|  |  |
| --- | --- |
| для прик эмбл | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждениевысшего профессионального образования"Московский технологический университет"МИРЭА | |
| Институт информационных технологий (ИТ) | |
| Кафедра *общей информатики* | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ** | |
| **по дисциплине** | |
| **«**Анализ сложности алгоритмов**»** | |
|  | |
| Выполнила студентка группы ИКБО-05-16 | Ступишина Е.А. |
| Принял преподаватель | Мирабо Е.И. |

|  |  |  |
| --- | --- | --- |
| Лабораторные работы выполнены | «\_\_»\_\_\_\_\_\_\_201\_\_ г. |  |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_201\_\_ г. |  |

Москва 2017

Оглавление

Цель работы3

Постановка задачи3

Тестирование программы 4

Таблицы 5

Графики 7

Вывод 9

Листинг 10

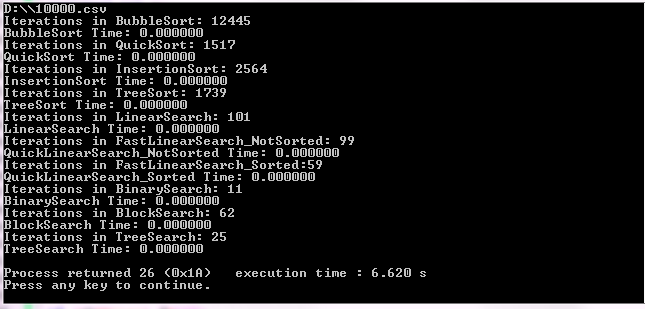
**Цель работы**: изучение алгоритмов сортировки и поиска элементов в массиве, и закрепление навыков в проведении сравнительного анализа алгоритмов.

**Постановка задачи**

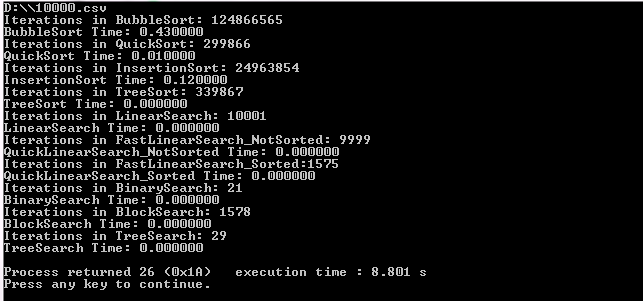
1. Изучить алгоритмы сортировки:
   1. Пузырьковая сортировка;
   2. Быстрая сортировка;
   3. Сортировка вставками;
   4. Сортировка с помощью двоичного дерева;
2. Изучить алгоритмы поиска:
   1. В неупорядоченном массиве
      1. Линейный поиск;
      2. Быстрый линейный поиск;
   2. В упорядоченном массиве:
      1. Быстрый линейный поиск;
      2. Бинарный поиск;
      3. Блочный поиск;
      4. Поиск деревом;
3. Построить таблицы, отображающие полученные данные (количество итераций и время выполнения алгоритмов);
4. Построить графики, основываясь на таблицы;
5. Сделать вывод.

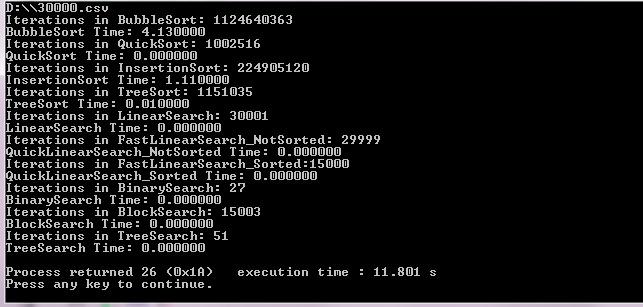
**Тестирование программы**

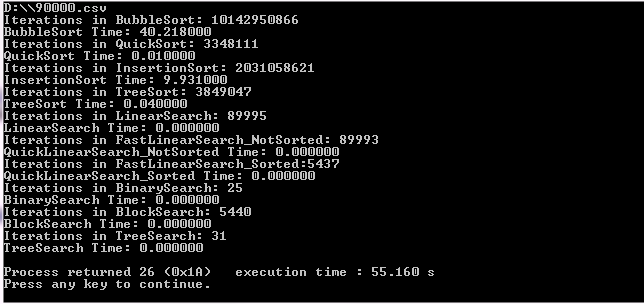
Для каждой сортировки и для каждого поиска поочередно выводится сначала количество итераций, затем время, затраченное на работу алгоритма.

100 элементов:

10000 элементов:



30000 элементов:

90000 элементов:

**Таблицы**

В приведённых ниже таблицах содержатся данные о количестве итераций, выполненных в ходе работы каждого алгоритма, а также о времени, затраченном на выполнение сортировок (время выполнения алгоритмов поиска равно нулю). Для каждого алгоритма указано по 3 значения. Средние значения выделены голубым цветом.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Алгоритмы | Количество элементов | | | | | | | |
| 10 | 100 | 1000 | 10000 | 30000 | 50000 | 70000 | 90000 |
| Bubble Sort | 121 | 12445 | 1264444 | 124866565 | 1124640363 | 3130922859 | 6135382435 | 10142950866 |
| 135 | 12648 | 1364584 | 134594568 | 1426778656 | 3245665758 | 6423279456 | 10856885638 |
| 120 | 11345 | 1294571 | 167854156 | 2158878685 | 3455465454 | 666789725 | 11453486441 |
| 125 | 12146 | 1307866 | 142438430 | 1570099235 | 3277351357 | 4408483872 | 10817774315 |
| Quick Sort | 82 | 1517 | 22131 | 299866 | 1002516 | 1770492 | 2514191 | 3348111 |
| 91 | 1348 | 23485 | 294568 | 1481503 | 1972397 | 2456897 | 3425627 |
| 80 | 1241 | 24783 | 464861 | 1310769 | 1547927 | 305678 | 4267652 |
| 84 | 1368 | 23466 | 353098 | 1264929 | 1763605 | 1758922 | 3680463 |
| Insertion Sort | 31 | 2564 | 255647 | 24963854 | 224905120 | 627015952 | 1228519144 | 2031058621 |
| 45 | 2631 | 264788 | 46788566 | 238646144 | 794638486 | 1594074686 | 2468688946 |
| 24 | 2486 | 246789 | 45086144 | 297436058 | 486846064 | 1468766478 | 3086864683 |
| 33 | 2560 | 255741 | 38946188 | 253662441 | 636166834 | 1430453436 | 2528870750 |
| Tree Sort | 97 | 1739 | 249857 | 339867 | 1151035 | 2020621 | 2923961 | 3849047 |
| 86 | 1669 | 312461 | 312597 | 1567664 | 2467854 | 2238846 | 3468746 |
| 87 | 1374 | 197654 | 564862 | 1456846 | 2564788 | 2429765 | 4686468 |
| 90 | 1594 | 253324 | 405775 | 1391848 | 2351087 | 2530857 | 4001420 |
| Linear Search | 10 | 101 | 1001 | 10001 | 30001 | 50001 | 70001 | 90001 |
| 11 | 101 | 995 | 9997 | 29998 | 49999 | 70001 | 90001 |
| 10 | 100 | 1000 | 10000 | 30000 | 50000 | 70000 | 90000 |
| 10 | 100 | 998 | 9999 | 29999 | 50000 | 70000, | 90000 |
| Fast Linear Search (ns) | 8 | 99 | 999 | 9999 | 29999 | 49999 | 69999 | 89999 |
| 11 | 101 | 1001 | 9995 | 29996 | 49997 | 69998 | 89999 |
| 11 | 101 | 1001 | 10001 | 30001 | 50001 | 70001 | 90001 |
| 10 | 100 | 1000 | 9998 | 29998 | 49999 | 69999 | 89999 |
| Fast Linear Search (s) | 4 | 59 | 555 | 1575 | 15000 | 44348 | 22976 | 80138 |
| 11 | 101 | 1001 | 523 | 3302 | 49934 | 25600 | 5437 |
| 1 | 14 | 226 | 2112 | 6296 | 10494 | 14697 | 189 |
| 5 | 58 | 594 | 1403 | 8199 | 34925 | 21091 | 28588 |
| Binary Search | 1 | 11 | 17 | 21 | 27 | 31 | 31 | 25 |
| 2 | 21 | 23 | 25 | 29 | 31 | 31 | 27 |
| 1 | 16 | 14 | 18 | 18 | 25 | 27 | 19 |
| 1 | 16 | 18 | 21 | 24 | 29 | 29 | 23 |
| Block Search | 7 | 62 | 558 | 1578 | 15003 | 44351 | 22979 | 5440 |
| 8 | 54 | 612 | 526 | 3305 | 49937 | 25603 | 80141 |
| 5 | 18 | 230 | 2116 | 6300 | 10498 | 14701 | 18933 |
| 6 | 44 | 466 | 1406 | 8202 | 34928 | 21094 | 34838 |
| Tree Search | 11 | 25 | 31 | 29 | 51 | 55 | 41 | 31 |
| 10 | 38 | 34 | 33 | 37 | 29 | 37 | 33 |
| 5 | 15 | 19 | 19 | 19 | 21 | 21 | 21 |
| 8 | 26 | 28 | 27 | 35 | 35 | 33 | 28 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алгоритмы сортировки | Количество элементов | | | | |
| 10000 | 30000 | 50000 | 70000 | 90000 |
| Bubble Sort | 0,43 | 4,13 | 11,60 | 24,26 | 40,21 |
| 0,46 | 4,38 | 12,31 | 24,25 | 40,18 |
| 0,46 | 4,40 | 12,38 | 24,38 | 40,38 |
| 0,45 | 4,30 | 12,43 | 24,30 | 40,26 |
| Quick Sort | 0,00 | 0,00 | 0,01 | 0,01 | 0,01 |
| 0,00 | 0,01 | 0,01 | 0,01 | 0,02 |
| 0,00 | 0,00 | 0,01 | 0,01 | 0,02 |
| 0,00 | 0,00 | 0,01 | 0,01 | 0,02 |
| Insertion Sort | 0,12 | 1,11 | 3,08 | 6,01 | 9,93 |
| 0,00 | 1,11 | 3,12 | 6,09 | 10,07 |
| 0,00 | 1,11 | 3,09 | 6,06 | 10,05 |
| 0,04 | 1,11 | 3,10 | 6,05 | 10,02 |
| Tree Sort | 0,00 | 0,01 | 0,02 | 0,03 | 0,04 |
| 0,00 | 0,02 | 0,02 | 0,04 | 0,05 |
| 0,00 | 0,01 | 0,02 | 0,03 | 0,04 |
| 0,00 | 0,01 | 0,02 | 0,03 | 0,04 |

**Графики**

Зависимость количества итераций от числа элементов:

Зависимость времени от количества элементов:

**Вывод**

В ходе работы удалось выяснить, что для массива менее 100 тыс. элементов выбор алгоритма поиска не влияет на скорость программы. Самым неэффективным по затратам памяти алгоритмом поиска оказался линейный (сложность O(n)). Самыми эффективными – бинарный поиск и поиск деревом (сложность порядка O(log2(n))).

Что же до алгоритмов сортировок, самые быстрые из них – это QuickSort и TreeSort.

Сложность этих алгоритмов – O(n log2(n)). При количестве элементов до 1000, сортировка вставками имеет практически одинаковую с указанными выше сортировками сложность. Однако при большем числе элементов, её сложность, как и у сортировки пузырьком,

становится равна O(n^2). BubbleSort является также самой медленной сортировкой, что заметно уже при количестве элементов более 10 тыс.

**Листинг**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#define N 256

int qsi=0, tsi=0, ts=0, trsi=0;

double cltsh=0;

typedef struct pack {

double num;

unsigned int date;

} PACK;

void reset(PACK \*a, PACK \*b, long K) {

int i;

for (i=0; i<K; i++)

a[i]=b[i];

}

unsigned int ForSearching (PACK \*a, int K) {

int i;

PACK p=a[0];

unsigned int x;

for (i=1; i<K; i++)

if (a[i].num<p.num) x=a[i].date;

return (x);

}

int main(int argc, char \*argv[]) {

clock\_t start, finish;

char ch, name[N];

PACK \*arr, \*copy, temp;

int i=0, j=0, k=0, p=0;

long K=0;

unsigned int x;

long int tt;

FILE \*File;

if (argc!=2) {

printf("Enter file name.\n");

scanf("%s", name);

}

else

strcpy(name, argv[1]);

File=fopen(name, "rb");

while (File == NULL) {

printf("File not found.\nTry again.\n");

scanf("%s", name);

File = fopen(name, "rb");

}

while (fscanf(File,"%c",&ch) != EOF)

if (ch=='\n') K++;

rewind (File);

arr=(PACK\*)malloc(K\*sizeof(PACK));

copy=(PACK\*)malloc(K\*sizeof(PACK));

while (fscanf(File, "%lf,%u", &arr[i].num, &arr[i].date)!=EOF) {

i++;

}

fclose(File);

x=ForSearching(arr, K);

reset (copy, arr, K);

start=clock();

BubbleSort (copy, K);

finish=clock();

printf("BubbleSort Time: %f\n", ((double)(finish - start)) / CLOCKS\_PER\_SEC);

reset (copy, arr, K);

start=clock();

QuickSort(copy, 0, K-1);

finish=clock();

printf ("Iterations in QuickSort: %u\n", qsi);

printf("QuickSort Time: %f\n", ((double)(finish - start)) / CLOCKS\_PER\_SEC);

reset (copy, arr, K);

start=clock();

InsertionSort(copy, K);

finish=clock();

printf("InsertionSort Time: %f\n", ((double)(finish - start)) / CLOCKS\_PER\_SEC);

reset (copy, arr, K);

start=clock();

TreeSort (copy, K, x);

finish=clock();

printf ("Iterations in TreeSort: %u\n", tsi);

printf("TreeSort Time: %f\n", (((double)(finish - start)) / CLOCKS\_PER\_SEC)-cltsh);

start=clock();

int ls=LinearSearch (arr, K, x);

finish=clock();

printf("LinearSearch Time: %f\n", ((double)(finish - start)) / CLOCKS\_PER\_SEC);

start=clock();

int fls\_ns=QuickLinearSearch\_NotSorted (arr, K, x);

finish=clock();

printf("QuickLinearSearch\_NotSorted Time: %f\n", ((double)(finish - start)) / CLOCKS\_PER\_SEC);

start=clock();

int fls\_s=QuickLinearSearch\_Sorted (copy, K, x);

finish=clock();

printf("QuickLinearSearch\_Sorted Time: %f\n", ((double)(finish - start)) / CLOCKS\_PER\_SEC);

start=clock();

int bs=BinarySearch (copy, K, x);

finish=clock();

printf("BinarySearch Time: %f\n", ((double)(finish - start)) / CLOCKS\_PER\_SEC);

start=clock();

int bls=BlockSearch(copy, K, x);

finish=clock();

printf("BlockSearch Time: %f\n", ((double)(finish - start)) / CLOCKS\_PER\_SEC);

printf ("Iterations in TreeSearch: %u\n", trsi);

printf("TreeSearch Time: %f\n", cltsh);

}

void BubbleSort (PACK \*a, long K) {

int i, j, k;

unsigned long long int iter=0;

PACK temp;

for (i=0; i<K; i++){

iter++;

for (j=0; j<K-i-1; j++){

iter++;

if (a[j].date>a[j+1].date) {

iter+=3;

temp=a[j];

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

a[j+1]=temp;

}

}

}

printf ("Iterations in BubbleSort: %llu\n", iter);

}

void QuickSort(PACK \*a, long l, long r) {

int j;

if(l<r) {

j = partition(a, l, r);

QuickSort(a, l, j-1);

QuickSort(a, j+1, r);

}

}

int partition(PACK \*a, long l, long r) {

int i, j;

PACK pivot, temp;

pivot=a[l];

i = l; j = r+1;

while(1) {

qsi++;

do {++i; qsi++;} while(a[i].date<=pivot.date && i<=r);

do {--j; qsi++;} while(a[j].date>pivot.date);

if(i>=j) {qsi++; break;}

temp=a[i]; a[i]=a[j]; a[j]=temp;

qsi+=3;

}

temp=a[l]; a[l]=a[j]; a[j]=temp;

qsi+=3;

return j;

}

void InsertionSort (PACK \*a, long K) {

int loc, i;

unsigned int iter=0;

PACK elem;

for (i=1; i<K; i++) {

iter++;

elem=a[i];

loc=i-1;

while(loc>=0 && a[loc].date>elem.date) {

iter++;

a[loc+1]=a[loc];

loc--;

}

a[loc+1]=elem;

}

printf ("Iterations in InsertionSort: %u\n", iter);

}

typedef struct tree {

PACK a;

struct tree \*left;

struct tree \*right;

} TREE;

TREE \*add\_to\_tree (TREE \*root, PACK new\_value) {

tsi++;

if (root==NULL)

{

root=(TREE\*)malloc(sizeof(TREE));

root->a = new\_value;

root->left = root->right = 0;

return root;

}

if (root->a.date<new\_value.date) {

root->right=add\_to\_tree(root->right, new\_value);

tsi++;

}

else {

root->left=add\_to\_tree(root->left, new\_value);

tsi++;

}

return root;

}

void tree\_to\_array (TREE \*root, PACK \*a) {

tsi++;

static max2=0;

if (root==NULL) return;

tree\_to\_array(root->left, a);

a[max2++]=root->a;

tree\_to\_array(root->right, a);

free(root);

}

void TreeSort (PACK \*a, int max, unsigned int x) {

int i;

TREE \*root = 0;

for (i=0; i<max; i++) {

tsi++;

root=add\_to\_tree(root, a[i]);

}

TreeSearch(root, x);

tree\_to\_array(root, a);

}

void TreeSearch(TREE \*root, unsigned int x) {

clock\_t begin, end;

trsi++;

begin=clock();

while ((root!=NULL) && root->a.date!=x) {

trsi++;

if (x<root->a.date)

return TreeSearch(root->left, x);

else if (x>root->a.date)

return TreeSearch(root->right, x);

}

end=clock();

cltsh=(double)(begin - end) / CLOCKS\_PER\_SEC;

}

int LinearSearch (PACK \*a, int K, unsigned int x) {

int i;

unsigned int iter=0;

for (i=0; i<K; i++) {

iter++;

if (a[i].date==x) {

iter++;

printf ("Iterations in LinearSearch: %u\n", iter);

return i;

}

}

return -1;

}

int QuickLinearSearch\_NotSorted (PACK \*a, int K, unsigned int x) {

int i=0;

unsigned int iter=0;

PACK New;

New.date=x;

a[K+1]=New;

while (a[i].date!=x) {++i; ++iter;}

printf ("Iterations in FastLinearSearch\_NotSorted: %u\n", iter);

if (i!=K+1) return i;

else return -1;

}

int QuickLinearSearch\_Sorted (PACK \*a, int K, unsigned int x) {

int i=0;

unsigned int iter=0;

PACK New;

New.date=x;

a[K+1]=New;

do {i++; iter++;}

while (a[i].date<x);

printf ("Iterations in FastLinearSearch\_Sorted:%u\n", iter);

if (i!=K+1) return i;

else return -1;

}

int BinarySearch(PACK \*a, int K, unsigned int x) {

int low, high, middle;

unsigned int iter=0;

low = 0;

high = K-1;

while (low<=high) {

iter++;

middle=(low + high)/2;

if (x<a[middle].date) {

high=middle-1;

iter++;

}

else if (x>a[middle].date) {

low=middle+1;

iter++;

}

else {

printf ("Iterations in BinarySearch: %u\n", iter);

if (x==a[middle].date) return middle;

else return -1;

}

}

}

int BlockSearch(PACK \*a, int K, unsigned int x) {

int f=1, i=0, l, in=(int)pow(K,(1/2));

unsigned int iter=0;

while ((i<=K)&&(f)) {

iter++;

i=i+in;

if (i>K) {i=K; iter++;}

int r=i;

l=i-3;

if (x<a[i].date) {

while ((l<=r)&&(x!=a[l].date)) {iter++; l++;}

f=0;

printf ("Iterations in BlockSearch: %u\n", iter);

if (x==a[l].date) return l;

else return -1;

}

}

}