Житомирський державний технологічний університет

Факультет інформаційно-комп’ютерних технологій

Кафедра інженерії програмного забезпечення

# **ЗВІТ**

**З ЛАБОРАТОРНИХ РОБІТ**

з «Алгоритми та структури даних»

(назва дисципліни)

Студента 1 курсу ВТ-21-1 групи

напряму підготовки Веб-технології

спеціальності 121 «Інженерія програмного забезпечення»

Маньківського Владислава Вячеславовича

(прізвище та ініціали)

Керівник Старший викладач кафедри ІПЗ Локтікова Т. М.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

Житомир – 2022 рік

Зміст

[**ЗВІТ** 1](#_Toc108458849)

[**Лабораторна робота № 1** 1](#_Toc108458850)

[**Лабораторна робота № 2** 1](#_Toc108458851)

[**Лабораторна робота № 3** 1](#_Toc108458852)

[**Лабораторна робота № 4** 1](#_Toc108458853)

[**Лабораторна робота № 5** 1](#_Toc108458854)

[**Лабораторна робота № 6** 1](#_Toc108458855)

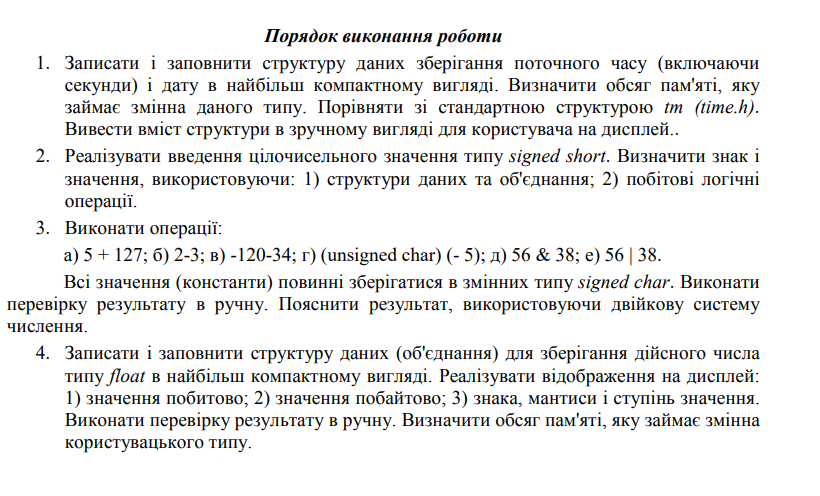
[**Лабораторна робота № 7-8** 1](#_Toc108458856)

# **Лабораторна робота № 1**

РОБОТА З БАЗОВИМИ ТИПАМИ ДАНИХ

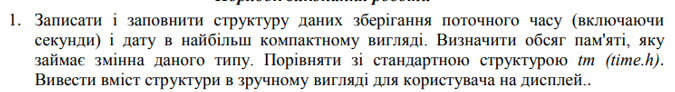
***Мета*** : отримати практичні навички по роботі з базовими типами даних (простими і складними типами даних).

**1.1 Хід роботи**

****

1.1.1

**Завдання 1**:



Лістинг:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <windows.h>

#include <stdlib.h>

#include <time.h>

struct date {

unsigned short Year : 12;

unsigned short Month : 4;

unsigned short Day : 5;

unsigned short Hour : 5;

unsigned short Min : 6;

unsigned short Sec : 6;

};

enum dow {

Mon = 1,

Tue,

Wed,

Thu,

Fri,

Sat,

Sun

};

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

time\_t now = time(0);

struct date data;

int y, m, d, h, mm, s;

char k[20];

printf("Введіть номер дня тижня ");

dow doo;

scanf\_s("%d", &doo);

switch (doo) {

case Mon:

strcpy\_s(k, "Понеділок");

break;

case Tue:

strcpy\_s(k, "Вівторок");

break;

case Wed:

strcpy\_s(k, "Середа");

break;

case Thu:

strcpy\_s(k, "Четвер");

break;

case Fri:

strcpy\_s(k, "П'ятниця");

break;

case Sat:

strcpy\_s(k, "Субота");

break;

case Sun:

strcpy\_s(k, "Неділя");

break;

default:break;

}

printf("Введіть рік:");

scanf\_s("%d", &y);

printf("Введіть місяць:");

scanf\_s("%d", &m);

printf("Введіть день:");

scanf\_s("%d", &d);

printf("Введіть годину:");

scanf\_s("%d", &h);

printf("Введіть хвилини:");

scanf\_s("%d", &mm);

printf("Введіть секунди:");

scanf\_s("%d", &s);

data.Year = y;

data.Month = m;

data.Day = d;

data.Hour = h;

data.Min = mm;

data.Sec = s;

time\_t rawtime;

struct tm\* timeinfo;

time(&rawtime);

timeinfo = localtime(&rawtime);

printf("%s", asctime(timeinfo));

printf("%s, %d.%d.%d %d:%d:%d", k, data.Day, data.Month, data.Year, data.Hour, data.Min, data.Sec);

int n1, n2;

n1 = sizeof(tm);

n2 = sizeof(date);

printf("\nРозмір tm-%d розмір власної структури-%d", n1, n2);

return 0;

}

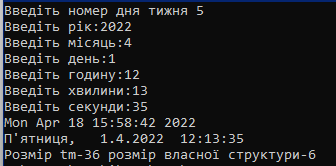
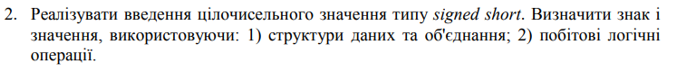


Рисунок 1.1 – Результат виконання завдання 1

1.1.2

**Завдання 2**:



Лістинг:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <string.h>

#include <windows.h>

#include <math.h>

#include <conio.h>

union binary {

signed short number\_binary;

struct {

unsigned char zero : 1;

unsigned char one : 1;

unsigned char two : 1;

unsigned char three : 1;

unsigned char four : 1;

unsigned char five : 1;

unsigned char six : 1;

unsigned char seven : 1;

unsigned char eight : 1;

unsigned char nine : 1;

unsigned char ten : 1;

unsigned char eleven : 1;

unsigned char twelve : 1;

unsigned char thirteen : 1;

unsigned char fourteen : 1;

unsigned char fiveteen : 1;

}bytes;

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

signed short num;

binary binary;

do {

printf("\n\nДля закінчення програми введіть 0.\nВведіть ціле значення = ");

scanf("%d", &num);

binary.number\_binary = num;

if (binary.bytes.fiveteen == 1) {

printf("\nЧисло %d є від'ємним, бо знаковий біт = %d.", binary.number\_binary, binary.bytes.fiveteen);

}

else {

printf("\nЧисло %d є додатнім, бо знаковий біт = %d.", binary.number\_binary, binary.bytes.fiveteen);

}

} while (num != 0);

return 0;

}

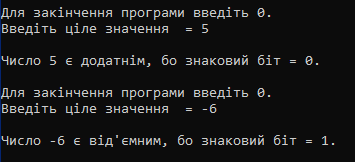
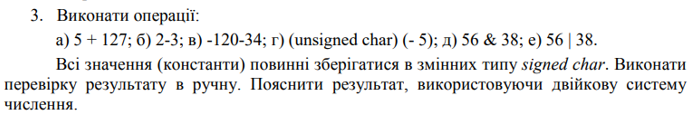


Рисунок 1.2 – Результат виконання завдання 2

1.1.2

**Завдання 3**:

Лістинг:



#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <math.h>

#include <Windows.h>

#include <string.h>

#include <conio.h>

#include <time.h>

struct ch {

signed char a = 5;

signed char b = 127;

signed char c = 2;

signed char d = -3;

signed char f = -120;

signed char e = -34;

signed char g = -5;

signed char l = 56;

signed char k = 38;

signed char rez;

}x;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

x.rez = x.a + x.b;

printf("5+127=%d", x.rez);

printf("\nВиходить за межі значень типу signed char");

x.rez = 0;

x.rez = x.c + x.d;

printf("\n\n2-3=%d", x.rez);

printf("\nНе виходить за межі значень типу signed char");

x.rez = 0;

x.rez = x.f + x.e;

printf("\n\n-120-34=%d", x.rez);

printf("\nВиходить за межі значень типу signed char");

x.rez = 0;

x.rez = (unsigned char)(x.g);

printf("\n\nunsigned char -5=%d", x.rez);

printf("\nНічого не відбувається");

x.rez = 0;

x.rez = x.l & x.k;

printf("\n\n56 & 38=%d", x.rez);

printf("\nЛогічна дія AND");

x.rez = 0;

x.rez = x.l | x.k;

printf("\n\n56 | 38=%d", x.rez);

printf("\nЛогічна дія OR\n");

return 0;

}

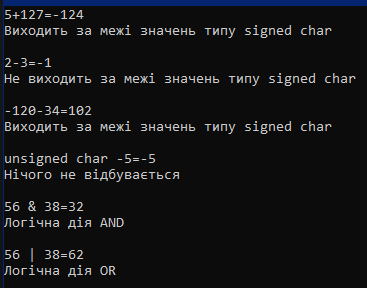
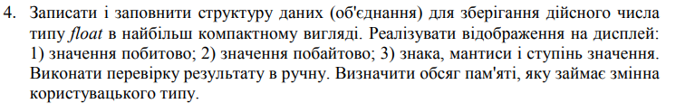


Рисунок 1.3 – Результат виконання завдання 3

1.1.4

**Завдання 4**:



Лістинг:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <math.h>

#include <Windows.h>

#include <string.h>

#include <conio.h>

#include <time.h>

union radiance {

float number;

struct {

unsigned char zero : 1;

unsigned char one : 1;

unsigned char two : 1;

unsigned char three : 1;

unsigned char four : 1;

unsigned char five : 1;

unsigned char six : 1;

unsigned char seven : 1;

unsigned char eight : 1;

unsigned char nine : 1;

unsigned char ten : 1;

unsigned char eleven : 1;

unsigned char twelve : 1;

unsigned char thirteen : 1;

unsigned char fourteen : 1;

unsigned char fiveteen : 1;

unsigned char sixteen : 1;

unsigned char seventeen : 1;

unsigned char eighteen : 1;

unsigned char nineteen : 1;

unsigned char twenty : 1;

unsigned char twenty\_one : 1;

unsigned char twenty\_two : 1;

unsigned char twenty\_three : 1;

unsigned char twenty\_four : 1;

unsigned char twenty\_five : 1;

unsigned char twenty\_six : 1;

unsigned char twenty\_seven : 1;

unsigned char twenty\_eight : 1;

unsigned char twenty\_nine : 1;

unsigned char thirty : 1;

unsigned char thirty\_one : 1;

}bytes;

}radiance;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

printf("Введіть змінну = "); scanf("%f", &radiance.number);

printf("\nЗначення побитово:\n");

printf("|%d|%d %d %d %d %d %d %d %d|%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d % d % d % d % d % d % d % d % d | \n", radiance.bytes.thirty\_one, radiance.bytes.thirty,radiance.bytes.twenty\_nine, radiance.bytes.twenty\_eight,radiance.bytes.twenty\_seven, radiance.bytes.twenty\_six,radiance.bytes.twenty\_five, radiance.bytes.twenty\_four,radiance.bytes.twenty\_three, radiance.bytes.twenty\_two,radiance.bytes.twenty\_one, radiance.bytes.twenty, radiance.bytes.nineteen,radiance.bytes.eighteen, radiance.bytes.seventeen, radiance.bytes.sixteen,radiance.bytes.fiveteen, radiance.bytes.fourteen, radiance.bytes.thirteen,radiance.bytes.twelve, radiance.bytes.eleven, radiance.bytes.ten,radiance.bytes.nine, radiance.bytes.eight, radiance.bytes.seven,radiance.bytes.six, radiance.bytes.five, radiance.bytes.four,radiance.bytes.three, radiance.bytes.two, radiance.bytes.one,radiance.bytes.zero);

printf(" | | |\n");

printf("знак ступінь мантиса\n");

printf("\nЗначення побайтово:\n");

printf("|%d %d %d %d %d %d %d %d|%d %d %d %d %d %d %d %d|%d %d %d %d %d %d %d % d | % d % d % d % d % d % d % d % d | \n", radiance.bytes.thirty\_one, radiance.bytes.thirty,radiance.bytes.twenty\_nine, radiance.bytes.twenty\_eight,radiance.bytes.twenty\_seven, radiance.bytes.twenty\_six,radiance.bytes.twenty\_five, radiance.bytes.twenty\_four,radiance.bytes.twenty\_three, radiance.bytes.twenty\_two,radiance.bytes.twenty\_one, radiance.bytes.twenty, radiance.bytes.nineteen,radiance.bytes.eighteen, radiance.bytes.seventeen, radiance.bytes.sixteen,radiance.bytes.fiveteen, radiance.bytes.fourteen, radiance.bytes.thirteen,radiance.bytes.twelve, radiance.bytes.eleven, radiance.bytes.ten,radiance.bytes.nine, radiance.bytes.eight, radiance.bytes.seven,radiance.bytes.six, radiance.bytes.five, radiance.bytes.four,radiance.bytes.three, radiance.bytes.two, radiance.bytes.one,radiance.bytes.zero);

float s;

printf("\nОбяг пам'яті змінної = %d;\nОбсяг пам'яті union = % d.", sizeof(s), sizeof(radiance));

return 0;

}

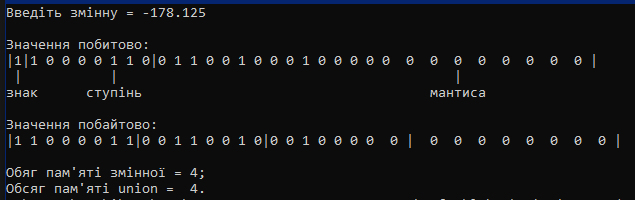


Рисунок 1.4 – Результат виконання завдання 4

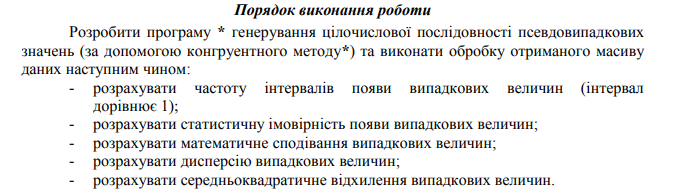
***Висновки:*** я отримав практичні навички по роботі з базовими типами даних (простими і складними типами даних) та навчився складати за ними програми.

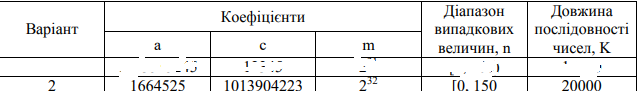
# **Лабораторна робота № 2**

ГЕНЕРУВАННЯ ПОСЛІДОВНОСТІ ПСЕВДОВИПАДКОВИХ ЗНАЧЕНЬ

***Мета*** : ознайомитись з методами генерування випадкових чисел, а також формуванням та обробкою масивів даних.

**2.1 Хід роботи**

****

****

2.1.1

**Завдання**:

Лістинг:

int a = 1664525, c = 1013904223, n = 150, x = 0;

double k = 20000, m = 0, d = 0, q = 0;

int mod = (int)Math.Pow(2, 32);

int[] l = new int[n];

double[] p = new double[n];

int[] mass = new int[20000];

for (int i = 0; i < k ;)

{

x = (a \* x + c) % mod;

mass[i] = x % n;

if (x >= 0)

{

Console.WriteLine($"{i+1} - {x % n}");

i++;

}

}

Console.WriteLine();

for (int i = 0; i < k; i++)

{

l[mass[i]] = l[mass[i]] + 1;

}

for (int i = 0; i < k; i++)

{

p[mass[i]] = l[mass[i]] / k;

}

for (int i = 0; i < n; i++)

Console.WriteLine($"P({i}) = {p[i]} ({l[i]})");

for (int i = 0; i < n; i++)

{

m = m + i \* p[i];

}

Console.WriteLine($"\nM(X)={m:f2}");

for (int i = 0; i < n; i++)

{

d = d + Math.Pow((i - m), 2) \* p[i];

}

Console.WriteLine($"D(X)={d:f2}");

q = Math.Sqrt(d);

Console.WriteLine($"q(X)={q:f2}");

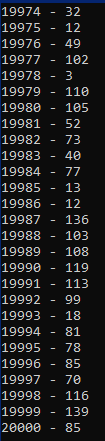
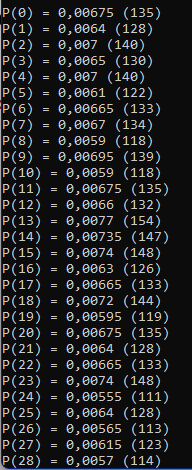
  

Рисунок 2.1 – Результат виконання завдання

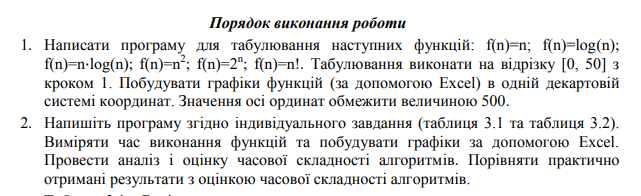
***Висновки:*** я ознайомився з методами генерування випадкових чисел, а також формуванням та обробкою масивів даних.

# **Лабораторна робота № 3**

ОЦІНКА ЧАСОВОЇ СКЛАДНОСТІ АЛГОРИТМІВ

***Мета*** : набуття навичок дослідження часової складності алгоритмів і визначення її асимптотичних оцінок.

**3.1 Хід роботи**

****

****

****

3.1.1

**Завдання 1**:

Лістинг:

Console.WriteLine("f(n) = n");

Console.WriteLine("x\ty");

double f;

for(int n = 0; n <= 50; n++)

{

f = n;

Console.WriteLine($"{n}\t{f}");

}

Console.WriteLine("f(n) = lg(n)");

Console.WriteLine("x\ty");

for (int n = 0; n <= 50; n++)

{

f = Math.Log10(n);

Console.WriteLine($"{n}\t{Math.Round(f, 4)}");

}

Console.WriteLine("f(n) = n \* lg(n)");

Console.WriteLine("x\ty");

for (int n = 0; n <= 50; n++)

{

f = n \* Math.Log10(n);

Console.WriteLine($"{n}\t{Math.Round(f, 4)}");

}

Console.WriteLine("f(n) = n^2");

Console.WriteLine("x\ty");

for (int n = 0; n <= 50; n++)

{

f = Math.Pow(n, 2);

Console.WriteLine($"{n}\t{Math.Round(f, 4)}");

}

Console.WriteLine("f(n) = 2^n");

Console.WriteLine("x\ty");

for (int n = 0; n <= 50; n++)

{

f = Math.Pow(2, n);

Console.WriteLine($"{n}\t{Math.Round(f, 4)}");

}

Console.WriteLine("f(n) = n!");

Console.WriteLine("x\ty");

Console.WriteLine($"{0}\t{1}");

for (int n = 1; n <= 50; n++)

{

f = 1;

for (int k = 1; k <= n; k++)

{

f = f \* k;

}

Console.WriteLine($"{n}\t{Math.Round(f, 4)}");

}

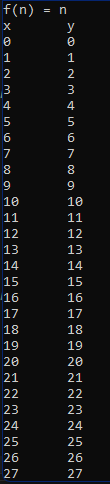
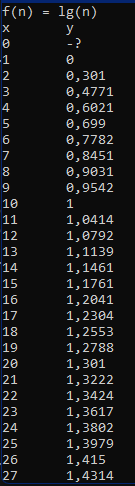
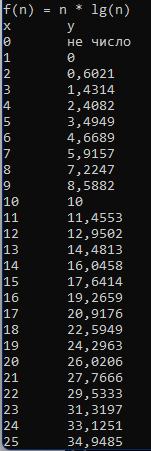
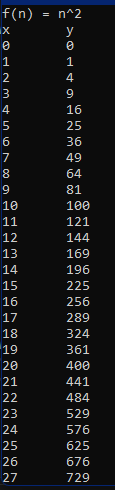
   

Рисунок 3.1 – Результат виконання завдання 1

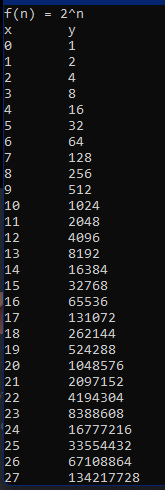
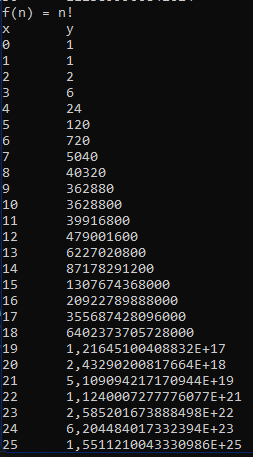
 

Рисунок 3.2 – Результат виконання завдання 1

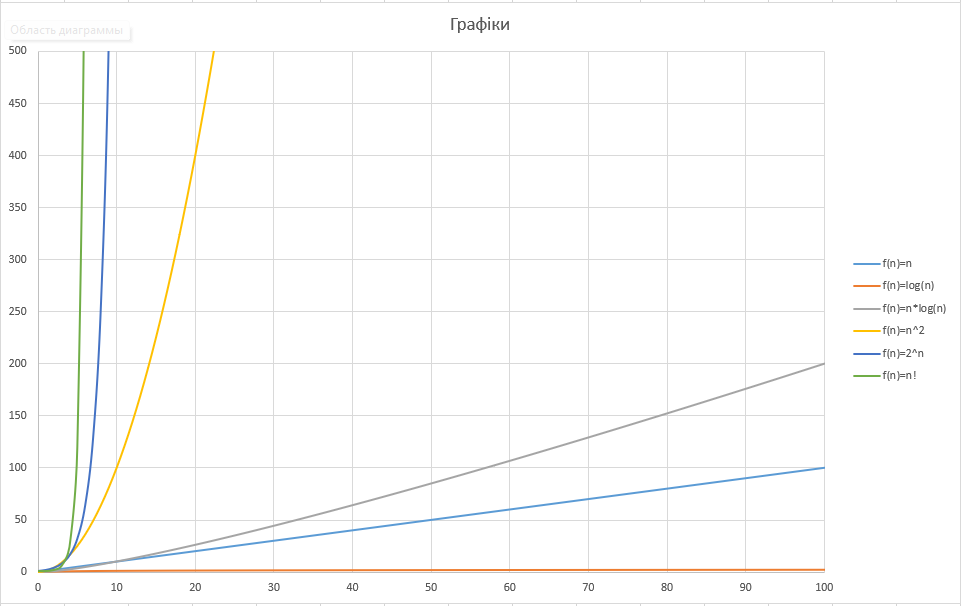


Рисунок 3.3 – Результат виконання завдання 1

**Завдання 2.1**:



Лістинг:

using System;

using System.Text;

using System.Diagnostics;

class Program

{

static void Main()

{

Console.OutputEncoding = System.Text.Encoding.Default;

Stopwatch stopwatch = new Stopwatch();

Console.WriteLine("Число a, де 0<=a<=20");

bool n0;

int a;

do

{

n0 = true;

if (int.TryParse(Console.ReadLine(), out a) && a >=0 && a<=20)

{

n0 = false;

}

else

{

Console.WriteLine("Введіть ще раз");

}

} while (n0);

stopwatch.Start();

Console.WriteLine($"{a}! = {fact(a)}");

stopwatch.Stop();

Console.WriteLine($"\nБуло витрачено {stopwatch.ElapsedMilliseconds \* 1000000} наносекунд;");

}

static double fact(int a)

{

if (a <= 1) return 1;

return a \* fact(a - 1);

}

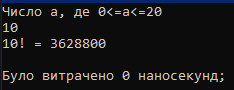
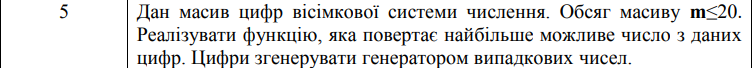


Рисунок 3.4 – Результат виконання завдання 2.1

**Завдання 2.2**:



Лістинг:

using System;

using System.Text;

using System.Diagnostics;

class Program

{

static void Main()

{

Console.OutputEncoding = System.Text.Encoding.Default;

Stopwatch stopwatch = new Stopwatch();

Console.WriteLine("Число m, де m<=20");

bool n0;

int m;

double max = 0, zero = 1, maxn = 0;

do

{

n0 = true;

if (int.TryParse(Console.ReadLine(), out m) && m >= 0 && m <= 20)

{

n0 = false;

}

else

{

Console.WriteLine("Введіть ще раз");

}

} while (n0);

stopwatch.Start();

int []mass = new int[m];

Random rnd = new Random();

for (int i = 0; i < m; i++)

{

mass[i] = rnd.Next(0, 8);

Console.Write($"{mass[i]}\t");

zero \*= 10;

}

int s = 0;

for (int i = 0; i < m; i++)

{

zero /= 10;

maxn = 0;

for (int j = 0; j < m; j++)

{

if (mass[j] > maxn && mass[j] != -1)

{

maxn = mass[j];

s = j;

}

}

max += maxn \* zero;

mass[s] = -1;

}

Console.WriteLine($"\nНайбільш можливе -> {max}");

stopwatch.Stop();

Console.WriteLine($"\nБуло витрачено {stopwatch.ElapsedMilliseconds \* 1000000} наносекунд;");

}

}

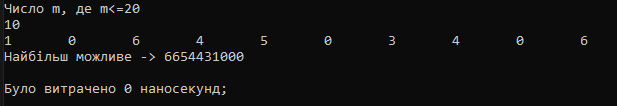


Рисунок 3.3 – Результат виконання завдання 2.1

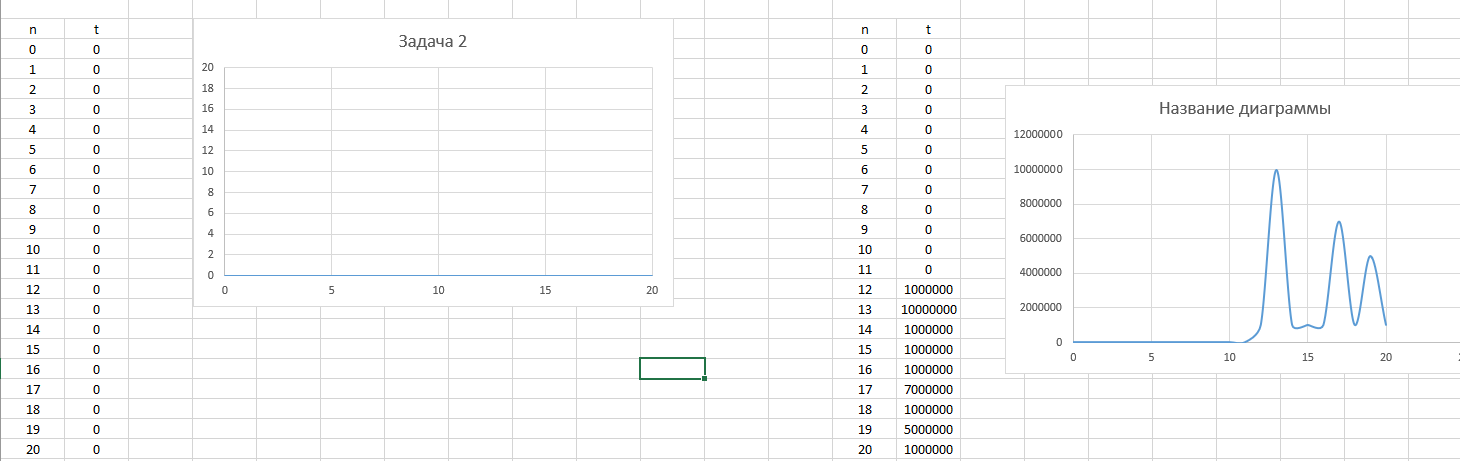


Рисунок 3.4 – Результат виконання завдання 2

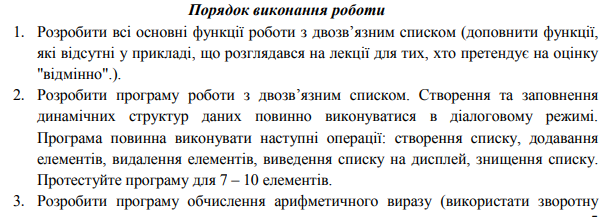
***Висновки:*** я набув навичок дослідження часової складності алгоритмів і визначення її асимптотичних оцінок.

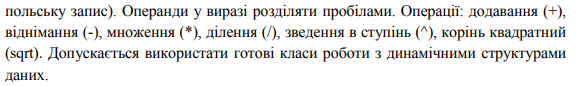
# **Лабораторна робота № 4**

ЗВ’ЯЗНИЙ СПИСОК, СТЕК, ЧЕРГА. ЗВОРОТНІЙ ПОЛЬСЬКИЙ ЗАПИС

***Мета*** : ознайомитися з основами роботи з двозв’язним списком, однозв’язним списком, стеком та чергою. Розробити основні функції для обчислення арифметичного виразу, записаного з використанням зворотного польського запису.

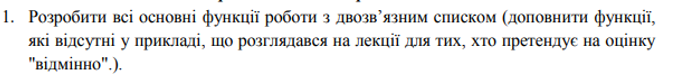
**4.1 Хід роботи**

****

****

4.1.1

**Завдання 1**:



Лістинг:

#include <stdio.h>

#include <stdlib.h>

typedef int elemtype;

struct elem {

elemtype value;

struct elem\* next;

struct elem\* prev;

};

struct myList {

struct elem\* head;

struct elem\* tail;

int size;

};

typedef struct elem cNode;

typedef struct myList cList;

cList\* createList(void) {

cList\* list = (cList\*)malloc(sizeof(cList));

if (list) {

list->size = 0;

list->head = list->tail = NULL;

}

return list;

}

bool isEmptyList(cList\* list) {

return ((list->head == NULL) || (list->tail == NULL));

}

cNode\* getNode(cList\* list, int index) {

cNode\* node = NULL;

int i;

if (index >= list->size) {

return (NULL);

}

if (index < list->size / 2) {

i = 0;

node = list->head;

while (node && i < index) {

node = node->next;

i++;

}

}

else {

i = list->size - 1;

node = list->tail;

while (node && i > index) {

node = node->prev;

i--;

}

}

return node;

}

int pushFront(cList\* list, elemtype\* data) {

cNode\* node = (cNode\*)malloc(sizeof(cNode));

if (!node) {

return(-1);

}

node->value = \*data;

node->next = list->head;

node->prev = NULL;

if (!isEmptyList(list)) {

list->head->prev = node;

}

else {

list->tail = node;

}

list->head = node;

list->size++;

return(0);

}

int pushBack(cList\* list, elemtype\* data) {

cNode\* node = (cNode\*)malloc(sizeof(cNode));

if (!node) {

return(-3);

}

node->value = \*data;

node->next = NULL;

node->prev = list->tail;

if (!isEmptyList(list)) {

list->tail->next = node;

}

else {

list->head = node;

}

list->tail = node;

list->size++;

return(0);

}

int pushPosition(cList\* list, elemtype\* data)

{

int index = (int)(list->size / 2);

if (index == 0)

{

return pushFront(list, data);

}

if (index<0 || index>list->size - 1)

{

return -1;

}

cNode\* next = getNode(list, index - 1)->next;

cNode\* prev = getNode(list, index)->prev;

cNode\* node = (cNode\*)malloc(sizeof(cNode));

node->value = \*data;

getNode(list, index - 1)->next = node;

getNode(list, index)->prev = node;

node->next = next;

node->prev = prev;

list->size++;

}

int popFront(cList\* list) {

cNode\* node;

if (isEmptyList(list)) {

return(-2);

}

node = list->head;

list->head = list->head->next;

if (!isEmptyList(list)) {

list->head->prev = NULL;

}

else {

list->tail = NULL;

}

list->size--;

free(node);

return(0);

}

int popBack(cList\* list) {

cNode\* node = NULL;

if (isEmptyList(list)) {

return(-4);

}

node = list->tail;

list->tail = list->tail->prev;

if (!isEmptyList(list)) {

list->tail->next = NULL;

}

else {

list->head = NULL;

}

list->size--;

free(node);

return(0);

}

int popPosition(cList\* list)

{

int index = (int)(list->size / 2);

if (index == 0)

{

elemtype tmp;

return popFront(list);

}

if (index == list->size - 1)

{

elemtype tmp;

return popBack(list);

}

if (index<0 || index>list->size - 1)

{

return -1;

}

cNode\* next = getNode(list, index)->next;

cNode\* prev = getNode(list, index)->prev;

free(getNode(list, index));

getNode(list, index - 1)->next = next;

getNode(list, index + 1)->prev = prev;

list->size--;

return 0;

}

void printList(myList\* numbers) {

if(!isEmptyList(numbers)){

cNode\* node = numbers->head;

for(int i = 1; i <= numbers->size; i++){

printf("\n%i. %i", i, node->value);

node = node->next;

}

} else {

printf("Список пустий");

}

}

int main()

{

system("chcp 1251");

system("cls");

createList();

int x = 0;

elemtype first = 1;

elemtype second = 2;

elemtype third = 3;

elemtype tmp;

cList\* list = createList();

do {

printList(list);

printf("\n\n1)додавання елемента на початої списку (1)\n");

printf("2)додавання елемента в середину списку (2)\n");

printf("3)додавання елемента в кінець списку (3)\n");

printf("4)видалення елемента з початку списку\n");

printf("5)видалення елемента з середини списку\n");

printf("6)видалення елемента з кінця списку\n");

printf("Введіть дію: ");

scanf\_s("%d", &x);

switch (x)

{

case 1:

pushFront(list, &first);

break;

case 2:

pushPosition(list, &second);

break;

case 3:

pushBack(list, &third);

break;

case 4:

popFront(list);

break;

case 5:

popPosition(list);

break;

case 6:

popBack(list);

break;

default:

x = -1;

break;

}

} while (x != -1);

return 0;

}

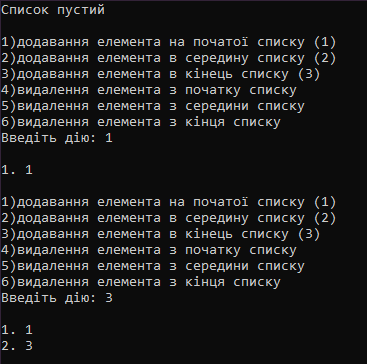


Рисунок 4.1 – Результат виконання завдання 1

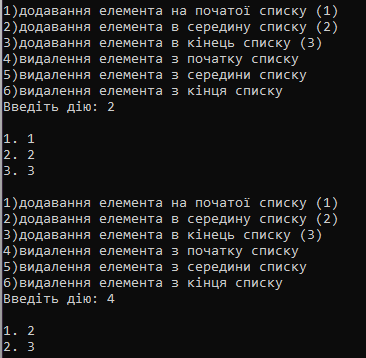


Рисунок 4.2 – Результат виконання завдання 1

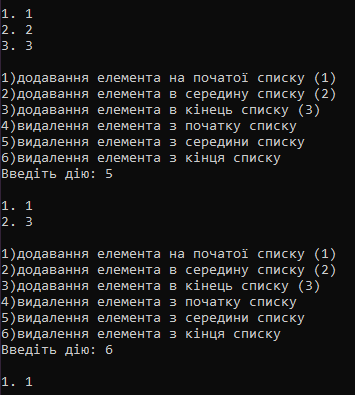
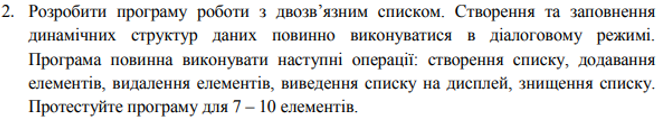


Рисунок 4.3 – Результат виконання завдання 1

**Завдання 2**:



Лістинг:

System.Globalization.CultureInfo customCulture = (System.Globalization.CultureInfo)

System.Threading.Thread.CurrentThread.CurrentCulture.Clone();

customCulture.NumberFormat.NumberDecimalSeparator = ".";

System.Threading.Thread.CurrentThread.CurrentCulture = customCulture;

Console.OutputEncoding = System.Text.Encoding.Default;

bool n0;

int x;

int n;

LinkedList<int> list = new LinkedList<int>();

Console.WriteLine("Введіть кількість елементів списку:");

do

{

n0 = true;

if (int.TryParse(Console.ReadLine(), out n) && n > 0)

{

n0 = false;

}

else

{

Console.WriteLine("Введіть ще раз");

}

} while (n0);

for (int i = 1; i <= n; i++)

{

list.AddLast(i);

}

do

{

Console.WriteLine();

foreach (int i in list)

{

Console.WriteLine(i);

}

Console.WriteLine("\n\n1)додати елемент\n2)видалити елемент списку\n3)видалити список");

do

{

n0 = true;

if (int.TryParse(Console.ReadLine(), out x))

{

n0 = false;

}

else

{

Console.WriteLine("Введіть ще раз");

}

} while (n0);

switch (x)

{

case 1:

Console.WriteLine("Введіть елемент після якого додати новий");

int k;

do

{

n0 = true;

if (int.TryParse(Console.ReadLine(), out k))

{

foreach (int i in list)

{

if (k == i)

{

n0 = false;

break;

}

}

}

else

{

Console.WriteLine("Введіть ще раз");

}

} while (n0);

Console.WriteLine("Введіть новий елемент");

int g;

do

{

n0 = true;

if (int.TryParse(Console.ReadLine(), out g))

{

n0 = false;

}

else

{

Console.WriteLine("Введіть ще раз");

}

} while (n0);

LinkedListNode<int> lln = list.Find(k);

list.AddAfter(lln, g);

break;

case 2:

Console.WriteLine("Введіть елемент який хочете видалити:");

int l;

do

{

n0 = true;

if (int.TryParse(Console.ReadLine(), out l))

{

n0 = false;

}

else

{

Console.WriteLine("Введіть ще раз");

}

} while (n0);

LinkedListNode<int> lll = list.Find(l);

list.Remove(lll);

break;

case 3:

list.Clear();

list.AddLast(0);

break;

default:

x = 0;

break;

}

} while (x != 0);

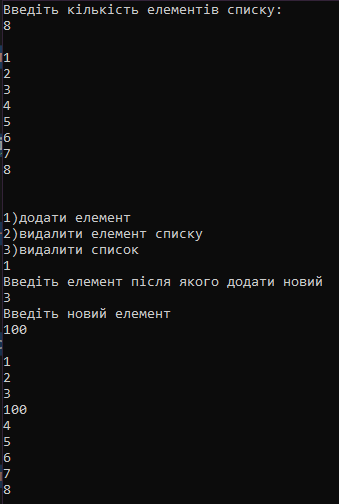


Рисунок 4.4 – Результат виконання завдання 2

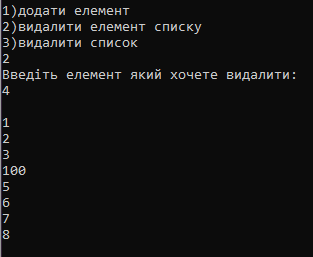


Рисунок 4.5 – Результат виконання завдання 2

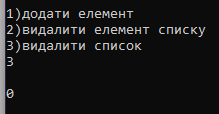
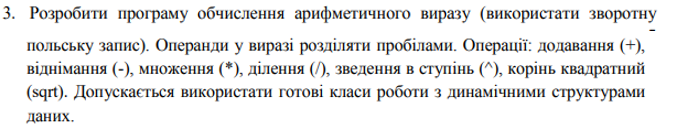


Рисунок 4.6 – Результат виконання завдання 2

**Завдання 3**:



Лістинг:

class Program

{

static int Priority(string symbol)

{

switch (symbol)

{

case ")": return 0;

case "(": return 1;

case "-": return 2;

case "+": return 2;

case "/": return 3;

case "\*": return 3;

case "^": return 4;

case "sqrt": return 5;

default: return 0;

}

}

static List<string> Station(string expression)

{

Stack<string> stack = new();

List<string> result = new();

string[] symbolsArr = expression.Split(" ");

for (int i = 0; i < symbolsArr.Length; i++)

{

if (float.TryParse(symbolsArr[i], out float num))

{

result.Add(symbolsArr[i]);

}

else if (symbolsArr[i] == "(")

{

stack.Push(symbolsArr[i]);

}

else if (symbolsArr[i] == ")")

{

string symbol = stack.Pop();

while (symbol != "(")

{

result.Add(symbol);

symbol = stack.Pop();

}

}

else

{

if (stack.Count > 0)

{

if (Priority(symbolsArr[i]) <= Priority(stack.Peek()))

{

result.Add(stack.Pop());

}

}

stack.Push(symbolsArr[i]);

}

Console.Write("Рядок: ");

for (int j = 0; j < result.Count; j++)

{

Console.Write($"{result[j]} ");

}

Console.Write("\nСтек: ");

foreach (Object obj in stack)

{

Console.Write($"{obj} ");

}

Console.Write("\n\n");

}

while (stack.Count > 0)

{

result.Add(stack.Pop());

}

return result;

}

static void Main()

{

Console.OutputEncoding = System.Text.Encoding.Default;

Stack<float> stack = new();

Console.Write("Введіть вираз: ");

string str = Console.ReadLine();

List<string> expressionList = Station(str);

float num1, num2;

Console.Write($"Зворотній польський запис: ");

for (int i = 0; i < expressionList.Count; i++)

{

Console.Write($"{expressionList[i]} ");

}

Console.Write("\n");

for (int i = 0; i < expressionList.Count; i++)

{

if (float.TryParse(expressionList[i], out float num))

{

stack.Push(num);

}

else

{

switch (expressionList[i])

{

case "+":

{

num1 = stack.Pop();

num2 = stack.Pop();

stack.Push(num2 + num1);

break;

}

case "-":

{

num1 = stack.Pop();

num2 = stack.Pop();

stack.Push(num2 - num1);

break;

}

case "\*":

{

num1 = stack.Pop();

num2 = stack.Pop();

stack.Push(num2 \* num1);

break;

}

case "/":

{

num1 = stack.Pop();

num2 = stack.Pop();

stack.Push(num2 / num1);

break;

}

case "^":

{

num1 = stack.Pop();

num2 = stack.Pop();

stack.Push((float)Math.Pow(num2, num1));

break;

}

case "sqrt":

{

stack.Push((float)Math.Sqrt(stack.Pop()));

break;

}

default: break;

}

}

}

Console.WriteLine($"\nОбчислення {str} = {stack.Peek()}\n");

}

}

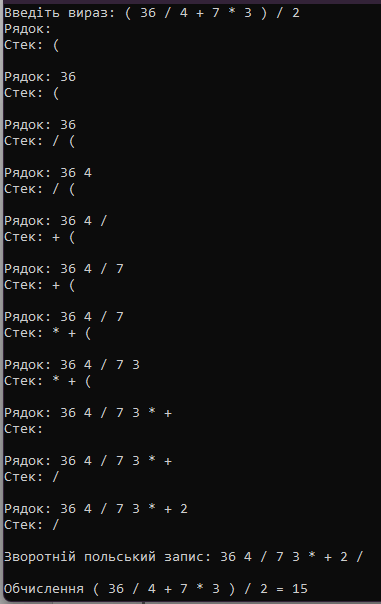


Рисунок 4.7 – Результат виконання завдання 3

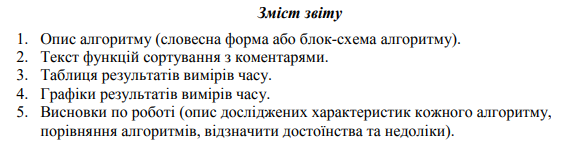
***Висновки:*** я ознайомитися з основами роботи з двозв’язним списком, однозв’язним списком, стеком та чергою. Розробив основні функції для обчислення арифметичного виразу, записаного з використанням зворотного польського запису.

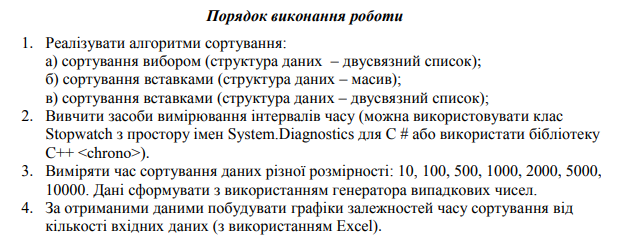
# **Лабораторна робота № 5**

ПРОСТІ МЕТОДИ СОРТУВАННЯ

***Мета*** : реалізація простих алгоритмів сортування та дослідження їх характеристик (швидкодія, необхідний обсяг пам'яті, застосування тощо).

**5.1 Хід роботи**

****

****

5.1.1

**Завдання**:

Лістинг:

using System.Diagnostics;

namespace lab\_5

{

class Program

{

static void Main()

{

Console.OutputEncoding = System.Text.Encoding.Unicode;

Console.InputEncoding = System.Text.Encoding.Unicode;

Stopwatch stopwatch = new Stopwatch();

Random random = new Random();

LinkedList<int> list = new LinkedList<int>();

int[] mass\_1 = new int[10];

int[] mass\_2 = new int[100];

int[] mass\_3 = new int[500];

int[] mass\_4 = new int[1000];

int[] mass\_5 = new int[2000];

int[] mass\_6 = new int[5000];

int[] mass\_7 = new int[10000];

Console.WriteLine("a) сортування вибором (структура даних – двусвязний список);");

Console.WriteLine("б) сортування вставками (структура даних – масив);");

Console.WriteLine("в) сортування вставками (структура даних – двусвязний список); ");

int x = 10, num;

Console.WriteLine($"\nРозмірність {x}:");

for (int i = 0; i < x; i++)

{

num = random.Next(-x, x);

list.AddLast(num);

mass\_1[i] = num;

}

stopwatch.Reset();

stopwatch.Start();

listSelectionSort(list, x);

stopwatch.Stop();

list.Clear();

for (int i = 0; i < x; i++)

{

list.AddLast(mass\_1[i]);

}

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

stopwatch.Start();

massInsertionSort(mass\_1);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

for (int i = 0; i < x; i++)

list.AddLast(random.Next(-x, x));

stopwatch.Reset();

stopwatch.Start();

listInsertionSort(list, x);

stopwatch.Stop();

list.Clear();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

////////////////////////////////////////////////////////////////////

x = 100;

Console.WriteLine($"\nРозмірність {x}:");

for (int i = 0; i < x; i++)

{

num = random.Next(-x, x);

list.AddLast(num);

mass\_2[i] = num;

}

stopwatch.Reset();

stopwatch.Start();

listSelectionSort(list, x);

stopwatch.Stop();

list.Clear();

for (int i = 0; i < x; i++)

{

list.AddLast(mass\_2[i]);

}

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

stopwatch.Start();

massInsertionSort(mass\_2);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

for (int i = 0; i < x; i++)

list.AddLast(random.Next(-x, x));

stopwatch.Reset();

stopwatch.Start();

listInsertionSort(list, x);

stopwatch.Stop();

list.Clear();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

////////////////////////////////////////////////////////////////////

x = 500;

Console.WriteLine($"\nРозмірність {x}:");

for (int i = 0; i < x; i++)

{

num = random.Next(-x, x);

list.AddLast(num);

mass\_3[i] = num;

}

stopwatch.Reset();

stopwatch.Start();

listSelectionSort(list, x);

stopwatch.Stop();

list.Clear();

for (int i = 0; i < x; i++)

{

list.AddLast(mass\_3[i]);

}

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

stopwatch.Start();

massInsertionSort(mass\_3);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

for (int i = 0; i < x; i++)

list.AddLast(random.Next(-x, x));

stopwatch.Reset();

stopwatch.Start();

listInsertionSort(list, x);

stopwatch.Stop();

list.Clear();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

////////////////////////////////////////////////////////////////////

x = 1000;

Console.WriteLine($"\nРозмірність {x}:");

for (int i = 0; i < x; i++)

{

num = random.Next(-x, x);

list.AddLast(num);

mass\_4[i] = num;

}

stopwatch.Reset();

stopwatch.Start();

listSelectionSort(list, x);

stopwatch.Stop();

list.Clear();

for (int i = 0; i < x; i++)

{

list.AddLast(mass\_4[i]);

}

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

stopwatch.Start();

massInsertionSort(mass\_4);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

for (int i = 0; i < x; i++)

list.AddLast(random.Next(-x, x));

stopwatch.Reset();

stopwatch.Start();

listInsertionSort(list, x);

stopwatch.Stop();

list.Clear();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

////////////////////////////////////////////////////////////////////

x = 2000;

Console.WriteLine($"\nРозмірність {x}:");

for (int i = 0; i < x; i++)

{

num = random.Next(-x, x);

list.AddLast(num);

mass\_5[i] = num;

}

stopwatch.Reset();

stopwatch.Start();

listSelectionSort(list, x);

stopwatch.Stop();

list.Clear();

for (int i = 0; i < x; i++)

{

list.AddLast(mass\_5[i]);

}

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

stopwatch.Start();

massInsertionSort(mass\_5);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

for (int i = 0; i < x; i++)

list.AddLast(random.Next(-x, x));

stopwatch.Reset();

stopwatch.Start();

listInsertionSort(list, x);

stopwatch.Stop();

list.Clear();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

////////////////////////////////////////////////////////////////////

x = 5000;

Console.WriteLine($"\nРозмірність {x}:");

for (int i = 0; i < x; i++)

{

num = random.Next(-x, x);

list.AddLast(num);

mass\_6[i] = num;

}

stopwatch.Reset();

stopwatch.Start();

listSelectionSort(list, x);

stopwatch.Stop();

list.Clear();

for (int i = 0; i < x; i++)

{

list.AddLast(mass\_6[i]);

}

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

stopwatch.Start();

massInsertionSort(mass\_6);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

for (int i = 0; i < x; i++)

list.AddLast(random.Next(-x, x));

stopwatch.Reset();

stopwatch.Start();

listInsertionSort(list, x);

stopwatch.Stop();

list.Clear();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

////////////////////////////////////////////////////////////////////

x = 10000;

Console.WriteLine($"\nРозмірність {x}:");

for (int i = 0; i < x; i++)

{

num = random.Next(-x, x);

list.AddLast(num);

mass\_7[i] = num;

}

stopwatch.Reset();

stopwatch.Start();

listSelectionSort(list, x);

stopwatch.Stop();

list.Clear();

for (int i = 0; i < x; i++)

{

list.AddLast(mass\_7[i]);

}

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

stopwatch.Start();

massInsertionSort(mass\_7);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

for (int i = 0; i < x; i++)

list.AddLast(random.Next(-x, x));

stopwatch.Reset();

stopwatch.Start();

listInsertionSort(list, x);

stopwatch.Stop();

list.Clear();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

//foreach (var person in list) Console.WriteLine(person);

}

static void listInsertionSort(LinkedList<int> list, int x)

{

int count = 1;

int k;

int j;

int temp;

var currentNode = list.First;

for (int i = 1; i < x; i++)//початок сортування вставок для масиву

{

currentNode = list.First;

for(count = 0; count < i; count++)//повернення до теперішнього елемента

currentNode = currentNode.Next;

k = currentNode.Value;//теперішній елемент

j = i;//індекс теперішнього елемента

while (j > 0 && currentNode.Previous.Value > k)//заміна

{

temp = currentNode.Value;

currentNode.Value = currentNode.Previous.Value;

currentNode.Previous.Value = temp;

j--;

currentNode = currentNode.Previous;

}

currentNode.Value = k;

}

}

static void massInsertionSort(int[] mass)

{

int k;

int j;

int temp;

for (int i = 1; i < mass.Length; i++)//початок сортування вставок для масиву

{

k = mass[i];//теперішній елемент

j = i;//індекс теперішнього елемента

while (j > 0 && mass[j - 1] > k)//заміна

{

temp = mass[j];

mass[j] = mass[j-1];

mass[j-1] = temp;

j--;

}

mass[j] = k;

}

}

static void listSelectionSort(LinkedList<int> list, int x, int currentIndex = 0, int count\_1 = 50000, int index = 0, int i = -1, int count\_2 = 50000)

{

for (currentIndex = 0; currentIndex < x; currentIndex++)//початок сортування вибором для списків

{

count\_2 = 50000;

i = -1;

index = 0;

foreach (var person in list)//пошук теперішнього та найменшого елемента

{

i++;//індекс теперішнього елемента

if (person < count\_2 && i > currentIndex)//найменший елемент

{

count\_2 = person;//найменший знайдений елемент

index = i;//індекс найменшого елемента

}

if (i == currentIndex)//теперішній елемент

count\_1 = person;

}

if (index != currentIndex && count\_2 <= count\_1)//заміна

{

i = -1;

var currentNode = list.First;

while (currentNode != null)//пошук цих елементів

{

i++;

if (i == currentIndex)//заміна

currentNode.Value = count\_2;

if (i == index)//заміна

currentNode.Value = count\_1;

currentNode = currentNode.Next;

}

}

}

}

}

}

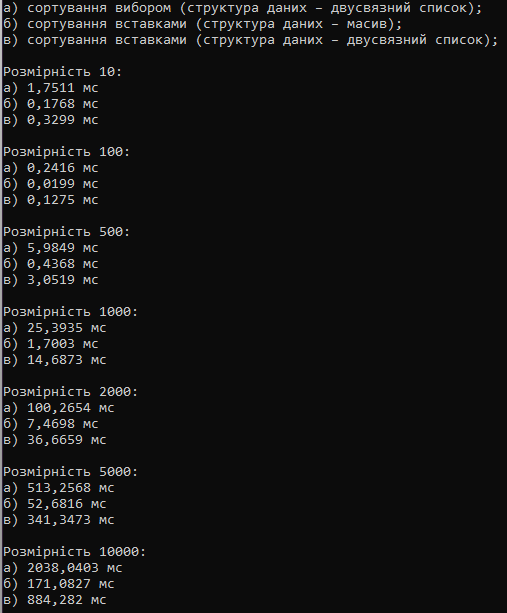


Рисунок 5.1 – Результат виконання завдання

***Словесний опис алгоритмів:***

**Сортування вибором**  – алгоритм сортування масиву, який по швидкості виконання можна зрівняти з сортуванням бульбашкою.

Алгоритм сортування вибором складається з наступних кроків:

* Для початку визначаємо позицію мінімального елементу масиву;
* Здійснюємо обмін мінімального елементу з елементом на початку масиву. Виходить, що перший елемент масиву вже відсортовано;
* Зменшуємо робочу область масиву, відкидаючи перший елемент, а для підмасиву, що залишився, повторюємо сортування.

Алгоритмічна складність: O(n) = n^2.

**Сортування вставками** - це алгоритм сортування, в якому всі елементи масиву почергово переглядаються, при цьому кожен елемент переміщається у відповідне місце серед раніше впорядкованих значень.

Алгоритм роботи сортування включенням наступний:

* На початку роботи впорядкована частина порожня;
* Додаємо в неї перший елемент масиву з не впорядкованих даних;
* Переходимо до наступного елементу в невідсортованих даних, і знаходимо йому правильну позицію у відсортованій частині масиву, цим ми розширюємо область впорядкованих даних;
* Повторюємо попередній крок для всіх елементів, що залишилися.

Алгоритмічна складність: O(n) = n^2.

***Графіки та таблиця:***

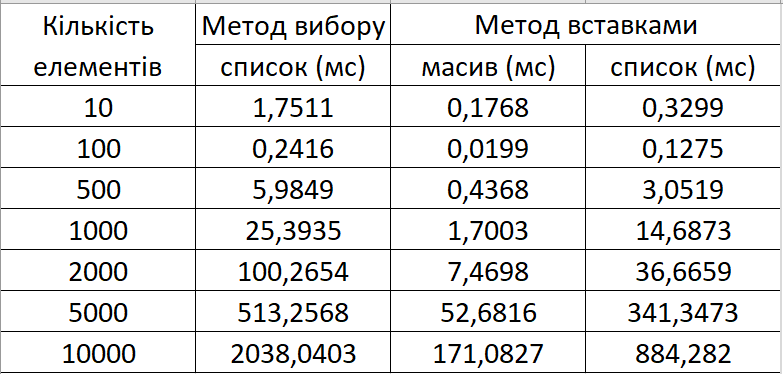


Рисунок 5.2 – Таблиця

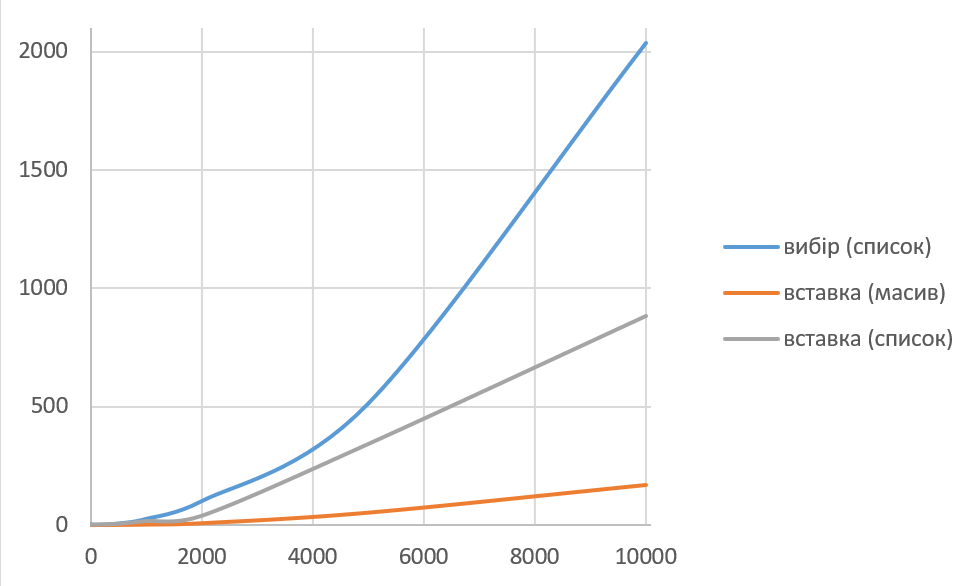


Рисунок 5.3 – Графіки

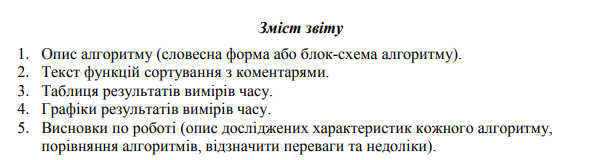
***Висновки:*** я дослідив та порівняв характеристики кожного з алгоритмів, та дійшов висновку, що сортування вибором найповільніше сортування, а вставка, навпаки виграє як для масиву, так і для списку, на мою думку, це відбувається так, тому що вставка, може бути частково відсортована, це значно допомагає в сортуванні, а під час вибірки, сортування не залежить від розміщення елемента, на малій кількості елементів, різниця невелика, але зі збільшенням їх кількості, збільшується і різниця, яка продемонстрована на графіку.

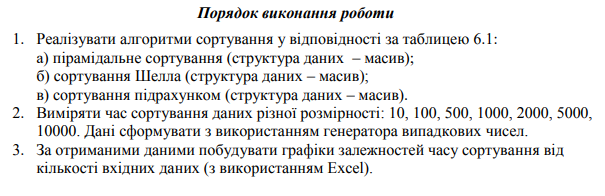
# **Лабораторна робота № 6**

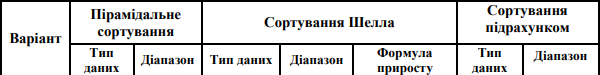
ШВИДКІ МЕТОДИ СОРТУВАННЯ

***Мета*** : реалізація швидких алгоритмів сортування та дослідження їх характеристик (швидкодія, необхідний обсяг пам'яті, застосування тощо).

**5.1 Хід роботи**

****

****

****

****

6.1.1

**Завдання**:

Лістинг:

using System.Diagnostics;

namespace lab\_5

{

class Program

{

static void Main()

{

Console.OutputEncoding = System.Text.Encoding.Unicode;

Console.InputEncoding = System.Text.Encoding.Unicode;

Stopwatch stopwatch = new Stopwatch();

Random random = new Random();

Console.WriteLine("a) пірамідальне сортування (структура даних – масив);");

Console.WriteLine("б) сортування Шелла(структура даних – масив);");

Console.WriteLine("в) сортування підрахунком(структура даних – масив);\n");

double[] piramida\_10 = new double[10];

double[] piramida\_100 = new double[100];

double[] piramida\_500 = new double[500];

double[] piramida\_1000 = new double[1000];

double[] piramida\_2000 = new double[2000];

double[] piramida\_5000 = new double[5000];

double[] piramida\_10000 = new double[10000];

float[] szela\_10 = new float[10];

float[] szela\_100 = new float[100];

float[] szela\_500 = new float[500];

float[] szela\_1000 = new float[1000];

float[] szela\_2000 = new float[2000];

float[] szela\_5000 = new float[5000];

float[] szela\_10000 = new float[10000];

char[] liczba\_10 = new char[10];

char[] liczba\_100 = new char[100];

char[] liczba\_500 = new char[500];

char[] liczba\_1000 = new char[1000];

char[] liczba\_2000 = new char[2000];

char[] liczba\_5000 = new char[5000];

char[] liczba\_10000 = new char[10000];

HeapSort heapSort = new HeapSort();

ShellSort shellSort = new ShellSort();

CountSort countSort = new CountSort();

/////////////////////////////////////////////////////////////////////////////////////////////////////////

int count = 10;

Console.WriteLine($"Розмірність: {count}");

for (int i = 0; i < count; i++)

{

piramida\_10[i] = random.NextDouble() \* (100 - (-10)) - 10;

}

stopwatch.Start();

heapSort.sort(piramida\_10);

stopwatch.Stop();

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

szela\_10[i] = (float)random.NextDouble() \* (200 - 0) - 0;

}

stopwatch.Start();

shellSort.Sort(szela\_10);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

liczba\_10[i] = (char)random.Next(-100, -10);

}

stopwatch.Start();

countSort.Sort(liczba\_10);

stopwatch.Stop();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

///////////////////////////////////////////////////////////////////////////////////////////////////////////

count = 100;

Console.WriteLine($"\nРозмірність: {count}");

for (int i = 0; i < count; i++)

{

piramida\_100[i] = random.NextDouble() \* (100 - (-10)) - 10;

}

stopwatch.Start();

heapSort.sort(piramida\_100);

stopwatch.Stop();

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

szela\_100[i] = (float)random.NextDouble() \* (200 - 0) - 0;

}

stopwatch.Start();

shellSort.Sort(szela\_100);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

liczba\_100[i] = (char)random.Next(-100, -10);

}

stopwatch.Start();

countSort.Sort(liczba\_100);

stopwatch.Stop();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

///////////////////////////////////////////////////////////////////////////////////////////////////////////

count = 500;

Console.WriteLine($"\nРозмірність: {count}");

for (int i = 0; i < count; i++)

{

piramida\_500[i] = random.NextDouble() \* (100 - (-10)) - 10;

}

stopwatch.Start();

heapSort.sort(piramida\_500);

stopwatch.Stop();

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

szela\_500[i] = (float)random.NextDouble() \* (200 - 0) - 0;

}

stopwatch.Start();

shellSort.Sort(szela\_500);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

liczba\_500[i] = (char)random.Next(-100, -10);

}

stopwatch.Start();

countSort.Sort(liczba\_500);

stopwatch.Stop();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

///////////////////////////////////////////////////////////////////////////////////////////////////////////

count = 1000;

Console.WriteLine($"\nРозмірність: {count}");

for (int i = 0; i < count; i++)

{

piramida\_1000[i] = random.NextDouble() \* (100 - (-10)) - 10;

}

stopwatch.Start();

heapSort.sort(piramida\_1000);

stopwatch.Stop();

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

szela\_1000[i] = (float)random.NextDouble() \* (200 - 0) - 0;

}

stopwatch.Start();

shellSort.Sort(szela\_1000);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

liczba\_1000[i] = (char)random.Next(-100, -10);

}

stopwatch.Start();

countSort.Sort(liczba\_1000);

stopwatch.Stop();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

///////////////////////////////////////////////////////////////////////////////////////////////////////////

count = 2000;

Console.WriteLine($"\nРозмірність: {count}");

for (int i = 0; i < count; i++)

{

piramida\_2000[i] = random.NextDouble() \* (100 - (-10)) - 10;

}

stopwatch.Start();

heapSort.sort(piramida\_2000);

stopwatch.Stop();

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

szela\_2000[i] = (float)random.NextDouble() \* (200 - 0) - 0;

}

stopwatch.Start();

shellSort.Sort(szela\_2000);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

liczba\_2000[i] = (char)random.Next(-100, -10);

}

stopwatch.Start();

countSort.Sort(liczba\_2000);

stopwatch.Stop();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

///////////////////////////////////////////////////////////////////////////////////////////////////////////

count = 5000;

Console.WriteLine($"\nРозмірність: {count}");

for (int i = 0; i < count; i++)

{

piramida\_5000[i] = random.NextDouble() \* (100 - (-10)) - 10;

}

stopwatch.Start();

heapSort.sort(piramida\_5000);

stopwatch.Stop();

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

szela\_5000[i] = (float)random.NextDouble() \* (200 - 0) - 0;

}

stopwatch.Start();

shellSort.Sort(szela\_5000);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

liczba\_5000[i] = (char)random.Next(-100, -10);

}

stopwatch.Start();

countSort.Sort(liczba\_5000);

stopwatch.Stop();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

///////////////////////////////////////////////////////////////////////////////////////////////////////////

count = 10000;

Console.WriteLine($"\nРозмірність: {count}");

for (int i = 0; i < count; i++)

{

piramida\_10000[i] = random.NextDouble() \* (100 - (-10)) - 10;

}

stopwatch.Start();

heapSort.sort(piramida\_10000);

stopwatch.Stop();

Console.WriteLine($"а) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

szela\_10000[i] = (float)random.NextDouble() \* (200 - 0) - 0;

}

stopwatch.Start();

shellSort.Sort(szela\_10000);

stopwatch.Stop();

Console.WriteLine($"б) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

for (int i = 0; i < count; i++)

{

liczba\_10000[i] = (char)random.Next(-100, -10);

}

stopwatch.Start();

countSort.Sort(liczba\_10000);

stopwatch.Stop();

Console.WriteLine($"в) {stopwatch.Elapsed.TotalMilliseconds} мс");

stopwatch.Reset();

}

}

public class HeapSort

{

public void sort(double[] arr)

{

int n = arr.Length;

//перегрупування масиву

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

for (int i = n - 1; i >= 0; i--)

{

var temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);//найбільше в кінець

}

}

void heapify(double[] arr, int n, int i)

{

int largest = i;

int l = 2 \* i + 1;//ліва частина

int r = 2 \* i + 2;//права частина

if (l < n && arr[l] > arr[largest])//якщо лівий елемент більше

largest = l;

if (r < n && arr[r] > arr[largest])//якщо правий елемент більше

largest = r;

if (largest != i)//якщо неайбільше не корінь

{

var swap = arr[i];

arr[i] = arr[largest];

arr[largest] = swap;

heapify(arr, n, largest);

}

}

}

public class ShellSort

{

static void Swap(ref float a, ref float b)//заміна

{

var t = a;

a = b;

b = t;

}

public void Sort(float[] array)

{

var d = array.Length / 2;//відстань між елементами, які порівнюються

while (d >= 1)

{

for (var i = d; i < array.Length; i++)

{

var j = i;

while ((j >= d) && (array[j - d] > array[j]))//пошук більшого

{

Swap(ref array[j], ref array[j - d]);

j = j - d;

}

}

d = d / 2;//зменшення відстані

}

}

}

public class CountSort

{

public void Sort(char[] inputArray)

{

int[] countArray = new int[inputArray.Max() + 1];//мазив з кінцевим індексом максимального елемента

for (int i = 0; i < inputArray.Length; i++)//елемент стає на індекс, відповідно свого числа

{

countArray[inputArray[i]]++;

}

int sortedArrayIndex = 0;//індекс відсортованого елемента

for (int i = countArray.Length - 1; i >= 0; i--)//сортування

{

for (int j = 0; j < countArray[i]; j++)

{

inputArray[sortedArrayIndex++] = (char)i;

}

}

}

}

}

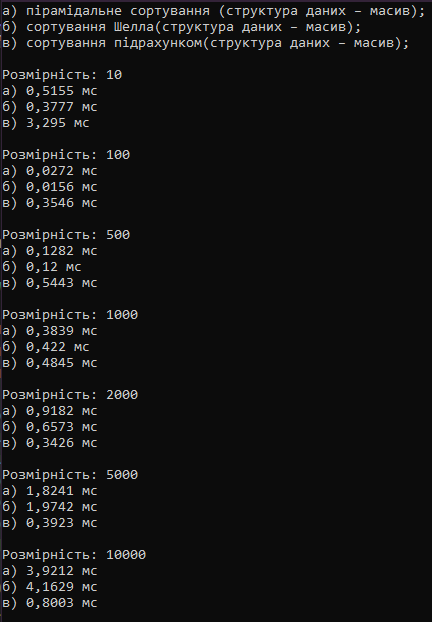


Рисунок 6.1 – Результат виконання завдання

***Словесний опис алгоритмів:***

**Пірамідальне сортування** - алгоритм сортування, який використовує бінарну кучу. Куча (англ. Heap) – структура даних типу дерево, яка задовольняє наступній властивості: для будь-якого заданого вузла В, який є нащадком вузла А виконується умова: ключ (A) ≥ ключ (B). Таким чином, кореневий вузол кучи буде зберігати найбільше значення, тому іноді таку кучу називають max-кучою. Якщо змінити порівняння на протилежне, то кореневий вузол буде зберігати найменше значення і таку кучу називають min-кучою. Поширеною реалізацією кучи є бінарна куча (англ. Binary heap), в якій дерево є двійковим деревом (рисунок 6.1 та рисунок 6.2). Двійкова куча задовольняє трьом умовам:

• Значення в будь-якій вершині не менш, ніж значення її нащадків (max-куча);

• Глибина всього листя відрізняється не більше ніж на 1 шар;

• Останній шар заповнюється зліва направо. Висота кореневого вузла (дерева): int(log2 N).

**Сортування Шелла** є модифікацією алгоритму сортування вставками та класифікується як сортування вставками з убиваючим кроком. Ефективність алгоритму полягає в тому, що на кожному з проміжних кроків сортується або невелике число елементів, або вже досить добре впорядковані набори елементів. Впорядкованість масиву зростає після кожного проходу. В ході вивчення алгоритму досліджувалася залежність середнього числа перестановок від розміру масивів при N від 100 до 60000 для декількох типів послідовностей кроків, для яких були отримані відповідні залежності часу роботи від розміру масиву.

**Сортування підрахунком** - алгоритм впорядкування, що застосовується при малій кількості різних елементів (ключів) у масиві даних. Час його роботи лінійно залежить як від загальної кількості елементів у масиві так і від кількості різних елементів.

***Графіки та таблиця:***

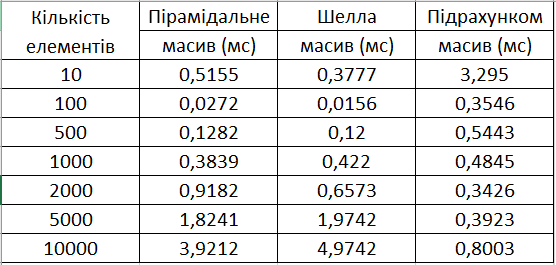


Рисунок 6.2 – Таблиця

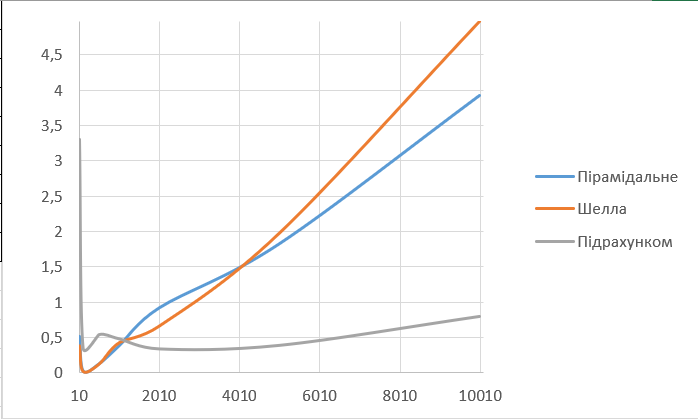


Рисунок 6.3 – Графіки

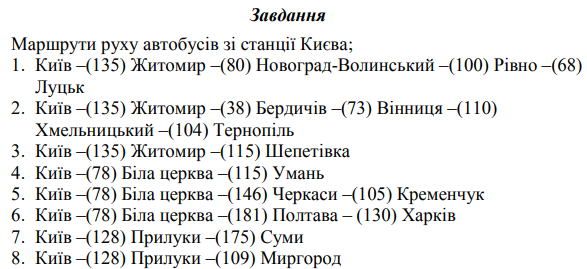
***Висновки:*** я дослідив та порівняв характеристики кожного з алгоритмів, та дійшов висновку, що на різних проміжках, швидкість сортування різна, наприклад, Шелла найшвидше від 10 до 500, пірамідальне на 1000, підрахунком від 2000 до 10000. Перш за все, ці результати не точні, окрім підрахунку, так як в різних масивах були різні числа. Отже необхідно проаналізувати кожен алгоритм. Пірамідальне сортування залежить від кількості першин під коренем, та чи відсортованні вже елементи, якщо так, то складність алгоритму збільшується до квадрату.Сортування Шелла навпаки більш ефективне для більш впорядкованих елементів. В свою чергу метод підрахунку не подібний на інші, так як він повністю залежить від кількості елементів, а не від їх значення.

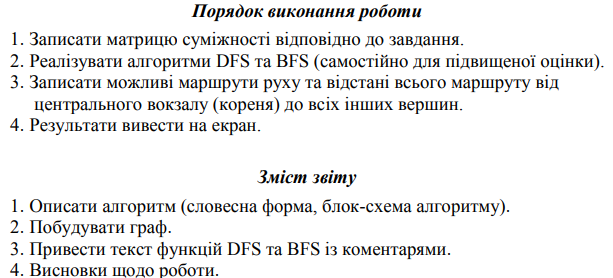
# **Лабораторна робота № 7-8**

ГРАФИ. ДЕРЕВА. АЛГОРИТМИ ПОШУКУ В ГЛИБИНУ ТА В ШИРИНУ

***Мета***: Освоїти та закріпити прийоми роботи з даними різного типу, організованими у вигляді дерев та їх окремого випадку – бінарних дерев. Здобути практичні навички роботи з графами.

**8.1 Хід роботи**

****

****

8.1.1

**Завдання**:

Лістинг:

namespace lab\_7\_8

{

class Program

{

static void Main()

{

Console.OutputEncoding = System.Text.Encoding.Unicode;

Console.InputEncoding = System.Text.Encoding.Unicode;

int v = 19;

int[] dfs = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18 };

string[] city = { "Київ", "Житомир", "Новоград-Волинський", "Рівне", "Луцьк",

"Бердичів","Вінниця","Хмельницький", "Тернопіль",

"Шепетівка",

"Біла церква", "Умань",

"Черкаси", "Кременчук",

"Полтава", "Харків",

"Прилуки", "Суми",

"Миргород"};

int[] km = { 135, 80, 100, 68, 38, 73, 110, 104, 115, 78, 115, 146, 105, 181, 130, 128, 175, 109 };

int[,] edges = new int[v, v];

bool[] visited = new bool[v];

edges[0, 1] = 1;

edges[1, 2] = 1;

edges[2, 3] = 1;

edges[3, 4] = 1;

edges[1, 5] = 1;

edges[5, 6] = 1;

edges[6, 7] = 1;

edges[7, 8] = 1;

edges[1, 9] = 1;

edges[0, 10] = 1;

edges[10, 11] = 1;

edges[10, 12] = 1;

edges[12, 13] = 1;

edges[10, 14] = 1;

edges[14, 15] = 1;

edges[0, 16] = 1;

edges[16, 17] = 1;

edges[16, 18] = 1;

Console.WriteLine("DFS");

for (int i = 0; i < v; i++)

{

if (!visited[i])

DFS(edges, v, visited, i, city);

}

for (int i = 0; i < v; i++)

{

visited[i] = false;

}

Console.WriteLine();

Console.WriteLine("BFS");

for (int i = 0; i < v; i++)

{

if (!visited[i])

BFS(edges, v, visited, i, dfs[i], city);

}

Console.WriteLine($"\nКиїв - Житомир:{km[0]}");

Console.WriteLine($"Київ - Житомир - Новоград-Волинський:{km[0] + km[1]}");

Console.WriteLine($"Київ - Житомир - Новоград-Волинський - Рівне:{km[0] + km[1] + km[2]}");

Console.WriteLine($"Київ - Житомир - Новоград-Волинський - Рівне - Луцьк:{km[0] + km[1] + km[2] + km[3]}");

Console.WriteLine($"Київ - Житомир - Бердичів:{km[0] + km[4]}");

Console.WriteLine($"Київ - Житомир - Бердичів - Вінниця:{km[0] + km[4] + km[5]}");

Console.WriteLine($"Київ - Житомир - Бердичів - Вінниця - Хмельницький:{km[0] + km[4] + km[5] + km[6]}");

Console.WriteLine($"Київ - Житомир - Бердичів - Вінниця - Хмельницький - Тернопіль:{km[0] + km[4] + km[5] + km[6] + km[7]}");

Console.WriteLine($"Київ - Житомир - Шепетівка: {km[0] + km[8]}");

Console.WriteLine($"Київ - Біла церква - Умань: {km[9] + km[10]}");

Console.WriteLine($"Київ - Біла церква - Черкаси - Кременчук: {km[9] + km[11] + km[12]}");

Console.WriteLine($"Київ - Біла церква - Полтава - Харків: {km[9] + km[13] + km[14]}");

Console.WriteLine($"Київ - Прилуки - Суми: {km[15] + km[16]}");

Console.WriteLine($"Київ - Прилуки - Миргород: {km[15] + km[17]}");

}

static void DFS(int[,] edges, int v, bool[] visited, int si, string[] city)

{

visited[si] = true;//відвідане

Console.Write($"{city[si]} ");

for (int i = 0; i < v; i++)

{

if (i == si)

continue;

if (!visited[i] && edges[si, i] == 1)

{

DFS(edges, v, visited, i, city);//пошук зв'язаниг графів

}

}

}

static void BFS(int[,] edges, int v, bool[] visited, int j, int si, string[] city)

{

Queue<int> queue = new Queue<int>();//створення черги (перший зайшов, перший вийшов)

queue.Enqueue(si);//запис в кінець черги

visited[si] = true;

while (queue.Count != 0)

{

int currentVertex = queue.Dequeue();

Console.Write($"{city[currentVertex]} ");

for (int i = 0; i < v; i++)//пошук графів на цьому рівні

{

if (i == j)

continue;

if (!visited[i] && edges[currentVertex, i] == 1)

{

queue.Enqueue(i);

visited[i] = true;

}

}

}

}

}

}

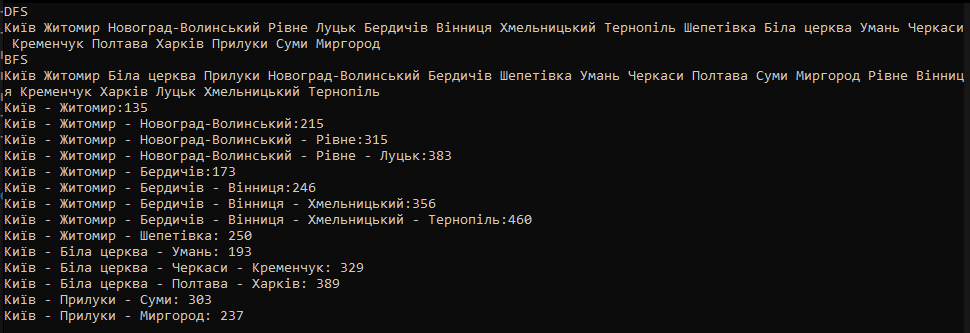


Рисунок 8.1 – Результат виконання завдання

***Словесний опис алгоритмів:***

**Алгоритм пошуку в глибину** (DFS) — алгортм для обходу дерева, структури подібної до дерева, або графу. Робота алгоритму починається з кореня дерева (або іншої обраної вершини в графі) і здійснюється обхід в максимально можливу глибину до переходу на наступну вершину.

Наведемо кроки алгоритму

1. Почати з довільної вершини v. Виконати DFS(v):=1. Включити цю вершину в стек.
2. Розглянути вершину у верхівці стеку: нехай це вершина х. Якщо всі ребра, інцидентні вершині х, позначено, то перейти до кроку 4, інакше — до кроку 3.
3. Нехай {*x*, *y*} — непозначене ребро. Якщо DFS(*у*) уже визначено, то позначити ребро {*x*, *y*} штриховою лінією та перейти до кроку 2.  Якщо DFS(*у*) не визначено, то позначити ребро {*x*, *y*} потовщеною суцільною лінією, визначити DFS(*у*) як черговий DFS-номер, включити цю вершину в стек і перейти до кроку 2.
4. Виключити вершину х зі стеку. Якщо стек порожній, то зупинитись, інакше — перейти до кроку 2.

Обчислювальна складність: O(n+m)

**Пошук у ширину** — алгоритм пошуку на графі.Алгоритм має назву пошуку в ширину, оскільки «фронт» пошуку (між пройденими та непройденими вершинами) одноманітно розширюється вздовж всієї своєї ширини. Тобто, алгоритм проходить всі вершини на відстані *k* перед тим як пройти вершини на відстані *k*+1.

Наведемо кроки алгоритму

1. Почати з довільної вершини v. Виконати BFS(v):=1. Включити вершину v у чергу.
2. Розглянути вершину, яка перебуває на початку черги; нехай це буде вершина х. Якщо для всіх вершин, суміжних із вершиною х, уже визначено BFS-номери, то перейти до кроку 4, інакше - до кроку 3.
3. Нехай {x,y} - ребро, у якому номер BFS(у) не визначено. Позначити це ребро потовщеною суцільною лінією, визначити BFS(у) як черговий BFS-номер, включити вершину у чергу й перейти до кроку 2.
4. Виключити вершину х із черги. Якщо черга порожня, то зупинитись, інакше - перейти до кроку 2.

Обчислювальна складність: O(|V|+|E|)

***Граф та матриця суміжності:***

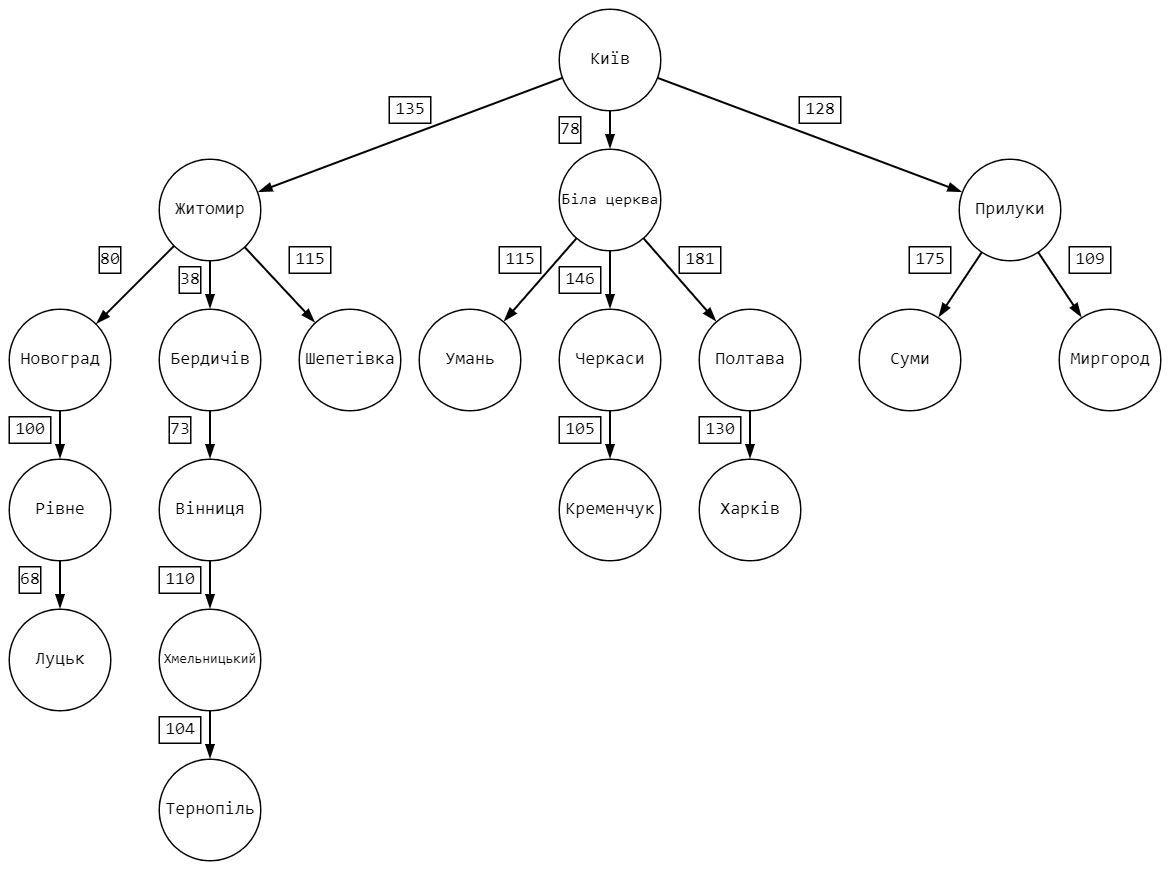


Рисунок 8.2 – Граф

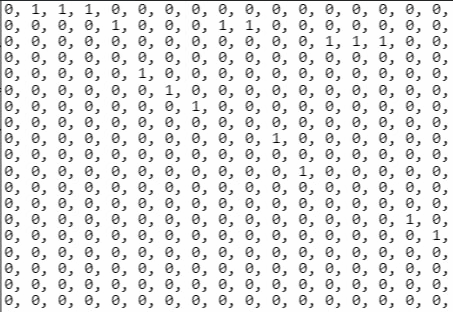


Рисунок 6.3 – Матриця суміжності

***Висновки:*** я освоїв та закріпив прийоми роботи з даними різного типу, організував у вигляді дерев та їх окремого випадку – бінарних дерев. Здобув практичні навички роботи з графами.