Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

**Лабораторна робота №2**

**«Методи сортування масивів»**

виконала:

ст. 3 курсу гр. ІС-зп61

Шуміліна У.О.

перевірив:

Селін О.М.

доцент, к.т.н.

Київ – 2016

**Мета роботи**: познайомитися з роботою поширених методів сортування, з критеріями та методикою їх порівняння.

**Хід роботи**

У далекій від нас країні жила собі одна заможна вдова з трьома дочками. Дві з них були їй рідні, а третя — пасербиця. Вдова своїх доньок дуже любила, гарно їх одягала й усі їхні примхи сповняла. А вже ту, нерідну доньку, так ненавиділа, що ані наїстись як слід їй не давала, тільки все лаяла та до праці силувала.

Одного дня було храмове свято, й людей посходилось із цілої країни видимо-невидимо. Кожному цікаво подивитись, коли кудись багато людей збереться, а до того на цей відпуст ще мав прибути й володар тієї країни.

Тому-то вже багато тижнів перед цим святом цілими ночами сиділа Попелюшка за дрібними вишиваннями та мережками для мачухи та її дочок. Попрала вона їм плаття, попрасувала нові горсетки та попередниці, що довгий час у скрині лежали. А як прийшла нарешті та неділя, то її так заганяли, що мало голови бідне дівча не згубило. За одним заходом на три сторони її кликали, трьома ротами лаяли, шістьма руками загрожували! Коли ж мачуха та її доньки були вже прибрані та сяяли всякими оздобами, жодна Попелюшці для годиться й «дякую!» не сказала. Набундючились і, мов пави, посунули до церкви, а Попелюшці наказали все в хаті й на подвір’ї поробити, обід зварити й ще, поки додому повернуться, цілу велику мірку засміченої пшениці по зернятку перебрати та за розміром відсортувати, щоб було на посів чисте зерно.

Крутилась Попелюшка по хаті, поки все поприбирала та обід варити настановила, а потім сіла на хвильку біля столу, поглянула на ту велетенську мірку пшениці, перемішаної з куколем і просом, та й зажурилась: коли ж вона з тією працею буде готова?! Отже ж, будуть її за те лаяти в святу неділю!

І вже зібралася вона йти до свого друга за мікрометром, щоб переміряти зернятка, як з’явився перед нею }{отт@бь)ч і каже:

* Не парься, серденько пшеницею, зараз я швиденько напишу алгоритм сортування методом простого включення і все норм буде. Я так дивлю у вас по хаті домашні мурахи повзають. От їх і змусимо алгоритм виконувати. Знаєш, ̶ сказав }{отт@бь)ч, дістаючи свого ноутбука, ̶ я ж як з Геною познайомився, взагалі перестав нормально чаклувати…. Тепер тільки так – через комп’ютер…. Такі от наслідки технологічного прогресу. А ти на чому програмуєш?
* …а я не програмую… ̶ здивовано відповіла Попелюшка.
* Хех, дарма….

}{отт@бь)ч швиденько накодив функцію сортування методом простого включення і зачаклував мурах на виконання, але процес йшов так повільно, що він навіть встиг згадати, що Гена лишив на його комп’ютері програмку з пірамідальним сортуванням.

Врешті, зернятка були відсортовані за розміром й Попелюшка, сяюча з радості, побігла на бал. А }{отт@бь)ч залишився порівнювати свій алгоритм методом включення з алгоритмом пірамідального сортування, що написав Гена. Результати дослідження він записав у табличку (додаток 0) і оформив протокол лабораторної роботи.

*Порівнюючи алгоритми сортування, можна побачити, що прямі алгоритми є дуже простими для написання, але на великих об’ємах даних працюють дуже повільно. В той же час, складні методи, такі як пірамідальне сортування, потребують тривалішої роботи з кодом, але на великих даних працюють значно швидше простих.*

**Додатки**

Додаток 0. Табличка

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Метод простого включення | | | | | Метод пірамідального сортування | | | | |
| N | К-ть копіювань (M) | | К-ть порівнянь (C) | | Час (T) | К-ть копіювань (M) | | К-ть порівнянь (C) | | Час (T) |
| Теор. | Експ. | Теор. | Експ. | Теор. | Експ. | Теор. | Експ. |
| 100 | 5250 | 2685 | 5050 | 2487 | 5,8е-4 | 2100 | 1719 | 1800 | 1526 | 4,2е-4 |
| 1000 | 502500 | 257261 | 500500 | 255263 | 5,9е-2 | 30000 | 27186 | 27000 | 25295 | 5,1е-3 |
| 10000 | 50025000 | 24967811 | 50005000 | 24947813 | 6,4 | 390000 | 372720 | 360000 | 352656 | 8,6е-2 |

Додаток 1. Генерований масив даних

24764 EMQ

30448 IPB

8966 XUR

8034 CKY

23878 YQB

27984 VJA

20804 CYR

6924 YNB

31997 QSS

12124 PME

20257 KMP

6650 EMF

11155 WTQ

8213 JJJ

16365 BCB

6350 VHC

10233 LVZ

31164 FIJ

25144 QPT

13160 RWU

5215 VDV

24017 JCY

8149 CMO

13545 JOH

1591 YWD

25879 VYE

7636 XLC

10163 FWX

5681 FKA

12173 DTA

19590 LKY

25958 EBP

28735 RUH

29813 RFI

30588 PEJ

9955 QRH

25490 WJX

4688 LBS

21896 ELL

5196 FDN

29338 IXX

7902 TUL

6329 YZR

21761 MUF

25353 MOT

30498 XCV

32353 LVS

8099 ABE

19919 ZHU

20315 FQI

24338 OVV

18964 WAX

12062 LXA

17163 NKP

16027 YFD

22900 MCT

30310 QTX

25843 RJS

23567 RMT

4644 WGI

5721 LVI

20244 VWD

16829 EVJ

1990 TZW

5352 KHV

12975 XLM

21816 WCY

11578 QNN

6979 CUJ

26692 VYV

26448 GTP

25761 KVT

1500 SNX

32661 TGN

17430 FHC

3932 MAF

29795 LZK

20388 PSR

17115 CSY

7693 TYN

22986 SQX

1712 QGD

22064 NIG

8341 UMF

3137 YEI

14906 LMI

7142 LKZ

22035 QQE

21818 LUW

30469 QQP

27918 ZKM

15594 EAB

358 XJC

20752 ULR

28154 BRA

8890 BZF

28853 PYQ

29598 CHR

7737 XJX

13413 QOF

Додаток 1. Масив даних, сортований методом простого включення

358 XJC

1500 SNX

1591 YWD

1712 QGD

1990 TZW

3137 YEI

3932 MAF

4644 WGI

4688 LBS

5196 FDN

5215 VDV

5352 KHV

5681 FKA

5721 LVI

6329 YZR

6350 VHC

6650 EMF

6924 YNB

6979 CUJ

7142 LKZ

7636 XLC

7693 TYN

7737 XJX

7902 TUL

8034 CKY

8099 ABE

8149 CMO

8213 JJJ

8341 UMF

8890 BZF

8966 XUR

9955 QRH

10163 FWX

10233 LVZ

11155 WTQ

11578 QNN

12062 LXA

12124 PME

12173 DTA

12975 XLM

13160 RWU

13413 QOF

13545 JOH

14906 LMI

15594 EAB

16027 YFD

16365 BCB

16829 EVJ

17115 CSY

17163 NKP

17430 FHC

18964 WAX

19590 LKY

19919 ZHU

20244 VWD

20257 KMP

20315 FQI

20388 PSR

20752 ULR

20804 CYR

21761 MUF

21816 WCY

21818 LUW

21896 ELL

22035 QQE

22064 NIG

22900 MCT

22986 SQX

23567 RMT

23878 YQB

24017 JCY

24338 OVV

24764 EMQ

25144 QPT

25353 MOT

25490 WJX

25761 KVT

25843 RJS

25879 VYE

25958 EBP

26448 GTP

26692 VYV

27918 ZKM

27984 VJA

28154 BRA

28735 RUH

28853 PYQ

29338 IXX

29598 CHR

29795 LZK

29813 RFI

30310 QTX

30448 IPB

30469 QQP

30498 XCV

30588 PEJ

31164 FIJ

31997 QSS

32353 LVS

32661 TGN

Додаток 3. Масив даних, сортований методом пірамідального сортування

358 XJC

1500 SNX

1591 YWD

1712 QGD

1990 TZW

3137 YEI

3932 MAF

4644 WGI

4688 LBS

5196 FDN

5215 VDV

5352 KHV

5681 FKA

5721 LVI

6329 YZR

6350 VHC

6650 EMF

6924 YNB

6979 CUJ

7142 LKZ

7636 XLC

7693 TYN

7737 XJX

7902 TUL

8034 CKY

8099 ABE

8149 CMO

8213 JJJ

8341 UMF

8890 BZF

8966 XUR

9955 QRH

10163 FWX

10233 LVZ

11155 WTQ

11578 QNN

12062 LXA

12124 PME

12173 DTA

12975 XLM

13160 RWU

13413 QOF

13545 JOH

14906 LMI

15594 EAB

16027 YFD

16365 BCB

16829 EVJ

17115 CSY

17163 NKP

17430 FHC

18964 WAX

19590 LKY

19919 ZHU

20244 VWD

20257 KMP

20315 FQI

20388 PSR

20752 ULR

20804 CYR

21761 MUF

21816 WCY

21818 LUW

21896 ELL

22035 QQE

22064 NIG

22900 MCT

22986 SQX

23567 RMT

23878 YQB

24017 JCY

24338 OVV

24764 EMQ

25144 QPT

25353 MOT

25490 WJX

25761 KVT

25843 RJS

25879 VYE

25958 EBP

26448 GTP

26692 VYV

27918 ZKM

27984 VJA

28154 BRA

28735 RUH

28853 PYQ

29338 IXX

29598 CHR

29795 LZK

29813 RFI

30310 QTX

30448 IPB

30469 QQP

30498 XCV

30588 PEJ

31164 FIJ

31997 QSS

32353 LVS

32661 TGN

Додаток 4. Лістинг програми

//+++++++++++++++++++++++++++++++++++++++++++++++++++//

// Meshkantsev Mykhaylo KA-04 (QuickSort, Selection) //

// Selin Olexander MMSA dept. (ShellSort, Insertion, //

// BubbleSort, ShakerSort) //

// Shumilina Uliana IS-zp61 (myInter) //

//+++++++++++++++++++++++++++++++++++++++++++++++++++//

#include <stdlib.h> // div\_t rand

#include <conio.h> // clrscr()

#include <math.h>

#include <iostream> // cin, cout

#include <fstream> // ofstream

#include <time.h>

using namespace std;

long int

N, // Array dimension

M; // Parameter for ShellSort. Never change it !

int shoo\_count;

unsigned long moves, compares;

struct tItem {

int key;

char data[4];

friend bool operator < (tItem x, tItem y) { return (x.key < y.key); }

friend bool operator <= (tItem x, tItem y) { return (x.key <= y.key); }

friend bool operator > (tItem x, tItem y) { return (x.key > y.key); }

friend bool operator >= (tItem x, tItem y) { return (x.key >= y.key); }

friend void swap(tItem& a, tItem& b);

friend void copy(tItem& a, tItem& b);

tItem& operator = (tItem& y) { copy(\*this, y); return \*this; };

};

void swap(tItem& x, tItem& y) {

int k = x.key; x.key = y.key; y.key = k;

char s[4];

//strcpy(s, x.data); strcpy(x.data, y.data); strcpy(y.data, s);

strcpy\_s(s, x.data); strcpy\_s(x.data, y.data); strcpy\_s(y.data, s);

moves++;

moves++;

moves++;

};

void copy(tItem& x, tItem& y) {

x.key = y.key;

//strcpy(x.data, y.data);

strcpy\_s(x.data, y.data);

moves++;

};

//ofstream fout("sort.txt");

ofstream fout;

void print(char \* text, tItem \* a, int N, bool sorted) {

fout << endl << text;

//if (N<101 && !sorted) {

fout << endl;

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

fout << a[i].key << " " << a[i].data << "\n";

if ((i + 1) % 10 == 0) fout << endl;

}

/\*}

if (sorted) {

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

if (i > 0 && a[i - 1] > a[i]) {

fout << a[i - 1].key << "(" << a[i - 1].data << ") > " << a[i].key << "(" << a[i].data << ") ! " << endl;

}

}

}\*/

}

//++++++++++++++++Quicksort++++++++++++++++++++++++++++

void QuickSort(tItem \* a, int L, int R) {

int m, i, j;

tItem x;

i = L;

j = R;

m = (L + R) / 2;

x = a[m];

while (i <= j) {

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

while (x<a[j]) { j--; }

if (i <= j) {

if (i<j) {

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

//c = a[i];

//a[i] = a[j];

//a[j] = c;

}

i++; j--;

}

}

if (L<j) { QuickSort(a, L, j); }

if (i<R) { QuickSort(a, i, R); }

}

//++++++++++++++++ShellSort++++++++++++++++++++++++++++

void ShellSort(tItem \* a) {

int i, t, k, m, l, j;

int \* h = new int[M];

tItem x;

t = (int)(log((double)N) / log((double)2) - 1);

h[t - 1] = 1;

for (k = t - 2; k >= 0; k--) {

h[k] = 2 \* h[k + 1] + 1;

}

for (m = 0; m<t; m++) { // Global iteration No

k = h[m]; // Step

for (l = 0; l <= k - 1; l++) { // SubArray No

for (i = 1; i <= (N - 1) / k; i++) {

if (l + i\*k < N) {

x = a[l + i\*k];

j = l + (i - 1)\*k;

while ((j >= 0) && (x<a[j])) {

a[j + k] = a[j];

j = j - k;

} // while ((j>=0)

a[j + k] = x;

} // if (l+i\*k

} // for (i=1;

} // SubArray

} // Global iteration

delete[] h;

}

//++++++++++++++++ShakerSort+++++++++++++++++++++++++++

void ShakerSort(tItem \* a) {

int i = 0, j, flag = 0;

while (flag == 0) {

flag = 1;

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

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

swap(a[j], a[j + 1]);

flag = 0;

}

}

if (flag == 0)

for (j = N - 2 - i; j>i; j--) {

if (a[j - 1]>a[j]) {

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

}

}

i++;

}

}

//++++++++++++++++MergeSort++++++++++++++++++++++++++++

void Merge(tItem \* a, int L, int m, int R) {

// merge a[L..m] and a[m+1..R]

int i1, i2, i;

tItem \* b = new tItem[R - L + 1];

for (i = 0; i<R - L + 1; ++i)

b[i] = a[i + L];

i = L; // index in a;

i1 = 0; // index in b[0..m-L];

i2 = m - L + 1; // index in b[m-L+1..R-L];

while (i <= R) {

if ((i1 <= m - L) && (i2 <= R - L)) {

if (b[i1]<b[i2])

a[i++] = b[i1++];

else

a[i++] = b[i2++];

continue;

}

if ((i1>m - L) && (i2 <= R - L)) {

a[i++] = b[i2++];

continue;

}

if ((i1 <= m - L) && (i2>R - L)) {

a[i++] = b[i1++];

continue;

}

if ((i1>m - L) && (i2>R - L)) {

cout << "!!!!!!!!!!!!! ";

continue;

}

}

delete[] b;

}

void MergeSort(tItem \* a, int L, int R) {

int m, i, j;

tItem x;

//cout << L << " " << R << endl;

if (R <= L) return;

if (R - L == 1) {

if (a[L] > a[R])

swap(a[L], a[R]);

return;

}

m = (R + L) / 2;

if (L<m) MergeSort(a, L, m);

if (R>m + 1) MergeSort(a, m + 1, R);

Merge(a, L, m, R);

}

//++++++++++++++++HeapSort+++++++++++++++++++++++++++++

void pushdown(tItem \* a, int L, int R) {

int i = L;

while (i + 1 <= (R + 1) / 2) {

if (R + 1 == 2 \* (i + 1)) {

if (a[i] < a[2 \* i + 1])

swap(a[i], a[2 \* i + 1]);

compares++;

i = R;

}

else {

if ((a[i] < a[2 \* i + 1]) && (a[2 \* i + 1] >= a[2 \* (i + 1)])) {

swap(a[i], a[2 \* i + 1]);

i = 2 \* i + 1;

} else {

if ((a[i] < a[2 \* (i + 1)]) && (a[2 \* (i + 1)] > a[2 \* i + 1])) {

swap(a[i], a[2 \* (i + 1)]);

i = 2 \* (i + 1);

}

else {

i = R;

} // if ((a[i] > a[2\*i+1]) ...

compares++;

compares++;

} // if ((a[i] > a[2\*i]) ...

compares++;

compares++;

} // if (R == 2\*i)

} // while

}

void HeapSort(tItem \* a) {

int i;

for (i = (N - 1) / 2; i >= 0; i--) {

pushdown(a, i, N - 1);

//print("", a, N, false);

}

//fout << "-----------------------------------" << endl;

for (i = N - 1; i>0; i--) {

swap(a[0], a[i]);

//print("Swap:", a, N, false);

pushdown(a, 0, i - 1);

//print("pushdown:", a, N, false);

//fout << endl;

}

}

void myInter(tItem \* a) {

int j;

tItem x;

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

copy(x, a[i]);

j = i;

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

compares++;

copy(a[j], a[j - 1]);

j = j - 1;

}

copy(a[j], x);

}

}

//+++++++++++++++++++++++++++++++++++++++++++++++++++//

//+++++++++++++++++++ +++++++++++++++++++++//

//+++++++++++++++++++ Main +++++++++++++++++++++//

//+++++++++++++++++++ +++++++++++++++++++++//

//+++++++++++++++++++++++++++++++++++++++++++++++++++//

void last\_main() {

tItem \* a, \*b; // Array and it's reserved copy

int i;

clock\_t start, finish;

double duration;

bool sorted = true;

a = new tItem[N];

b = new tItem[N];

srand(time(0));

//++++++++++++++++Generation+++++++++++++++++++++++

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

a[i].key = rand();

a[i].data[0] = 'A' + rand() % ('Z' - 'A' + 1);

a[i].data[1] = 'A' + rand() % ('Z' - 'A' + 1);

a[i].data[2] = 'A' + rand() % ('Z' - 'A' + 1);

a[i].data[3] = '\0';

b[i].key = a[i].key;

//strcpy(b[i].data, a[i].data);

strcpy\_s(b[i].data, a[i].data);

}

print("Array generated", a, N, !sorted);

//++++++++++++++++Quicksort++++++++++++++++++++++++

if (false) {

moves = 0; // кількість обмінів (копіювань)

compares = 0; // кількість порівнянь ключів елементів

start = clock();

QuickSort(a, 0, N - 1);

finish = clock();

print("Array sorted by QuickSort", a, N, sorted);

cout << "Time to sort array of " << N << " elements QuickSort:";

duration = (double)(finish - start) / CLOCKS\_PER\_SEC;

cout << duration << " seconds" << endl;

fout << endl << "Time to sort array of " << N << " elements QuickSort: "

<< duration << endl;

cout << compares << " - compares; " << moves << " moves \n";

fout << compares << " - compares; " << moves << " moves \n";

}

//============== my Insertion sort ============

if (true) {

start = clock();

for (int shoo = 0; shoo < shoo\_count; shoo++) {

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

a[i] = b[i];

moves = 0; // кількість обмінів (копіювань)

compares = 0; // кількість порівнянь ключів елементів

myInter(a);

}

finish = clock();

print("Array sorted by Intersort", a, N, sorted);

cout << "Time to sort array of " << N << " elements by Intersort:";

duration = (double)(finish - start) / CLOCKS\_PER\_SEC;

duration /= shoo\_count;

cout << duration << " seconds" << endl;

fout << endl << "Time to sort array of " << N << " elements by Intersort: "

<< duration << endl;

cout << compares << " compares; " << moves << " moves \n";

fout << compares << " compares; " << moves << " moves \n";

}

//++++++++++++++++ShellSort++++++++++++++++++++++++

if (false) {

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

a[i] = b[i];

moves = 0; // кількість обмінів (копіювань)

compares = 0; // кількість порівнянь ключів елементів

start = clock();

ShellSort(a);

finish = clock();

print("Array sorted by ShellSort", a, N, sorted);

cout << "Time to sort array of " << N << " elements by ShellSort:";

duration = (double)(finish - start) / CLOCKS\_PER\_SEC;

cout << duration << " seconds" << endl;

fout << endl << "Time to sort array of " << N << " elements by ShellSort: "

<< duration << endl;

cout << compares << " compares; " << moves << " moves \n";

fout << compares << " compares; " << moves << " moves \n";

}

//++++++++++++++++ShakerSort+++++++++++++++++++++++

if (false) {

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

a[i] = b[i];

moves = 0; // кількість обмінів (копіювань)

compares = 0; // кількість порівнянь ключів елементів

start = clock();

ShakerSort(a);

finish = clock();

print("Array sorted by ShakerSort", a, N, sorted);

cout << "Time to sort array of " << N << " elements by ShakerSort:";

duration = (double)(finish - start) / CLOCKS\_PER\_SEC;

cout << duration << " seconds" << endl;

fout << endl << "Time to sort array of " << N << " elements by ShakerSort: "

<< duration << endl;

cout << compares << " compares; " << moves << " moves \n";

fout << compares << " compares; " << moves << " moves \n";

}

//++++++++++++++++HeapSort+++++++++++++++++++++++++

if (true) {

start = clock();

for (int shoo = 0; shoo < shoo\_count; shoo++) {

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

a[i] = b[i];

moves = 0; // кількість обмінів (копіювань)

compares = 0; // кількість порівнянь ключів елементів

HeapSort(a);

}

finish = clock();

print("Array sorted by HeapSort", a, N, sorted);

cout << "Time to sort array of " << N << " elements by HeapSort:";

duration = (double)(finish - start) / CLOCKS\_PER\_SEC;

duration /= shoo\_count;

cout << duration << " seconds" << endl;

fout << endl << "Time to sort array of " << N << " elements by HeapSort: "

<< duration << endl;

cout << compares << " compares; " << moves << " moves \n";

fout << compares << " compares; " << moves << " moves \n";

}

//++++++++++++++++MergeSort++++++++++++++++++++++++

if (false) {

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

a[i] = b[i];

moves = 0; // кількість обмінів (копіювань)

compares = 0; // кількість порівнянь ключів елементів

start = clock();

MergeSort(a, 0, N - 1);

finish = clock();

print("Array sorted by MergeSort", a, N, sorted);

cout << "Time to sort array of " << N << " elements by MergeSort:";

duration = (double)(finish - start) / CLOCKS\_PER\_SEC;

cout << duration << " seconds" << endl;

fout << endl << "Time to sort array of " << N << " elements by MergeSort: "

<< duration << endl;

cout << compares << " compares; " << moves << " moves \n";

fout << compares << " compares; " << moves << " moves \n";

}

delete[] a;

delete[] b;

}

void set\_values(long int countm, int sh) {

N = countm;

M = int(log((double)N) / log((double)2) + 1);

shoo\_count = sh;

}

int main() {

fout.open("sort100.txt");

set\_values(100, 100);

last\_main();

fout.close();

fout.open("sort1000.txt");

set\_values(1000, 10);

last\_main();

fout.close();

fout.open("sort10000.txt");

set\_values(10000, 1);

last\_main();

fout.close();

\_getch();

return 0;

}