

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ ТА
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

КУРСОВА РОБОТА
з дисципліни «Структури даних та алгоритми»

Виконав:
Горпинич-Радуженко І.О.
Група КВ-41
Залікова книжка: КВ-4106

Допущений до захисту

2 семестр 2014/2015 навч. року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ ТА
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

<i>Узгоджено</i>	<i>Захищено</i>
<i>Керівник роботи</i>	«__» _____ 20__ р.
_____/Марченко	з оцінкою _____
O.I./	_____/Марченко
	O.I./

**Дослідження ефективності методів сортування
на багатовимірних масивах:**

**Гібридний алгоритм "вставка – обмін", здійснюючи обхід з
використанням додаткового одновимірного масиву;**

**Гібридний алгоритм "вставка – обмін", здійснюючи обхід
перетворюючи один індекс елементів "уявного" вектора у
відповідні індекси елементів заданого двовимірного масиву;**

**Гібридний алгоритм "вставка – обмін", здійснюючи обхід
безпосередньо по елементах заданого двовимірного масиву**

Виконавець роботи: _____

Гопинич-Радуженко Іван Олександрович
_____ 20__ р.

ПЛАН РОБОТИ

1. Технічне завдання.
2. Теоретичні положення.
3. Схема імпорту/експорту модулів та структурна схема взаємовикликів процедур та функцій.
4. Опис призначення процедур та функцій.
5. Текст програми.
6. Тести.
7. Таблиці виміру часу.
8. Порівняльний аналіз.
9. Висновки.

Технічне завдання на курсову роботу

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 одновимірного масиву відносно загальновідомої теорії.

2. Для багатовимірних масивів відносно результатів для одновимірного 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 поведуть себе саме так, а не інакше.

VII. Зробити висновки за виконаним порівняльним аналізом.

Варіант №106

Задача

Впорядкувати окремо кожен переріз тривимірного масиву $A[p,m,n]$ наскрізно по рядках за незменшенням.

Досліджувані методи та алгоритми

1. Гібридний алгоритм №17 "вставка – обмін".

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.

Випадки дослідження

1. Елементи початкового масиву впорядковані відповідно до заданої ознаки.
2. Елементи початкового масиву неупорядковані.
3. Елементи початкового масиву впорядковані за протилежно заданою ознакою.

Окремо з кожним випадком дослідження необхідно провести дослідження залежності часу виконання алгоритму від розмірів масиву:

- а) Кількість перерізів: $p = \text{const} = 4$;
Розміри перерізу:
- I. $m=4; n=400$;
 - II. $m=40; n=40$;
 - III. $m=400; n=4$;

b) Розміри масиву: $m = \text{const} = 20$; $n = \text{const} = 20$;

Кількість перерізів:

- I. $p=1$;
- II. $p=2$;
- III. $p=4$;
- IV. $p=8$;
- V. $p=16$;
- VI. $p=32$;

c) Кількість перерізів: $p = \text{const} = 4$;

Розміри перерізу:

- I. $m = n = 4$;
- II. $m = n = 8$;
- III. $m = n = 16$;
- IV. $m = n = 32$;

ТЕОРЕТИЧНІ ПОЛОЖЕННЯ

Розглянемо роботу гібридного алгоритму №17 "вставка – обмін" на прикладі сортування одновимірного масиву (вектора).

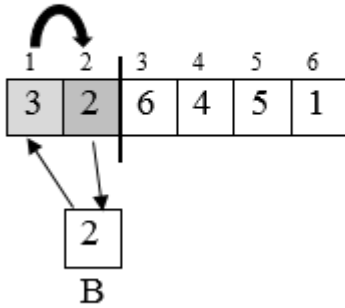
Принцип роботи:

1. Починаючи з 2 і до n {for $i := 2$ to n }:
2. Змінна місця проходу j спочатку має значення лічильника, тобто $j := i$;
3. Якщо змінна місця проходу більша за одиницю і якщо даний елемент $A[j]$ менший за попередній $A[j-1]$, то:
4. Запам'ятовуємо елемент $A[j]$ як B ;
5. На місце елемента $A[j]$ стає $A[j-1]$, а елементу $A[j-1]$ присвоюється значення B , та місце проходу зменшуємо на 1 .
6. Якщо змінна місця проходу дорівнює одиниці або ж даний елемент більший за попередній, то переходимо до (1).

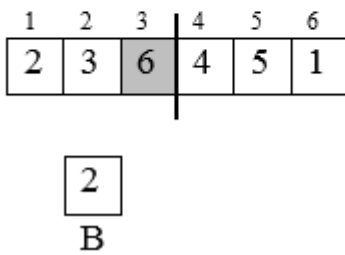
З початку кожної ітерації циклу **for** з індексом j масив складається з двох частин. Підмасив $A[1 \dots j-1]$ складається з елементів, які початково знаходились у $A[1 \dots j-1]$, але тепер розташовані в відсортованому порядку, а елементи $A[j+1 \dots n]$ відповідають не відсортованій частині.

Схема роботи алгоритму

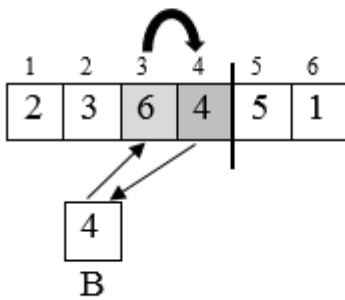
1. for $i:=2$ to n do ($i=2$)
 $j:=i$ ($j=2$)



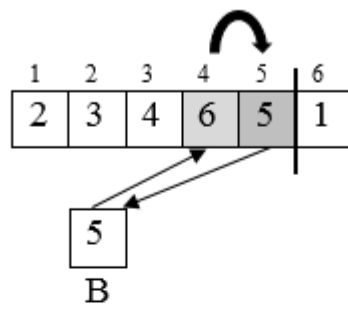
2. for $i:=2$ to n do ($i=3$)
 $j:=i$ ($j=3$)



3. for $i:=2$ to n do ($i=4$)
 $j:=i$ ($j=4$)

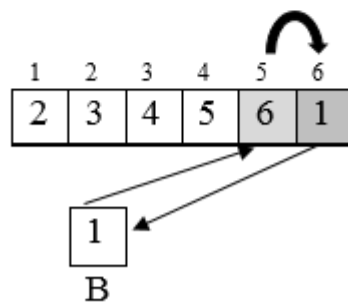


4. for $i:=2$ to n do ($\underline{i=5}$)
 $\underline{j:=i}$ ($\underline{j=5}$)



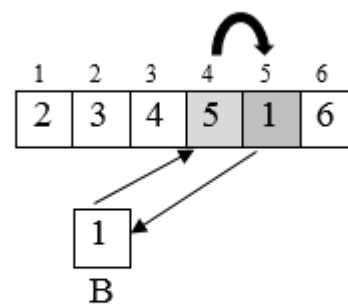
5. for $i:=2$ to n do ($\underline{i=6}$)
 $\underline{j:=i}$ ($\underline{j=6}$)

a)



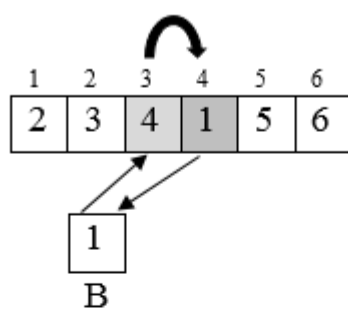
$\underline{j=j-1=5};$

b)



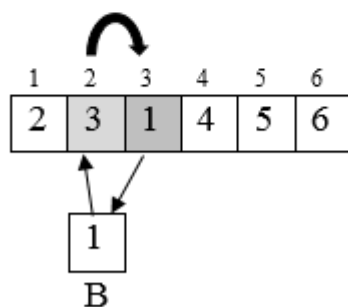
$\underline{j=j-1=4};$

c)



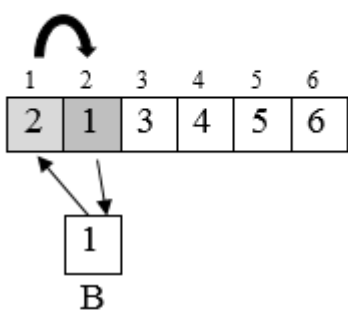
$j=j-1=3;$

d)



$j=j-1=2;$

e)



$j=j-1=1;$

Кінцевий результат:

1	2	3	4	5	6
1	2	3	4	5	6

СТРУКТУРНІ СХЕМИ ПРОГРАМИ

Схема імпорту/експорту модулів та структурна схема взаємовикликів
процедур і функцій

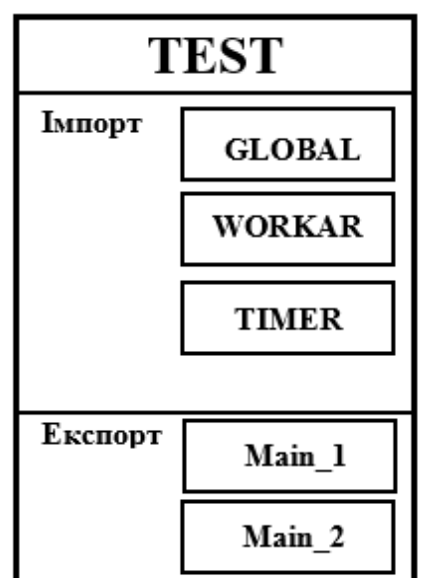
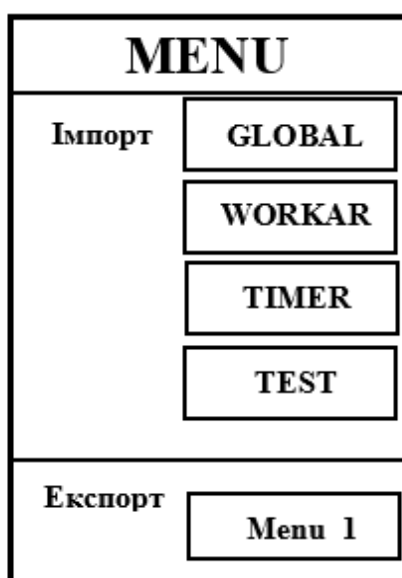
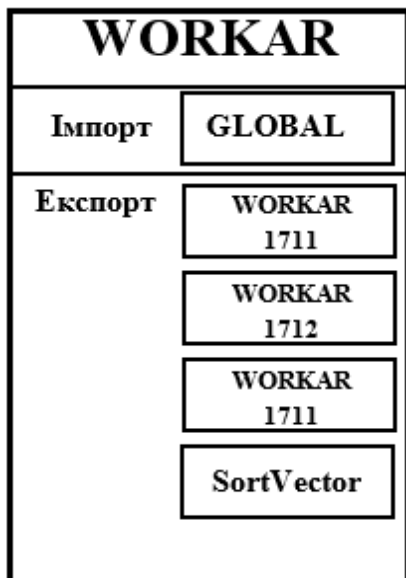
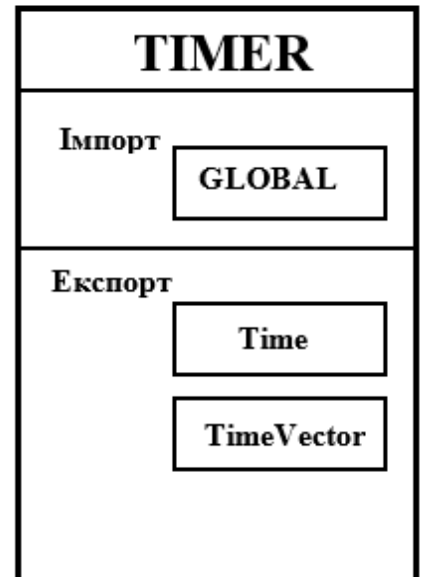
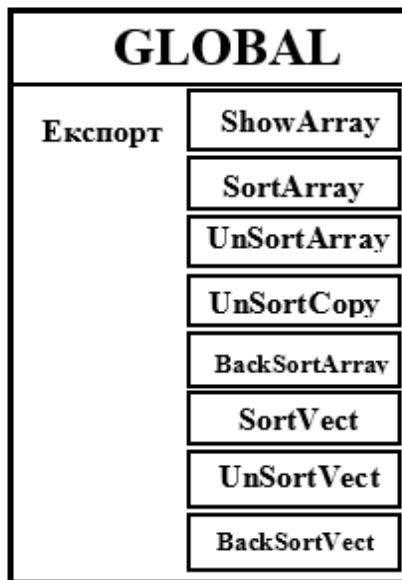
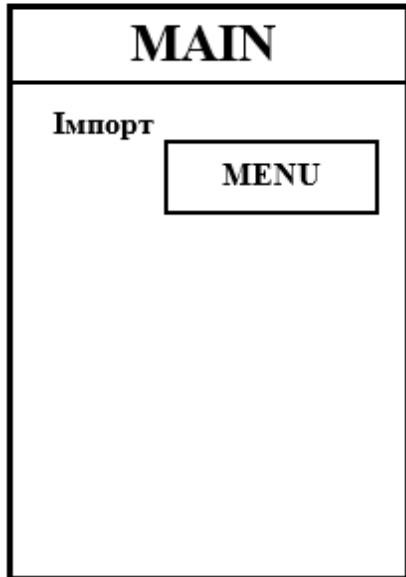
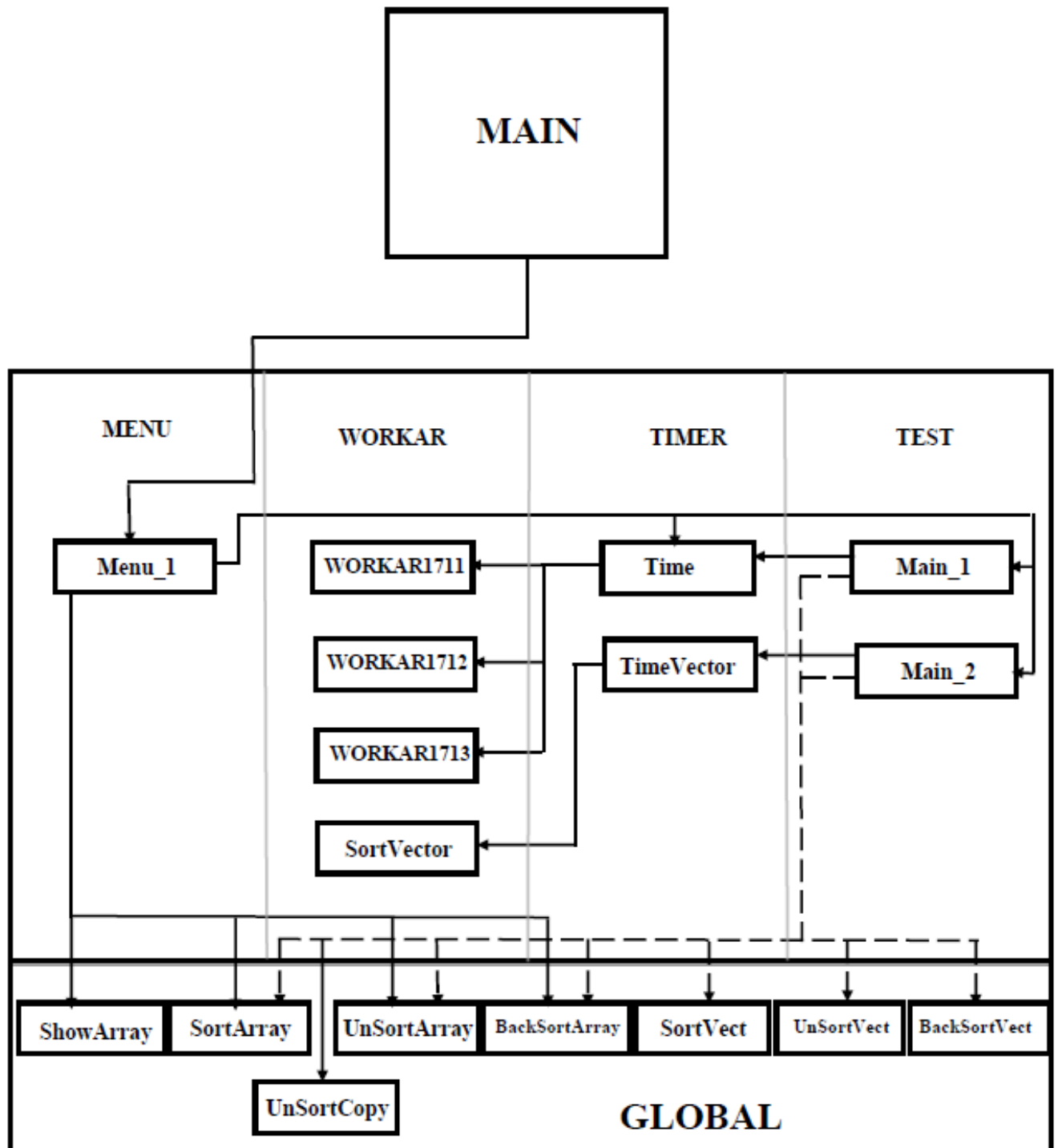


Схема викликів процедур та функцій



Опис призначення модулів, процедур і функцій

Головна програма MAIN:

Використання головної програми MAIN насамперед призначено для використання поняття принципу приховування інформації.

1. Модуль MENU:

Модуль MENU призначений для виклику головного меню;

- **procedure Menu_1:**

процедура відповідає за виклик необхідних процедур та функцій, необхідних для виконання тестування різних випадків за допомогою зрозумілого інтерфейсу.

2. Модуль GLOBAL:

Модуль GLOBAL призначений для змінних та процедур, які використовуються на багатьох етапах роботи програми, насамперед це процедури заповнення масиву та вектора;

- **procedure ShowArray:**

виводить на екран збережений у пам'яті масив;

- **procedure SortArray:**

заповнює масив впорядкованими значеннями: (від 1 до $m*n*p$);

- **procedure UnSortArray:**

заповнює масив рандомними значеннями;

- **procedure UnSortCopy:**

копіює рандомний масив, для того, щоб усі вимірювання відбувалися з однаковими значеннями;

- **procedure BackSortArray:**

заповнює масив обернено впорядкованими значеннями:

(від $m*n*p$ до 1);

- **procedure SortVect:**

заповнює вектор впорядкованими значеннями: (від 1 до $m*n$);

- **procedure UnSortVect:**

заповнює вектор рандомними значеннями;

- **procedure BackSortVect:**

заповнює вектор обернено впорядкованими значеннями:

(від $m*n$ до 1);

3. Модуль TIMER:

Модуль TIMER призначений для функцій, які підраховують час роботи алгоритмів;

- **function ResTime:**

знаходить різницю між часом початку роботи алгоритму та часом закінчення;

- **function Time:**

запам'ятовує час початку роботи алгоритму та час закінчення, та за допомогою функції ResTime знаходить точний час роботи (для алгоритмів, які працюють з масивом);

- **function TimeVector:**

запам'ятовує час початку роботи алгоритму та час закінчення, та за допомогою функції ResTime знаходить точний час роботи (для алгоритмів, які працюють з вектором);

4. Модуль TEST:

Модуль TEST призначений для процедур, які виконують пакетний замір часу;

- **procedure Main_1:**

знаходить середнє значення часу роботи алгоритмів з масивами, та виводить значення часу виконання роботи алгоритму з вектором розмірами одного перерізу, та помножене на кількість перерізів;

- **procedure Main_2:**

знаходить значення часу роботи алгоритмів з масивами, та виводить значення часу виконання роботи алгоритму з вектором розмірами одного перерізу, та помножене на кількість перерізів;

5. Модуль WORKAR:

Модуль WORKAR призначений для процедур сортування;

- **procedure WORKAR1711:**

перепис елементів перерізу масиву у додатковий одновимірний масив, виконання сортування, повернення результату у переріз масиву;

- **procedure WORKAR1712:**

не використовуючи додаткового масиву, виконується сортування перетворюючи один індекс елементів "уявного" вектора у відповідні індекси елементів заданого перерізу масиву;

- **procedure WORKAR1713:**

виконується сортування, здійснюючи обхід безпосередньо по елементах заданого двовимірного масиву, не використовуючи додаткових масивів і перетворень індексів;

- **procedure SortVector:**

виконується сортування вектора.

ТЕКСТ ПРОГРАМИ

Головна програма MAIN

```
program main;
uses MENU,crt; {Виклик необхідних модулів}
begin
  repeat {Повернення меню на головну сторінку, доки на головній
сторінці меню не буде натиснений Esc}
    clrscr;
    Menu_1; {Виклик процедури меню}
    clrscr;
  until keypressed;
end.
```

Модуль MENU

```
Unit MENU;

interface {Описання доступної інформації для інших модулів}

procedure Menu_1;

implementation {Описання прихованої інформації від інших модулів}

uses crt,GLOBAL,WORKAR,TIMER,TEST;

procedure Menu_1;
var ch, ch1:char; {Змінні, необхідні для роботи процедури Menu_1}
    T:integer; {Змінна, яка передає час виконання обраного метода
обходу}
begin
  repeat {Цикл повернення головного меню на початкову сторінку}
    clrscr;
    gotoxy(35,12);
    Writeln('Pressed:');
    gotoxy(35,13);
    Writeln('1 - Sort array');
    gotoxy(35,14);
    Writeln('2 - Unsort array');
    gotoxy(35,15);
    Writeln('3 - Backsort array');
    gotoxy(35,16);
    Writeln('4 - Packed-mode launch');
    gotoxy(35,17);
    Writeln('5 - Average packed-mode launch');
    gotoxy(35,18);
    Writeln('Esc - Exit');
    ch:=readkey;
  case ch of {Вибір методу заповнення масиву}
    '1': SortArray(A);
    '2': UnSortArray(A);
    '3': BackSortArray(A);
    '4': begin{Вибір методу пакетного запуску}
        Main_2(A,V,C);{Прохід по алгоритмах по одному разу}
        exit;
    end;
  end;
end;
```



```

        end;
        '5': begin
            Main_1(A,V,C); {Прохід по алгоритмах з пошуком середнього
значення}
            exit;
        end;
        #27: halt; {При натисканні клавіші Esc програма завершиться}
    end;

    if (ch='1') or (ch='2') or (ch='3') then {Вибір методу обходу}
        clrscr;
        gotoxy(35,12);
        Writeln('Pressed:');
        gotoxy(35,13);
        Writeln('1 - Bypass 1 (With additional array)'); {Обхід з
використанням додаткового масиву}
        gotoxy(35,14);
        Writeln('2 - Bypass 2 (With indexes imaginary vector)'); {Обхід з
використанням індексів уявного масиву}
        gotoxy(35,15);
        Writeln('3 - Bypass 3 (Around it elements of array)'); {Обхід
безпосередньо по елементах масиву}
        gotoxy(35,16);
        Writeln('Esc - Back');
        ch:=readkey;
        case ch of
            '1': begin {Безпосередній виклик виконання обраного обходу}
                clrscr;
                Writeln('Press Space if you want to show array before
sorting');
                Writeln('Press Enter for continue without showing an
array');
                ch1:=readkey;
                if ch1=#32 then {Вивід первозданного заданого масиву(на
вибір)}
                    begin
                        ShowArray(A); {Безпосередній вивід первозданного
заданого масиву}
                        Writeln('Press Enter for continue');
                        readkey;
                    end;

                    pr:=WORKAR1711; {Змінний процедурного типу присвоюється
значення процедури обраного методу обходу; означення чисел: 17__ :
номер алгоритму за методичними вказівками; __ 1 __ номер задачі за
методичними вказівками; __ 1: номер метода обходу за методичними
вказівками}
                    Writeln('Please wait...');
                    T:=Time(pr); {Присвоєння часу виконання алгоритму за
заданим видом оходу}
                    clrscr;
                    Writeln('Press Space if you want to show array after
sorting');
                    Writeln('Press Enter for continue without showing an
array');
                    ch1:=readkey;
                    if ch1=#32 then {Вивід відсортованого масиву(на вибір)}
                        begin

```

```

        ShowArray(A); {Безпосередній вивід відсортованого масиву}
        Writeln('Press Enter for continue');
        readkey;
    end;
end;
'2': begin {Безпосердній виклик виконання обраного обходу}
    clrscr;
    Writeln('Press Space if you want to show array before
sorting');
    Writeln('Press Enter for continue without showing an
array');
    ch1:=readkey;
    if ch1=#32 then {Вивід первозданного заданого масиву(на
вибір)}
        begin
            ShowArray(A); {Безпосередній вивід первозданного
заданого масиву}
            Writeln('Press Enter for continue');
            readkey;
        end;
    end;

    pr:=WORKAR1712; {Змінний процедурного типу присвоюється
значення процедури обраного методу обходу; означення чисел: 17__ :
номер алгоритму за методичними вказівками; __ 1 номер задачі за
методичними вказівками; __ 1: номер метода обходу за методичними
вказівками}
    Writeln('Please wait...');
    T:=Time(pr); {Присвоєння часу виконання алгоритму за
заданим видом оходу}
    clrscr;
    Writeln('Press Space if you want to show array after
sorting');
    Writeln('Press Enter for continue without showing an
array');
    ch1:=readkey;
    if ch1=#32 then {Вивід відсортованого масиву(на вибір)}
        begin
            ShowArray(A); {Безпосередній вивід відсортованого
масиву}
            Writeln('Press Enter for continue');
            readkey;
        end;
    end;
'3': begin {Безпосердній виклик виконання обраного обходу}
    clrscr;
    Writeln('Press Space if you want to show array before
sorting');
    Writeln('Press Enter for continue without showing an
array');
    ch1:=readkey;
    if ch1=#32 then {Вивід первозданного заданого масиву(на
вибір)}
        begin
            ShowArray(A); {Безпосередній вивід первозданного
заданого масиву}
            Writeln('Press Enter for continue');
            readkey;
        end;
    end;
end;

```

```

        pr:=WORKAR1713; {Змінний процедурного типу присвоюється
значення процедури обраного методу обходу; означення чисел: 17__ :
номер алгоритму за методичними вказівками; _ 1 _ номер задачі за
методичними вказівками; __ 1: номер метода обходу за методичними
вказівками}
        Writeln('Please wait...');
        T:=Time(pr); {Присвоєння часу виконання алгоритму за
заданим видом оходу}
        clrscr;
        Writeln('Press Space if you want to show array after
sorting');
        Writeln('Press Enter for continue without showing an
array');
        ch1:=readkey;
        if ch1=#32 then {Вивід відсортованого масиву(на вибір)}
            begin
                ShowArray(A); {Безпосередній вивід відсортованого
масиву}
                Writeln('Press Enter for continue');
                readkey;
            end;
        end;
        #27: exit; {Вихід з процедури Menu_1}
    end; {case}
    if ch<>#27 then
        begin
            clrscr;
            gotoxy(40,12);
            Writeln('Algorithm time - ',T); {Вивід часу виконання заданого
алгоритму}
            gotoxy(40,13);
            Writeln('Press Enter for continue');
            readkey;
        end;
    until (ch=#27); {Цикл повернення головного меню на початкову
сторінку, доки не буде натиснута Esc}
end;
end.

```

Модуль GLOBAL

```

unit GLOBAL;

interface {Описання доступної інформації для інших модулів}

const
    p = 1;
    m = 10;
    n = 10;

type
    arr = array[1..p, 1..m, 1..n] of integer; {Задання користувацького
типу "масив"}

```

```

    vector=array [1..n*m] of integer; {Задання користувацького типу
"вектор"}
    var A:arr; {Змінна типу масив}
        V:arr; {Змінна типу масив}
        C:vector; {Змінна типу вектор}

    procedure ShowArray(const a: arr); {Процедура виведення масиву на
екран}
    procedure SortArray(var a: arr); {Процедура впорядкованого
заповнення масиву}
    procedure UnSortArray(var a: arr); {Процедура неупорядкованого
заповнення масиву}
    procedure UnSortCopy(var a: arr); {Процедура копіювання рандомного
масиву}
    procedure BackSortArray(var a: arr); {Процедура заповнення обернено
впорядкованого масиву}
    procedure SortVect(var C: vector); {Процедура впорядкованого
заповнення вектора}
    procedure UnSortVect(var C: vector); {Процедура неупорядкованого
заповнення вектора}
    procedure BackSortVect(var C: vector); {Процедура заповнення
обернено впорядкованого вектора}
    implementation

    procedure ShowArray(const a: arr); {Процедура виведення масиву на
екран}
    var
        i, j, k: word;
    begin
        for k := 1 to p do {Лічильники проходу по координатам масиву}
        begin
            for i := 1 to m do
            begin
                for j := 1 to n do
                write(a[k, i, j]:8); {Виведення елементу на екран}
                writeln;
            end;
            writeln;
        end;
    end;

    procedure UnSortArray(var a: arr); {Процедура неупорядкованого
заповнення масиву}
    var
        i, j, k: word; {Змінні координат}
    begin
        randomize;
        for k := 1 to p do {Лічильники проходу по координатам масиву}
        for i := 1 to m do
        for j := 1 to n do
            a[k, i, j] := random(p * m * n); {Присвоєння комірці масиву
рандомного значення}
        end;

    procedure UnSortCopy(var a: arr); {Процедура копіювання рандомного
масиву}
    var
        i, j, k: word; {Змінні координат}

```

```

begin
  for k := 1 to p do {Лічильники проходу по координатам масиву}
    for i := 1 to m do
      for j := 1 to n do
        a[k, i, j] := V[k,i,j]; {Присвоєння запам'ятованих рандомних
занчень}
      end;
    end;

  procedure SortArray(var a: arr); {Процедура впорядкованного
заповнення масиву}
  var
    i, j, k: word; {Змінні кординат}
    l: integer; {Змінна, за допомогою якої буде заповнюватися масив}
  begin
    l := 1;
    for k := 1 to p do {Лічильники проходу по координатам масиву}
      for i := 1 to m do
        for j := 1 to n do
          begin
            a[k, i, j] := l; {Присвоєння комірці масиву значення l}
            inc(l); {При кожному проході лічильника, змінна l буде
збільшуватися на 1}
          end;
        end;
      end;

    procedure BackSortArray(var a: arr); {Процедура заповнення обернено
впорядкованного масиву}
    var
      i, j, k: word; {Змінні кординат}
      l: integer; {Змінна, за допомогою якої буде заповнюватися масив}
    begin
      l := p * m * n; {Змінній l присвоюється значення кількості
елементів масиву}
      for k := 1 to p do {Лічильники проходу по координатам масиву}
        for i := 1 to m do
          for j := 1 to n do
            begin
              a[k, i, j] := l; {Присвоєння комірці масиву значення l}
              dec(l); {При кожному проході лічильника, змінна l буде
зменьшуватися на 1}
            end;
          end;
        end;

      procedure SortVect(var C: vector); {Процедура впорядкованного
заповнення вектора}
      var
        i, j: integer;
      begin
        for i:=1 to (n*m) do {Лічильник проходу по вектору}
          C[i]:=i; {Заповнення вектора впорядкованими числами}
        end;

        procedure UnSortVect(var C: vector); {Процедура невпорядкованного
заповнення вектора}
        var
          i, j: integer;
        begin
          j:=n*m;

```

```

for i:=1 to (j) do {Лічильник проходу по вектору}
C[i]:=random(j); {Присвоєння рандомного значення комірці вектора}
end;

procedure BackSortVect(var C: vector); {Процедура заповнення
обернено впорядкованого вектора}
var
  i,j,g:integer;
begin
  j:=n*m; g:=j;
  for i:=1 to j do {Лічильник проходу по вектору}
  begin
    C[i]:=g; {Заповнення вектора обернено впорядкованими
числами}
    dec(g);
  end;
end;
end.

```

Модуль WORKAR

```

unit WORKAR; {Процедура Workaround, у якій приведений алгоритм з
необхідними методами обхода}

interface

uses GLOBAL; {Описання доступної інформації для інших модулів}

procedure WORKAR1711(var A: arr); {Алгоритм "вставка - обмін" з
використанням додаткового масиву}
procedure WORKAR1712(var A: arr); {Алгоритм "вставка - обмін" з
використанням елементів "уявного" вектора і переведення у відповідні
індекси елементів заданого двовимірного масиву}
procedure WORKAR1713(var A: arr); {Алгоритм "вставка - обмін",
здійснюючи обхід безпосередньо по елементах заданого двовимірного
масиву}
procedure SortVector(var C: vector); {Алгоритм "вставка - обмін"
сортування вектора}

implementation

procedure WORKAR1711(var a: arr); {Алгоритм "вставка - обмін" з
використанням додаткового масиву}

  var ii, k, j, i, B:integer; {ii: координата одновимірного масиву;
k,i,j: координати 3-вимірного масиву; B-додаткова комірка пам'яті}
  Z:vector; {Додатковий одновимірний масив}

begin
  for k:=1 to p do {Лічильники проходу по координатам 3-вимірного
масиву}
  begin

```

```

    ii:=1;
    for i := 1 to m do
        for j := 1 to n do
            begin
                Z[ii] := a[k, i, j]; {Переписування елементів двовимірного
масиву до одновимірного}
                inc(ii); {Після кожного проходу лічильника, координата
одновимірного масиву ii збільшується на 1}
            end;

        for i:=2 to (n*m) do {Прохід по одновимірному масиву}
            begin
                j:=i;
                while (j>1) and (Z[j]<Z[j-1]) do {Починається робота
безпосередньо гібридного алгоритму "вставка - обмін"}
                    begin
                        B:=Z[j]; {Зміна елементів місцями}
                        Z[j]:=Z[j-1];
                        Z[j-1]:=B;
                        j:=j-1;
                    end;
                end;
                ii := 1;
                for i := 1 to m do {Лічильники проходу по координатам 2-
вимірного масиву}
                    for j := 1 to n do
                        begin
                            a[k, i, j]:= Z[ii]; {Переписування елементів одновимірного
масиву до двовимірного}
                            inc(ii); {Після кожного проходу лічильника, координата
одновимірного масиву ii збільшується на 1}
                        end;
                    end;
                end;
            end;

    procedure WORKAR1712(var a: arr); {Алгоритм "вставка - обмін" з
використанням елементів "уявного" вектора і переведення у відповідні
індекси елементів заданого двовимірного масиву}

    var k,j, i, B:integer; {k,i,j: координати 3-вимірного масиву; B-
додаткова комірка пам'яті}
    begin
        for k:=1 to p do {Лічильники проходу по перерізам 3-вимірного
масиву}
            for i:=2 to (n*m) do {Лічильники проходу по уявному вектору}
                begin
                    j:=i;

                    while (j>1) and (A[k, ((j-1) div n)+1, ((j-1) mod n)+1]<A[k, ((j-
2) div n)+1, ((j-2) mod n)+1]) do {На місцях координат 3-вимірного
масиву стоять формули переведення елементів "уявного" вектора у
відповідні індекси двовимірного масиву}
                        begin
                            B:=A[k, ((j-1) div n)+1, ((j-1) mod n)+1]; {Зміна елементів
місцями}
                            A[k, ((j-1) div n)+1, ((j-1) mod n)+1]:=A[k, ((j-2) div
n)+1, ((j-2) mod n)+1];
                            A[k, ((j-2) div n)+1, ((j-2) mod n)+1]:=B;

```

```

        j:=j-1;
    end;
end;
end;

procedure WORKAR1713(var a: arr); {Алгоритм "вставка - обмін",
здійснюючи обхід безпосередньо по елементах заданого двовимірного
масиву}
var
    i, j, k, Z, H, f, g, B: integer; {k,i,j: координати 3-вимірного
масиву; B-додаткова комірка пам'яті; Z, H, f, g: додаткові змінні для
проходу алгоритму по координатам перерізу}
begin
    for k:=1 to p do {Лічильник проходу по перерізам 3-вимірного
масиву}
        for i:=1 to m do {Лічильники проходу по координатам перерізу}
            for j:=1 to n do
                begin
                    Z:=i; H:=j; {Копіювання координат елемента, для можливості
їх зміни}
                    if H<>1 then {перевірка на першість елемента у рядку}
                        begin
                            f:=Z; g:=H-1; {Якщо елемент не перший, то перехід
додаткових координат g,f на попередій рядок не відбувається,
відбувається перехід на сусідній елемент зліва}
                        end
                    else
                        begin
                            f:=Z-1; g:=n; {Якщо елемент перший, то відбувається
перехід додаткових координат g,f на останній елемент попереднього
рядка}
                        end
                    end;

                    while (H>=1) and (f>=1) and (A[k,Z,H]<A[k,f,g]) do {Перевірка за
умовою алгоритму; H,f- перевірка для запобігання виходу за рамки
двовимірного масиву}
                        begin
                            B:=A[k,Z,H]; {Зміна елементів місцями}
                            A[k,Z,H]:=A[k,f,g];
                            A[k,f,g]:=B;

                            dec(H); dec(g); {Зменшення координат, відповідаючих за
прохід по стовпцям елемента}

                            if H=1 then {Перевірка на першість координати елемента}
                                begin
                                    g:=n; f:=Z-1; {Якщо елемент перший, то відбувається
перехід додаткових координат g,f на останній елемент попереднього
рядка}
                                end
                            else
                                if H=0 then {Перевірка на закінчення рядка}
                                    begin
                                        H:=n; Z:=Z-1; {Якщо елементи у рядку закінчуються, то
додаткові координати H,Z переходять на останній елемент попереднього
рядка}
                                    end
                                end;
                            end;
                        end;
                    end;
                end;
            end;
        end;
    end;

```



```

    end;
end;

procedure SortVector(var C:vector); {Алгоритм "вставка - обмін"
сортування вектора}
    var i,j,B: integer; {i,j: координати проходу по вектору; B-
додаткова комірка пам'яті;}
    begin

        for i:=2 to (m*n) do {Лічильник проходу по вектору}
        begin
            j:=i;
            while (j>1) and (C[j]<C[j-1]) do {Починається робота
безпосередньо гібридного алгоритму "вставка - обмін"}
            begin
                B:=C[j]; {Зміна елементів місцями}
                C[j]:=C[j-1];
                C[j-1]:=B;
                j:=j-1; {Зміщення координати вліво}
            end;
        end;
    end;

end.

```

Модуль TIMER

```

unit TIMER;

interface

uses GLOBAL;

type proc=procedure(var A:arr);
    vect=procedure(var C:vector);
var pr:proc; {Змінна процедурного типу}
    vec:vect; {Змінна процедурного типу}

    function Time(pr:proc):longint; {Функція знаходження часу роботи
алгоритму з масивом}
    function TimeVector(vec:vect):longint; {Функція знаходження часу
роботи алгоритму з вектором}

implementation

uses dos,crt;

type TTime=record {Користувацький тип "запис" для запам'ятовування
часу}
    Hours,Min,Sec,HSec:word;
end;

    function ResTime(const STime,FTime:TTime):longint; {Функція
знаходження різниці часу}

```

```

begin
  ResTime:=360000*Longint (FTime.Hours)+
           6000*Longint (FTime.Min)+
           100*Longint (FTime.Sec)+
           Longint (FTime.HSec)-
           360000*Longint (STime.Hours)-
           6000*Longint (STime.Min)-
           100*Longint (STime.Sec)-
           Longint (STime.HSec);
end;

function Time(pr:proc):longint;

var StartTime,FinishTime:TTime;

begin
  with StartTime do
    GetTime(Hours,Min,Sec,HSec);
  pr(A); {Запуск обраної процедури сортування масиву}
  with FinishTime do
    GetTime(Hours,Min,Sec,HSec);
  Time:=ResTime (StartTime,FinishTime);
end;

function TimeVector (vec:vect):longint;

var StartTime,FinishTime:TTime;

begin
  with StartTime do
    GetTime(Hours,Min,Sec,HSec);
  vec(C); {Запуск обраної процедури сортування вектора}
  with FinishTime do
    GetTime(Hours,Min,Sec,HSec);
  TimeVector:=ResTime (StartTime,FinishTime);
end;

end.

```

Модуль TEST

```

unit TEST;

interface

  uses
    GLOBAL,WORKAR,TIMER,crt,dos;

  procedure Main_1(var a:arr; var C:vector); {Пакетний вивід середніх
значень часу роботи алгоритмів}
  procedure Main_2(var a:arr; var C:vector);{Пакетний вивід значень
часу роботи алгоритмів}

implementation

```

```

var ch:char;

procedure Main_1(var a:arr; var C:vector); {Пакетний вивід середніх
значень часу роботи алгоритмів}
const n=14;
var var_sort,var_alg,i:integer;
    Min,Max:longint;
    B:array[1..n]of longint; {Вектор для запам'ятовування часу
роботи алгоритмів}
    Sum:real;
    T:longint; {Комірка запам'ятовування часу}
begin
clrscr;
writeln('                Sort      UnSort      BackSort');

UnSortArray(V); {Заповнює додатковий масив випадковими значеннями}

var_alg:=1; {Початкові значення вибору алгоритму}
repeat
var_sort:=1;{Початкові значення вибору методу заповнення масиву}
case var_alg of {Вибір алгоритму}
1: begin
write('Workaround 1:');
pr:=WORKAR1711;{Процедурному вказівнику присвоюється адреса
алгоритму "вставка - обмін" з використанням додаткового масиву}
end;
2: begin
write('Workaround 2:');
pr:=WORKAR1712; {Процедурному вказівнику присвоюється адреса
алгоритму "вставка - обмін" з використанням елементів "уявного"
вектора і переведення у відповідні індекси елементів заданого
двовимірного масиву}
end;

3: begin
write('Workaround 3:');
pr:=WORKAR1713; {Процедурному вказівнику присвоюється адреса
алгоритму "вставка - обмін" здійснюючи обхід безпосередньо по
елементах заданого двовимірного масиву}
end;
end;{case}

repeat
case var_sort of{Вибір методу заповнення масиву}
1: for i:=1 to n do {Лічильник кількості виконання алгоритму}
begin
SortArray(A);{Процедура впорядкованого заповнення масиву}
B[i]:=Time(pr); {Присвоєння комірці вектора часу роботи
алгоритма}
end;
2: for i:=1 to n do {Лічильник кількості виконання алгоритму}
begin
UnSortCopy(A); {Процедура копіювання з додаткового
випадкового масиву у необхідний}
B[i]:=Time(pr);{Присвоєння комірці вектора часу роботи
алгоритма}
end;
end;

```

```

3: for i:=1 to n do {Лічильник кількості виконання алгоритму}
    begin
        BackSortArray(A); {Процедура заповнення обернено
впорядкованного масиву}
        B[i]:=Time(pr); {Присвоєння комірці вектора часу роботи
алгоритма}
    end;
end; {case}

Min:=B[3];
Max:=B[3];
for i:=4 to n do {Пошук мінімального та максимального значення
часу роботи}
    begin
        if B[i]>Max then Max:=B[i];
        if B[i]<Min then Min:=B[i];
    end;
Sum:=0;
for i:=3 to n do Sum:=Sum+B[i];
Sum:=(Sum-Min-Max)/10; {Знаходження середнього значення роботи
алгоритму}
if var_sort<>3 then Write(Sum:10:1)
else Writeln(Sum:10:1);

    inc(var_sort); {Зміна вибору методу заповнення масиву}
until var_sort>3;
inc(var_alg); {Зміна вибору алгоритму}
until var_alg>3;
Writeln('Press Enter for continue');
readln;

vec:=SortVector; {Процедурному вказівнику присвоюється адреса
алгоритму "вставка - обмін" сортування вектора}
SortVect(C); {Процедура впорядкованого заповнення вектора}
T:=TimeVector(vec); {Комірці запам'ятовування часу присвоюється
значення часу роботи алгоритму}
writeln('Sort Vector:',T*p); {Вивід на екран теоретичного значення
роботи алгоритму з вектором}
UnSortVect(C); {Процедура невлпорядкованого заповнення вектора}
T:=TimeVector(vec); {Комірці запам'ятовування часу присвоюється
значення часу роботи алгоритму}
writeln('UnSort Vector:',T*p); {Вивід на екран теоретичного
значення роботи алгоритму з вектором}
BackSortVect(C); {Процедура заповнення обернено впорядкованого
вектора}
T:=TimeVector(vec); {Комірці запам'ятовування часу присвоюється
значення часу роботи алгоритму}
writeln('BackSort Vector:',T*p); {Вивід на екран теоретичного
значення роботи алгоритму з вектором}

Writeln('Press Enter for continue');
ch:=readkey;
if ch=#32 then exit; {Вихід з роботи процедури}
end;

procedure Main_2(var a:arr; var C:vector); {Пакетний вивід значень
часу роботи алгоритмів}
var var_sort,var_alg,i:integer;

```

```

T,B:longint; {Комірки запам'ятовування часу}
begin
clrscr;
writeln('                Sort      UnSort      BackSort');

UnSortArray(V); {Заповнює додатковий масив випадковими значеннями}

var_alg:=1; {Початкові значення вибору алгоритму}
repeat
var_sort:=1; {Початкові значення вибору методу заповнення масиву}
case var_alg of {Вибір алгоритму}
1: begin
write('Workaround 1:');
pr:=WORKAR1711; {Процедурному вказівнику присвоюється адреса
алгоритму "вставка - обмін" з використанням додаткового масиву}
end;
2: begin
write('Workaround 2:');
pr:=WORKAR1712; {Процедурному вказівнику присвоюється адреса
алгоритму "вставка - обмін" з використанням елементів "уявного"
вектора і переведення у відповідні індекси елементів заданого
двовимірного масиву}
end;

3: begin
write('Workaround 3:');
pr:=WORKAR1713; {Процедурному вказівнику присвоюється адреса
алгоритму "вставка - обмін" здійснюючи обхід безпосередньо по
елементах заданого двовимірного масиву}
end;
end; {case}

repeat
case var_sort of {Вибір методу заповнення масиву}
1: begin
SortArray(A); {Процедура впорядкованого заповнення масиву}
B:=Time(pr); {Присвоєння комірці значення часу роботи
алгоритма}
end;
2: begin
UnSortCopy(A); {Процедура копіювання з додаткового
випадкового масиву у необхідний}
B:=Time(pr); {Присвоєння комірці значення часу роботи
алгоритма}
end;
3: begin
BackSortArray(A); {Процедура заповнення обернено
впорядкованого масиву}
B:=Time(pr); {Присвоєння комірці вектора часу роботи
алгоритма}
end;
end; {case}

if var_sort<>3 then Write(B:10)
else Writeln(B:10);
inc(var_sort);
until var_sort>3;
inc(var_alg);

```

```

    until var_alg>3;
    Writeln('Press Enter for continue');
    readln;

    vec:=SortVector; {Процедурному вказівнику присвоюється адреса
алгоритму "вставка - обмін" сортування вектора}
    SortVect(C); {Процедура впорядкованого заповнення вектора}
    T:=TimeVector(vec);{Комірці запам'ятовування часу присвоюється
значення часу роботи алгоритму}
    writeln('Sort Vector:',T*p); {Вивід на екран теоретичного значення
роботи алгоритму з вектором}
    UnSortVect(C);{Процедура неупорядкованого заповнення вектора}
    T:=TimeVector(vec);{Комірці запам'ятовування часу присвоюється
значення часу роботи алгоритму}
    writeln('UnSort Vector:',T*p);{Вивід на екран теоретичного
значення роботи алгоритму з вектором}
    BackSortVect(C);{Процедура заповнення обернено впорядкованого
вектора}
    T:=TimeVector(vec);{Комірці запам'ятовування часу присвоюється
значення часу роботи алгоритму}
    writeln('BackSort Vector:',T*p); {Вивід на екран теоретичного
значення роботи алгоритму з вектором}

    Writeln('Press Enter for continue');
    ch:=readkey;
    if ch=#32 then exit; {Вихід з роботи процедури}
    end;

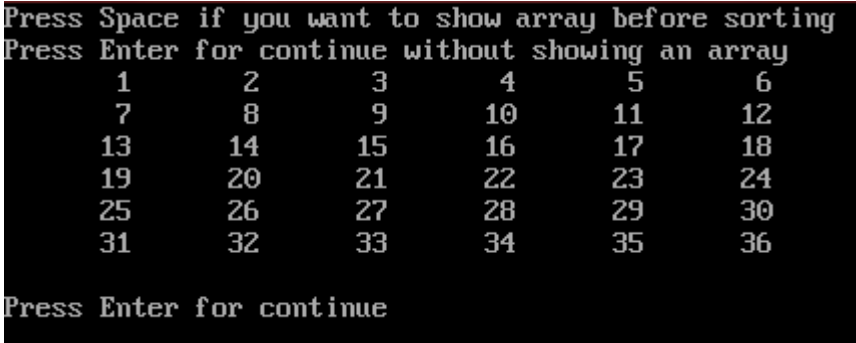
end.

```

ТЕСТИ

1. Тестування процедури, що заповнює масив впорядкованими значеннями:

```
procedure SortArray(var a: arr); var
  i, j, k: word;
  l: integer;
begin
  l := 1;
  for k := 1 to p do
    for i := 1 to m do
      for j := 1 to n do
        begin
          a[k, i, j] := l;
          inc(l);
        end;
      end;
    end;
  end;
```

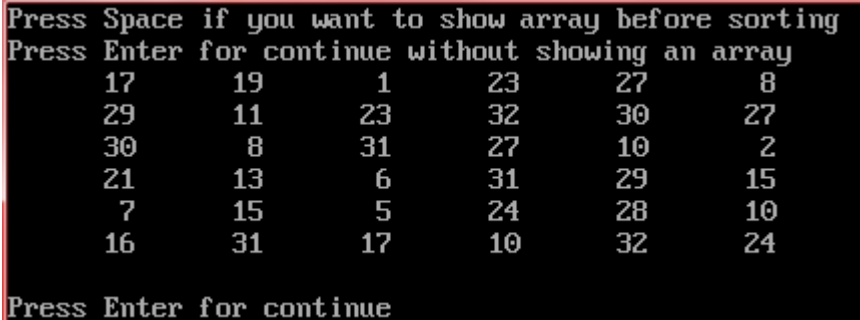


```
Press Space if you want to show array before sorting
Press Enter for continue without showing an array
  1      2      3      4      5      6
  7      8      9     10     11     12
 13     14     15     16     17     18
 19     20     21     22     23     24
 25     26     27     28     29     30
 31     32     33     34     35     36

Press Enter for continue
```

2. Тестування процедури, що заповнює масив не впорядкованими (рандомними) значеннями:

```
procedure UnSortArray(var a: arr);
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(p * m * n);
      end;
    end;
  end;
```

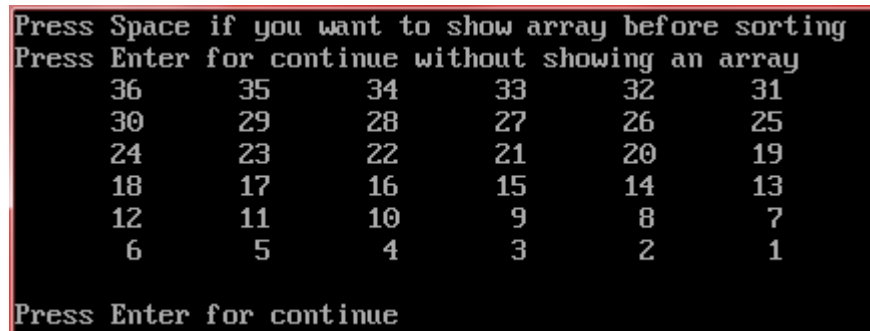


```
Press Space if you want to show array before sorting
Press Enter for continue without showing an array
 17     19      1     23     27      8
 29     11     23     32     30     27
 30      8     31     27     10      2
 21     13      6     31     29     15
  7     15      5     24     28     10
 16     31     17     10     32     24

Press Enter for continue
```

3. Тестування процедури, що заповнює масив обернено впорядкованими значеннями:

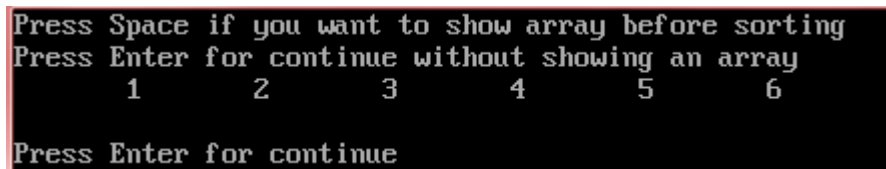
```
procedure BackSortArray(var a: arr);  
var  
    i, j, k: word;  
    l: integer;  
begin  
    l := p * m * n;  
    for k := 1 to p do  
        for i := 1 to m do  
            for j := 1 to n do  
                begin  
                    a[k, i, j] := l;  
                    dec(l);  
                end;  
            end;  
        end;  
    end;
```



```
Press Space if you want to show array before sorting  
Press Enter for continue without showing an array  
36    35    34    33    32    31  
30    29    28    27    26    25  
24    23    22    21    20    19  
18    17    16    15    14    13  
12    11    10    9     8     7  
6     5     4     3     2     1  
Press Enter for continue
```

4. Тестування процедури, що заповнює вектор впорядкованими значеннями:

```
procedure SortVect(var C: vector);  
var  
    i, j: integer;  
begin  
    for i:=1 to (n*m) do  
        C[i]:=i;  
    end;
```



```
Press Space if you want to show array before sorting  
Press Enter for continue without showing an array  
1     2     3     4     5     6  
Press Enter for continue
```

5. Тестування процедури, що заповнює вектор не впорядкованими (рандомними) значеннями:

```
procedure UnSortVect(var C: vector);
```



```

var
  i,j:integer;
begin
  j:=n*m;
  for i:=1 to (j) do
    C[i]:=random(j);
  end;

```

```

Press Space if you want to show array before sorting
Press Enter for continue without showing an array
      5      3      5      3      0      4
Press Enter for continue

```

6. Тестування процедури, що заповнює масив обернено впорядкованими значеннями:

```

procedure BackSortVect(var C: vector); var
  i,j,g:integer;
begin
  j:=n*m; g:=j;
  for i:=1 to j do
    begin
      C[i]:=g;
      dec(g);
    end;
  end;
end.

```

```

Press Space if you want to show array before sorting
Press Enter for continue without showing an array
      6      5      4      3      2      1
Press Enter for continue

```

7. Тестування процедури, яка сортує масив за допомогою гібридного алгоритму "вставка – обмін", здійснюючи обхід з використанням додаткового одновимірного масиву:

```

procedure WORKAR1711(var a: arr);
var ii, k, j, i, B:integer;
    Z:vector;

begin
  for k:=1 to p do
    begin
      ii:=1;
      for i := 1 to m do
        for j := 1 to n do
          begin
            Z[ii] := a[k, i, j];

```

```

inc(ii);
end;

for i:=2 to (n*m) do
begin
j:=i;
while (j>1) and (Z[j]<Z[j-1]) do
begin
B:=Z[j];
Z[j]:=Z[j-1];
Z[j-1]:=B;
j:=j-1;
end;
end;
ii := 1;
for i := 1 to m do
for j := 1 to n do
begin
a[k, i, j]:= Z[ii];
inc(ii);
end;
end;
end;
end;

```

```

Press Space if you want to show array before sorting
Press Enter for continue without showing an array
  9      3      29      1      30      0
 25      6       4     27      9     10
  2     10     31      5     24      7
 19     25     19     25     19     19
 15      4     16     20     25     15
  9     32      5     29     12     12

Press Enter for continue

Press Space if you want to show array after sorting
Press Enter for continue without showing an array
  0       1       2       3       4       4
  5       5       6       7       9       9
  9      10      10      12      12      15
 15      16      19      19      19      19
 20      24      25      25      25      25
 27      29      29      30      31      32

Press Enter for continue

```

8. Тестування процедури, яка сортує масив за допомогою гібридного алгоритму "вставка – обмін", здійснюючи бохід перетворюючи один індекс елементів "уявного" вектора у відповідні індекси елементів заданого двовимірного масиву:

```

procedure WORKAR1712(var a: arr);
var k,j, i, B:integer;
begin

```

```

for k:=1 to p do
  for i:=2 to (n*m) do
    begin
      j:=i;

      while (j>1) and (A[k, ((j-1) div n)+1, ((j-1) mod
n)+1]<A[k, ((j-2) div n)+1, ((j-2) mod n)+1]) do
        begin
          B:=A[k, ((j-1) div n)+1, ((j-1) mod n)+1];
          A[k, ((j-1) div n)+1, ((j-1) mod n)+1]:=A[k, ((j-2) div
n)+1, ((j-2) mod n)+1];
          A[k, ((j-2) div n)+1, ((j-2) mod n)+1]:=B;
          j:=j-1;
        end;
      end;
    end;
  end;
end;

```

```

Press Space if you want to show array before sorting
Press Enter for continue without showing an array
  35      35      20      23      23      6
  17      19      19      30      23      10
  24      13      14       7      26      25
  18      20      17      20       7       1
  33      14      29      18       8       9
  16      31      26      32      12      29

Press Enter for continue

Press Space if you want to show array after sorting
Press Enter for continue without showing an array
   1       6       7       7       8       9
  10      12      13      14      14      16
  17      17      18      18      19      19
  20      20      20      23      23      23
  24      25      26      26      29      29
  30      31      32      33      35      35

Press Enter for continue

```

9. Тестування процедури, яка сортує масив за допомогою гібридного алгоритму "вставка – обмін", здійснюючи обхід безпосередньо по елементах заданого двовимірного масиву:

```

procedure WORKAR1713(var a: arr);
var
  i, j, k, Z, H, f,g, B: integer;
begin
  for k:=1 to p do
    for i:=1 to m do
      for j:=1 to n do
        begin
          Z:=i; H:=j;
          if H<>1 then

```

```

        begin
            f:=Z; g:=H-1;
        end
    else
        begin
            f:=Z-1; g:=n
        end;

while (H>=1) and (f>=1) and (A[k,Z,H]<A[k,f,g]) do
    begin
        B:=A[k,Z,H];
        A[k,Z,H]:=A[k,f,g];
        A[k,f,g]:=B;

        dec(H);    dec(g);

        if H=1 then
            begin
                g:=n; f:=Z-1;
            end
        else
            if H=0 then
                begin
                    H:=n; Z:=Z-1;
                end;
            end;
        end;
    end;
end;
end;

```

```

Press Space if you want to show array before sorting
Press Enter for continue without showing an array
    4      28      30      27      34      2
   16      18       8      15      11      6
   30      18       1      34      20      33
   23      12      20      11      22      3
   29      25      35       5      22      25
   27      30      19      18      31      13

Press Enter for continue

Press Space if you want to show array after sorting
Press Enter for continue without showing an array
    1       2       3       4       5       6
    8      11      11      12      13      15
   16      18      18      18      19      20
   20      22      22      23      25      25
   27      27      28      29      30      30
   30      31      33      34      34      35

Press Enter for continue

```

10. Тестування процедури, яка сортує вектор за допомогою Гібридного алгоритму "вставка – обмін":

```
procedure SortVector(var C:vector);  
var i,j,B: integer;  
begin  
    for i:=2 to (m*n) do  
    begin  
        j:=i;  
        while (j>1) and (C[j]<C[j-1]) do  
        begin  
            B:=C[j];  
            C[j]:=C[j-1];  
            C[j-1]:=B;  
            j:=j-1;  
        end;  
    end;  
end;
```

```
Press Space if you want to show array before sorting  
Press Enter for continue without showing an array  
    0      1      2      1      5      5  
  
Press Enter for continue  
  
Press Space if you want to show array after sorting  
Press Enter for continue without showing an array  
    0      1      1      2      5      5  
  
Press Enter for continue
```

Результати тестування швидкодії алгоритмів

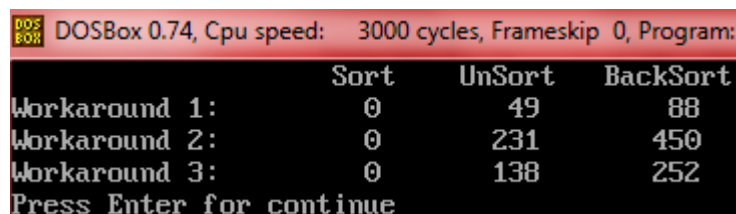
При тестуванні швидкодії алгоритмів використовувався такий вид змінної, як процедурний тип, для уніфікації та облегшення коду програми. Процедурний тип викликається у процедурах Time та TimeVector між стандартними процедурами модуля Dos, GetTime, які призначені для точного заміру часу роботи алгоритму. Практичним шляхом перевірено, що при швидкості роботи процесора у середовищі DosBox у **50 cycles**, при виклику процедурного типу, який вказує на пусту процедуру, час роботи даної структури дорівнює 0.



```
DOSBox 0.74, Cpu speed: 50 cycles, Frameskip 0, Program: TURBO

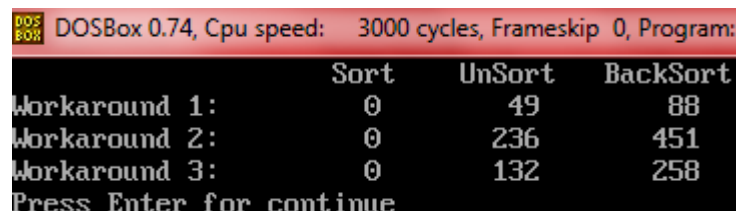
Algorithm time - 0
Press Enter for continue
```

Була виконана і перевірка від зворотнього: було перевірено роботу алгоритмів при однаковій швидкості процесора з використанням процедурного типу та без нього, з використанням однакових вхідних даних (тести у відповідному порядку).



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program:

      Sort      UnSort      BackSort
Workaround 1:      0         49         88
Workaround 2:      0        231        450
Workaround 3:      0        138        252
Press Enter for continue
```



```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program:

      Sort      UnSort      BackSort
Workaround 1:      0         49         88
Workaround 2:      0        236        451
Workaround 3:      0        132        258
Press Enter for continue
```

Так, ми бачимо, що при реалізації деяких алгоритмів, час відрізняється, але це не дає повної конкретики, тому що при роботі середовища DosBox час роботи може відрізнятися, це залежить від роботи віртуального

процесора, тому цей фактор ми віднесемо до фактору похибки роботи віртуального середовища.

Як висновок, час виклику процедурного вказівника займає дуже мало часу, тому що він передає лише адресу процедури, до якої потрібно звернутися.

Ми будемо працювати зі швидкістю роботи процесора у середовищі DosBox у **3000 cycles** та **1000 cycles**, що набагато більша ніж та, з якою ми проводили перше тестування (**50 cycles**), тому будемо вважати, що виклик процедурного типу не буде впливати на час виконання 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. Значення для невідсортованого вектора кожен раз будуть обиратися нові, тому що розмір вектора з кожною перевіркою змінюється. Тестування виконувалось при швидкості процесора **1000 cycles**.

Пояснювання для скорочень у таблиці:

- Вектор: час виконання алгоритму "вставка – обмін", здійснюючи обхід по одновимірному масиву (вектору) ;

Таблиця №1.1 для вектора C[N]; N=200			
	Відсортований	Невідсортований	Обернено відсортований
Вектор	0	44	83

Таблиця №1.2 для вектора C[N]; N=300			
	Відсортований	Невідсортований	Обернено відсортований
Вектор	0	99	192

Таблиця №1.3 для вектора C[N]; N=400			
	Відсортований	Невідсортований	Обернено відсортований
Вектор	0	165	340

Таблиця №1.4 для вектора C[N]; N=600			
	Відсортований	Невідсортований	Обернено відсортований
Вектор	0	379	769

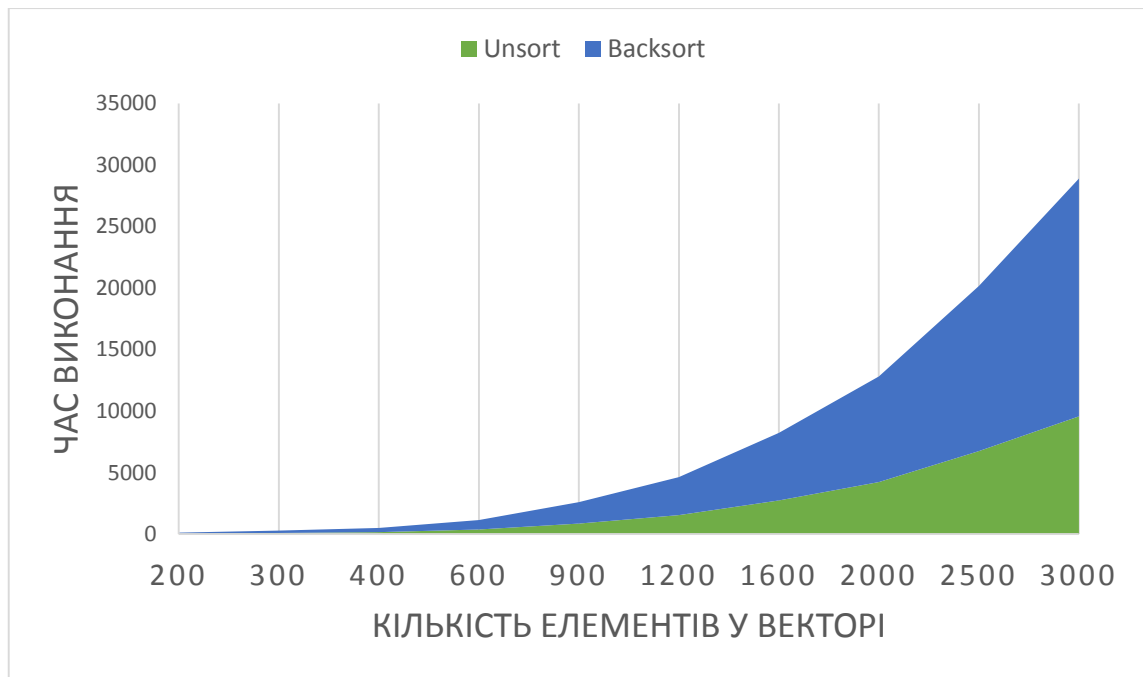
Таблиця №1.5 для вектора C[N]; N=900			
	Відсортований	Невідсортований	Обернено відсортований
Вектор	0	862	1741
Таблиця №1.6 для вектора C[N]; N=1200			
	Відсортований	Невідсортований	Обернено відсортований
Вектор	5	1549	3087

Таблиця №1.7 для вектора C[N]; N=1600			
	Відсортований	Невідсортований	Обернено відсортований
Вектор	5	2741	5492

Таблиця №1.8 для вектора C[N]; N=2000			
	Відсортований	Невідсортований	Обернено відсортований
Вектор	6	4235	8585

Таблиця №1.9 для вектора C[N]; N=2500			
	Відсортований	Невідсортований	Обернено відсортований
Вектор	5	6751	13419

Таблиця №1.10 для вектора C[N]; N=3000			
	Відсортований	Невідсортований	Обернено відсортований
Вектор	5	9573	19322



2. Масив глибиною P та перерізами різної форми, з однаковою кількістю елементів

Результати тестування знову ж таки приведені у таблицях (заміри – у сотих долях секунди). Для коректності тестування та зменшення похибки роботи процесора, ми використовуємо середні значення часу виконання роботи алгоритмів з масивом (процедура пошуку середнього значення **Main_1** наведена у модулі **TEST**). Пошук часу виконання алгоритму у векторі ми виконуємо за методичними вказівниками викладача (сортування вектора розмірністю ($m \cdot n$) одного перерізу, і помножене на кількість перерізів (p)) для порівняння роботи алгоритму у трьовимірному масиві з теоретичним значенням сортування цього алгоритму у векторі). Тестування виконувалось при швидкості процесора **3000 cycles**.

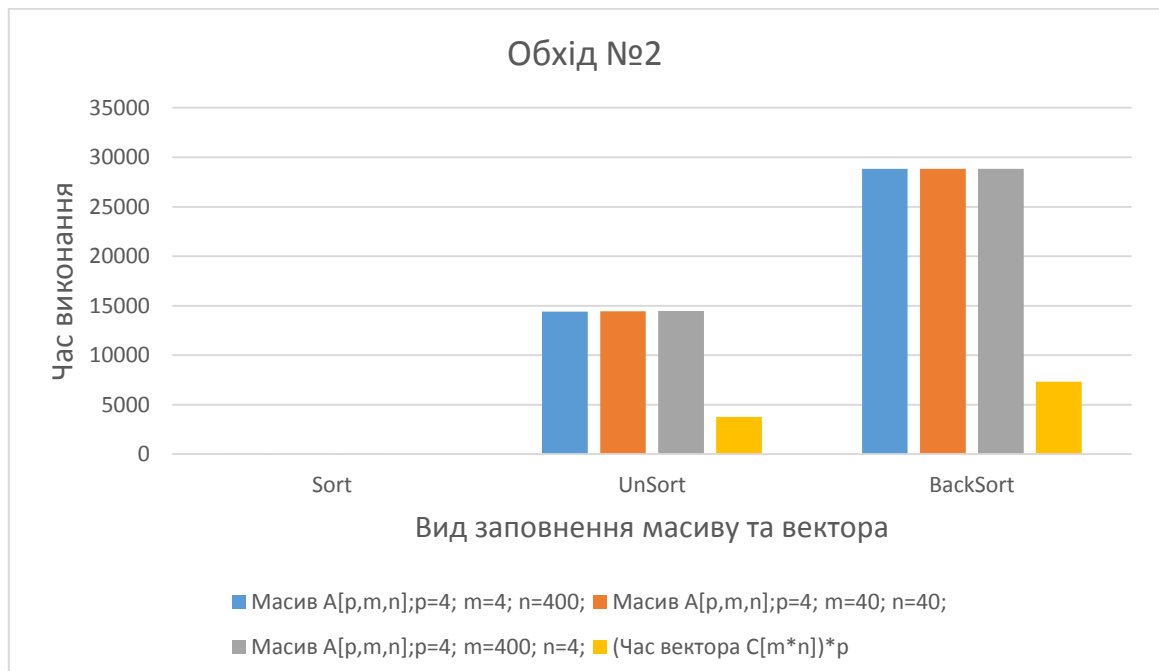
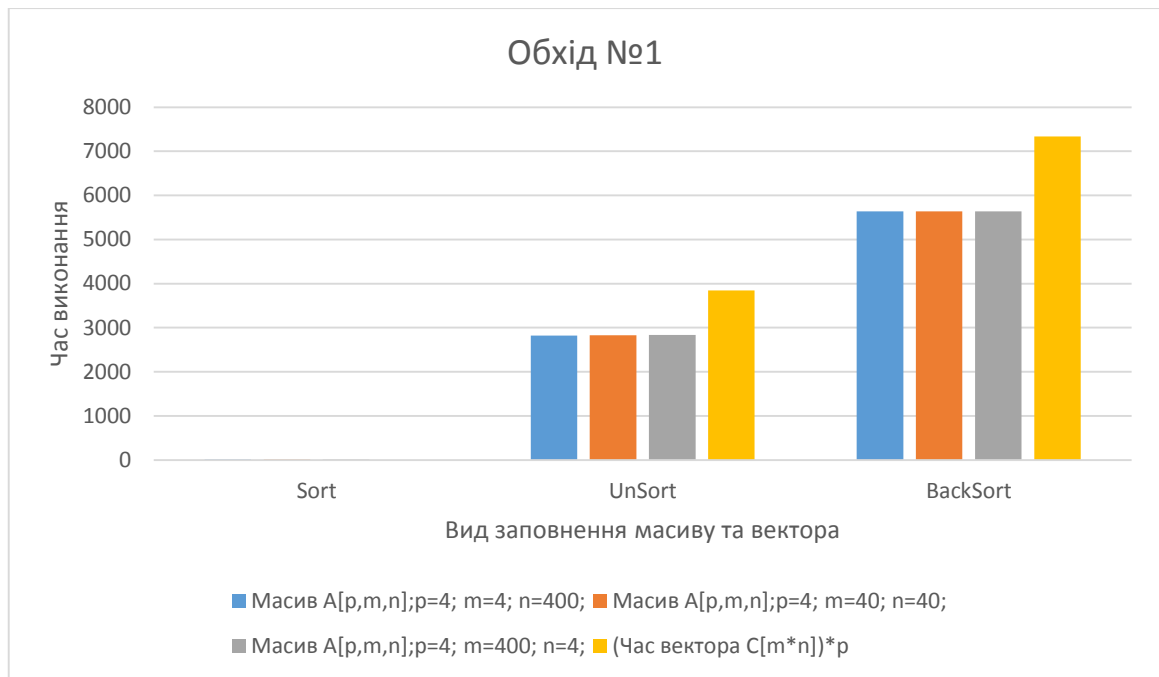
Пояснювання до скорочень у таблиці:

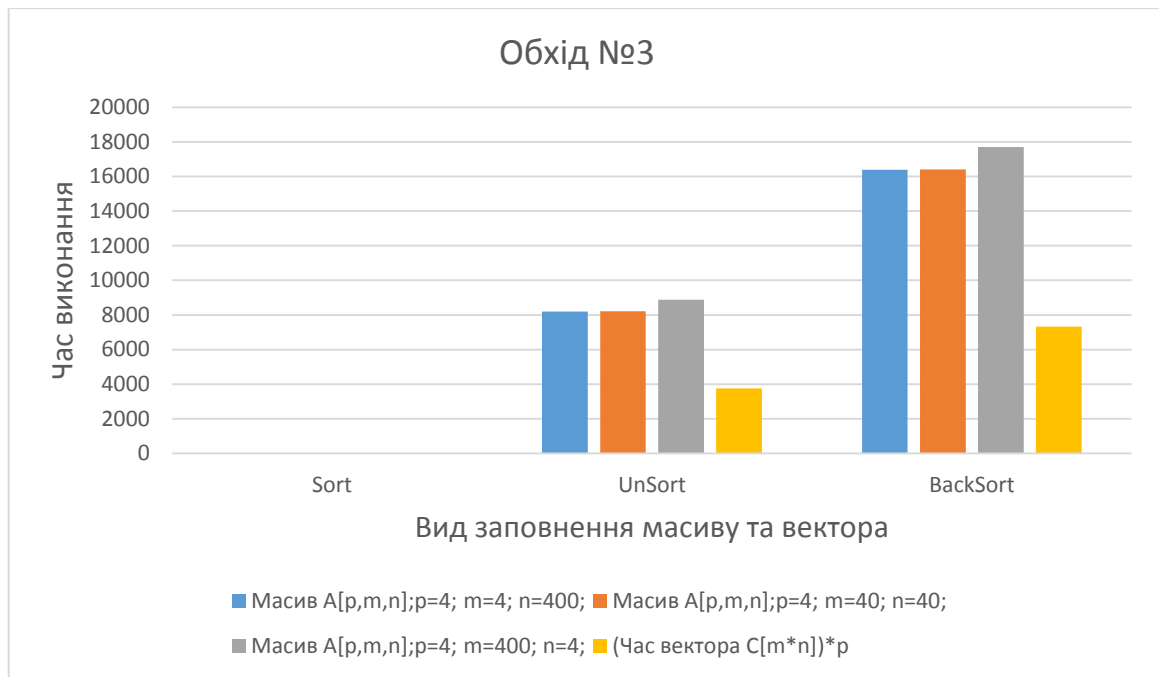
- Обхід №1: час виконання алгоритму "вставка – обмін", здійснюючи обхід з використанням додаткового одновимірного масиву;
- Обхід №2: час виконання алгоритму "вставка – обмін", здійснюючи обхід перетворюючи один індекс елементів "уявного" вектора у відповідні індекси елементів заданого двовимірного масиву;
- Обхід №3: час виконання алгоритму "вставка – обмін", здійснюючи обхід безпосередньо по елементах заданого двовимірного масиву;
- Вектор: час виконання алгоритму "вставка – обмін", здійснюючи обхід по одновимірному масиву (вектору) розміром ($m \cdot n$), тобто розміром одного перерізу, та помножене на (p), тобто кількість перерізів;

Таблиця №2.1 для масива $A[p,m,n]$; $p=4;m=4;n=400$;			
	Відсортований	Невідсортований	Обернено відсортований
Обхід №1	11.0	2824.8	5640.9
Обхід №2	14.2	14418.0	28835.4
Обхід №3	11.0	8193.2	16386.3
Вектор	0	3844	7340

Таблиця №2.2 для масива $A[p,m,n]$; $p=4;m=40;n=40$;			
	Відсортований	Невідсортований	Обернено відсортований
Обхід №1	11.0	2828.0	5640.9
Обхід №2	13.1	14431.5	28835.3
Обхід №3	10.9	8216.0	16412.3
Вектор	20	3624	7340

Таблиця №2.3 для масива $A[p,m,n]$; $p=4;m=400;n=4$;			
	Відсортований	Невідсортований	Обернено відсортований
Обхід №1	12.0	2834.1	5641.4
Обхід №2	13.8	14464.2	28836.0
Обхід №3	11.0	8882.5	17706.4
Вектор	0	3756	7336





3. Масив глибиною P та квадратними перерізами (m=n=const)

Результати тестування знову ж таки приведені у таблицях (заміри – у сотих долях секунди, розміри перерізів - (20*20)). Для коректності тестування та зменшення похибки роботи процесора, ми використовуємо середні значення часу виконання роботи алгоритмів з масивом(процедура пошуку середнього значення **Main_1** наведена у модулі **TEST**). Пошук часу виконання алгоритму у векторі ми виконуємо за методичними вказівниками викладача(сортування вектора розмірністю (400) одного перерізу, і помножене на кількість перерізів (p)) для порівняння роботи алгоритму у трьовимірному масиві з теоретичним значенням сортування цього алгоритму у векторі). Тестування виконувалось при швидкості процесора **3000 cycles**.

Пояснювання до скорочень у таблиці:

- Обхід №1: час виконання алгоритму "вставка – обмін", здійснюючи обхід з використанням додаткового одновимірного масиву;
- Обхід №2: час виконання алгоритму "вставка – обмін", здійснюючи обхід перетворюючи один індекс елементів "уявного" вектора у відповідні індекси елементів заданого двовимірного масиву;
- Обхід №3: час виконання алгоритму "вставка – обмін", здійснюючи обхід безпосередньо по елементах заданого двовимірного масиву;
- Вектор: час виконання алгоритму "вставка – обмін", здійснюючи обхід по одновимірному масиву (вектору) розміром (m*n), тобто розміром одного перерізу, та помножене на (p), тобто кількість перерізів;

Таблиця №3.1 для масива $A[p,m,n]$; $p=1;m=20;n=20$;			
	Відсортований	Невідсортований	Обернено відсортований
Обхід №1	0,5	44	87,8
Обхід №2	0	222,5	449,7
Обхід №3	0	127,5	257
Вектор	0	55	115

Таблиця №3.2 для масива $A[p,m,n]$; $p=2;m=20;n=20$;			
	Відсортований	Невідсортований	Обернено відсортований
Обхід №1	1,1	87,8	176,9
Обхід №2	1	445,6	900,2
Обхід №3	1,6	225,5	513,7
Вектор	0	110	230

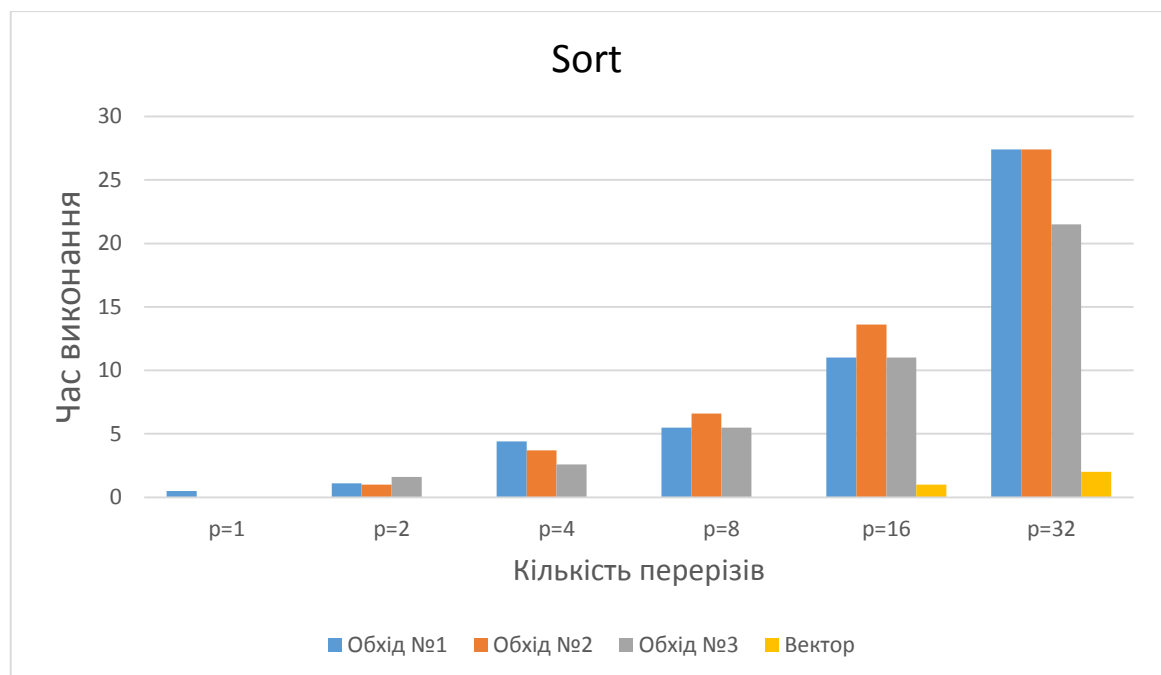
Таблиця №3.3 для масива $A[p,m,n]$; $p=4;m=20;n=20$;			
	Відсортований	Невідсортований	Обернено відсортований
Обхід №1	4,4	179,5	353,7
Обхід №2	3,7	901,9	1798
Обхід №3	2,6	515,2	1027
Вектор	0	244	460

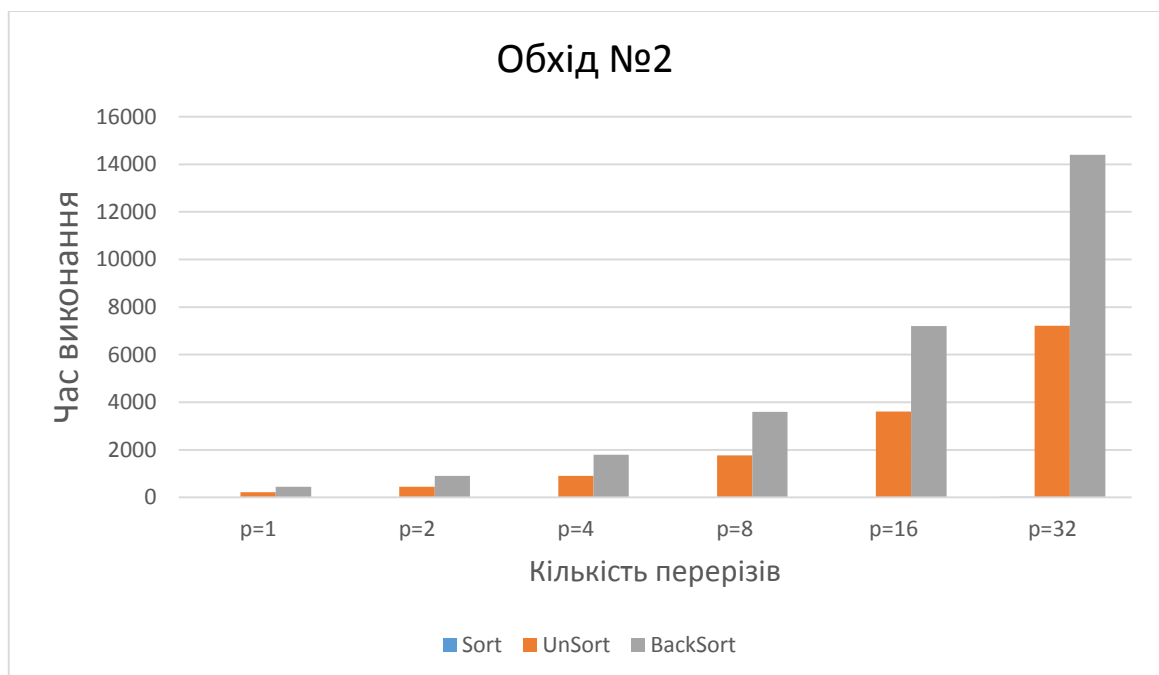
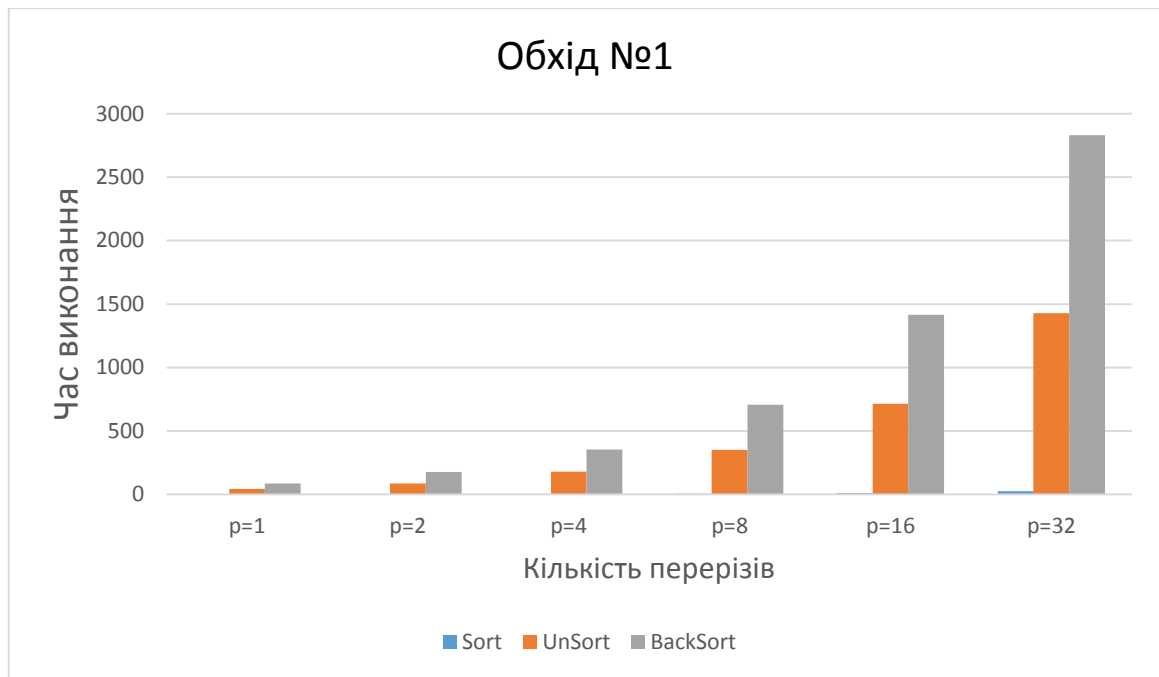
Таблиця №3.4 для масива $A[p,m,n]$; $p=8;m=20;n=20$;			
	Відсортований	Невідсортований	Обернено відсортований
Обхід №1	5,5	351,5	707,4
Обхід №2	6,6	1772,5	3600,4
Обхід №3	5,5	1012,3	2053,6
Вектор	0	480	928

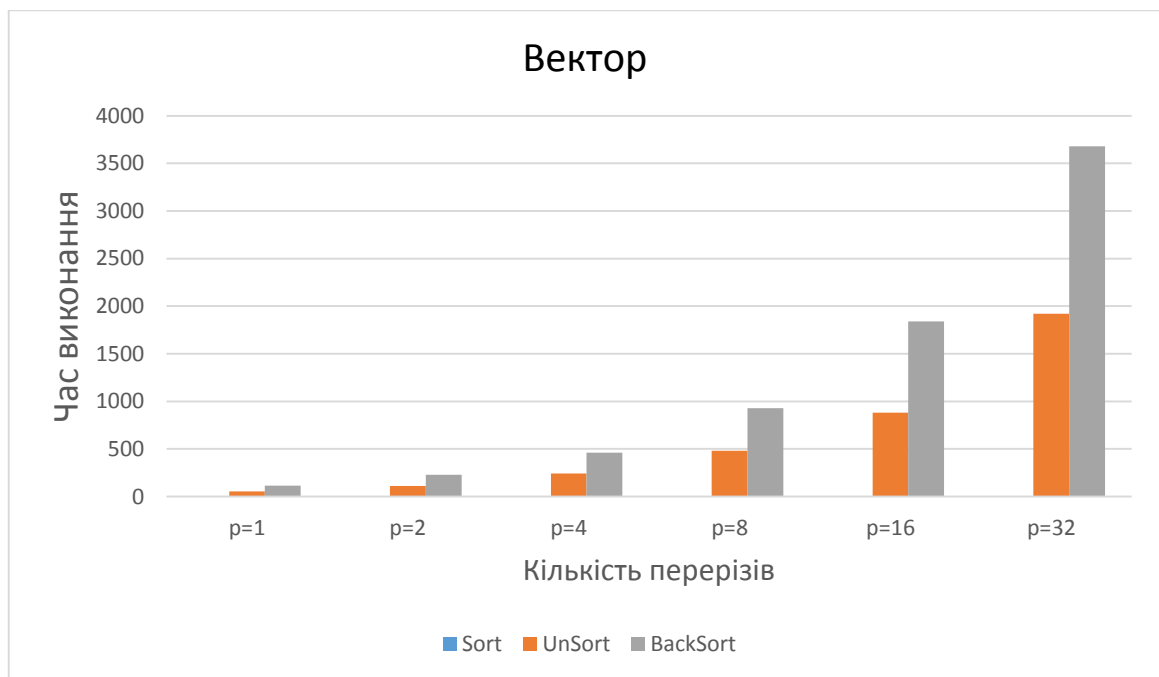
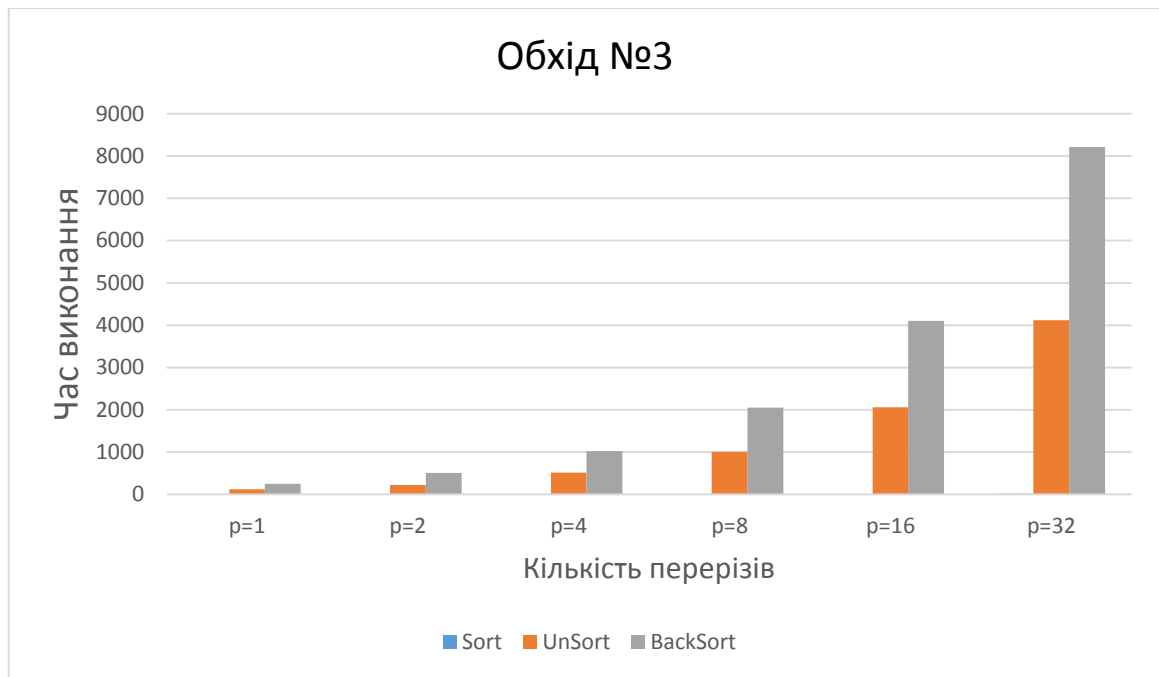
Таблиця №3.5 для масива $A[p,m,n]$; $p=16;m=20;n=20$;			
	Відсортований	Невідсортований	Обернено відсортований
Обхід №1	11	714,5	1415,5
Обхід №2	13,6	3611,2	7195,2
Обхід №3	11	2062,4	4105,6
Вектор	1	880	1840

Таблиця №3.6 для масива $A[p,m,n]$; $p=32;m=20;n=20$;			
	Відсортований	Невідсортований	Обернено відсортований
Обхід №1	27,4	1428	2829,9
Обхід №2	27,4	7213,9	14392,2
Обхід №3	21,5	4118,8	8212,4
Вектор	2	1920	3680

Так як час роботи алгоритмів при відсортованому масиві дуже малий порівняно з іншими видами заповнення масиву, то винесемо цей випадок у окремий графік.







4. Масив глибиною P та квадратними перерізами різної розмірності ($p=const$)

Результати тестування знову ж таки приведені у таблицях (заміри – у сотих долях секунди). Для коректності тестування та зменшення похибки роботи процесора, ми використовуємо середні значення часу виконання роботи алгоритмів з масивом (процедура пошуку середнього значення **Main_1** наведена у модулі **TEST**). Тестування виконувалось при швидкості процесора **3000 cycles**.

Пояснювання до скорочень у таблиці:

- Обхід №1: час виконання алгоритму "вставка – обмін", здійснюючи обхід з використанням додаткового одновимірного масиву;
- Обхід №2: час виконання алгоритму "вставка – обмін", здійснюючи обхід перетворюючи один індекс елементів "уявного" вектора у відповідні індекси елементів заданого двовимірного масиву;
- Обхід №3: час виконання алгоритму "вставка – обмін", здійснюючи обхід безпосередньо по елементах заданого двовимірного масиву;

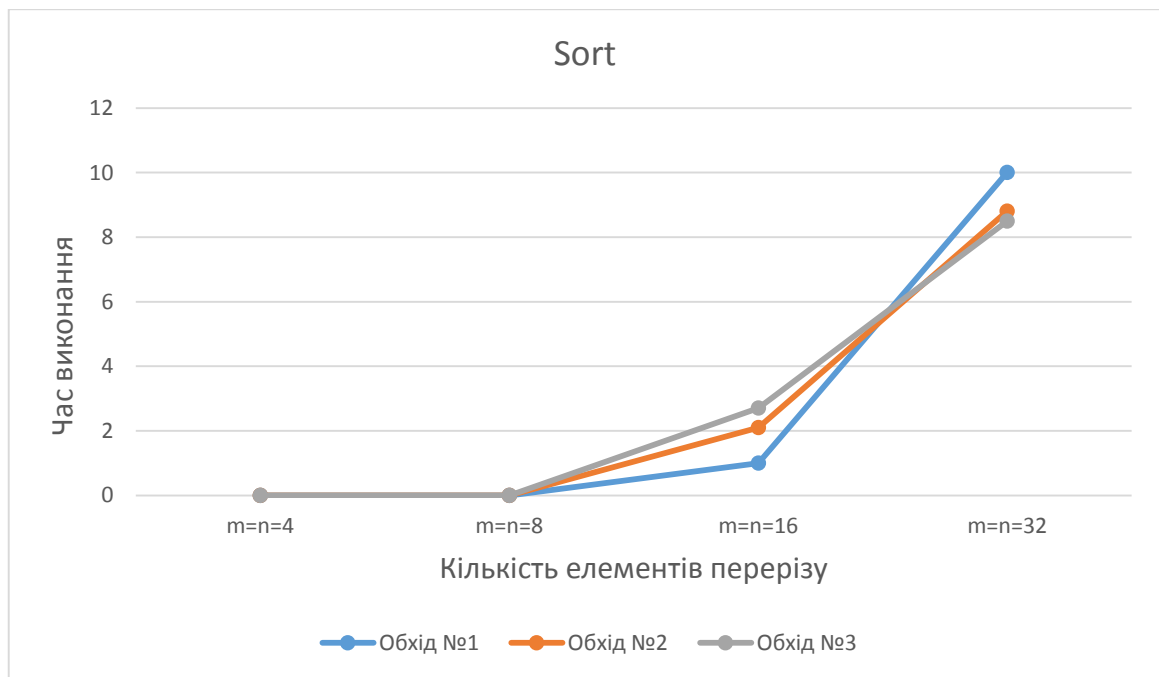
Таблиця №4.1 для масива $A[p,m,n]$; $p=4;m=4;n=4$;			
	Відсортований	Невідсортований	Обернено відсортований
Обхід №1	0	0,1	0,5
Обхід №2	0	1,1	2,7
Обхід №3	0	0,5	1,6

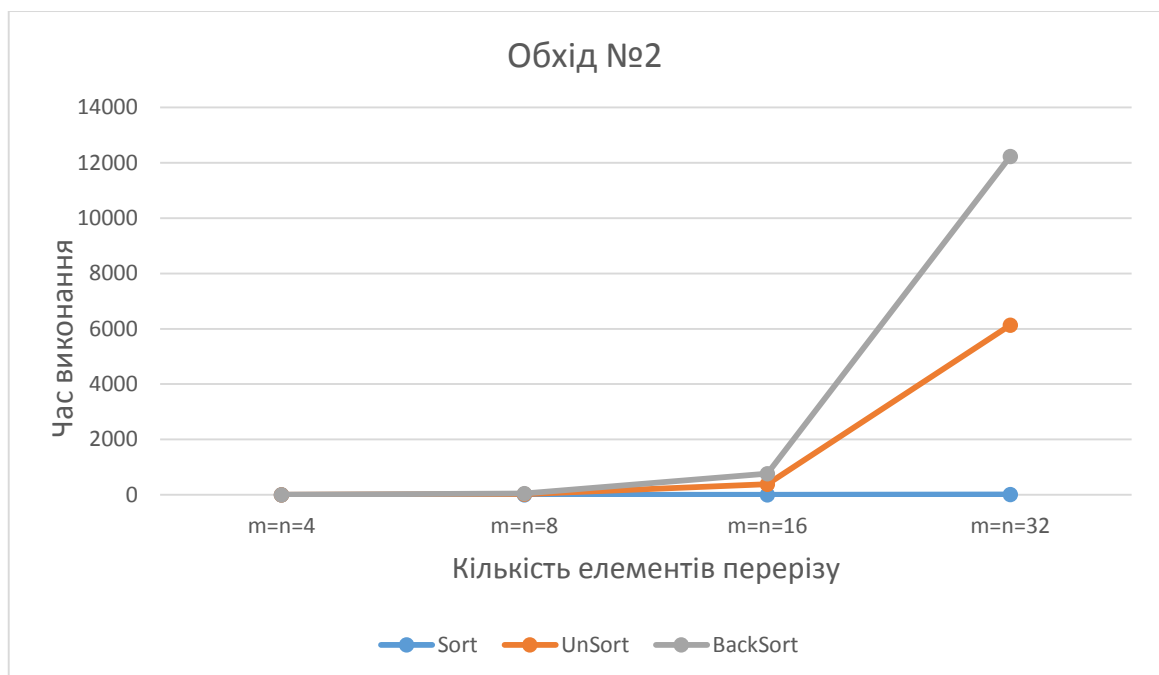
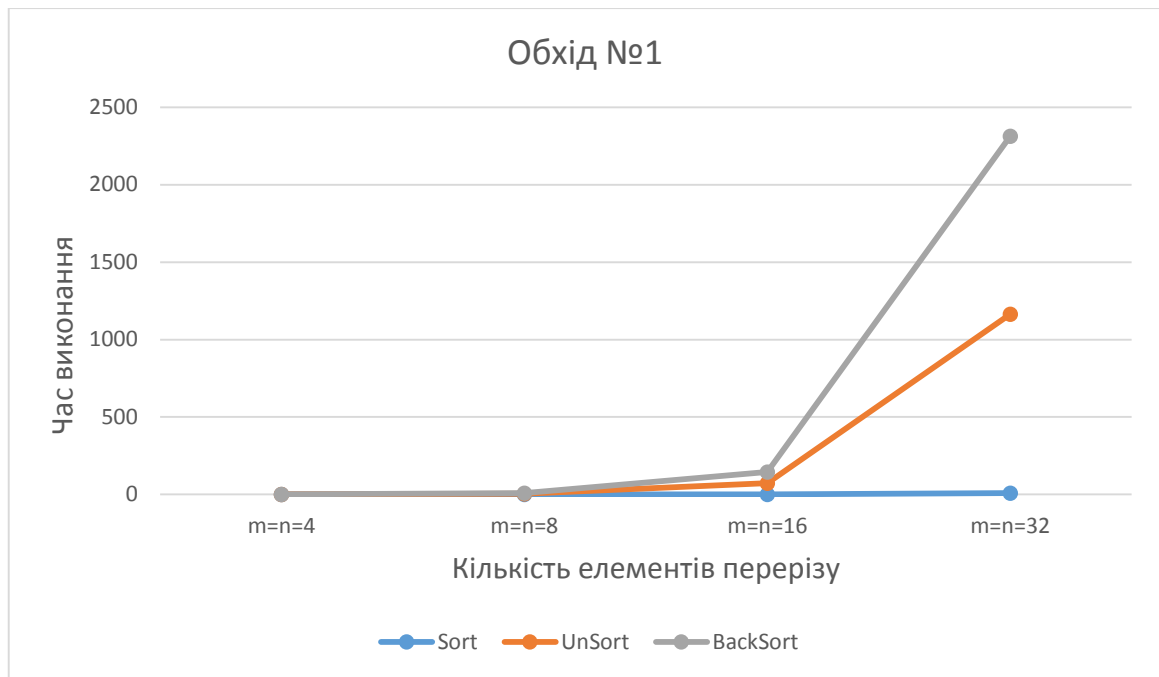
Таблиця №4.2 для масива $A[p,m,n]$; $p=4;m=8;n=8$;			
	Відсортований	Невідсортований	Обернено відсортований
Обхід №1	0	5	9,4
Обхід №2	0	22,5	47,8
Обхід №3	0	13,8	28

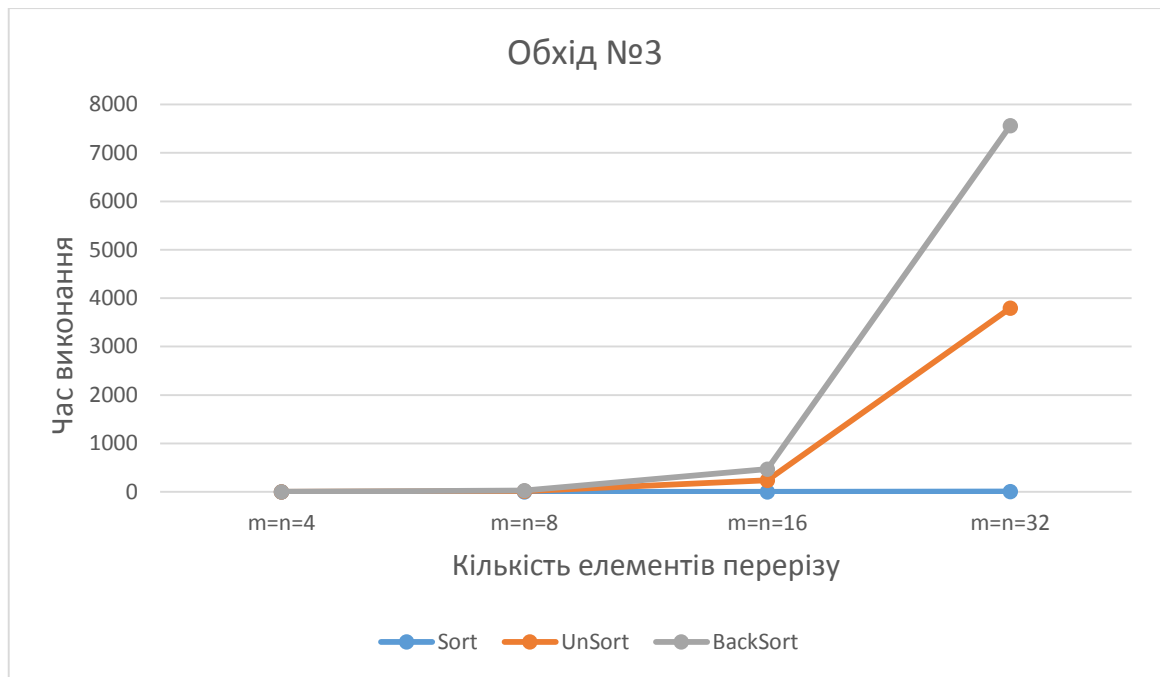
Таблиця №4.3 для масива $A[p,m,n]$; $p=4;m=16;n=16$;			
	Відсортований	Невідсортований	Обернено відсортований
Обхід №1	1	73	145,6
Обхід №2	2,1	379,5	762,2
Обхід №3	2,7	236,1	472,4

Таблиця №4.4 для масива $A[p,m,n]$; $p=4;m=32;n=32$;			
	Відсортований	Невідсортований	Обернено відсортований
Обхід №1	10	1164,3	2312,5
Обхід №2	8,8	6136,8	12226,5
Обхід №3	8,5	3797,1	7563

Так як час роботи алгоритмів при відсортованому масиві дуже малий порівняно з іншими видами заповнення масиву, то винесемо цей випадок у окремий графік.







Порівняльний аналіз алгоритмів

I. Порівняльний аналіз алгоритму відносно загальновідомої теорії

Результати для сортування одновимірного масива цілком відповідають загальній теорії.

На відсортованому векторі 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 $O(n)$ від довжини вектора.

Для обернено відсортованого вектора 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 $O(n^2)$. Така ситуація справді є найгіршою для алгоритму.

На випадково сформованому векторі 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.

II. Порівняльний аналіз алгоритму для багатовимірних масивів відносно результатів для одновимірного масиву

Сортування тривимірного масива відносно сортування вектора практично нічим у характері швидкодії не відрізняється, але є деякі нюанси.

З таблиці №2.1, №2.2 та таблиці №2.3 бачимо: за однакових значень 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). А таблиці №3.1-3.6, 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 практично лінійний характер $O(p)$.

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 напряду і для нього краще у плані швидкодії аби переріз був «ширшим», ніж «вищим». Це пов'язано з тим, що для даної задачі сортування, «наступним» елементом у більшості випадків є елемент попереднього рядка. Робота з переходу на новий рядок вимагає певного часу. У «високих» перерізів таких переходів набагато більше, ніж у

«широких», а, отже, і додаткового витраченого часу на перехід між рядками більше.

Одною з цікавих особливостей є порівняння часу сортування вектора та тривимірного масиву з використанням додаткового масиву. З таблиць №2.1-2.3 та таблиць №3.1-3.6 бачимо, що при однакових розмірах перерізу та вектора, час сортування вектора відбувається довше, ніж переписування елементів у вектор, їх сортування, та повернення їх назад до масиву.

По-перше, ця різниця пов'язана з похибками роботи компілятора. Час сортування масиву підраховується повністю, а час виконання сортування вектора підраховується за таким принципом: сортується вектор розмірністю одного перерізу ($m*n$), потім цей час помножається на кількість перерізів; тому навіть якщо була невелика похибка, при помноженні вона могла збільшитися.

По-друге, додатковий вектор, з яким працює алгоритм при обході №1, описується локально, і знаходиться він у власній внутрішній швидкій пам'яті процесора (кеші), і як висновок, час роботи сортування менший, навіть з додатковими операціями переписування (оскільки час на початкові і кінцеві копіювання елементів – малий у порівнянні з часом на виконання основної частини алгоритму), ніж сортування вектора, який описаний глобально, і вимагатиме запиту до повільної відносно частоти роботи процесора оперативної пам'яті.

Більший час сортування тривимірного масива (у нашому випадку, окрім сортування за допомогою додаткового вектора) пояснюється деякими додатковими операціями, що виконуються під час роботи алгоритмів. При обході з перетворенням координат виконуються додаткові «важкі» арифметичні операції (цілочисельне ділення, взяття остачі, множення, додавання). При прямому обході виконуються «зайві присвоєння» (координати на попередній елемент, різного роду межі роботи циклів) та порівняння (наприклад, для продовження внутрішнього циклу з передумовою необхідно зробити три перевірки, а не дві, тощо).

III. Порівняльний аналіз алгоритмів на багатовимірних масивах між собою

На відсортованому масиві час сортування усіма методами обходу 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 $O(n)$ від кількості елементів масиву.

Для обернено впорядкованого масива залежність часу від кількості елементів різко відрізняється від лінійної і, судячи з швидкості росту значень по вертикальній осі графіка, можна стверджувати, що вона наближається до квадратичної $O(n^2)$. 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.

Як можна помітити на графіках, найкращий алгоритм сортування – з використанням додаткового вектора. Найгіршим випадком є сортування з проходом по масиву з перетворенням координат.

Алгоритм з використанням додаткового вектора іноді програє іншим лише у випадках 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 внутрішнього циклу з передумовою і, за рахунок цього можуть «обіграти» перший за швидкістю, 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 та звертання пам'яті до сортованої частини відбувається швидше, тому що вектор, у який переписуються елементи, описується локально, а при виконанні інших алгоритмів звертання відбувається до глобальних даних. 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 або такий же, або навіть кращий результат, ніж перший, лише тоді, коли масив відсортований. Проблема у багатьох додаткових присвоєннях, порівняннях, перевірках. Проте даний алгоритм прямого обходу не потребує великої кількості додаткової пам'яті (на рівні локальних змінних). Також цей алгоритм, як видно, швидший за алгоритм з перетвореннями координат. Це пояснюється тим, що арифметичні операції ділення/взяття остачі і множення «важчі» за операції присвоєння чи, тим більше, порівняння (останнє виконується дуже швидко).

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.

На відсортованому масиві швидкодія має практично лінійну залежність $O(n)$ від кількості елементів масиву.

Для обернено впорядкованого масива залежність часу від кількості елементів наближається до квадратичної $O(n^2)$.

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.

За однакових значень кількості перерізів і кількості елементів у перерізі результати майже однакові, незалежно від конкретної кількості елементів (виняток становить сортування масивів з використанням прямого обходу, висота і ширина перерізів яких суттєво відрізняються), тобто геометричний розмір майже не впливає на час сортування.

Параметр кількості перерізів є лише коефіцієнтом, а, значить, за умови незмінної кількості елементів у перерізі, залежність швидкодії від кількості перерізів (P) має практично лінійний характер $O(p)$.

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 порівнянні з перетвореннями координат і іноді 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 швидкістю й може використовуватись лише на невеликих масивах в системах не критичних до часу виконання або для учбових цілей.

Отже, цими тестами ще раз було підтверджено «золоте» правило алгоритмізації:

Виграти можливо у чомусь одному: або у використанні пам'яті, або у швидкодії, або у простоті реалізації.

Список використаної літератури

1. Методичні матеріали для виконання курсової роботи.
2. Марченко О., Марченко Л. Turbo Pascal 7.0. Базовий курс. – К.: ВЕК, 2004. – С. 359-362.
3. Т.Кормен, Ч.Лейзерсон, Р.Ривест, К.Штайн - Алгоритмы. Построение и анализ, 3-е изд : Пер. с англ. – М. : ООО «И. Д. Вильямс», 2013. – 1328с. : ил. – Парал. тит. англ.