НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра спеціалізованих комп’ютерних систем

##### КУРСОВА РОБОТА

***з дисципліни "Структури даних і алгоритми"***

Виконав: Стасюк А. В.

#### Група: КB-02

Номер залікової книжки: КВ-0217

#### Допущений до захисту

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2 семестр 2010/2011

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КПІ»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра спеціалізованих комп’ютерних систем

Узгоджено ЗАХИЩЕНА "\_\_"\_\_\_\_\_\_\_\_\_2011р.

Керівник роботи з оцінкою\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_/Марченко О.І./ \_\_\_\_\_\_\_\_ /Марченко О.І./

***Дослідження ефективності способів обходу (використання додаткового одновимірного масива, перетворення координат, безпосередній обхід без додаткових масивів і перетворень координат) при сортуванні на багатовимірних масивах («шейкерне» сортування)***

#### Виконавець роботи:

Стасюк Андрій Вікторович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_2011р.

***ТЕХНІЧНЕ ЗАВДАННЯ***

***на курсову роботу з дисципліни***

***«Структури даних і алгоритми»***

I. Описати принцип та схему роботи досліджуваного методу сортування для одновимірного масиву.

II. Скласти алгоритми сортування в багатовимірному масиві заданим методом, згідно з варіантом, та написати відповідну програму на мові програмування.

Програма повинна задовольняти наступні вимоги:

1. Всі алгоритми повинні бути реалізовані в рамках ОДНІЄЇ програми з діалоговим інтерфейсом для вибору варіантів тестування та виміру часу кожного алгоритму.
2. Одним з варіантів запуску програми має бути режим запуску виміру часу всіх алгоритмів у пакетному режимі, тобто запуск всіх алгоритмів для всіх випадків і побудова результуючої таблиці за наведеним нижче зразком для масиву з заданими геометричними розмірами.
3. При реалізації програми повинні бути використані модулі (unit).
4. Програма повинна мати коментарі для всіх структур даних, процедур та функцій, а також до основних смислових фрагментів алгоритмів.

III. Виконати налагодження та тестування коректності роботи написаної програми.

IV. Провести практичні дослідження швидкодії складених алгоритмів.

V. За результатами досліджень скласти порівняльні таблиці за різними ознаками.

VI. Виконати порівняльний аналіз поведінки заданих алгоритмів за отриманими результатами:

1. Для одномірного масиву відносно загальновідомої теорії.
2. Для багатовимірних масивів відносно результатів для одномірного масиву.
3. Для заданих алгоритмів на багатовимірних масивах між собою.
4. Дослідити вплив різних геометричних розмірів багатовимірних масивів на поведінку алгоритмів та їх взаємовідношення між собою.
5. Для всіх вищезазначених пунктів порівняльного аналізу пояснити, ЧОМУ алгоритми в розглянутих ситуаціях поводять себе саме так, а не інакше.

VII. Зробити висновки за виконаним порівняльним аналізом.

**Варіант № 50**

**Задача**

Впорядкувати окремо кожен переріз тривимірного масиву А [p,m,n] наскрізно по стовпчиках за незменшенням.

**Досліджуваний алгоритм**

Алгоритм сортування №4 методу прямого обміну(«шейкерне» сортування).

**Способи обходу**

1. Переписати елементи заданого двовимірного масиву у додатковий одновимірний масив. Виконати сортування. Повернути результат у початковий масив.

2. Не використовуючи додаткового масиву, виконати сортування перетворюючи один індекс елементів "уявного" вектора у відповідні індекси елементів заданого двовимірного масиву.

3. Виконати сортування, здійснюючи обхід безпосередньо по елементах заданого двовимірного масиву, не використовуючи додаткових масивів і перетворень індексів.**Принцип роботи алгоритму «шейкерного» сортування для одновимірного масива**

Алгоритм «шейкерного» сортування базується на алгоритмі прямого обміну і є його модифікацією. Якщо проаналізувати останній, то можна помітити дві особливості: по-перше, у разі проходу по частині масива без обмінів ця частина вже відсортована і її можна не розглядати у подальшому; по-друге, під час роботи алгоритму найбільший елемент масива «опускається» на останню позицію, а найменший «виштовхується» на першу. Отже, якщо зробити межі розгляду масива динамічними, змінюючи їх при кожному обміні, а напрям самого проходження масивом поперемінно змінювати (зліва направо і навпаки), то отримаємо новий більш ефективний алгоритм, який було названо «шейкерним» сортуванням.

Найкращим випадком для «шейкерного» сортування є прямо відсортований, а найгіршим – обернено відсортований масив. Швидкодія у першому разі має лінійну, а у другому – квадратичну залежність від розміру масива.

Алгоритм на мові програмування можна записати наступним чином:

program ShakerSort;

const n=10;

type

TVector=array[1..n] of integer;

var

A : TVector;

B : integer;

i, L, R, k : word;

begin

for i:=1 to n do read(A[i]);

readln;

L:=1; R:=n; k:=1;

while L<R do begin

for i:=L to R-1 do

if A[i] > A[i+1] then begin

B:=A[i];

A[i]:=A[i+1];

A[i+1]:=B;

k:=i;

end;

R:=k;

for i:=R-1 downto L do

if A[i] > A[i+1] then begin

B:=A[i];

A[i]:=A[i+1];

A[i+1]:=B;

k:=i;

end;

L:=k+1;

end;

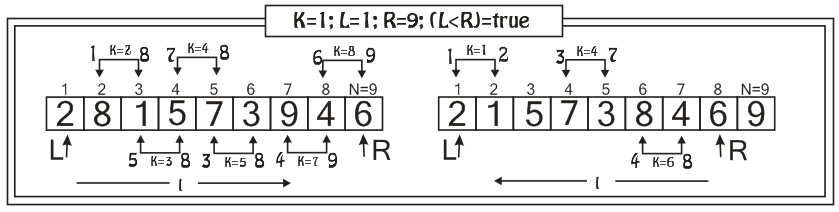
for i:=1 to n do write(A[i]:8);

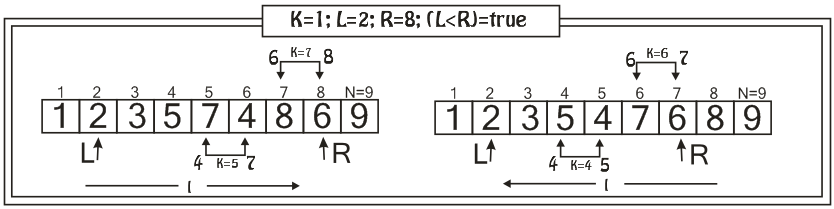
writeln;

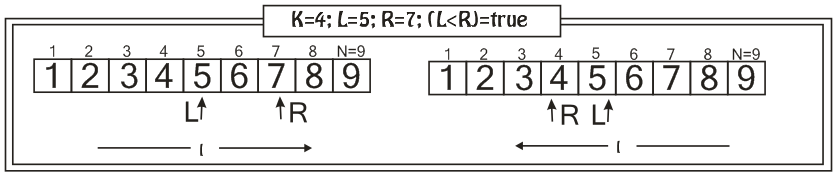
end.

Наведемо схему сортування для конкретного вектора:



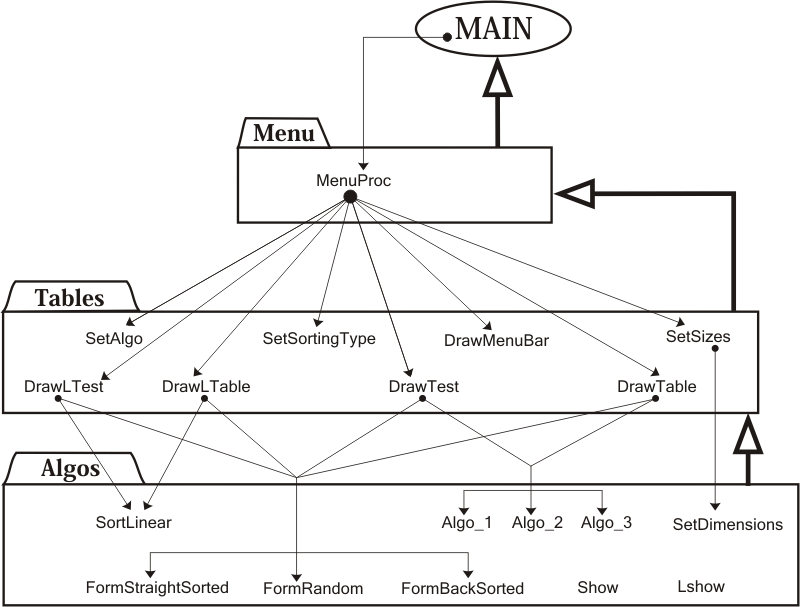


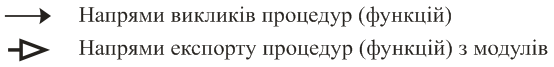






**Структурна схема взаємозв’язків модулів та взаємовикликів процедур (функцій)**

****

****

**Призначення модулів і процедур (функцій)**

*Модуль “Algos”* надає процедури та функції, необхідні для реалізації сортування з різними типами обходу елементів масива, а також для визначення часу роботи кожного з них. Фактично, цей модуль є основною частиною усієї програми. Інтерфейс складається з таких процедур (функцій):

* *procedure LShow* – процедура, додана для налагодження і в кінцевій версії програми ніде не викликається. Виводить значення елементів вектора;
* *procedure Show* – процедура, додана для налагодження і в кінцевій версії програми ніде не викликається. Виводить значення елементів кожного з перерізів масива;
* *procedure SetDimensions(newP, newM, newN, newLen:word)* – встановлює нові геометричні розміри тривимірного масива та вектора, які не можуть перевищувати значень відповідних констант, що описані в інтерфейсі модуля. Перевірка на коректність в цій процедурі не проводиться, уся відповідальність за це покладається на користувача. Параметри (зліва направо): нові кількість перерізів, висота (к-ість рядків) та ширина (к-ість стовпців), довжина вектора відповідно;
* *function SortLinear:longint* – сортує вектор і повертає час роботи алгоритму у сотих долях секунди;
* *function Algo\_1:longint* – сортує тривимірний масив відповідно до задачі варіанта. Обхід елементів проводиться з допомогою додаткового вектора (обхід №1). Повертає час роботи алгоритму у сотих долях секунди;
* *function Algo\_2:longint* – сортує тривимірний масив відповідно до задачі варіанта. Обхід елементів проводиться через перетворення координат (обхід №2). Повертає час роботи алгоритму у сотих долях секунди;
* *function Algo\_3:longint* – сортує тривимірний масив відповідно до задачі варіанта. Обхід елементів проводиться безпосередньо, не використовуючи додаткових масивів і перетворень координат (обхід №3). Повертає час роботи алгоритму у сотих долях секунди;
* *procedure FormStraightSorted* – формує відсортовані тривимірний масив і вектор;
* *procedure FormBackSorted ­–* формує обернено відсортовані тривимірний масив і вектор;
* *procedure FormRandom –* формує невідсортовані (випадкові) тривимірний масив і вектор.

*Модуль “Tables” –* реалізовуєпроцедури для красивого форматованого виводу результатів тестів, в основному, у вигляді таблиць, а також для відображення поточних налаштувань програми (геометричні розміри масивів, упорядкованість їх елементів, алгоритм обходу) і меню користувача. Інтерфейс складається з процедур:

* *procedure DrawTable* – малює таблицю результатів тестування на тривимірному масиві в пакетному режимі;
* *procedure DrawLTable –* малює таблицю результатів тестування на векторі в пакетному режимі;
* *procedure DrawTest –* відображає результати одиничного тесту на тривимірному масиві, використовуючи поточні налаштування алгоритму, упорядкованості та розмірів;
* *procedure DrawLTest –* відображає результати одиничного тесту на векторі, використовуючи поточні налаштування алгоритму, упорядкованості та розміру (довжини);
* *procedure SetSizes(newP, newM, newN, newLen:word)* – допоміжна процедура-посередник, що встановлює поточні геометричні розміри. Не несе відповідальності за коректність значень вхідних параметрів, викликає відповідну процедуру із попереднього модуля. Була введена для збереження рівневої структури взаємозв’язків модулів. Параметри (зліва направо): нові кількість перерізів, висота (к-ість рядків) та ширина (к-ість стовпців) масива, довжина вектора відповідно;
* *procedure SetAlgo(newAlgo:word)* – встановлює поточний алгоритм обходу, зберігаючи його номер у внутрішній змінній модуля. Єдиний параметр – номер алгоритму обходу (можливі коректні значення задані у константах відповідного модуля);
* *procedure SetSortingType(newType:word)* – встановлює поточний тип упорядкованості елементів масива та вектора, зберігаючи його номер у внутрішній змінній модуля. Єдиний параметр – номер типу відсортованості (можливі коректні значення задані у константах відповідного модуля);
* *procedure DrawMenuBar –* відображає у вигляді таблиці зверху екрана меню користувача, поточні налаштування тощо.

*Модуль “Menu”* – реалізує безпосереднє спілкування з користувачем: введення ним вхідних даних, вибір пунктів меню тощо. Це найменший модуль програми, що складається лише з однієї процедури:

* *procedure MenuProc* – забезпечує циклічне опитування користувача і ввід ним даних. Саме ця процедура відповідальна за коректність параметрів, переданих у процедури попереднього модуля.

**Текст модулів та головної програми**

1. *Модуль “Algos”.*

unit algos;

interface

procedure SetDimensions(newP, newM, newN, newLen:word);

{встановлює поточні геометричні розміри}

function SortLinear:longint; {сортує вектор, повертає час сортування}

function Algo\_1:longint; {сортує масив з викор. дод. вектора, повертає час сортування}

function Algo\_2:longint; {сортує масив з перетв. координат, повертає час сортування}

function Algo\_3:longint; {сортує масив з прямим обходом, повертає час сортування}

procedure FormStraightSorted; {формує відсорт. масив і вектор}

procedure FormBackSorted; {формує оберн. відсорт. масив і вектор}

procedure FormRandom; {формує невідсорт. масив і вектор}

procedure Show; {виводить усі перерізи масива}

procedure LShow; {виводить усі елементи вектора}

const MAXM=40; MAXN=40;

MAXP=20; MAXLEN=20000; {константи макс. геометр. розмірів

масивів}

implementation

uses dos;

type TTime=record {тип для замірів часу}

hours:word;

mins:word;

secs:word;

hsecs:word;

end;

type TArrType=integer; {тип елементів масивів}

PArrType=^TArrType;

TLArray=array [1..MAXLEN] of TArrType; {тип для вектора

(розподіл з кучі)}

PLArray=^TLArray;

var M, N, P, Len:word; {змінні для геометричних розмірів

масивів}

A:array [1..MAXP, 1..MAXM, 1..MAXN] of TArrType;

{робочий тривимірний масив}

C:PLArray; {робочий вектор}

procedure FormSorted(straight:boolean); {внутрішня допоміжна

процедура}

{параметр – істина, якщо сортування пряме}

var i, j, k:word;

dv: integer;

value:TArrType;

begin

if straight = true then begin

value:=1;

dv:=1;

end else begin

value:=M\*N;

dv:=-1;

end;

for j:=1 to N do

for i:= 1 to M do begin

for k:= 1 to P do

A[k, i, j]:=value;

value:=value+dv;

end;

if straight = true then value:=1

else value:=Len;

for i:=1 to Len do begin

C^[i]:=value;

value:=value+dv;

end;

end;

procedure LShow;

var i:word;

begin

for i:=1 to Len do write(C^[i], ‘ ‘);

writeln;

end;

procedure Show;

var i, j, k:word;

begin

for k:=1 to P do begin

for i:=1 to M do begin

for j:=1 to N do write(A[k, i, j], ‘ ‘);

writeln;

end;

writeln;

end;

writeln;

end;

procedure SetDimensions(newP, newM, newN, newLen:word);

begin

M:=newM;

N:=newN;

P:=newP;

Len:=newLen;

end;

procedure FormStraightSorted;

begin

FormSorted(true);

end;

procedure FormBackSorted;

begin

FormSorted(false);

end;

procedure FormRandom;

var i, j, k:word;

begin

randomize;

for k:=1 to P do

for i:=1 to M do

for j:=1 to N do

A[k, i, j]:=random(8112);

for i:=1 to Len do C^[i]:=random(8112);

end;

function Algo\_1:longint;

var startTime, finTime:TTime;

B:array [1..MAXN\*MAXM] of longint; {дод. масив}

i, j, k, R, L,

x, {для збереження позиції останнього обміну}

pos:word; {для роботи з додатковим масивом}

tmp:TArrType;

begin

with startTime do GetTime(hours, mins, secs, hsecs);

for k:=1 to P do begin

pos:=1;

for j:=1 to N do {переписуємо елементи по стовпцях у дод.

вектор}

for i:=1 to M do begin

B[pos]:=A[k, i, j];

pos:=pos+1;

end;

L:=1; R:=M\*N; x:=1; {сортуємо дод. лінійний масив

(стандарт. алгоритм)}

while L < R do begin

for i:=L to R-1 do

if B[i] > B[i+1] then begin

tmp:=B[i];

B[i]:=B[i+1];

B[i+1]:=tmp;

x:=i;

end;

R:=x;

for i:=R-1 downto L do

if B[i] > B[i+1] then begin

tmp:=B[i];

B[i]:=B[i+1];

B[i+1]:=tmp;

x:=i+1;

end;

L:=x;

end;

pos:=1;

for j:=1 to N do {повертаємо елементи з вектора у масив}

for i:=1 to M do begin

A[k, i, j]:=B[pos];

pos:=pos+1;

end;

end;

with finTime do GetTime(hours, mins, secs, hsecs);

Algo\_1:=

((finTime.hours\*3600+finTime.mins\*60+finTime.secs)\*100+

finTime.hsecs) -

((startTime.hours\*3600+startTime.mins\*60+startTime.secs)\*100+

startTime.hsecs);

end;

function Algo\_2:longint;

var startTime, finTime:TTime;

i, k, R, L, x:word;

tmp:TArrType;

begin

with startTime do GetTime(hours, mins, secs, hsecs);

for k:=1 to P do begin

L:=1; R:=M\*N; x:=1;

while L < R do begin

for i:=L to R-1 do

if A[k, (i-1)mod(M)+1, (i-1)div(M)+1] >

A[k,(i)mod(M)+1, (i)div(M)+1] then begin

tmp:=A[k, (i-1)mod(M)+1, (i-1)div(M)+1];

A[k, (i-1)mod(M)+1, (i-1)div(M)+1]:=

A[k,(i)mod(M)+1, (i)div(M)+1];

A[k,(i)mod(M)+1, (i)div(M)+1]:=tmp;

x:=i;

end;

R:=x;

for i:=R-1 downto L do

if A[k, (i-1)mod(M)+1, (i-1)div(M)+1] >

A[k,(i)mod(M)+1, (i)div(M)+1] then begin

tmp:=A[k, (i-1)mod(M)+1, (i-1)div(M)+1];

A[k, (i-1)mod(M)+1, (i-1)div(M)+1]:=

A[k,(i)mod(M)+1, (i)div(M)+1];

A[k,(i)mod(M)+1, (i)div(M)+1]:=tmp;

x:=i;

end;

L:=x+1;

end;

end;

with finTime do GetTime(hours, mins, secs, hsecs);

Algo\_2:=

((finTime.hours\*3600+finTime.mins\*60+finTime.secs)\*100+

finTime.hsecs) -

((startTime.hours\*3600+startTime.mins\*60+startTime.secs)\*100+

startTime.hsecs);

end;

function Algo\_3:longint;

var startTime, finTime:TTime;

i, j, k,

rI, {права границя сортування для рядків}

rJ, {права границя сортування для стовпців}

lI, {ліва границя сортування для рядків}

lJ, {ліва границя сортування для стовпців}

xI, {остання позиція обміну, рядкова координата}

xJ, {остання позиція обміну, стовпцева координата}

start, fin: {межі проходу рядками у внутрішньому циклі}

word;

tmp:TArrType;

next:PArrType; {вказівник на наступний за даним елемент}

begin

with startTime do GetTime(hours, mins, secs, hsecs);

for k:=1 to P do begin

lI:=1; lJ:=1; rI:=M; rJ:=N; xI:=1; xJ:=1;

while lJ <= rJ do begin

if (lJ = rJ) and (lI >= rI) then break;

for j:=lJ to rJ do begin

start:=1; fin:=M;

if j = lJ then start:=lI;

if j = rJ then fin:=rI-1;

for i:=start to fin do begin

if i < M then next:=@ A[k, i+1, j]

else next:=@ A[k, 1, j+1];

if A[k, i, j] > next^ then begin

tmp:=A[k, i, j];

A[k, i, j]:=next^;

next^:=tmp;

xI:=i;

xJ:=j;

end;

end;

end;

rI:=xI;

rJ:=xJ;

for j:=rJ downto lJ do begin

start:=1; fin:=M;

if j = lJ then start:=lI;

if j = rJ then fin:=rI-1;

for i:=fin downto start do begin

if i < M then next:=@ A[k, i+1, j]

else next:=@ A[k, 1, j+1];

if A[k, i, j] > next^ then begin

tmp:=A[k, i, j];

A[k, i, j]:=next^;

next^:=tmp;

xI:=i;

xJ:=j;

end;

end;

end;

if xI < M then begin

lI:=xI+1;

lJ:=xJ;

end else begin

lI:=1;

lJ:=xJ+1;

end;

end;

end;

with finTime do GetTime(hours, mins, secs, hsecs);

Algo\_3:=((finTime.hours\*3600+finTime.mins\*60+finTime.secs)\*100+

finTime.hsecs) - ((startTime.hours\*3600+startTime.mins\*60+startTime.secs)\*100+

startTime.hsecs);

end;

function SortLinear:longint; {стандартний алгоритм}

var startTime, finTime:TTime;

L, R,

x, {для збереження позиції останнього обміну}

i:word;

tmp:TArrType;

begin

with startTime do GetTime(hours, mins, secs, hsecs);

L:=1; R:=Len; x:=1;

while L < R do begin

for i:=L to R-1 do

if C^[i] > C^[i+1] then begin

tmp:=C^[i];

C^[i]:=C^[i+1];

C^[i+1]:=tmp;

x:=i;

end;

R:=x;

for i:=R-1 downto L do

if C^[i] > C^[i+1] then begin

tmp:=C^[i];

C^[i]:=C^[i+1];

C^[i+1]:=tmp;

x:=i;

end;

L:=x+1;

end;

with finTime do GetTime(hours, mins, secs, hsecs);

SortLinear:=((finTime.hours\*3600+finTime.mins\*60+finTime.secs)\*

100+finTime.hsecs) - ((startTime.hours\*3600+startTime.mins\*60+startTime.secs)\*100+

startTime.hsecs);

end;

begin

new(C); {розподіляємо вектор з кучі}

SetDimensions(1, 1, 1, 1);

FormRandom;

end.

1. *Модуль “Tables”.*

unit tables;

interface

procedure DrawTable; {виводить таблицю результатів тестування масива у пакетному режимі}

procedure DrawLTable; {виводить таблицю результатів тестування вектора у пакетному режимі}

procedure DrawTest {виводить результати одиничного тесту для масива};

procedure DrawLTest; {виводить результати одиничного тесту для вектора}

procedure SetSizes(newP, newM, newN, newLen:word); {встановлює геометричні розміри масива і вектора}

procedure SetAlgo(newAlgo:word); {встановлює тип алгоритму обходу}

procedure SetSortingType(newType:word); {встановлює тип відсортованості масива і вектора}

procedure DrawMenuBar {виводить заголовок меню зверху екрана};

var algoTypes:array [1..3] of string [24]; {назви типів

алгоритму обходу}

implementation

uses crt, algos;

type ALGOFUNC=function:longint; {вказівник на функцію сортування}

FORMPROC=procedure;

{вказівник на процедуру формування елементів масивів}

var linId, {поточний порядк. номер тестування вектора у пак.

режимі}

id, {поточний порядк. номер тестування масива у

пак. режимі}

testId, {поточний порядк. номер тесту для масива}

lTestId, {поточний порядк. номер тесту для вектора}

M, N, P, Len, {поточні геометр. розміри}

sorting, {поточний тип упорядкованості масивів}

algo: {поточний тип алгоритму обходу}

word;

algFuncs:array [1..3] of ALGOFUNC; {масив функцій алгоритмів обходу}

formProcs:array[1..3] of FORMPROC; {масив процедур формування масивів}

busy:boolean; {статус, істина у разі зайнятості}

menuStrs:array [1..5, 1..4] of string [24];

{таблиця рядків заголовка меню}

sortTypes:array [1..3] of string [16]; {назви типів

упорядков.}

function Bigger(x, y:longint):longint;}

begin

if x > y then Bigger:=x

else Bigger:=y;

end;

procedure SetSizes(newP, newM, newN, newLen:word);

begin

M:=newM;

N:=newN;

P:=newP;

Len:=newLen;

SetDimensions(P, M, N, Len);

DrawMenuBar;

end;

procedure DrawTest;

var tm:longint;

begin

busy:=true;

DrawMenuBar;

testId:=testId+1;

formProcs[sorting];

writeln(' Тест №', testId, 'для масива A[P, M, N]; P=',

P, ' M=', M, ' N=', N, '.');

writeln(' Алгоритм обходу: ', algoTypes[algo],

'. Упорядкування: ',

sortTypes[sorting], '.');

write('Час сортування: ', algFuncs[algo], '.');

writeln;

busy:=false;

DrawMenuBar;

end;

procedure SetAlgo(newAlgo:word);

begin

algo:=newAlgo;

DrawMenuBar;

end;

procedure SetSortingType(newType:word);

begin

sorting:=newType;

DrawMenuBar;

end;

procedure DrawLTest;

var tm:longint;

begin

busy:=true;

DrawMenuBar;

lTestId:=lTestId+1;

formProcs[sorting];

tm:=SortLinear;

writeln(' Тест №', lTestId, 'для вектора C[N]; N=',

Len, '.');

writeln(' Алгоритм обходу: ', algoTypes[algo],

'. Упорядкування: ', sortTypes[sorting], '.');

write('Час сортування: ', tm, '.');

writeln;

busy:=false;

DrawMenuBar;

end;

procedure DrawTable;

var cellStrs:array [1..4, 1..4] of string [32];

{комірки таблиці тестування}

colWids:array [1..4] of word; {ширина кожного стовпця

таблиці}

max,

tabWid: {ширина усієї таблиці}

word;

i, j, k:word;

begin

busy:=true;

DrawMenuBar;

id:=id+1;

cellStrs[1, 1]:=''; cellStrs[1, 2]:='Відсорт.';

cellStrs[1, 3]:='Невідсорт.';

cellStrs[1, 4]:='Обeрн. відсорт.';

cellStrs[2, 1]:='Дод. масив';

cellStrs[3, 1]:='Перетв. коорд.';

cellStrs[4, 1]:='Прямий обхід';

for j:=1 to 3 do begin

for i:=1 to 3 do begin

formProcs[j];

str(longint(algFuncs[i]), cellStrs[i+1, j+1]);

end;

tabWid:=0;

for j:=1 to 4 do begin

max:=0;

for i:=1 to 4 do

if length(cellStrs[i, j]) > max then

max:=length(cellStrs[i, j]);

colWids[j]:=max+2;

tabWid:=tabWid+colWids[j];

end;

writeln(' Таблиця №', id,

' для масива A[P, M, N]; P=', P, ' M=', M, ' N=',

N, '.');

tabWid:=tabWid+5;

for i:=1 to 4 do begin

for j:=1 to tabWid do write('-');

writeln; write('|');

for j:=1 to 4 do begin

for k:=1 to (colWids[j]-length(cellStrs[i, j])-1) do

write(' ');

write(cellStrs[i, j]);

write(' |');

end;

writeln;

end;

for i:=1 to tabWid do write('-');

writeln;

busy:=false;

DrawMenuBar;

end;

procedure DrawLTable;

var cellStrs:array [1..2, 1..3] of string [32]; {змінні такі

ж, як і у попер. проц.}

colWids:array [1..3] of word;

max, tabWid:word;

i, j, k:word;

begin

busy:=true;

DrawMenuBar;

linId:=linId+1;

cellStrs[1, 1]:='Відсорт.'

cellStrs[1, 2]:='Невідсорт.';

cellStrs[1, 3]:='Оберн. відсорт.';

for i:=1 to 3 do begin

formProcs[i];

str(SortLinear, cellStrs[2, i]);

end;

writeln(' Таблиця №', linId, ' для вектора C[N]; N=',

Len, '.');

tabWid:=0;

for i:=1 to 3 do begin

colWids[i]:=Bigger(length(cellStrs[1, i]),

length(cellStrs[2, i]))+2;

tabWid:=tabWid+colWids[i];

end;

tabWid:=tabWid+4;

for i:=1 to 2 do begin

for j:=1 to tabWid do write('-');

writeln; write('|');

for j:=1 to 3 do begin

for k:=1 to (colWids[j]-length(cellStrs[i, j])-1) do

write(' ');

write(cellStrs[i, j], ' |');

end;

writeln;

end;

for i:=1 to tabWid do write('-');

writeln;

busy:=false;

DrawMenuBar;

end;

procedure DrawMenuBar;

var i, j, k, tabWid, max:word;

colWids:array [1..4] of word;

x, y:word;

begin

x:=WhereX; y:=WhereY; {збереж. поточних координат курсора}

if y < 8 then y:=8;

for i:=1 to 8 do begin {очищ. область зверху під

заголовок меню}

GotoXY(1, i);

for j:=1 to 80 do write(' ');

end;

GotoXY(1, 1);

str(P, menuStrs[1, 3]);

menuStrs[1, 3]:='P='+menuStrs[1, 3];

str(M, menuStrs[2, 3]);

menuStrs[2, 3]:='M='+menuStrs[2, 3];

str(N, menuStrs[3, 3]);

menuStrs[3, 3]:='N='+menuStrs[3, 3];

menuStrs[4, 3]:='';

str(Len, menuStrs[5, 3]);

menuStrs[5, 3]:='Len='+menuStrs[5, 3];

menuStrs[1, 4]:='Упорядк.: '+sortTypes[sorting];

menuStrs[2, 4]:='Oбхід: '+algoTypes[algo];

menuStrs[3, 4]:='';

if busy = true then menuStrs[4, 4]:='Працюю...'

else menuStrs[4, 4]:='Вільний';

menuStrs[4, 4]:='Статус: '+menuStrs[4, 4];

menuStrs[5, 4]:='';

tabWid:=0;

for j:=1 to 4 do begin

max:=0;

for i:=1 to 5 do

if length(menuStrs[i, j]) > max then

max:=length(menuStrs[i, j]);

colWids[j]:=max+2;

tabWid:=tabWid+colWids[j];

end;

tabWid:=tabWid+3;

for i:=1 to tabWid do write('=');

writeln;

for i:=1 to 5 do begin

write('| ');

for j:= 1 to 4 do begin

write(menuStrs[i, j]);

for k:=1 to (colWids[j]-length(menuStrs[i, j])-1) do

write(' ');

if (j=2) or (j=4) then write('|');

write(' ');

end;

writeln;

end;

for i:=1 to tabWid do write('=');

GotoXY(x, y); {повертаємо курсор у стару позицію}

end;

begin

algFuncs[1]:=Algo\_1;

algFuncs[2]:=Algo\_2;

algFuncs[3]:=Algo\_3;

formProcs[1]:=FormStraightSorted;

formProcs[2]:=FormRandom;

formProcs[3]:=FormBackSorted;

menuStrs[1, 1]:='1-Розміри';

menuStrs[2, 1]:='2-Алг. обходу'

menuStrs[3, 1]:='3-Відсорт.';

menuStrs[4, 1]:='4-Невідсорт.';

menuStrs[5, 1]:='5-Оберн. відсорт.';

menuStrs[1, 2]:='6-Тест. масив';

menuStrs[2, 2]:='7-Тест. вектор';

menuStrs[3, 2]:='8-Пак. режим (масив)';

menuStrs[4, 2]:='9-Пак. режим (вектор)';

menuStrs[5, 2]:='0-Вихід';

sortTypes[1]:='Відсорт.';

sortTypes[2]:='Невідсорт.';

sortTypes[3]:='Оберн.';

algoTypes[1]:='Дод. масив';

algoTypes[2]:='Перет. коорд.';

algoTypes[3]:='Прямий обхід';

id:=0;

linId:=0;

testId:=0;

lTestId:=0;

busy:=false;

algo:=1;

sorting:=2;

SetSizes(1, 1, 1, 1);

end.

1. *Модуль “Menu”.*

unit menu;

interface

procedure MenuProc; {забезпечує циклічне опитування користувача і введення ним даних; слідкує за коректністю введених даних}

implementation

uses crt, tables, algos;

procedure MenuProc;

var i:word;

maxes, {масив макс. значень геометр. розмірів масивів}

sizes: {масив введених значень геометр. розмірів масивів}

array [1..4] of word;

names:array [1..4] of string [4]; {масив імен відпов.

геометр. розмірів масивів}

sel:char; {символ користувацького вибору}

begin

clrscr;

DrawMenuBar;

GotoXY(1, 9);

maxes[1]:=MAXP; maxes[2]:=MAXM;

maxes[3]:=MAXN; maxes[4]:=MAXLEN;

names[1]:='P'; names[2]:='M';

names[3]:='N'; names[4]:='Len';

sel:='0';

repeat

if (sel >= '3') and (sel <= '9') then begin {перевірка потрібна для форматування

виводу (дод. відступ)}

writeln;

DrawMenuBar;

end;

write('Ваш вибір: ');

sel:=readkey;

writeln(sel); DrawMenuBar;

case sel of

'1': begin

writeln(' \*\*\*\*\* Налаштування розмірів \*\*\*\*\*');

DrawMenuBar;

for i:=1 to 4 do repeat

write(names[i], '='); read(sizes[i]);

DrawMenuBar;

if sizes[i] > maxes[i] then begin

writeln('Занадто велике значення!');

DrawMenuBar;

end;

until sizes[i] <= maxes[i];

writeln;

SetSizes(sizes[1], sizes[2], sizes[3], sizes[4]);

end;

'2': begin

writeln(' \*\*\*\*\* Вибір алгоритму обходу \*\*\*\*\*');

DrawMenuBar;

for i:=1 to 3 do writeln(i, '-', algoTypes[i]);

DrawMenuBar;

repeat

sel:=readkey;

if not ((sel >= '1') and (sel <= '3')) then begin

writeln('Неправильне значення!');

DrawMenuBar;

end;

until ((sel >= '1') and (sel <= '3'));

SetAlgo(ord(sel)-ord('0'));

sel:='3';

end;

'3': SetSortingType(1);

'4': SetSortingType(2);

'5': SetSortingType(3);

'6': DrawTest;

'7': DrawLTest;

'8': DrawTable;

'9': DrawLTable;

'0': begin

repeat

write('Вийти? (y,n) ');

sel:=readkey;

writeln(sel); DrawMenuBar;

if (sel <> 'y') and (sel <> 'n') then begin

writeln('Лише ''y''  або ''n''!');

DrawMenuBar;

end;

until not ((sel <> 'y') and (sel <> 'n'));

writeln; DrawMenuBar;

end;

else begin

writeln('Неправильне значення!');

sel:='3'; {щоб зробити відступ (див. зверху) + страховка

на випадок sel=’y’}

DrawMenuBar;

end;

end;

until sel='y';

end;

begin end.

1. *Головна програма.*

program main;

uses menu, crt;

begin

clrscr;

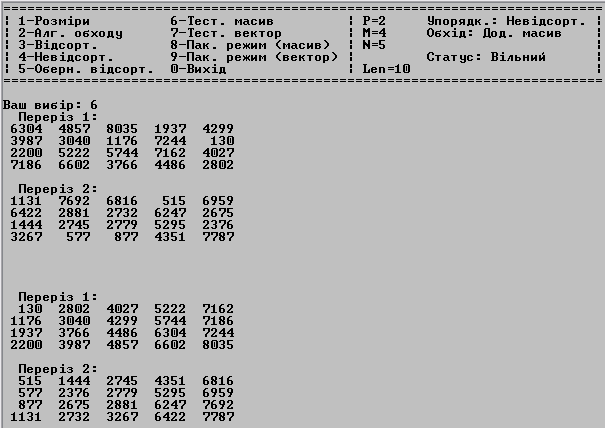
MenuProc;

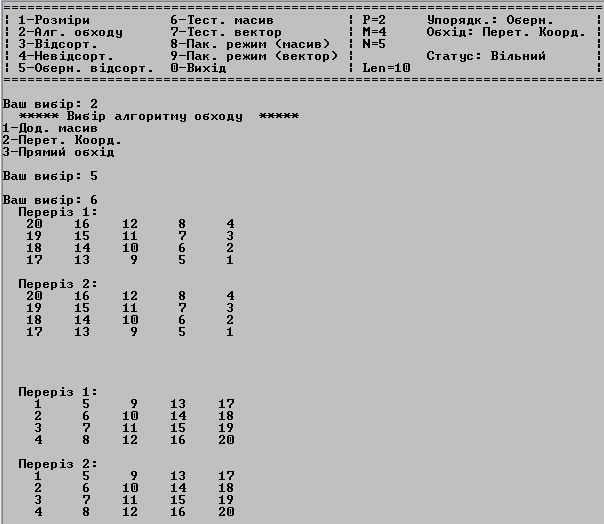
end.

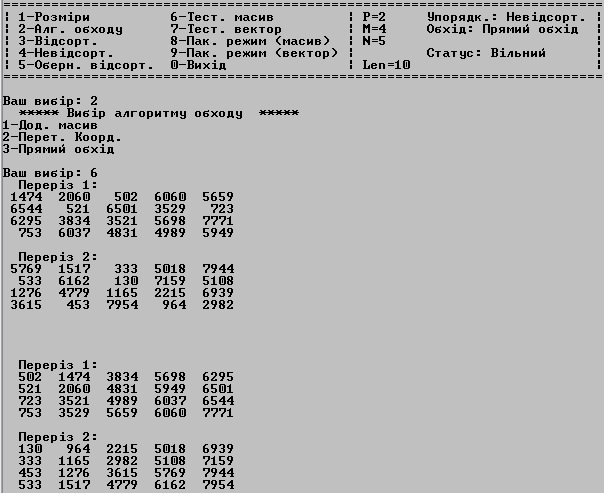
**Тестування та налагодження програми**

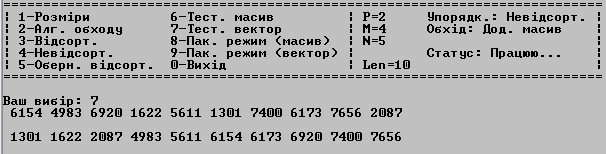
Для налагодження програми у модуль *Algos* були додані процедури *Show* та *LShow*, які виконують вивід масива та вектора відповідно. Дані процедури увійшли до кінцевого варіанту програми, проте ніде не викликаються.

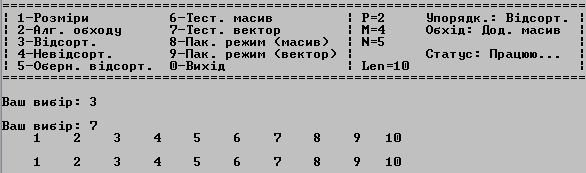
При налагодженні *Show* і *LShow* викликались до і після кожного сортування, дані про час виконання не виводились. Тестування проводилось для масива *A[2, 4, 5]* та вектора *C[10].* Результати показані на малюнках. Тип алгоритму обходу і упорядкованості елементів можна побачити у заголовку меню.

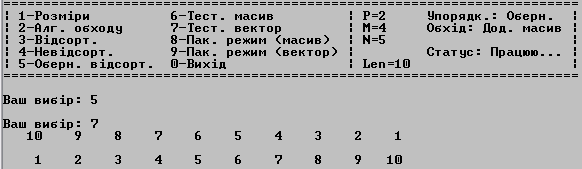
**

**

**

**

**

**

Отже, як можна переконатися, програма коректно формує масиви, а також сортує їх відповідно до умови задачі, використовуючи для обходу елементів різні типи алгоритмів.

**Результати тестування швидкодії алгоритмів**

1. *Тестування стандартного алгоритму для вектора*. Часові показники при сортуванні вектора різних розмірів наведені у таблицях нижче (заміри – у сотих долях секунди). Далі наведено графік залежності швидкодії від довжини масива для випадків не відсортованого та обернено відсортованого векторів. Час роботи алгоритму для прямо відсортованого вектора майже в усіх випадках дорівнює нулю, тому немає сенсу показувати залежність графічно.

|  |  |  |
| --- | --- | --- |
| Таблиця №1 для вектора C[N]; N=200 | | |
| Відсорт. | Невідсорт. | Оберн. відсорт. |
| 0 | 203 | 340 |
| Таблиця №2 для вектора C[N]; N=300 | | |
| Відсорт. | Невідсорт. | Оберн. відсорт. |
| 0 | 423 | 775 |

|  |  |  |
| --- | --- | --- |
| Таблиця №3 для вектора C[N]; N=400 | | |
| Відсорт. | Невідсорт. | Оберн. відсорт. |
| 0 | 796 | 1379 |

|  |  |  |
| --- | --- | --- |
| Таблиця №4 для вектора C[N]; N=600 | | |
| Відсорт. | Невідсорт. | Оберн. відсорт. |
| 0 | 1736 | 3103 |

|  |  |  |
| --- | --- | --- |
| Таблиця №5 для вектора C[N]; N=900 | | |
| Відсорт. | Невідсорт. | Оберн. відсорт. |
| 6 | 3971 | 6981 |

|  |  |  |
| --- | --- | --- |
| Таблиця №6 для вектора C[N]; N=1200 | | |
| Відсорт. | Невідсорт. | Оберн. відсорт. |
| 11 | 6920 | 12413 |

1. *Тестування алгоритмів обходу при сортуванні тривимірного масива*. Результати тестування знову ж таки приведені у таблицях (заміри – у сотих долях секунди).

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця №1 для масива A[P, M, N]; P=2, M=20, N=20 | | | |
|  | Відсорт. | Невідсорт. | Оберн. відсорт. |
| Дод. масив | 22 | 1818 | 3251 |
| Пертв. коорд. | 16 | 5058 | 8964 |
| Прямий обхід | 11 | 2895 | 4916 |

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця №2 для масива A[P, M, N]; P=3, M=10, N=20 | | | |
|  | Відсорт. | Невідсорт. | Оберн. відсорт. |
| Дод. масив | 16 | 670 | 1219 |
| Пертв. коорд. | 16 | 1873 | 3356 |
| Прямий обхід | 5 | 1154 | 1862 |

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця №3 для масива A[P, M, N]; P=2, M=20, N=30 | | | |
|  | Відсорт. | Невідсорт. | Оберн. відсорт. |
| Дод. масив | 33 | 4124 | 7310 |
| Пертв. коорд. | 22 | 11139 | 20191 |
| Прямий обхід | 16 | 6619 | 11067 |

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця №4 для масива A[P, M, N]; P=2, M=30, N=20 | | | |
|  | Відсорт. | Невідсорт. | Оберн. відсорт. |
| Дод. масив | 32 | 4163 | 7311 |
| Пертв. коорд. | 22 | 11413 | 20190 |
| Прямий обхід | 16 | 6503 | 11035 |

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця №5 для масива A[P, M, N]; P=3, M=40, N=10 | | | |
|  | Відсорт. | Невідсорт. | Оберн. відсорт. |
| Дод. масив | 33 | 2703 | 4878 |
| Пертв. коорд. | 28 | 7557 | 13446 |
| Прямий обхід | 16 | 4389 | 7349 |

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця №6 для масива A[P, M, N]; P=6, M=10, N=40 | | | |
|  | Відсорт. | Невідсорт. | Оберн. відсорт. |
| Дод. масив | 66 | 5559 | 9754 |
| Пертв. коорд. | 49 | 15346 | 26897 |
| Прямий обхід | 38 | 8981 | 14885 |

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця №7 для масива A[P, M, N]; P=1, M=30, N=40 | | | |
|  | Відсорт. | Невідсорт. | Оберн. відсорт. |
| Дод. масив | 33 | 8321 | 14599 |
| Пертв. коорд. | 22 | 22327 | 40403 |
| Прямий обхід | 17 | 12749 | 22075 |

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця №8 для масива A[P, M, N]; P=6, M=20, N=10 | | | |
|  | Відсорт. | Невідсорт. | Оберн. відсорт. |
| Дод. масив | 39 | 1411 | 2444 |
| Пертв. коорд. | 28 | 3707 | 6712 |
| Прямий обхід | 16 | 2224 | 3686 |

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця №9 для масива A[P, M, N]; P=20, M=40, N=2 | | | |
|  | Відсорт. | Невідсорт. | Оберн. відсорт. |
| Дод. масив | 44 | 786 | 1323 |
| Пертв. коорд. | 33 | 1999 | 3554 |
| Прямий обхід | 28 | 1192 | 1961 |

|  |  |  |  |
| --- | --- | --- | --- |
| Таблиця №10 для масива A[P, M, N]; P=20, M=2, N=40 | | | |
|  | Відсорт. | Невідсорт. | Оберн. відсорт. |
| Дод. масив | 50 | 791 | 1329 |
| Пертв. коорд. | 33 | 2005 | 3554 |
| Прямий обхід | 28 | 1258 | 2115 |

З табл. №3 та табл. №4 бачимо: за однакових значення *P* і добутку *M\*N* результати майже однакові, незалежно від конкретних *M* і *N* (виняток становить сортування масивів з використанням прямого обходу, висота і ширина перерізів яких суттєво відрізняються – табл. 9 і табл. 10). А табл. №5 й табл. №6, окрім цього, вказують ще й на той цілком закономірний факт, що параметр *P* є лише коефіцієнтом, а, значить, за умови *M\*N=const*, залежність швидкодії від *P* має лінійний характер. Усе вищесказане підводить до думки, що цілком правомірно досліджувати залежність: . Спираючись на такий висновок побудуємо відповідні графіки:

**Порівняльний аналіз алгоритмів**

І. Результати для *сортування одновимірного масива* цілком відповідають *загальній теорії*.

На відсортованому масиві час сортування майже завжди був нульовим, а тому можна стверджувати, що, по-перше, цей випадок дійсно є найкращим для заданого алгоритму і, по-друге, швидкодія має практично лінійну залежність від довжини вектора.

Для обернено відсортованого масива залежність часу від розміру різко відрізняється від лінійної і, судячи з швидкості росту значень по вертикальній осі графіка, можна стверджувати, що вона наближається до квадратичної. Така ситуація справді є найгіршою для алгоритму.

На випадково сформованому масиві алгоритм дав проміжні результати. Залежність аж ніяк не можна назвати лінійною, проте ріст часу з ростом довжини вектора не був настільки стрімким, як у випадку обернено відсортованого масива.

*ІІ. Сортування тривимірного масива відносно сортування вектора* практично нічим у характері швидкодії не відрізняється.

Як було показано вище, головний параметр, що визначає час роботи алгоритмів усіх типів, – добуток *M\*N*, який, фактично, є аналогом довжини для вектора, а розмір *P* відіграє лише роль коефіцієнта і не впливає на порядок залежності.

Винятком є лише сортування перерізів із суттєво різними геометричними розмірами при прямому обході. Така поведінка є закономірною. Для інших видів обходів різниця між геометричними розмірами перерізів неважлива, оскільки вони працюють або з реальним, або з уявним, та все ж вектором *M\*N.* Алгоритм же прямого обходу працює з масивом напряму і для нього краще у плані швидкодії аби переріз був «вищим», ніж «ширшим». Це пов’язано з тим, що для даної задачі сортування «наступним» елементом у більшості випадків є елемент того ж стовпця, але на рядок нижче, і лише в «крайовому» випадку «наступний» елемент знаходиться в іншому стовпці. Робота з переходу на новий стовпець вимагає певного часу. У «широких» перерізів таких переходів набагато більше, ніж у «високих», а, отже, і додаткового витраченого часу на перехід між стовпцями більше.

Більший час сортування тривимірного масива пояснюється деякими додатковими операціями, що виконуються під час роботи алгоритмів. Для обходу з використанням додаткового одновимірного масива – це *О(M\*N\*P)* додаткових присвоєнь і інкрементів (декрементів), необхідних для копіювання елементів з одного масива в інший. При обході з перетворенням координат виконуються додаткові «важкі» арифметичні операції (цілочисельне ділення, взяття остачі, множення, додавання). При прямому обході виконуються «зайві присвоєння» (вказівник на наступний елемент, різного роду межі роботи циклів – див. лістинг) та порівняння (наприклад, для продовження зовнішнього циклу з передумовою необхідно зробити 2 перевірки, а не одну тощо – див. лістинг).

*ІІІ.* Як можна помітити на графіках, найкращий алгоритм сортування – з використанням додаткового вектора. Найгіршою ідеєю є перетворення координат.

Алгоритм з використанням додаткового вектора програє іншим лише у випадку відсортованого масива. Це пояснюється тим, що він виконує копіювання елементів незалежно від вхідних даних, навіть якщо жодних обмінів виконувати не потрібно (а як відомо, операція присвоєння є досить тривалою). Другий та третій алгоритми ж закінчують свою роботу на першій ітерації зовнішнього циклу з передумовою і, за рахунок цього, «обігрують» перший за швидкодією, незважаючи на усі «важкі» операції, що їм доводиться виконувати. Проте, вже у випадку не відсортованого масива, обхід з використанням додаткового вектора стає найшвидшим, оскільки час на початкові і кінцеві копіювання елементів – малий у порівнянні з часом на виконання основної частини алгоритму. Єдиним недоліком є використання додаткової пам’яті для розміщення вектора, яка може досягати значних розмірів.

Обхід через перетворення координат у будь-якому випадку, зважаючи на результати дослідження, не рекомендується до використання. Він потребує виконання занадто великої кількості тривалих арифметичних операцій, а тому, навіть на відсортованому масиві дає гірший результат, ніж прямий обхід елементів.

Третій алгоритм дає кращий результат, ніж перший, лише тоді, коли масив відсортований. Проблема у багатьох додаткових присвоюваннях, порівняннях, перевірках, операціях з вказівником тощо. Проте даний алгоритм прямого обходу не потребує великої кількості додаткової пам’яті (на рівні локальних змінних, а не цілих масивів). Також цей алгоритм, як видно, швидший за алгоритм з перетвореннями координат. Це пояснюється тим, що, по-перше, арифметичні операції ділення/взяття остачі і множення «важчі» за операції присвоєння чи, тим більше, порівняння (останнє виконується дуже швидко), а, по-друге, досить імовірно, що необхідні для роботи програми фрагменти оперативної пам’яті знаходяться у власній внутрішній швидкій пам’яті процесора (кеші) і те ж присвоювання чи «розіменування» вказівника не вимагатиме запиту до повільної відносно частоти роботи процесора оперативної пам’яті.

**Висновок**

В процесі виконання даної курсової роботи була досліджена швидкодія різних методів обходу елементів тривимірного масива при його сортуванні. На основі даних про час роботи кожного з них були побудовані відповідні графіки і складені таблиці, а тому можемо зробити деякі висновки.

Результатами досліджень, показали, що найшвидшим типом обходу є обхід з використанням додаткового вектора. Він дає найкращі результати для випадків не відсортованого та обернено відсортованого масивів. Програшем у випадку відсортованого масива можна знехтувати, оскільки такі масиви зустрічаються надзвичайно рідко, а сортування все одно закінчується досить швидко.

Незважаючи на усі переваги першого типу обходу, він потребує багато додаткової пам’яті, а тому у системах з нестачею пам’яті використовуватись не може. Замість нього пропонується використовувати прямий обхід масива, що дає кращі результати у порівняння з перетвореннями координат і найкращі у випадку відсортованого масива.

Алгоритм обходу з перетвореннями координат, незважаючи на свою простоту та «економне» використання пам’яті, не в стані конкурувати із двома іншими за швидкодією й може використовуватись лише на невеликих масивах в системах не критичних до часу виконання або для учбових цілей.

Отже, ще раз було підтверджено «золоте» правило про те, що виграти можливо або у використанні пам’яті, або у швидкодії, або у простоті реалізації.

**Список джерел**

1. Седжвік Р. Фундаментальні алгоритми на С++. – К.: ДіаСофт, 2001. – С. 261-263.
2. Марченко О., Марченко Л. Turbo Pascal 7.0. Базовий курс. – К.: ВЕК, 2004. – С. 359-362.
3. Конспект лекцій з дисципліни «Структури даних та алгоритми».