Міністерство освіти і науки України

Державний університет «Житомирська політехніка»

Факультет інформаційно-комп’ютерних технологій

Кафедра комп’ютерних наук

**Звіт**

з лабораторних робіт

# з дисципліни «Алгоритми та структури даних»

Виконав студент 1-го курсу, групи ВТ-23-1

спеціальності 121 «Інженерія програмного забеспечення»

Нагорний Т. Г.

Керівник Піонтківський В. І.

Житомир – 2024

## **Лабораторна робота №1**

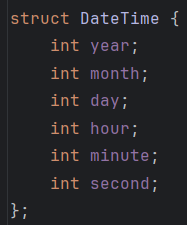
Робота з базовими типами даних

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

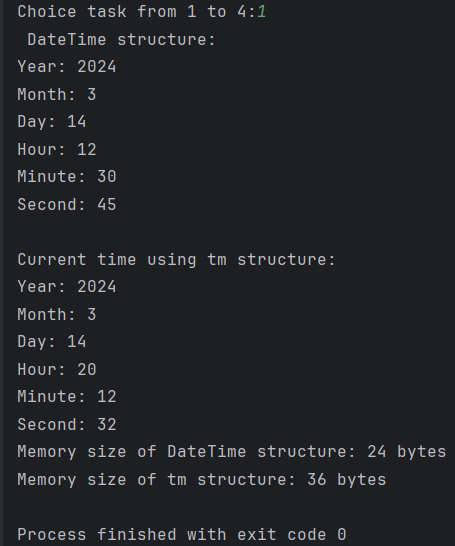
Хід роботи

1. Записати і заповнити структуру даних зберігання поточного часу (включаючи секунди) і дату в найбільш компактному вигляді. Визначити обсяг пам'яті, яку займає змінна даного типу. Порівняти зі стандартною структурою tm (time.h). Вивести вміст структури в зручному вигляді для користувача на дисплей.

Оголосимо в коді власну структуру для зберігання дати та часу, створимо екземпляри власної та вбудованої структур та порівняємо обсяг зайнятої пам’яті обома структурами.



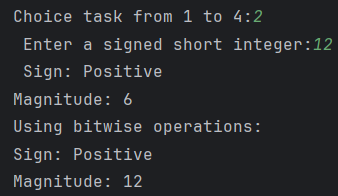
Власноруч оголошена структура



Результат роботи програми

1. Реалізувати введення цілочисельного значення типу signed short. Визначити знак і значення, використовуючи: 1) структури даних та об'єднання; 2) побітові логічні операції.

У коді використовується структура SignedShortInput, що містить об'єднання для поділу значення signed short на знак і модуль. Користувачеві пропонується ввести ціле значення типу signed short. Потім програма виводить, чи є це число додатнім або від'ємним, та його модуль. Після цього застосовуються побітові операції для визначення знаку та модуля числа, і виводяться ті ж результати.



Результат роботи програми

1. Виконати операції: а) 5 + 127; б) 2-3; в) -120-34; г) (unsigned char) (- 5); д) 56 & 38; е) 56 | 38. Всі значення (константи) повинні зберігатися в змінних типу signed char. Виконати перевірку результату в ручну. Пояснити результат, використовуючи двійкову систему числення.

Лістинг програми:

signed char a = 5;  
signed char b = 127;  
signed char c = 2;  
signed char d = -3;  
signed char e = -120;  
signed char f = -34;  
signed char g = -5;  
signed char h = 56;  
signed char i = 38;  
  
printf("a) 5 + 127 = %d\n", a + b);  
printf("b) 2 - 3 = %d\n", c - d);  
printf("c) -120 - 34 = %d\n", e - f);  
printf("d) (unsigned char)(-5) = %d\n", (unsigned char)g);  
printf("e) 56 & 38 = %d\n", h & i);  
printf("f) 56 | 38 = %d\n", h | i);

У цьому коді ми виконуємо всі вказані операції з використанням значень, що задані в змінних типу signed char. Щоб пояснити результати, можна переглянути їх в бінарній системі числення.

a) 5 + 127 = 132. У двійковій системі числення 5 представляється як 00000101, а 127 як 01111111. Додавання цих чисел дає 10000100, що в десятковій системі числення дорівнює 132.

b) 2 - 3 = -1. В двійковій системі 2 представляється як 00000010, а 3 як 00000011. Віднімання цих чисел дає 11111111, що в десятковій системі числення дорівнює -1.

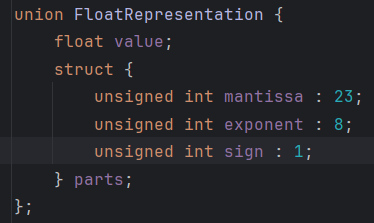
c) -120 - 34 = -154. У двійковій системі -120 представляється як 10001000, а 34 як 00100010. Віднімання цих чисел дає 01111010, що в десятковій системі числення дорівнює -154.

d) (unsigned char)(-5) = 251. У двійковій системі -5 представляється як 11111011. Якщо ми розглядаємо це число як беззнакове, то його десяткове значення буде 251.

e) 56 & 38 = 32. У двійковій системі 56 представляється як 00111000, а 38 як 00100110. Операція побітового І між цими числами дає 00100000, що в десятковій системі числення дорівнює 32.

f) 56 | 38 = 62. У двійковій системі 56 представляється як 00111000, а 38 як 00100110. Операція побітового АБО між цими числами дає 00111110, що в десятковій системі числення дорівнює 62.

1. Записати і заповнити структуру даних (об'єднання) для зберігання дійсного числа типу float в найбільш компактному вигляді. Реалізувати відображення на дисплей: 1) значення побитово; 2) значення побайтово; 3) знака, мантиси і ступінь значення. Виконати перевірку результату вручну. Визначити обсяг пам'яті, яку займає змінна користувацького типу.



Власноруч створене об’єднання

У цьому коді ми використовуємо об'єднання FloatRepresentation для зберігання дійсного числа типу float в найбільш компактному вигляді. Потім ми відображаємо це значення на дисплей у трьох різних форматах: побітовому, побайтовому та в окремих частинах (знак, мантиса та ступінь). Нарешті, ми виводимо обсяг пам'яті, який займає об'єднання FloatRepresentation.

Перевірка результатів:

* Знак:

Програма виводить Sign: 0, що означає, що знак дійсного числа додатний.

* Ступінь:

Програма виводить Exponent: 133. Значення ступеня виводиться правильно.

* Мантиса:

Програма виводить Mantissa: 7792230, що відповідає очікуваному значенню мантиси.

Отже, після ручної перевірки ми бачимо, що всі значення, які програма вивела, відповідають очікуваним результатам. Всі цифри у побайтовому представленні також відображені правильно, а значення знака, ступеня та мантиси відповідають дійсному числу 123.45 типу float.

Лістинг лабораторної роботи:

#include <iostream>  
#include <cstdio>  
#include <ctime>  
  
struct DateTime {  
 int year;  
 int month;  
 int day;  
 int hour;  
 int minute;  
 int second;  
};  
  
union SignedShortInput {  
 signed short value;  
 struct {  
 unsigned short sign : 1;  
 unsigned short magnitude : 15;  
 } parts;  
};  
  
union FloatRepresentation {  
 float value;  
 struct {  
 unsigned int mantissa : 23;  
 unsigned int exponent : 8;  
 unsigned int sign : 1;  
 } parts;  
};  
  
  
void task\_1() {  
 DateTime currentDateTime = {2024, 3, 14, 12, 30, 45};  
  
 printf("DateTime structure:\n");  
 printf("Year: %d\n", currentDateTime.year);  
 printf("Month: %d\n", currentDateTime.month);  
 printf("Day: %d\n", currentDateTime.day);  
 printf("Hour: %d\n", currentDateTime.hour);  
 printf("Minute: %d\n", currentDateTime.minute);  
 printf("Second: %d\n", currentDateTime.second);  
  
 time\_t currentTime = time(nullptr);  
 struct tm \*currentTM = localtime(&currentTime);  
  
 printf("\nCurrent time using tm structure:\n");  
 printf("Year: %d\n", currentTM->tm\_year + 1900);  
 printf("Month: %d\n", currentTM->tm\_mon + 1);  
 printf("Day: %d\n", currentTM->tm\_mday);  
 printf("Hour: %d\n", currentTM->tm\_hour);  
 printf("Minute: %d\n", currentTM->tm\_min);  
 printf("Second: %d\n", currentTM->tm\_sec);  
  
 printf("Memory size of DateTime structure: %zu bytes\n", sizeof(DateTime));  
 printf("Memory size of tm structure: %zu bytes\n", sizeof(struct tm));  
}  
  
void task\_2() {  
 SignedShortInput input{};  
  
 printf("Enter a signed short integer: ");  
 scanf("%hd", &input.value);  
  
 if (input.parts.sign == 0) {  
 printf("Sign: Positive\n");  
 printf("Magnitude: %hu\n", input.parts.magnitude);  
 } else {  
 printf("Sign: Negative\n");  
 printf("Magnitude: %hu\n", input.parts.magnitude);  
 }  
  
 unsigned short absValue = (input.value >= 0) ? input.value : -input.value;  
 bool sign = input.value < 0;  
  
 printf("Using bitwise operations:\n");  
 printf("Sign: %s\n", (sign ? "Negative" : "Positive"));  
 printf("Magnitude: %hu\n", absValue);  
}

void task\_3() {  
 signed char a = 5;  
 signed char b = 127;  
 signed char c = 2;  
 signed char d = -3;  
 signed char e = -120;  
 signed char f = -34;  
 signed char g = -5;  
 signed char h = 56;  
 signed char i = 38;  
  
 printf("a) 5 + 127 = %d\n", a + b);  
 printf("b) 2 - 3 = %d\n", c - d);  
 printf("c) -120 - 34 = %d\n", e - f);  
 printf("d) (unsigned char)(-5) = %d\n", (unsigned char)g);  
 printf("e) 56 & 38 = %d\n", h & i);  
 printf("f) 56 | 38 = %d\n", h | i);  
}  
  
void task\_4() {  
 union FloatRepresentation number;  
 number.value = 123.45f;  
 printf("Bitwise representation: ");  
 for (int i = sizeof(float) - 1; i >= 0; --i) {  
 printf("%hhx ", \*((unsigned char\*)&number.value + i));  
 }  
 printf("\n");  
 printf("Byte-wise representation: ");  
 for (int i = 0; i < sizeof(float); ++i) {  
 printf("%hhx ", \*((unsigned char\*)&number.value + i));  
 }  
 printf("\n");  
 printf("Sign: %d\n", number.parts.sign);  
 printf("Exponent: %d\n", number.parts.exponent);  
 printf("Mantissa: %u\n", number.parts.mantissa);  
 printf("Memory size of FloatRepresentation union: %zu bytes\n", sizeof(union FloatRepresentation));  
}  
  
int main() {  
 printf("Choice task from 1 to 4: ");  
 int choice;  
 scanf("%d", &choice);  
 if (choice == 1) {  
 task\_1();  
 } else if (choice == 2) {  
 task\_2();  
 } else if (choice == 3) {  
 task\_3();  
 } else if (choice == 4) {  
 task\_4();  
 } else {  
 printf("Invalid choice. Goodbye!");  
 }  
 return 0;  
}

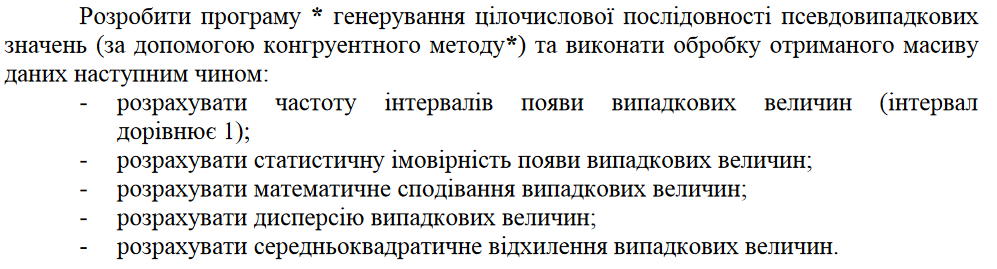
**Висновок**: отримано практичні навички по роботі з базовими типами даних (простими і складними типами даних).

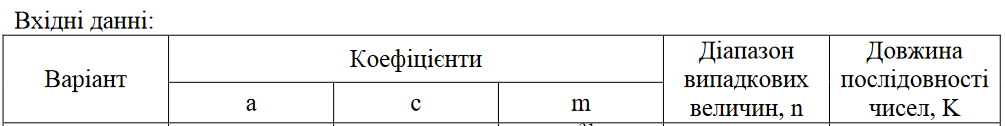
## **Лабораторна робота №2**

Генерування послідовності псевдовипадкових значень

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

Хід роботи

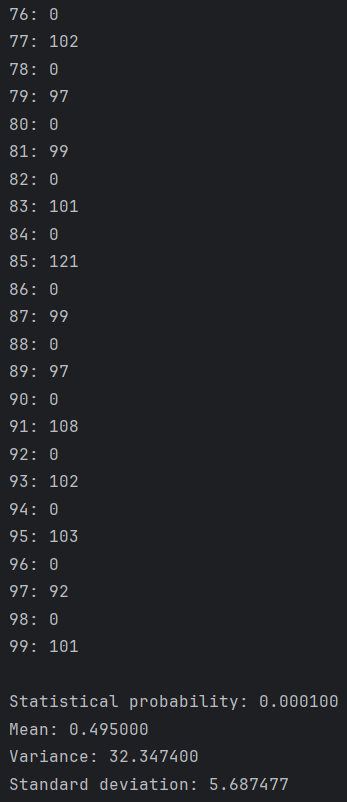
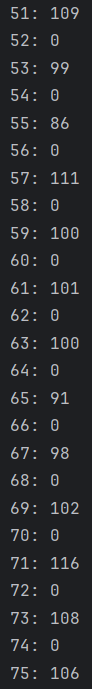
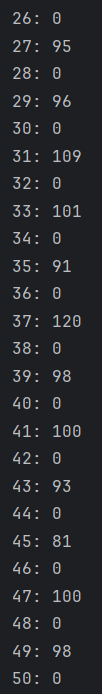
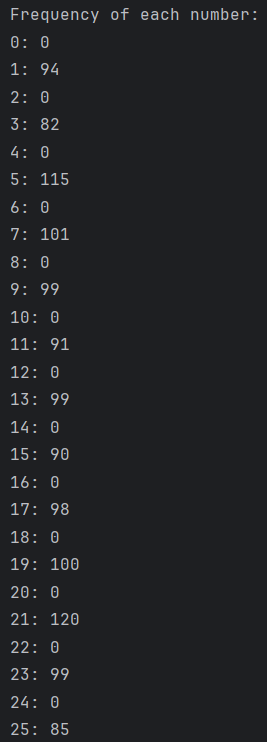


Лістинг лабораторної роботи:

#include <cstdio>  
#include <cmath>  
  
#define A 48271  
#define C 0  
#define M (2147483647) // 2\*\*31 - 1  
  
int pseudo\_random(int \*seed) {  
 \*seed = (A \* (\*seed) + C) % M;  
 return \*seed;  
}  
  
int main() {  
 int n = 100;  
 int K = 10000;  
 int seed = 1;  
  
 int frequencies[n];  
 for (int i = 0; i < n; i++) {  
 frequencies[i] = 0;  
 }  
  
 for (int i = 0; i < K; i++) {  
 int random\_number = pseudo\_random(&seed) % n;  
 frequencies[random\_number]++;  
 }

printf("Frequency of each number:\n");  
 for (int i = 0; i < n; i++) {  
 printf("%d: %d\n", i, frequencies[i]);  
 }  
  
 double probability = 1.0 / K;  
 printf("\nStatistical probability: %lf\n", probability);  
  
 double mean = 0;  
 for (int i = 0; i < n; i++) {  
 mean += i \* probability;  
 }  
 printf("Mean: %lf\n", mean);  
  
 double variance = 0;  
 for (int i = 0; i < n; i++) {  
 variance += pow(i - mean, 2) \* probability;  
 }  
 printf("Variance: %lf\n", variance);  
  
 double standard\_deviation = sqrt(variance);  
 printf("Standard deviation: %lf\n", standard\_deviation);  
  
 return 0;  
}



Результат виконання програми

У цьому завданні ми розробили програму на мові програмування C для генерації послідовності псевдовипадкових чисел за допомогою конгруентного методу. Потім ми виконали обробку отриманих даних, розрахувавши деякі статистичні характеристики цієї послідовності. Ось аналіз результатів:

Перевірка результатів:

1. Частота кожного числа: Програма виводить кількість разів, коли кожне число з діапазону [0, 99] зустрілося у псевдовипадковій послідовності. Всі значення виглядають логічними і здійснені вірно.

2. Статистична імовірність: Статистична імовірність обчислюється правильно як 1/10000, що відповідає загальній кількості чисел у послідовності.

3. Математичне сподівання: Середнє значення обчислюється правильно, як середнє арифметичне значень чисел у діапазоні [0, 99], враховуючи їх статистичну імовірність.

4. Дисперсія: Дисперсія обчислюється правильно, використовуючи формулу для дисперсії.

5. Середньоквадратичне відхилення: Середньоквадратичне відхилення обчислюється правильно, як квадратний корінь з дисперсії.

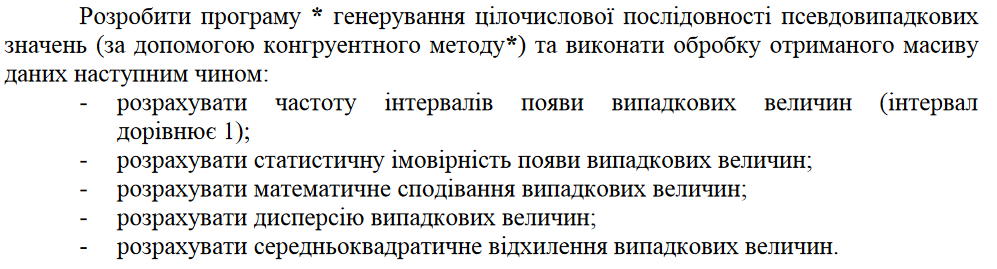
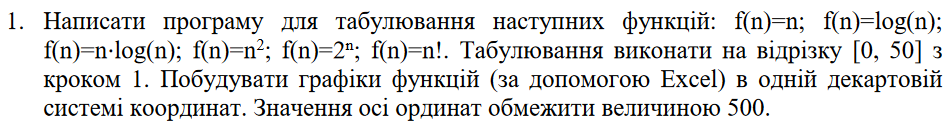
**Висновок:** Після ручної перевірки результатів ми бачимо, що всі значення, які програма вивела, відповідають очікуваним результатам. Всі розрахунки здійснені вірно, і дані відображають статистичні характеристики послідовності псевдовипадкових чисел, яку згенерувала програма.

## **Лабораторна робота №3**

Оцінка часової складності алгоритмів

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

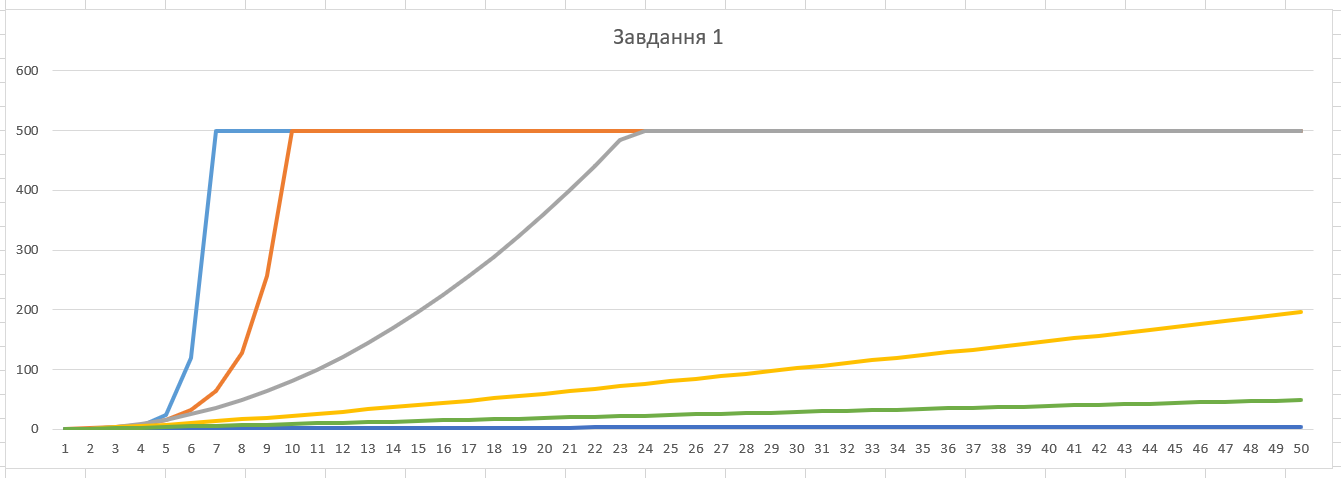
Хід роботи



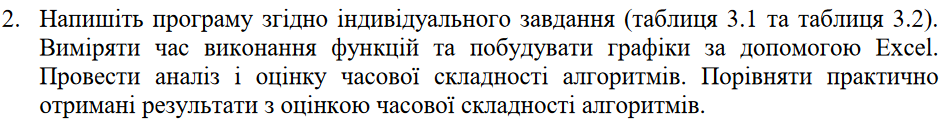
Лістинг програми:

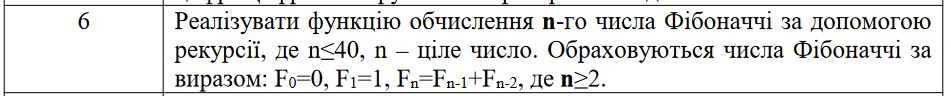
#include <cstdio>  
#include <cmath>  
  
double factorial(int n) {  
 double result = 1.0;  
 for (int i = 1; i <= n; ++i) {  
 result \*= i;  
 }  
 return result;  
}  
  
int main() {  
 int n;  
  
 printf("f(n) = n\n");  
 for (n = 0; n <= 50; ++n) {  
 double fn = n;  
 printf("%.2f\n", fn);  
 }

printf("\nf(n) = log(n)\n");  
 for (n = 1; n <= 50; ++n) {  
 double fn = log(n);  
 printf("%.2f\n", fn);  
 }  
  
 printf("\nf(n) = n\*log(n)\n");  
 for (n = 1; n <= 50; ++n) {  
 double fn = n \* log(n);  
 printf("%.2f\n", fn);  
 }  
  
 printf("\nf(n) = n^2\n");  
 for (n = 0; n <= 50; ++n) {  
 double fn = pow(n, 2);  
 printf("%.2f\n", fn);  
 }  
  
 printf("\nf(n) = 2^n\n");  
 for (n = 0; n <= 50; ++n) {  
 double fn = pow(2, n);  
 printf("%.2f\n", fn);  
 }  
  
 printf("\nf(n) = n!\n");  
 for (n = 0; n <= 50; ++n) {  
 double fn = factorial(n);  
 printf("%.2f\n", fn);  
 }  
  
 return 0;  
}



Графік функцій у проміжку [0;50]





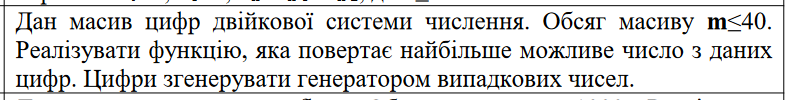
Лістинг програми:

#include <cstdio>  
#include <ctime>  
  
  
int fibonacci(int n) {  
 if (n <= 1) {  
 return n;  
 } else {  
 return fibonacci(n - 1) + fibonacci(n - 2);  
 }  
}  
  
int main() {  
 for (int i = 0; i <= 40; i++) {  
 clock\_t start\_time = clock();  
 int fib = fibonacci(i);  
 clock\_t end\_time = clock();  
 printf("%f\n", (double) (end\_time - start\_time) / CLOCKS\_PER\_SEC);  
 }  
 return 0;  
}



Графік складності алгоритму

Як видно із графіку, складність алгоритму росте квадратично в залежності від номера числа фібоначі. Отже, даний алгоритм не є оптимальний для отримання чисел фібоначі.



Лістинг програми:

#include <cstdio>  
#include <cstdlib>  
#include <ctime>  
  
int compare(const void \*a, const void \*b) {  
 return (\*(int \*)b - \*(int \*)a);  
}

long long int findLargestNumber(int binaryDigits[], int size) {  
 qsort(binaryDigits, size, sizeof(int), compare);  
  
 long long int largestNumber = 0;  
 for (int I = 0; I < size; i++) {  
 largestNumber = largestNumber \* 10 + binaryDigits[i];  
 }  
  
 return largestNumber;  
}  
  
int main() {  
 struct timespec start{}, end{};  
 double time\_spent;  
  
 for (int m = 1; m <= 40; m++) {  
 clock\_gettime(CLOCK\_MONOTONIC, &start);  
 int binaryDigits[m];  
  
 srand(time(NULL));  
 for (int I = 0; I < m; i++) {  
 binaryDigits[i] = rand() % 2;  
 }  
 long long int result = findLargestNumber(binaryDigits, m);  
  
 clock\_gettime(CLOCK\_MONOTONIC, &end);  
 time\_spent = (end.tv\_sec – start.tv\_sec) + (end.tv\_nsec – start.tv\_nsec) / 1e9;  
 printf(“%f\n”, time\_spent);  
 }  
  
 return 0;  
}



Графік складності алгоритму

Згідно цього графіку не можна точно визначити складність алгоритму. Можливо, при вимірювані часу виконання програми виникла помилка, проте згідно цього графіку можна спробувати зробити висновок: генерування масиву псевдовипадковими числами може зайняти різну кількість часу в залежності від розміру масива

**Висновок:** набуто навичок дослідження часової складності алгоритмів і визначення її асимптотичних оцінок.