i - 1];

A[i - 1] = tmp;

--i;

}

}

}

Записанный программой файл, содержащий размеры массивов и времена сортировок, загружаем в gnuplot. Используем следующие команды:

set lmargin 24 (левое поле)

set title font "calibri, 25"

set tics font "calibri, 25"

set key font "calibri, 25" (шрифт и размер)

set logscale x (логарифмический масшаб по оси х)

set format x "%9.0e" (подписи по х в научном формате)

set xrange [10000; 8800000] (диапазон значений по оси х)

set yrange [0; 10000] (диапазон значений по оси у)

set xlabel "Кол-во элементов" font "calibri, 25" offset 0.0,-1.5 (подпись оси х, шрифт, размер, смещение)

set ylabel "Время сортировки, мс" font "calibri, 25" offset -7.5,0.0 (подпись оси у, шрифт, размер, смещение)

plot "SortTime88.txt" using 1:2 with lp lw 3 title "Пузырьковая",\

"SortTime88.txt" using 1:3 with lp lw 3 title "Быстрая", \

"SortTime88.txt" using 1:4 with lp lw 3 title "Выбором", \

"SortTime88.txt" using 1:5 with lp lw 3 title "Вставками", \

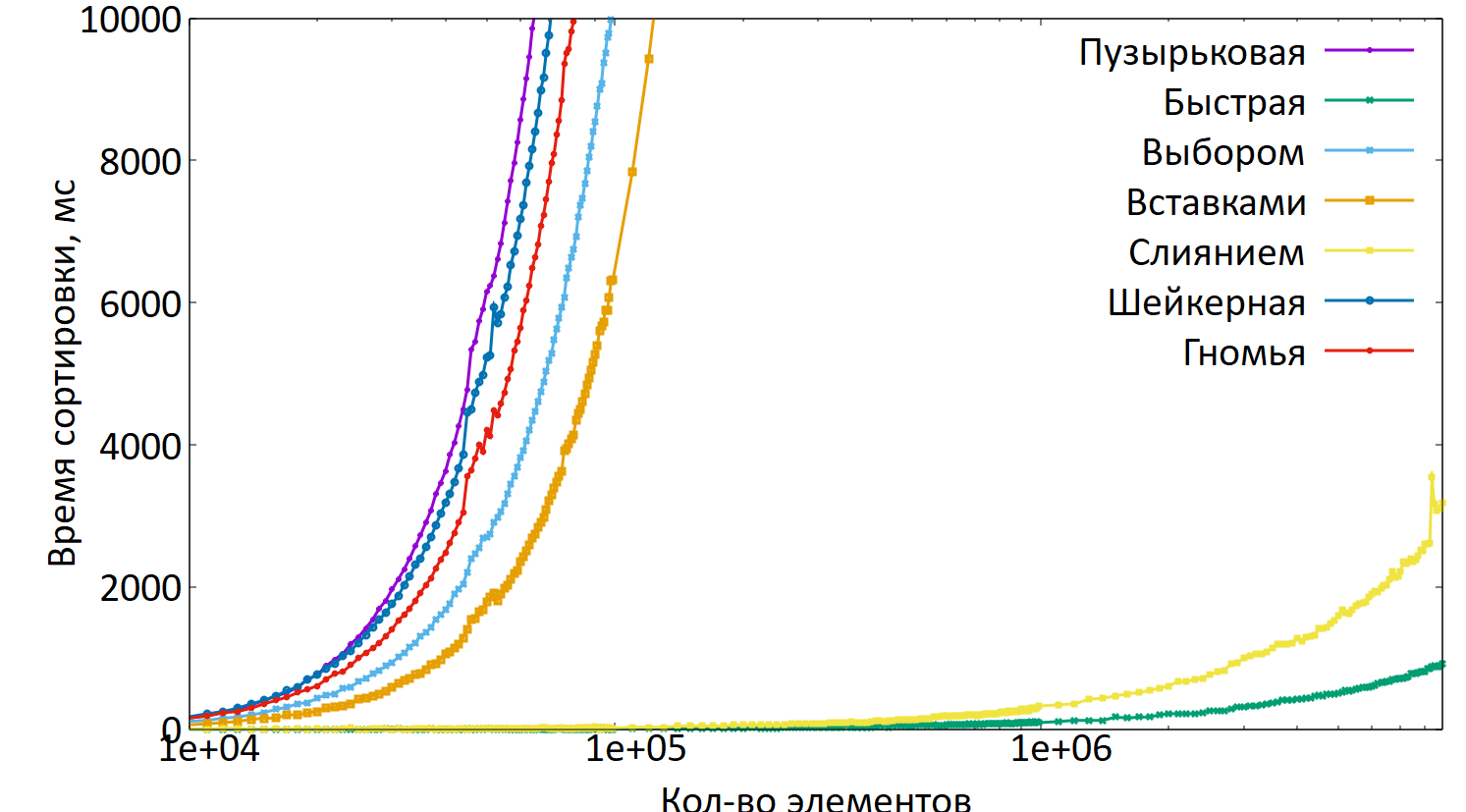
"SortTime88.txt" using 1:6 with lp lw 3 title "Слиянием",\

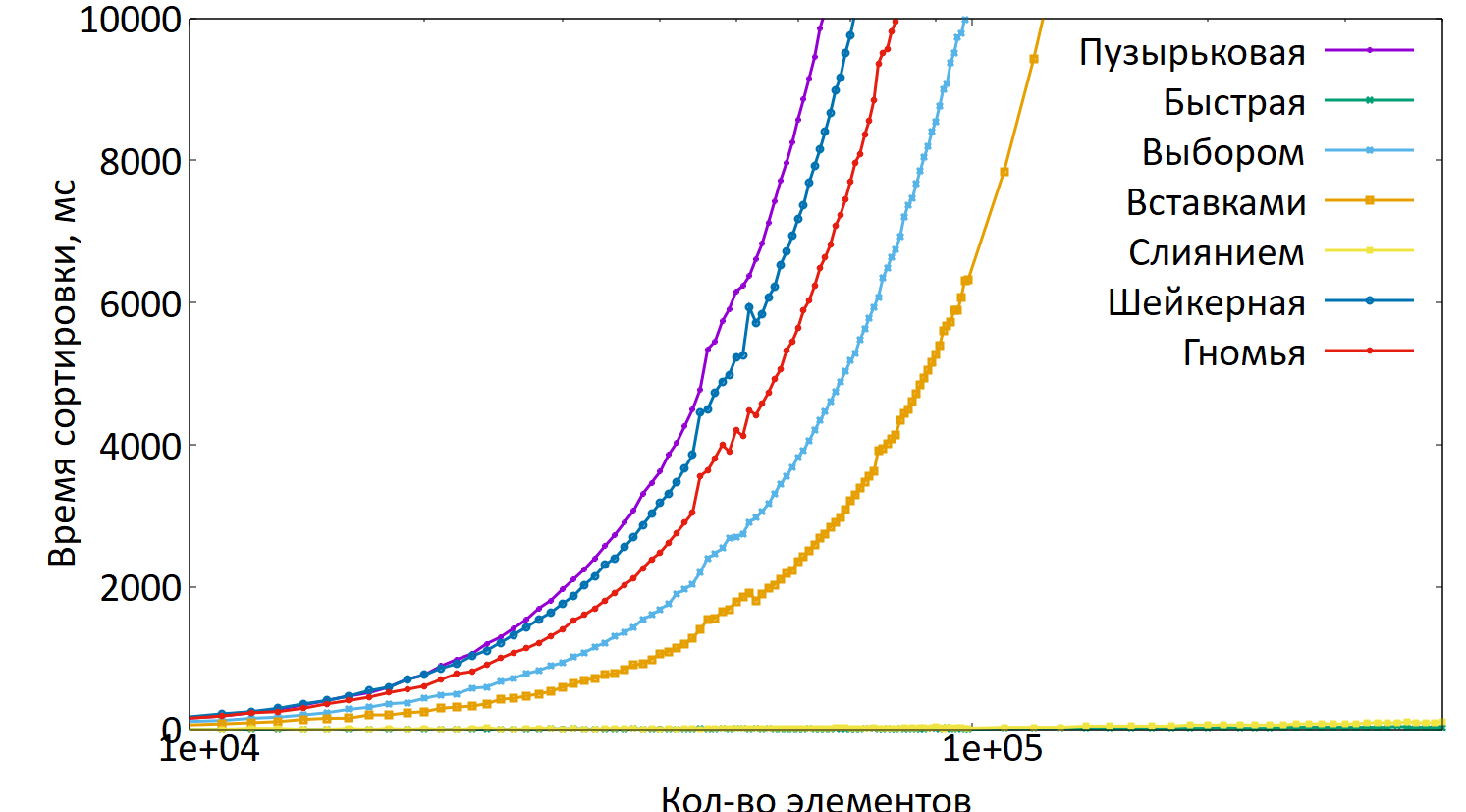
"SortTime88.txt" using 1:7 with lp lw 3 title "Шейкерная", \

"SortTime88.txt" using 1:8 with lp lw 3 title "Гномья"

(строим графики, используя колонку 1 как х, колонки 2-8 как у, точки отмечаются, проводятся линии толщиной 3, каждый график подписывается).

**Результаты и выводы**





Места сортировок (от самой быстрой к самой медленной):

1. Быстрая (quickSort);
2. Слиянием (mergeSort);
3. Вставками (insertSort);
4. Выбором (selectionSort);
5. Гномья (gnomeSort);
6. Шейкерная (shakerSort);
7. Пузырьковая (bubbleSort).