ЛАБОРАТОРНА РОБОТА №1

Назва роботи: алгоритм; властивості, параметри та характеристики складності алгоритму.

Мета роботи: проаналізувати складність заданих алгоритмів.

1. Загальні відомості.

Здавна найбільшу увагу приділяли дослідженням алгоритму з метою мінімізації часової складності розв'язання задач. Але зміст складності алгоритму не обмежується однією характеристикою. В ряді випадків не менше значення має складність логіки побудови алгоритму, різноманітність його операцій, зв'язаність їх між собою. Ця характеристика алгоритму називається програмною складністю. В теорії алгоритмів, крім часової та програмної складності, досліджуються також інші характеристики складності, наприклад, місткісна, але найчастіше розглядають дві з них – часову і програмну. Якщо у кінцевому результаті часова складність визначає час розв'язання задачі, то програмна складність характеризує ступінь інтелектуальних зусиль, що потрібні для синтезу алгоритму. Вона впливає на витрати часу проектування алгоритму.

Вперше значення зменшення програмної складності продемонстрував аль-Хорезмі у своєму трактаті "Про індійський рахунок". Алгоритми реалізації арифметичних операцій, описані аль-Хорезмі у словесній формі, були першими у позиційній десятковій системі числення. Цікаво спостерігати, як точно і послідовно описує він алгоритм сумування, користуючись арабською системою числення і кільцем (нулем). В цьому опису є всі параметри алгоритму. Це один з перших відомих у світі вербальних арифметичних алгоритмів.

Схема розроблення будь-якого об'єкту складається з трьох операцій: синтез, аналіз та оптимізація *(Рис.1.)*.

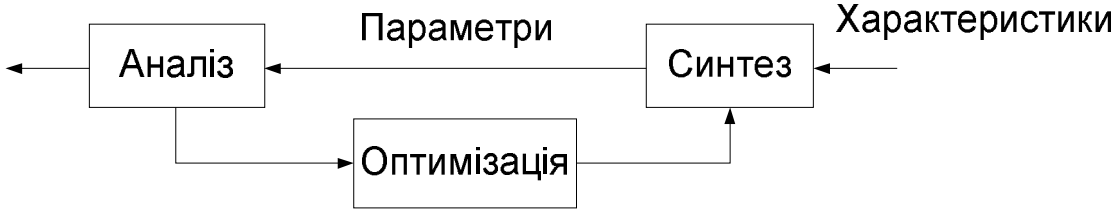


Рис. 1.

Існує два види синтезу: структурний і параметричний. Вихідними даними для структурного синтезу є параметри задачі – сформульоване намагання і набори вхідних даних. В результаті структурного синтезу отримують алгоритм, який розв'язує задачу в принципі.

Параметричний синтез змінами параметрів створює таку його структуру, яка дозволяє зменшити часову складність попередньої моделі. Існує багато способів конструювання ефективних алгоритмів на основі зміни параметрів. Розглянемо спосіб зміни правила безпосереднього перероблення на прикладі задачі знаходження найбільшого спільного дільника двох натуральних чисел..

Алгоритм знаходження найбільшого спільного дільника, яким ми користуємось для цієї цілі і донині, був запропонований Евклідом приблизно в 1150 році до н.е. у геометричній формі, в ньому порівняння величин проводилося відрізками прямих, без

використання арифметичних операцій Алгоритм розв'язку передбачав повторювання віднімання довжини коротшого відрізка від довжини довшого.

Опис алгоритму. Маючи два натуральні числа *a* та *b*, повторюємо *обчислення* для пари значень *b* та залишку від ділення *a* на *b* (тобто *a* mod *b*). Якщо *b*=0, то *a* є шуканим НСД.

Обчислення

Ітераційна версія:

```
НСД( a, b )
      поки b ≠ 0
           c = остача від ділення a на b
           a = b
           b = c
      поверни a
```

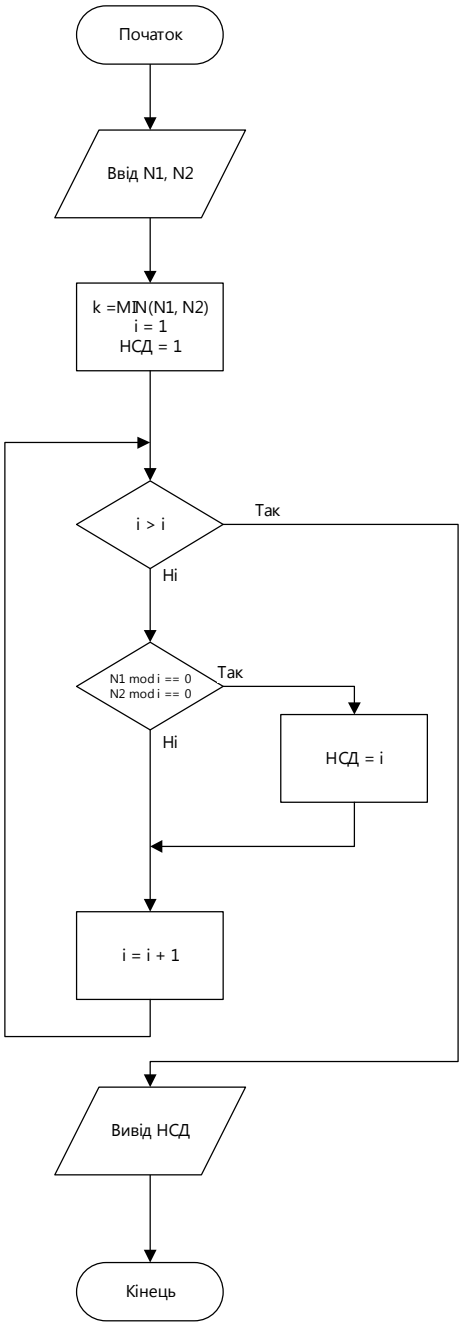
Рекурсивна версія:

```
НСД( a, b )
      якщо b == 0
         поверни a
      інакше
         поверни  НСД( b, остача від ділення a на b )
```

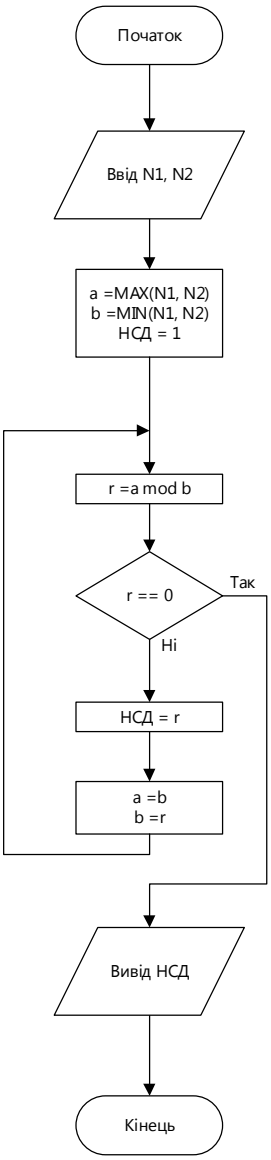
Для того, щоб довести ефективність алгоритму, потрібно порівняти його з таким, який приймається за неефективний. Прикладом такого неефективного алгоритму є процедура послідовного перебору можливих розв’язань задачі. Будемо вважати, що алгоритм перебору утворений в результаті структурного синтезу, на основі вхідних даних та намагання знайти серед всіх допустимих чисел такє, що є найбільшим дільником двох заданих чисел.

Ефективність, як правило, визначається такою характеристикою як часова складність, що вимірюється кількістю операцій, необхідних для розв’язання задачі.

Дослідимо розв’язання задачі знаходження найбільшого спільного дільника двох цілих чисел ($N_1>0$, $N_2>0$, $N_1 \geq N_2$) алгоритмом перебору і алгоритмом Евкліда. Алгоритм перебору заснований на операції інкременту змінної n від одиниці до меншого (N_2) з двох заданих чисел і перевірці, чи ця змінна є дільником заданих чисел. Якщо це так, то значення змінної запам’ятовується і операції алгоритму продовжуються. Якщо ні, то операції алгоритму продовжуються без запам’ятовування. Операції алгоритму закінчуються видачею з пам’яті знайденого останнім спільного дільника. Блок-схема алгоритму приведена на *рис.2(а)*.



а)



б)

Рис2. Блок-схема алгоритму перебору (а) і Евкліда (б).

Адаптований до сучасної арифметики алгоритм Евкліда використовує циклічну операцію ділення більшого числа на менше, знаходження остачі (r) і заміну числа, яке було більшим, на число, яке було меншим, а меншого числа на остачу. Всі перераховані операції виконуються в циклі. Операції циклу закінчуються, коли остача дорівнює нулю. Останній дільник є найбільшим спільним дільником. Блок-схема алгоритму приведена на *рис.2(б)*.

Аналіз цих двох алгоритмів показує, що часова складність алгоритму перебору значно перевищує часову складність алгоритму Евкліда. Для обох алгоритмів часова складність є функцією від вхідних даних, а не їх розміру. В таких випадках при порівнянні ефективності алгоритмів користуються порівнянням часових складностей визначених для найгіршого випадку. Часова складність для найгіршого випадку (Lmax) представляє собою максимальну часову складність серед всіх вхідних даних розміру N.

Часова складність $L_{\max}$ для алгоритму перебору:

$$L_{\max} = C \cdot N_2 \qquad (1)$$

де C – константа , яка дорівнює кількості операцій в кожній ітерації.

Для цілих чисел n $(1 \le n < r_i)$ алгоритм Евкліда знаходження найбільшого спільного дільника має найбільшу часову складність для пари чисел r_{i-1} і r_{i-2} , де 1 , 2 , 3 , … , r_{i-2}, r_{i-1} , r_i – числа Фібоначчі.

Алгоритм Евкліда є ефективним за часовою складністю у порівняні з алгоритмом перебору. Мінімізація часової складності дозволяє за всіх інших рівних умова збільшити продуктивність розв'язання задачі.

2. Приклад програми.

Лістинг 2.1

```
//to measure time it is better to run with Linux

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N1  18
#define N2  27

#define MIN(a,b) (((a)<(b))?(a):(b))
#define MAX(a,b) (((a)>(b))?(a):(b))

#define REPEAT_COUNT 1000000
#define REPEATOR(count, code) \
for (unsigned int indexIteration = (count); indexIteration--;){ code; }
#define TWO_VALUES_SELECTOR(variable, firstValue, secondValue) \
       (variable) = indexIteration % 2 ? (firstValue) : (secondValue);

double getCurrentTime(){
       clock_t time = clock();
       if (time != (clock_t)-1) {
              return ((double)time / (double)CLOCKS_PER_SEC);
       }
       return 0.; // else
}

unsigned long long int f1_GCD(unsigned long long int variableN1, unsigned long long int
variableN2){
    unsigned long long int returnValue = 1;

       for(unsigned long long int i = 1, k = MIN(variableN1, variableN2); i <= k; i++){
           if(!(variableN1 % i || variableN2 % i)){
               returnValue = i;
           }
       }

       return returnValue;
}

unsigned long long int f2_GCD(unsigned long long int a, unsigned long long int b){
       for(unsigned long long int aModB;
              aModB = a % b,
              a = b,
              b = aModB;
       );
```

```
        return a;
}

unsigned long long int f3_GCD(unsigned long long int a, unsigned long long int b){
    if(!b){
            return a;
        }

        return f3_GCD(b, a % b); // else
}

int main() {
    unsigned long long int vN1 = N1, vN2 = N2, a = MAX(vN1, vN2), b = MIN(vN1, vN2),
        vN1_ = vN1, vN2_ = vN2, a_ = a, b_ = b,
        returnValue;

        double startTime, endTime;

        // f1_GCD
        startTime = getCurrentTime();
        REPEATOR(REPEAT_COUNT,
            TWO_VALUES_SELECTOR(vN1, 16, vN1_);
            TWO_VALUES_SELECTOR(vN2, 4, vN2_);
            returnValue = f1_GCD(a, b);
        )
        endTime = getCurrentTime();
        printf("f1_GCD return %d \r\nrun time: %dns\r\n\r\n",
        returnValue,
        (unsigned int)((endTime - startTime) * (double)(1000000000 / REPEAT_COUNT)));

        // f2_GCD
        startTime = getCurrentTime();
        REPEATOR(REPEAT_COUNT,
            TWO_VALUES_SELECTOR(a, 16, a_);
            TWO_VALUES_SELECTOR(b, 4, b_);
            returnValue = f2_GCD(a, b);
        )
        endTime = getCurrentTime();
        printf("f2_GCD return %d \r\nrun time: %dns\r\n\r\n",
        returnValue,
        (unsigned int)((endTime - startTime) * (double)(1000000000 / REPEAT_COUNT)));

        // f3_GCD
        startTime = getCurrentTime();
        REPEATOR(REPEAT_COUNT,
            TWO_VALUES_SELECTOR(a, 16, a_);
            TWO_VALUES_SELECTOR(b, 4, b_);
            returnValue = f3_GCD(a, b);
        )
        endTime = getCurrentTime();
        printf("f3_GCD return %d \r\nrun time: %dns\r\n\r\n",
        returnValue,
        (unsigned int)((endTime - startTime) * (double)(1000000000 / REPEAT_COUNT)));

        printf("Press any key to continue . . .");
        getchar();

        return 0;
}
```

3. Спрощене завдання.

Скласти програму (C/C++), яка дозволяє провести порівняння трьох алгоритмів(алгоритму перебору, ітераційної версії алгоритму Евкліда та рекурсивної версії алгоритму Евкліда) знаходження НСД за характеристикою часової складності для таких вхідних даних:

Варіант	**Вхідні дані**	
	N1	**N2**
1	112	107
2	146	141
3	166	161
4	186	181
5	206	201
6	226	221
7	246	241
8	266	261
9	286	281
10	306	301
11	326	321
12	346	341
13	366	361
14	386	381
15	406	401
16	426	421
17	446	441
18	466	461
19	486	481
20	506	501
21	526	521
22	546	541
23	566	561
24	586	581
25	606	601
26	626	621
27	646	641
28	666	661
29	686	681
30	706	701

**** Для отримання 50% балів за лабораторну роботу можна використати наявний програмний код** з лістингу 2.1. Для отримання 100% балів за лабораторну роботу потрібно написати власний код.*

4. Завдання базового рівня.

Скласти програму (C/C++), яка реалізовує заданий згідно варіанту алгоритм та дозволяє провести порівняння цього алгоритму з аналогічним алгоритмом прямого перебору за характеристикою часової складності для вхідних даних, що вводяться під час роботи програми.

Варіант	**Алгоритм**
1	Алгоритм Косараджу
2	Алгоритм Борувки
3	Алгоритм Крускала
4	Алгоритм Прима
5	Алгоритм Дейкстри
6	Алгоритм Флойда-Воршала
7	Алгоритм Джонсона
8	Алгоритм Беллмана-Форда
9	Алгоритм Данцига
10	Алгоритм Лі
11	Алгоритм Ахо-Корасік
12	Алгоритм Бояра-Мура
13	Алгоритм Бояра-Мура-Горспула
14	Алгоритм Кнута-Моріса-Прата
15	Алгоритм Коменц-Вальтер
16	Алгоритм Рабіна-Карпа
17	Алгоритм Нідлмана-Вунша
18	Алгоритм Барнса-Хата
19	Алгоритм Діксона
20	Алгоритм Луна
21	Алгоритм Штрассена
22	Алгоритм Копперсміта-Віноградa
23	Алгоритм Пана
24	Алгоритм Біні
25	Алгоритми Шенхаге
26	Алгоритм Малхотри-Кумара-Махешварі
27	Алгоритм Галіла-Наамада
28	Алгоритм Ахьюа-Орліна-Тар'яна
29	Алгоритм Черіяна-Хейджрапа-Мехлхорна
30	Алгоритм Келнера-Мондри-Спілман-Тена

5. Зміст звіту

- Титульний лист;
- Завдання;
- Алгоритм рішення завдання;
- Код програми;
- Екранна форма з результатами роботи програми;
- Висновки.

ЛАБОРАТОРНА РОБОТА №2

Назва роботи: асимптотичні характеристики складності алгоритму; алгоритми з поліноміальною та експоненціальною складністю.

Мета роботи: ознайомитись з асимптотичними характеристиками складності та класами складності алгоритмів.

1.1. Часова складність.

В процесі розв'язку задачі вибір алгоритму викликає певні труднощі. Алгоритм повинен задовільняти вимогам, які часом суперечать одна одній:

- бути простим для розуміння, переводу в програмний код, відлагодження;
- ефективно використовувати комп'ютерні ресурси і виконуватись швидко.

Якщо програма повинна виконуватись декілька разів, то перша вимога більш важлива. Вартість робочого часу програміста перевищує вартість машинного часу виконання програми, тому вартість програми оптимізується по вартості написання, а не виконання програми. Якщо задача вимагає значних обчислювальних витрат, то вартість виконання програми може перевищити вартість написання програми, особливо коли програма повинна виконуватись багаторазово. Але навіть в цій ситуації доцільно спочатку реалізувати простий алгоритм, і з'ясувати яким чином повинна себе поводити більш складна програма.

На час виконання програми впливають наступні чинники:

- ввід інформації в програму;
- якість скомпільованого коду;
- машинні інструкції, які використовуються для виконання програми;
- часова складність алгоритму(ЧС).

Часова складність є функцією від вхідних даних. Для деяких задач ЧС залежить від самих вхідних даних (знаходження найбільшого спільного дільника двох чисел), для інших – від їх "розміру" (задачі сортування).

Коли ЧС є функцією від самих даних, її визначають як ЧС для найгіршого випадку, тобто як найбільшу кількість інструкцій програми серед всіх можливих вхідних даних для цього алгоритму.

Використовується також ЧС в середньому випадку (в статистичному сенсі), як середня кількість інструкцій по всім можливим вхідним даним. На практиці ЧС в середньому випадку важче визначити ніж ЧС для найгіршого випадку, через те що це математично важка для розв'язання задача. Крім того, іноді важко визначити, що означає "середні" вхідні дані.

Коли ЧС є функцією від кількості вхідних даних, аналізується швидкість зростання цієї функції.

1.2. Асимптотичні співвідношення

Для опису швидкості зростання функцій використовується О-символіка. Функція f(n) має порядок зростання O(g(n)), якщо існують додатні константи C і $\mathbf{n}_0$ такі, що:

$$f(n) <= C*g(n), \quad \text{для } n > n_0.$$

Позначемо функцію яка виражає залежність часової складності від кількості вхідних даних (**n**) через **L(n)**. Тоді, наприклад, коли говорять, що часова складність **L(n)** алгоритму має порядок(степінь) зростання $O(n^2)$ (читається як "О велике від **n** в квадраті", або просто як "о від **n** в квадраті", то вважається, що існують додатні константи **c** і $\mathbf{n}_0$ такі, що для всіх **n**, більших або рівних $\mathbf{n}_0$, виконується нерівність $\mathbf{L(n)<=cn^2}$.

Наприклад, функція **L(n) = $3n^3+2n^2$** має порядок зростання **O(n^3)**. Нехай **n_0**=0 і **c**=5. Очевидно, що для всіх цілих n>=0 виконується нерівність **$3n^3+2n^2$ <= $5n^3$.**

Коли кажуть, що **L(n)** має степінь зростання **O(f(n))**, то вважається, що **f(n)** є верхньою границею швидкості зростання **L(n).** Щоби вказати нижню границю швидкості зростання **L(n)** використовують позначення **Ω(g(n))**, що означає існування такої константи **c**, що для нескінченої кількості значень **n** виконується нерівність **L(n)>=c*g(n).**

Теоретичне визначення порядку зростання функції є складною математичною задачею. На практиці визначення порядку зростання є задачею, що цілком вирішується за допомогою кількох базових принципів. Існують три правила для визначення складності:

1. **O(c* f(n))=O(f(n))**
2. **O(f(n) + g(n)) = O(max(f(n), g(n)))**
3. **O(f(n) * g(n)) = O(f(n)) * O(g(n))**

Перше правило декларує, що постійні множники не мають значення для визначення порядку зростання.

Друге правило називається ***"Правило сум".*** Це правило використовується для послідовних програмних фрагментів з циклами та розгалуженнями. Порядок зростання скінченої послідовності програмних фрагментів (без врахування констант) дорівнює порядку зростання фрагменту з найбільшою часовою складністю. Якщо алгоритм складається з двох фрагментів, функції часових складностей яких **L_1(n)** і **L_2(n)** мають ступені зростання **O(f(n))** і **O(g(n))** відповідно, то алгоритм має степінь зростання **O(max(f(n),g(n))).**

Третє правило називається ***"Правило добутків"***. Якщо **L_1(n)** і **L_2(n)** мають ступені зростання **O(f(n))** і **O(g(n))** відповідно, то добуток **L_1(n) L_2(n)** має степінь зростання **O(f(n)g(n))** . Прикладом може бути фрагмент програми "цикл в циклі".

2. Приклад.

Задані функції часової складності L(n) для чотирьох алгоритмів:

1. $L_1(n) = n\sqrt{n}$ 2. $L_2(n) = 2^n + n$ 3. $L_3(n) = 3n^2 + 2n^3$ 4. $L_4(n) = n + \log_2 n$

Використавши правило сум і правило добутків знайдемо O(n) :

$O_1(n) = n\sqrt{n}$ $O_2(n) = 2^n$ $O_3(n) = n^3$ $O_4(n) = n$

Розташуємо функції $O_i(n)$ у порядку зростання:

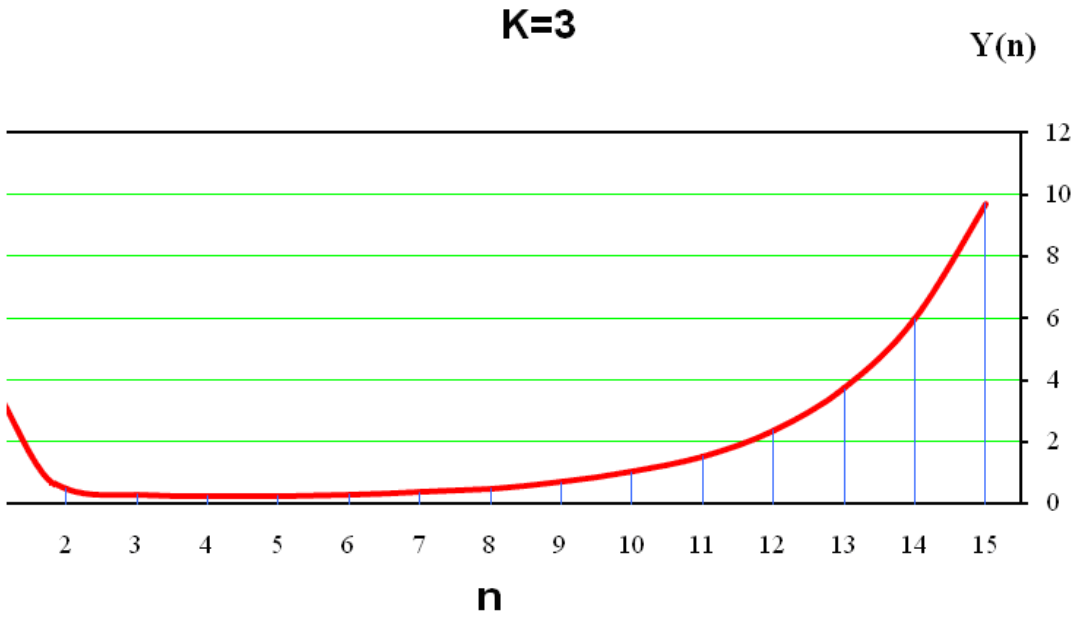
1. $O_4(n) = n$ 2. $O_1(n) = n\sqrt{n}$ 3. $O_3(n) = n^3$ 4. $O_2(n) = 2^n$

Функція $O_2(n) = 2^n$ має найбільший степінь зростання.

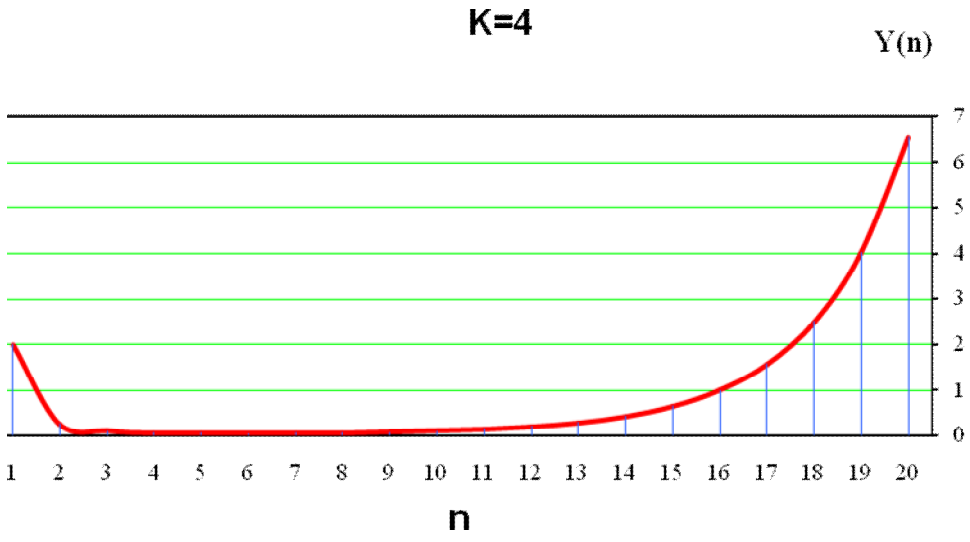
Побудуємо графіки $Y(n)=\frac{O_2(n)}{P_k(n)}$ для n = (1,2,…,10); k = 3,4,5

Для спрощення будемо вважати що поліном для відповідних значень **K** буде прирівнюватися до n^3, n^4 та n^5, оскільки ці значення є тою адитивною складовою в поліномі, яка найшвидше зростає.

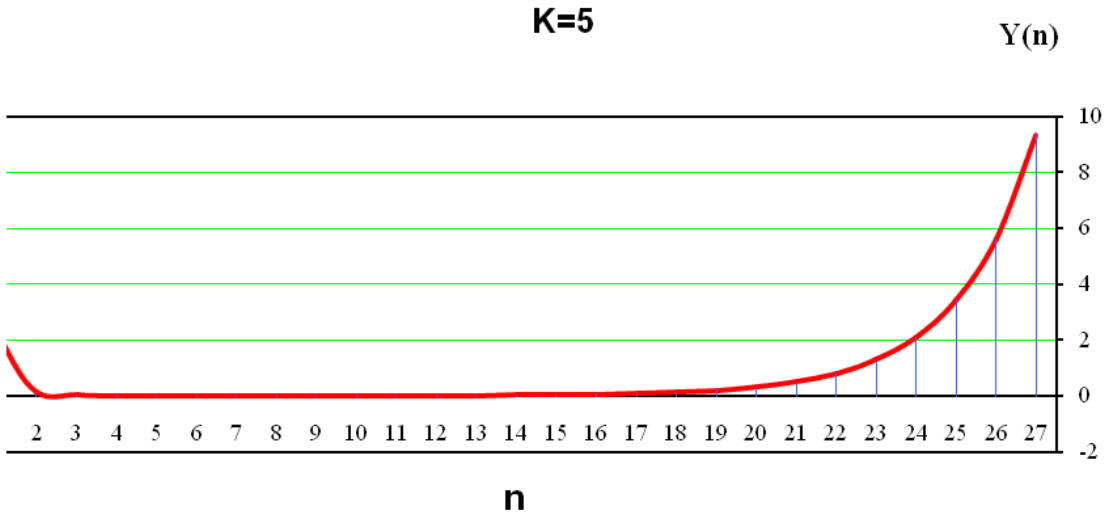
$$Y(n)=\frac{2^n}{n^3}:$$



$$Y(n)=\frac{2^n}{n^4}:$$



$$Y(n) = \frac{2^n}{n^5}:$$



Графіки показують, що існують такі значення $\mathbf{n}_0$ (при зростанні **K** значення $\mathbf{n}_0$ теж зростає)**,** починаючи з яких значення функції порядку зростання часової складності буде приймати більші значення ніж значення відповідного поліному. Це ілюструє приналежність алгоритму до класу алгоритмів з експоненціальною складністю.

3. Приклад програми

Лістинг 3.1

```
// don't forget to use compilation key for Linux: -lm

#define _CRT_SECURE_NO_WARNINGS  // for using fopen in VS

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

#ifndef UINT
#define UINT  unsigned long int
#endif

#ifndef USHORT
#define USHORT      unsigned short
#endif

#ifndef UCHAR
#define UCHAR unsigned char
#endif

#define QDBMP_VERSION_MAJOR       1
#define QDBMP_VERSION_MINOR       0
#define QDBMP_VERSION_PATCH       1

typedef enum
{
       BMP_OK = 0,
       BMP_ERROR,
```

```
        BMP_OUT_OF_MEMORY,
        BMP_IO_ERROR,
        BMP_FILE_NOT_FOUND,
        BMP_FILE_NOT_SUPPORTED,
        BMP_FILE_INVALID,
        BMP_INVALID_ARGUMENT,
        BMP_TYPE_MISMATCH,
        BMP_ERROR_NUM
} BMP_STATUS;

typedef struct _BMP BMP;

BMP*                BMP_Create(UINT width, UINT height, USHORT depth);
void                BMP_Free(BMP* bmp);

BMP*                BMP_ReadFile(const char* filename);
void                BMP_WriteFile(BMP* bmp, const char* filename);

UINT                BMP_GetWidth(BMP* bmp);
UINT                BMP_GetHeight(BMP* bmp);
USHORT              BMP_GetDepth(BMP* bmp);

void                BMP_GetPixelRGB(BMP* bmp, UINT x, UINT y, UCHAR* r, UCHAR* g, UCHAR*
b);
void                BMP_SetPixelRGB(BMP* bmp, UINT x, UINT y, UCHAR r, UCHAR g, UCHAR b);
void                BMP_GetPixelIndex(BMP* bmp, UINT x, UINT y, UCHAR* val);
void                BMP_SetPixelIndex(BMP* bmp, UINT x, UINT y, UCHAR val);

void                BMP_GetPaletteColor(BMP* bmp, UCHAR index, UCHAR* r, UCHAR* g, UCHAR*
b);
void                BMP_SetPaletteColor(BMP* bmp, UCHAR index, UCHAR r, UCHAR g, UCHAR
b);

BMP_STATUS          BMP_GetError();
const char*         BMP_GetErrorDescription();



#define OUTPUT_SIZE   512
#define OUTPUT_WIDTH  OUTPUT_SIZE
#define OUTPUT_HEIGHT OUTPUT_SIZE
#define NEIGHBOURHOOD 0
#define OUTPUT_SCALE 32


#define BMP_CHECK_ERROR( output_file, return_value ) \
if (BMP_GetError() != BMP_OK)
                            \
{
                                                    \
       fprintf((output_file), "BMP error: %s\n", BMP_GetErrorDescription());      \
       return(return_value);
                                    \
}
                                                    \

void mark(BMP * bmpPtr, UINT neighbourhood, UINT xSize, UINT ySize, UCHAR pt, UINT
scale){
       UINT neighbourhood2 = 2 * neighbourhood;
       UINT   xO, yO;

       for (UINT x = 0; x < xSize; ++x){
              for (UINT y = 0; !y || (!(x % scale) && y < 10); ++y){
                     for (xO = 0; xO <= neighbourhood2; ++xO){
                            for (yO = 0; yO <= neighbourhood2; ++yO){
```

```
                                        if (x + xO >= neighbourhood && x + xO - neighbourhood <
xSize && y + yO >= neighbourhood && y + yO - neighbourhood < ySize){
                                                BMP_SetPixelIndex(bmpPtr, x + xO - neighbourhood,
ySize - (y + yO - neighbourhood), pt);
                                        }
                                }
                        }
                }
        }

        for (UINT y = 0; y < ySize; ++y){
                for (UINT x = 0; !x || (!(y % scale) && x < 10); ++x){
                        for (xO = 0; xO <= neighbourhood2; ++xO){
                                for (yO = 0; yO <= neighbourhood2; ++yO){
                                        if (x + xO >= neighbourhood && x + xO - neighbourhood <
xSize && y + yO >= neighbourhood && y + yO - neighbourhood < ySize){
                                                BMP_SetPixelIndex(bmpPtr, x + xO - neighbourhood,
ySize - (y + yO - neighbourhood), pt);
                                        }
                                }
                        }
                }
        }
}

double functionForTabulateK3(double arg){
        return pow(2., arg) / pow(arg, 3.);
}

double functionForTabulateK4(double arg){
        return pow(2., arg) / pow(arg, 4.);
}

double functionForTabulateK5(double arg){
        return pow(2., arg) / pow(arg, 5.);
}

void tabulate(double(*functionPtr)(double arg), BMP * bmpPtr, UINT neighbourhood, UINT
xSize, UINT ySize, UCHAR pt, UINT scale){
        UINT neighbourhood2 = 2 * neighbourhood;
        UINT   xO, yO;
        UINT   fraction;
        UINT   scaledXSize = (UINT)((double)xSize / (double)scale);
        for (UINT x_ = 0; x_ < scaledXSize; x_++){
                for (fraction = 0; fraction < scale; ++fraction){
                        double fX = (double)fraction / (double)scale + (double)x_;
                        UINT x = (UINT)(fX * (double)scale);
                        UINT y = (UINT)(functionPtr(fX) * (double)scale);
                        for (xO = 0; xO <= neighbourhood2; xO++){
                                for (yO = 0; yO <= neighbourhood2; yO++){
                                        if (x + xO >= neighbourhood && x + xO - neighbourhood <
xSize && y + yO >= neighbourhood && y + yO - neighbourhood < ySize){
                                                BMP_SetPixelIndex(bmpPtr, x + xO - neighbourhood,
ySize - (y + yO - neighbourhood), pt);
                                        }
                                }
                        }
                }
        }
}

int main(int argc, char* argv[])
{
        BMP *outputForK3, *outputForK4, *outputForK5;
        UCHAR  r = 0xff, g = 0xff, b = 0xff;
```

```
	outputForK3 = BMP_Create(OUTPUT_WIDTH, OUTPUT_HEIGHT, 8);
	BMP_CHECK_ERROR(stderr, -3);

	outputForK4 = BMP_Create(OUTPUT_WIDTH, OUTPUT_HEIGHT, 8);
	BMP_CHECK_ERROR(stderr, -3);

	outputForK5 = BMP_Create(OUTPUT_WIDTH, OUTPUT_HEIGHT, 8);
	BMP_CHECK_ERROR(stderr, -3);

	BMP_SetPaletteColor(outputForK3, 2, 0, 255, 0);
	BMP_SetPaletteColor(outputForK3, 1, 255, 0, 0);
	BMP_SetPaletteColor(outputForK3, 0, 0, 0, 0);

	BMP_SetPaletteColor(outputForK4, 2, 0, 255, 0);
	BMP_SetPaletteColor(outputForK4, 1, 255, 0, 0);
	BMP_SetPaletteColor(outputForK4, 0, 0, 0, 0);

	BMP_SetPaletteColor(outputForK5, 2, 0, 255, 0);
	BMP_SetPaletteColor(outputForK5, 1, 255, 0, 0);
	BMP_SetPaletteColor(outputForK5, 0, 0, 0, 0);

	mark(outputForK3, 0, OUTPUT_WIDTH, OUTPUT_HEIGHT, 1, OUTPUT_SCALE);
	tabulate(functionForTabulateK3, outputForK3, NEIGHBOURHOOD, OUTPUT_WIDTH,
OUTPUT_HEIGHT, 2, OUTPUT_SCALE);

	mark(outputForK4, 0, OUTPUT_WIDTH, OUTPUT_HEIGHT, 1, OUTPUT_SCALE);
	tabulate(functionForTabulateK4, outputForK4, NEIGHBOURHOOD, OUTPUT_WIDTH,
OUTPUT_HEIGHT, 2, OUTPUT_SCALE);

	mark(outputForK5, 0, OUTPUT_WIDTH, OUTPUT_HEIGHT, 1, OUTPUT_SCALE);
	tabulate(functionForTabulateK5, outputForK5, NEIGHBOURHOOD, OUTPUT_WIDTH,
OUTPUT_HEIGHT, 2, OUTPUT_SCALE);

	BMP_WriteFile(outputForK3, "K3.bmp");
	BMP_CHECK_ERROR(stderr, -5);

	BMP_WriteFile(outputForK4, "K4.bmp");
	BMP_CHECK_ERROR(stderr, -5);

	BMP_WriteFile(outputForK5, "K5.bmp");
	BMP_CHECK_ERROR(stderr, -5);

	BMP_Free(outputForK3);
	BMP_Free(outputForK4);
	BMP_Free(outputForK5);

	printf("Result writed to \"out.bmp\".\r\n");
	printf("An example file may be located:\r\n");
	printf("                \"..\\Visual Studio "
"2013\\Projects\\amolab2\\amolab2\\out.bmp\"\r\n");
	printf("After closing the program, open it manually.\r\n\r\n");
	printf("Press any key to continue . . .");
	getchar();

	return 0;
}

/******************************** BMP Implementation *******************************/

typedef struct _BMP_Header
{
	USHORT		Magic;
	UINT		FileSize;
	USHORT		Reserved1;
	USHORT		Reserved2;
	UINT		DataOffset;
	UINT		HeaderSize;
```

```
        UINT            Width;
        UINT            Height;
        USHORT          Planes;
        USHORT          BitsPerPixel;
        UINT            CompressionType;
        UINT            ImageDataSize;
        UINT            HPixelsPerMeter;
        UINT            VPixelsPerMeter;
        UINT            ColorsUsed;
        UINT            ColorsRequired;
} BMP_Header;

/* Private data structure */
struct _BMP
{
        BMP_Header      Header;
        UCHAR*          Palette;
        UCHAR*          Data;
};

static BMP_STATUS BMP_LAST_ERROR_CODE = 0;

static const char* BMP_ERROR_STRING[] =
{
        "",
        "General error",
        "Could not allocate enough memory to complete the operation",
        "File input/output error",
        "File not found",
        "File is not a supported BMP variant (must be uncompressed 8, 24 or 32 BPP)",
        "File is not a valid BMP image",
        "An argument is invalid or out of range",
        "The requested action is not compatible with the BMP's type"
};

#define BMP_PALETTE_SIZE    ( 256 * 4 )

int             ReadHeader(BMP* bmp, FILE* f);
int             WriteHeader(BMP* bmp, FILE* f);

int             ReadUINT(UINT* x, FILE* f);
int             ReadUSHORT(USHORT *x, FILE* f);

int             WriteUINT(UINT x, FILE* f);
int             WriteUSHORT(USHORT x, FILE* f);

BMP* BMP_Create(UINT width, UINT height, USHORT depth)
{
        BMP*    bmp;
        int             bytes_per_pixel = depth >> 3;
        UINT    bytes_per_row;

        if (height <= 0 || width <= 0)
        {
                BMP_LAST_ERROR_CODE = BMP_INVALID_ARGUMENT;
                return NULL;
        }

        if (depth != 8 && depth != 24 && depth != 32)
        {
                BMP_LAST_ERROR_CODE = BMP_FILE_NOT_SUPPORTED;
                return NULL;
        }

        bmp = calloc(1, sizeof(BMP));
        if (bmp == NULL)
        {
                BMP_LAST_ERROR_CODE = BMP_OUT_OF_MEMORY;
                return NULL;
        }

        bmp->Header.Magic = 0x4D42;
        bmp->Header.Reserved1 = 0;
        bmp->Header.Reserved2 = 0;
        bmp->Header.HeaderSize = 40;
        bmp->Header.Planes = 1;
        bmp->Header.CompressionType = 0;
        bmp->Header.HPixelsPerMeter = 0;
        bmp->Header.VPixelsPerMeter = 0;
        bmp->Header.ColorsUsed = 0;
        bmp->Header.ColorsRequired = 0;

        bytes_per_row = width * bytes_per_pixel;
        bytes_per_row += (bytes_per_row % 4 ? 4 - bytes_per_row % 4 : 0);

        bmp->Header.Width = width;
        bmp->Header.Height = height;
        bmp->Header.BitsPerPixel = depth;
```

```
	bmp->Header.ImageDataSize = bytes_per_row * height;
	bmp->Header.FileSize = bmp->Header.ImageDataSize + 54 + (depth == 8 ? BMP_PALETTE_SIZE : 0);
	bmp->Header.DataOffset = 54 + (depth == 8 ? BMP_PALETTE_SIZE : 0);

	if (bmp->Header.BitsPerPixel == 8)
	{
		bmp->Palette = (UCHAR*)calloc(BMP_PALETTE_SIZE, sizeof(UCHAR));
		if (bmp->Palette == NULL)
		{
			BMP_LAST_ERROR_CODE = BMP_OUT_OF_MEMORY;
			free(bmp);
			return NULL;
		}
	}
	else
	{
		bmp->Palette = NULL;
	}

	bmp->Data = (UCHAR*)calloc(bmp->Header.ImageDataSize, sizeof(UCHAR));
	if (bmp->Data == NULL)
	{
		BMP_LAST_ERROR_CODE = BMP_OUT_OF_MEMORY;
		free(bmp->Palette);
		free(bmp);
		return NULL;
	}

	BMP_LAST_ERROR_CODE = BMP_OK;

	return bmp;
}

void BMP_Free(BMP* bmp)
{
	if (bmp == NULL)
	{
		return;
	}

	if (bmp->Palette != NULL)
	{
		free(bmp->Palette);
	}

	if (bmp->Data != NULL)
	{
		free(bmp->Data);
	}

	free(bmp);

	BMP_LAST_ERROR_CODE = BMP_OK;
}

BMP* BMP_ReadFile(const char* filename)
{
	BMP*	bmp;
	FILE*	f;

	if (filename == NULL)
	{
		BMP_LAST_ERROR_CODE = BMP_INVALID_ARGUMENT;
		return NULL;
	}

	bmp = calloc(1, sizeof(BMP));
	if (bmp == NULL)
	{
		BMP_LAST_ERROR_CODE = BMP_OUT_OF_MEMORY;
		return NULL;
	}

	f = fopen(filename, "rb");
	if (f == NULL)
	{
		BMP_LAST_ERROR_CODE = BMP_FILE_NOT_FOUND;
		free(bmp);
		return NULL;
	}

	if (ReadHeader(bmp, f) != BMP_OK || bmp->Header.Magic != 0x4D42)
	{
		BMP_LAST_ERROR_CODE = BMP_FILE_INVALID;
		fclose(f);
		free(bmp);
		return NULL;
	}
```

```
	if ((bmp->Header.BitsPerPixel != 32 && bmp->Header.BitsPerPixel != 24 && bmp->Header.BitsPerPixel != 8)
		|| bmp->Header.CompressionType != 0 || bmp->Header.HeaderSize != 40)
	{
		BMP_LAST_ERROR_CODE = BMP_FILE_NOT_SUPPORTED;
		fclose(f);
		free(bmp);
		return NULL;
	}

	if (bmp->Header.BitsPerPixel == 8)
	{
		bmp->Palette = (UCHAR*)malloc(BMP_PALETTE_SIZE * sizeof(UCHAR));
		if (bmp->Palette == NULL)
		{
			BMP_LAST_ERROR_CODE = BMP_OUT_OF_MEMORY;
			fclose(f);
			free(bmp);
			return NULL;
		}

		if (fread(bmp->Palette, sizeof(UCHAR), BMP_PALETTE_SIZE, f) != BMP_PALETTE_SIZE)
		{
			BMP_LAST_ERROR_CODE = BMP_FILE_INVALID;
			fclose(f);
			free(bmp->Palette);
			free(bmp);
			return NULL;
		}
	}
	else
	{
		bmp->Palette = NULL;
	}

	bmp->Data = (UCHAR*)malloc(bmp->Header.ImageDataSize);
	if (bmp->Data == NULL)
	{
		BMP_LAST_ERROR_CODE = BMP_OUT_OF_MEMORY;
		fclose(f);
		free(bmp->Palette);
		free(bmp);
		return NULL;
	}

	if (fread(bmp->Data, sizeof(UCHAR), bmp->Header.ImageDataSize, f) != bmp->Header.ImageDataSize)
	{
		BMP_LAST_ERROR_CODE = BMP_FILE_INVALID;
		fclose(f);
		free(bmp->Data);
		free(bmp->Palette);
		free(bmp);
		return NULL;
	}

	fclose(f);

	BMP_LAST_ERROR_CODE = BMP_OK;

	return bmp;
}

void BMP_WriteFile(BMP* bmp, const char* filename)
{
	FILE*	f;

	if (filename == NULL)
	{
		BMP_LAST_ERROR_CODE = BMP_INVALID_ARGUMENT;
		return;
	}

	f = fopen(filename, "wb");
	if (f == NULL)
	{
		BMP_LAST_ERROR_CODE = BMP_FILE_NOT_FOUND;
		return;
	}

	if (WriteHeader(bmp, f) != BMP_OK)
	{
		BMP_LAST_ERROR_CODE = BMP_IO_ERROR;
		fclose(f);
		return;
	}

	if (bmp->Palette)
	{
```

```
                    if (fwrite(bmp->Palette, sizeof(UCHAR), BMP_PALETTE_SIZE, f) != BMP_PALETTE_SIZE)
                    {
                            BMP_LAST_ERROR_CODE = BMP_IO_ERROR;
                            fclose(f);
                            return;
                    }
            }

            if (fwrite(bmp->Data, sizeof(UCHAR), bmp->Header.ImageDataSize, f) != bmp->Header.ImageDataSize)
            {
                    BMP_LAST_ERROR_CODE = BMP_IO_ERROR;
                    fclose(f);
                    return;
            }

            BMP_LAST_ERROR_CODE = BMP_OK;
            fclose(f);
}

UINT BMP_GetWidth(BMP* bmp)
{
            if (bmp == NULL)
            {
                    BMP_LAST_ERROR_CODE = BMP_INVALID_ARGUMENT;
                    return -1;
            }

            BMP_LAST_ERROR_CODE = BMP_OK;

            return (bmp->Header.Width);
}

UINT BMP_GetHeight(BMP* bmp)
{
            if (bmp == NULL)
            {
                    BMP_LAST_ERROR_CODE = BMP_INVALID_ARGUMENT;
                    return -1;
            }

            BMP_LAST_ERROR_CODE = BMP_OK;

            return (bmp->Header.Height);
}

USHORT BMP_GetDepth(BMP* bmp)
{
            if (bmp == NULL)
            {
                    BMP_LAST_ERROR_CODE = BMP_INVALID_ARGUMENT;
                    return -1;
            }

            BMP_LAST_ERROR_CODE = BMP_OK;

            return (bmp->Header.BitsPerPixel);
}

void BMP_GetPixelRGB(BMP* bmp, UINT x, UINT y, UCHAR* r, UCHAR* g, UCHAR* b)
{
            UCHAR*    pixel;
            UINT      bytes_per_row;
            UCHAR     bytes_per_pixel;

            if (bmp == NULL || x < 0 || x >= bmp->Header.Width || y < 0 || y >= bmp->Header.Height)
            {
                    BMP_LAST_ERROR_CODE = BMP_INVALID_ARGUMENT;
            }
            else
            {
                    BMP_LAST_ERROR_CODE = BMP_OK;

                    bytes_per_pixel = bmp->Header.BitsPerPixel >> 3;

                    bytes_per_row = bmp->Header.ImageDataSize / bmp->Header.Height;

                    pixel = bmp->Data + ((bmp->Header.Height - y - 1) * bytes_per_row + x * bytes_per_pixel);


                    if (bmp->Header.BitsPerPixel == 8)
                    {
                            pixel = bmp->Palette + *pixel * 4;
                    }

                    if (r)    *r = *(pixel + 2);
                    if (g)    *g = *(pixel + 1);
                    if (b)    *b = *(pixel + 0);
            }
```

```
}

void BMP_SetPixelRGB(BMP* bmp, UINT x, UINT y, UCHAR r, UCHAR g, UCHAR b)
{
	UCHAR*	pixel;
	UINT	bytes_per_row;
	UCHAR	bytes_per_pixel;

	if (bmp == NULL || x < 0 || x >= bmp->Header.Width || y < 0 || y >= bmp->Header.Height)
	{
		BMP_LAST_ERROR_CODE = BMP_INVALID_ARGUMENT;
	}

	else if (bmp->Header.BitsPerPixel != 24 && bmp->Header.BitsPerPixel != 32)
	{
		BMP_LAST_ERROR_CODE = BMP_TYPE_MISMATCH;
	}

	else
	{
		BMP_LAST_ERROR_CODE = BMP_OK;

		bytes_per_pixel = bmp->Header.BitsPerPixel >> 3;

		bytes_per_row = bmp->Header.ImageDataSize / bmp->Header.Height;

		pixel = bmp->Data + ((bmp->Header.Height - y - 1) * bytes_per_row + x * bytes_per_pixel);

		*(pixel + 2) = r;
		*(pixel + 1) = g;
		*(pixel + 0) = b;
	}
}

void BMP_GetPixelIndex(BMP* bmp, UINT x, UINT y, UCHAR* val)
{
	UCHAR*	pixel;
	UINT	bytes_per_row;

	if (bmp == NULL || x < 0 || x >= bmp->Header.Width || y < 0 || y >= bmp->Header.Height)
	{
		BMP_LAST_ERROR_CODE = BMP_INVALID_ARGUMENT;
	}

	else if (bmp->Header.BitsPerPixel != 8)
	{
		BMP_LAST_ERROR_CODE = BMP_TYPE_MISMATCH;
	}

	else
	{
		BMP_LAST_ERROR_CODE = BMP_OK;

		bytes_per_row = bmp->Header.ImageDataSize / bmp->Header.Height;

		pixel = bmp->Data + ((bmp->Header.Height - y - 1) * bytes_per_row + x);


		if (val) *val = *pixel;
	}
}

void BMP_SetPixelIndex(BMP* bmp, UINT x, UINT y, UCHAR val)
{
	UCHAR*	pixel;
	UINT	bytes_per_row;

	if (bmp == NULL || x < 0 || x >= bmp->Header.Width || y < 0 || y >= bmp->Header.Height)
	{
		BMP_LAST_ERROR_CODE = BMP_INVALID_ARGUMENT;
	}

	else if (bmp->Header.BitsPerPixel != 8)
	{
		BMP_LAST_ERROR_CODE = BMP_TYPE_MISMATCH;
	}

	else
	{
		BMP_LAST_ERROR_CODE = BMP_OK;

		bytes_per_row = bmp->Header.ImageDataSize / bmp->Header.Height;

		pixel = bmp->Data + ((bmp->Header.Height - y - 1) * bytes_per_row + x);

		*pixel = val;
	}
}
```

```
void BMP_GetPaletteColor(BMP* bmp, UCHAR index, UCHAR* r, UCHAR* g, UCHAR* b)
{
	if (bmp == NULL)
	{
		BMP_LAST_ERROR_CODE = BMP_INVALID_ARGUMENT;
	}

	else if (bmp->Header.BitsPerPixel != 8)
	{
		BMP_LAST_ERROR_CODE = BMP_TYPE_MISMATCH;
	}

	else
	{
		if (r)	*r = *(bmp->Palette + index * 4 + 2);
		if (g)	*g = *(bmp->Palette + index * 4 + 1);
		if (b)	*b = *(bmp->Palette + index * 4 + 0);

		BMP_LAST_ERROR_CODE = BMP_OK;
	}
}

void BMP_SetPaletteColor(BMP* bmp, UCHAR index, UCHAR r, UCHAR g, UCHAR b)
{
	if (bmp == NULL)
	{
		BMP_LAST_ERROR_CODE = BMP_INVALID_ARGUMENT;
	}

	else if (bmp->Header.BitsPerPixel != 8)
	{
		BMP_LAST_ERROR_CODE = BMP_TYPE_MISMATCH;
	}

	else
	{
		*(bmp->Palette + index * 4 + 2) = r;
		*(bmp->Palette + index * 4 + 1) = g;
		*(bmp->Palette + index * 4 + 0) = b;

		BMP_LAST_ERROR_CODE = BMP_OK;
	}
}

BMP_STATUS BMP_GetError()
{
	return BMP_LAST_ERROR_CODE;
}

const char* BMP_GetErrorDescription()
{
	if (BMP_LAST_ERROR_CODE > 0 && BMP_LAST_ERROR_CODE < BMP_ERROR_NUM)
	{
		return BMP_ERROR_STRING[BMP_LAST_ERROR_CODE];
	}
	else
	{
		return NULL;
	}
}

int	ReadHeader(BMP* bmp, FILE* f)
{
	if (bmp == NULL || f == NULL)
	{
		return BMP_INVALID_ARGUMENT;
	}

	if (!ReadUSHORT(&(bmp->Header.Magic), f))		return BMP_IO_ERROR;
	if (!ReadUINT(&(bmp->Header.FileSize), f))		return BMP_IO_ERROR;
	if (!ReadUSHORT(&(bmp->Header.Reserved1), f))		return BMP_IO_ERROR;
	if (!ReadUSHORT(&(bmp->Header.Reserved2), f))		return BMP_IO_ERROR;
	if (!ReadUINT(&(bmp->Header.DataOffset), f))		return BMP_IO_ERROR;
	if (!ReadUINT(&(bmp->Header.HeaderSize), f))		return BMP_IO_ERROR;
	if (!ReadUINT(&(bmp->Header.Width), f))		return BMP_IO_ERROR;
	if (!ReadUINT(&(bmp->Header.Height), f))		return BMP_IO_ERROR;
	if (!ReadUSHORT(&(bmp->Header.Planes), f))		return BMP_IO_ERROR;
	if (!ReadUSHORT(&(bmp->Header.BitsPerPixel), f))	return BMP_IO_ERROR;
	if (!ReadUINT(&(bmp->Header.CompressionType), f))	return BMP_IO_ERROR;
	if (!ReadUINT(&(bmp->Header.ImageDataSize), f))	return BMP_IO_ERROR;
	if (!ReadUINT(&(bmp->Header.HPixelsPerMeter), f))	return BMP_IO_ERROR;
	if (!ReadUINT(&(bmp->Header.VPixelsPerMeter), f))	return BMP_IO_ERROR;
	if (!ReadUINT(&(bmp->Header.ColorsUsed), f))		return BMP_IO_ERROR;
	if (!ReadUINT(&(bmp->Header.ColorsRequired), f))	return BMP_IO_ERROR;

	return BMP_OK;
}
```

```
int	WriteHeader(BMP* bmp, FILE* f)
{
	if (bmp == NULL || f == NULL)
	{
		return BMP_INVALID_ARGUMENT;
	}

	if (!WriteUSHORT(bmp->Header.Magic, f))			return BMP_IO_ERROR;
	if (!WriteUINT(bmp->Header.FileSize, f))		return BMP_IO_ERROR;
	if (!WriteUSHORT(bmp->Header.Reserved1, f))		return BMP_IO_ERROR;
	if (!WriteUSHORT(bmp->Header.Reserved2, f))		return BMP_IO_ERROR;
	if (!WriteUINT(bmp->Header.DataOffset, f))		return BMP_IO_ERROR;
	if (!WriteUINT(bmp->Header.HeaderSize, f))		return BMP_IO_ERROR;
	if (!WriteUINT(bmp->Header.Width, f))			return BMP_IO_ERROR;
	if (!WriteUINT(bmp->Header.Height, f))			return BMP_IO_ERROR;
	if (!WriteUSHORT(bmp->Header.Planes, f))		return BMP_IO_ERROR;
	if (!WriteUSHORT(bmp->Header.BitsPerPixel, f))		return BMP_IO_ERROR;
	if (!WriteUINT(bmp->Header.CompressionType, f))		return BMP_IO_ERROR;
	if (!WriteUINT(bmp->Header.ImageDataSize, f))		return BMP_IO_ERROR;
	if (!WriteUINT(bmp->Header.HPixelsPerMeter, f))		return BMP_IO_ERROR;
	if (!WriteUINT(bmp->Header.VPixelsPerMeter, f))		return BMP_IO_ERROR;
	if (!WriteUINT(bmp->Header.ColorsUsed, f))		return BMP_IO_ERROR;
	if (!WriteUINT(bmp->Header.ColorsRequired, f))		return BMP_IO_ERROR;

	return BMP_OK;
}

int	ReadUINT(UINT* x, FILE* f)
{
	UCHAR little[4];

	if (x == NULL || f == NULL)
	{
		return 0;
	}

	if (fread(little, 4, 1, f) != 1)
	{
		return 0;
	}

	*x = (little[3] << 24 | little[2] << 16 | little[1] << 8 | little[0]);

	return 1;
}

int	ReadUSHORT(USHORT *x, FILE* f)
{
	UCHAR little[2];

	if (x == NULL || f == NULL)
	{
		return 0;
	}

	if (fread(little, 2, 1, f) != 1)
	{
		return 0;
	}

	*x = (little[1] << 8 | little[0]);

	return 1;
}

int	WriteUINT(UINT x, FILE* f)
{
	UCHAR little[4];

	little[3] = (UCHAR)((x & 0xff000000) >> 24);
	little[2] = (UCHAR)((x & 0x00ff0000) >> 16);
	little[1] = (UCHAR)((x & 0x0000ff00) >> 8);
	little[0] = (UCHAR)((x & 0x000000ff) >> 0);

	return (f && fwrite(little, 4, 1, f) == 1);
}

int	WriteUSHORT(USHORT x, FILE* f)
{
	UCHAR little[2];

	little[1] = (UCHAR)((x & 0xff00) >> 8);
	little[0] = (UCHAR)((x & 0x00ff) >> 0);

	return (f && fwrite(little, 2, 1, f) == 1);
}
```

4. Спрощене завдання

1. Відповідно до варіанту завдання для заданих функцій часової складності L(n) визначити асимптотичні характеристики O(n).
2. Розташувати отримані функції в порядку зростання асимптотичних характеристик O(n).
3. Скласти програму (C/C++), яка ілюструє клас (поліноміальний чи експоненціальний) складності алгоритму, асимптотична складність якого була визначена в п2, як найбільша. При складанні програми передбачити можливість зміни значень **K** та **n.**

1	$2^n + n$	$n\sqrt{n}$	$n + \log_2 n$	$3n^2 + 2n^3$
2	$\log_2 n + n^2$	$n! + n^2$	$5n^3 + \sqrt{n}\log_2 n$	$4n^7 + 7n^4 + 7\sqrt{n}$
3	$15n^7 + 3n^5 + n^3$	$n + 2$	$n! + n^2 \log_2 n$	$15 + \log_2 n$
4	$\sqrt{n!} + 5n$	$\log_2(\log_2 n) + 5n^2$	$\log_2 n + n^2$	$35n + 53$
5	$\frac{n}{\log_2 n} + n$	$n^3 + 13n$	$6e^n + 3n^7$	$\log_2^2 n + 3n^2$
6	$2^{n^2} + 3^n$	$n^3 \log_2 n$	$\sqrt{n} + 5n^7$	$n^3 + n^2 + n$
7	$(n!)^2 + \log_2^2 n + n^3$	$\frac{n^2}{\log_2 n} + n$	$\left(\frac{1}{3}\right)^n + n^2$	$\sqrt{n}$
8	$7n^5 + 15n^3 + n^2$	$\frac{n!}{\log_2 n}$	$17n + 5$	$\log_2(\log_2 n)$
9	$(\log_2 n)^{n+1} + 5n$	$5n^3 + \log_2 n + 121$	$n^2 \log_2 n + n^2$	$\frac{n}{2} + 2$
10	$\frac{n^3}{3} + \frac{n^2}{2} + n + 1$	$n^5 + 2n^3 + 8$	$e^n + n^{12}$	$\sqrt{n}\log_2 n$

11	$2^{(n-1)^2} + 2^{2n} + n^3$	$\frac{3n^2}{2} + \frac{n^3}{2} + n$	$n^5\sqrt{n}$	$\frac{\log_2 n^2}{n^2} + \sqrt{n}$
12	$\log_2(\log_2 n^2) + n! + 5$	$7n^7 + 5n^5 + 3$	$\frac{n^8}{\sqrt{n}} + \frac{n^6}{\sqrt{n}} + \log_2 n$	$3\sqrt{n} + 3$
13	$17n^2 + 2n^{17} + 34n$	$n^2 \log_2(\sqrt{n-1})$	$\frac{(n-1)^2}{\sqrt{n}-1} + (n-1)^3$	$2^{(n+2)} + (n+2)^2$
14	$(\log_2 n)^{n+1} + n^7 + 7n^2$	$(2n^2+3n^3)^2$	$6n^5 + 5n^4$	$(\log_2(n+2)+n)^3$
15	$e^n + n^{12}$	$\log_2 n^3 + 5$	$n^7 + n^5 + 3$	$(n+3)^5 + (n^5+5)^2 + 15$
16	$(n+3)^5 + (n^2+5)^3 + (n^3+7)^2$	$n^2 \log_2(n+1) + n^3$	$(n+2)! + (n+2)^2$	$\log_2(\log_2 n)$
17	$n \log_2 n + \sqrt{n-1}$	$5 + 3n$	$(\log_2 n)^n - 1$	$(\frac{n!}{(n-1)!})^3 + 3n$
18	$3n^5 + \sqrt{n-1} + 1$	$n! + (\log_2(n+1))^3$	$n+1$	$(n-1)^2$
19	$(\log_2 n)^{n+1} + n^{n+1} + n$	$(n^2+2)^3$	$\sqrt{n}\log_2 n$	$\frac{n^6+1}{\sqrt{n}} + n^5 + n^3$
20	$5n^7 + 7n^5 + 3$	$2^{(n+1)} + (n+1)^2 + n + 1$	$\frac{\log_2 n}{n}$	$(n^3+3)^3$
21				

	$(n^3+n^2+n)^2+(\log_2 n)^2$	$\frac{5n^7+7n^5}{n^2}+1$	$2^{(n+1)}+(n+1)^2+n+1$	$\sqrt{n}+2n$
22	$7n^5+\sqrt{n}$	$\frac{n}{\log_2 n}$	$\frac{n!}{n-1}+n^3$	$(n^2+2)^3$
23	$\log_2(\log_2 n^2)+n^2$	$n2^{n-1}$	$\frac{(n+1)^3}{n}+n^3$	$3\sqrt{n}+3n$
24	$\left(\frac{1}{2}\right)^n+n^3+2$	$n^3\log_2 n+n^3$	$\frac{n!}{n+1}+(n+1)^3$	$\sqrt{n}+1$
25	$7n+1$	$n^5\sqrt{n}$	$(n^3+2)^2+n^5$	$e^{n+1}+1$

**** Для отримання 50% балів за лабораторну роботу можна використати наявний програмний код з лістингу 3.1. Для отримання 100% балів за лабораторну роботу потрібно написати власний код.***

5. Завдання базового рівня

1. Для алгоритму реалізованого відповідно до завдань базового рівня з лабораторної роботи №1 сформувати функцію часової складності L(n) та визначити асимптотичну характеристику O(n).
2. Скласти програму (C/C++), яка ілюструє клас (поліноміальний чи експоненціальний) складності алгоритму. При складанні програми передбачити можливість зміни значень **K** та **n.**

6. Зміст звіту

- Титульний лист;
- Завдання;
- Алгоритм рішення завдання;
- Код програми;
- Екранна форма з результатами роботи програми;
- Висновки.

ЛАБОРАТОРНА РОБОТА №3

Назва роботи: використання потокового графу алгоритму при проектуванні паралельних обчислень.

Мета роботи: ознайомитися з використанням потокового графу алгоритму при паралельному програмуванні.

*(детальніша інформація про **паралельні обчислення**, **паралельне програмування** та **потоковий граф алгоритму** в матеріалах лекційних занять)*

1.1. Паралельні обчислення.

Паралельні обчислення — це форма обчислень, в яких кілька дій проводяться одночасно. Є кілька різних рівнів паралельних обчислень: ***бітовий***, ***інструкцій***, ***даних*** та ***паралелізм задач***. Паралельні обчислення застосовуються вже протягом багатьох років, в основному в високопродуктивних обчисленнях, але зацікавлення ними зросло тільки недавно, через фізичні обмеження зростання частоти. Оскільки споживана потужність (і відповідно виділення тепла) комп'ютерами стало проблемою в останні роки, паралельне програмування стає домінуючою парадигмою в комп'ютерній архітектурі, в основному в формі багатоядерних процесорів.

Паралельні комп'ютери системи можуть бути грубо класифіковані згідно з рівнем, на якому апаратне забезпечення підтримує паралелізм: багатоядерні процесори; багатопроцесорні комп'ютери — комп'ютери, що мають багато обчислювальних елементів в межах одної машини, також сюди відносимо кластери; MPP та GRID — системи що використовують багато комп'ютерів для роботи над одним завданням. Також Іноді, поряд з цими традиційними комп’ютерними системами, використовуються спеціалізовані паралельні архітектури для прискорення особливих задач.

Програми для паралельних комп'ютерів писати значно складніше, ніж для послідовних, бо паралелізм додає кілька нових класів потенційних помилок, серед яких найпоширенішою є стан гонитви(конкуренції обчислювальних процесів). Комунікація, та синхронізація процесів зазвичай одна з найбільших перешкод для досягнення хорошої продуктивності паралельних програм.

1.2. Паралельні обчислення рівня машинних інструкцій.

Комп'ютерна програма, по суті, є потоком інструкцій, які виконуються процесором. Іноді ці інструкції можна перевпорядкувати, та об'єднати в групи, які потім виконувати паралельно, без зміни результату роботи програми, що відомо як паралелізм на рівні інструкцій. Такий підхід до збільшення продуктивності обчислень переважав з середини 80-тих до середини 90-тих. Сучасні процесори мають багатоетапні конвеєри команд. Кожен етап конвеєра відповідає іншій дії, що виконує процесор. Процесор що має конвеєр з N-ступенями, може одночасно обробляти N інструкцій, кожну на іншій стадії обробки. Класичним прикладом процесора з конвеєром є процесор архітектури RISC, що має п'ять етапів: завантаження інструкції, декодування, виконання, доступ до пам'яті, та запис результату. Процесор Pentium 4 має конвеєр з 35 етапами. На додачу до паралелізму на рівні інструкцій деякі процесори можуть виконувати більш ніж одну інструкцію за раз. Вони відомі як суперскалярні процесори. Інструкції групуються разом, якщо між ними не існує залежності даних. Щоб реалізувати паралелізм на рівні інструкцій використовують алгоритми Scoreboarding та Tomasulo algorithm (який аналогічний до попереднього, проте використовує перейменування регістрів).

1.3. Використання потокового графу алгоритму для паралельного програмування.

Потоковий граф алгоритму описує паралельні обчислення, в яких на заданому фрагменті виконання відсутні залежності керування за даними. Кожна вершина графу на одному ярусі виконується одночасно. Приміром на *рис.1.1* показано виконання швидкого перетворення Фур'є для 8-ми відліків.

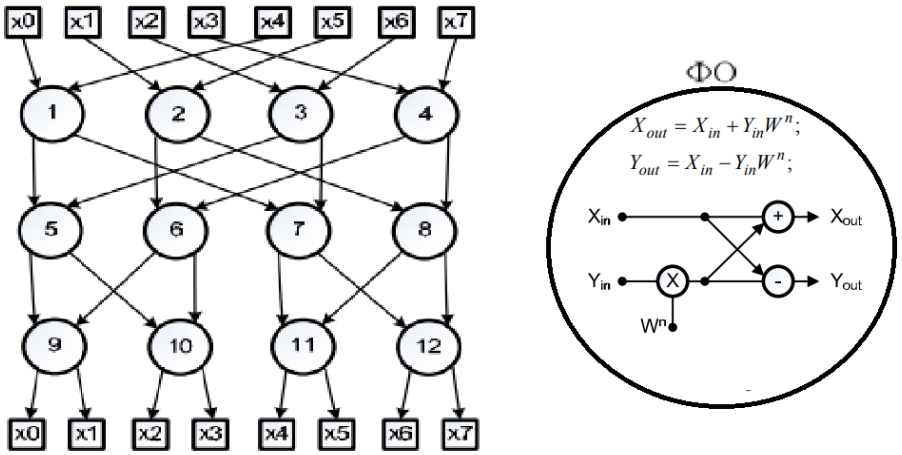
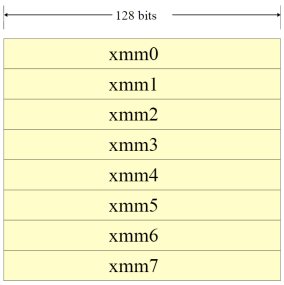


Рис.1.1. Приклад потокового графу алгоритму

1.4. Набір інструкцій SSE2.

SSE (англ. Streaming SIMD Extensions, потокове SIMD-розширення процесора) — це SIMD (англ. Single Instruction, Multiple Data, Одна інструкція — багато даних) набір інструкцій, розроблених Intel, і вперше представлених у процесорах серії Pentium III як відповідь на аналогічний набір інструкцій 3DNow! від AMD, який був представлений роком раніше. Цікаво, що початкова назва цих інструкцій була KIN, що розшифровувалася як Katmai New Instructions (Katmai — назва першої версії ядра процесора Pentium III).



SSE2 використовує вісім 128-бітних регістри (xmm0 до xmm7), що увійшли до архітектури x86 з вводом розширення SSE, кожний з яких трактується як послідовність 2 значень з плаваючою комою подвійної точності. SSE2 включає в себе набір інструкцій, який виконує дії зі скалярними і упакованими типами даних. Також SSE2 містить інструкції для потокової обробки даних з фіксованою комою у тих же 128-бітних xmm регістрах.

Лістинг 1.1

```
float a[4] = { 300.0, 4.0, 4.0, 12.0 };
float b[4] = {   1.5, 2.5, 3.5,  4.5 };
_asm {
movups xmm0, a   ; // помістити 4 змінні з рухомою комою із a в регістр xmm0
movups xmm1, b   ; // помістити 4 змінні з рухомою комою із b в регістр xmm1
mulps xmm1, xmm0 ; // перемножити пакети: xmm1=xmm1*xmm0
                 ; // xmm10 = xmm10*xmm00
                 ; // xmm11 = xmm11*xmm01
                 ; // xmm12 = xmm12*xmm02
                 ; // xmm13 = xmm13*xmm03
movups a, xmm1   ; // вивантажити результати із регістра xmm1 в a
};
```

1.5. Робата з SSE2 на рівні мов програмування високого рівня.

Для роботи з SSE, SSE2, SSE3, SSE4 та AVX в C/C++ зручно використовувати обгортки над цими інструкціями процесора. Їх реалізовують ***intrinsics-функції***, вичерпну інформацію по яких можна отримати за лінком: https://software.intel.com/sites/landingpage/IntrinsicsGuide/ .

2. Приклад.

Математичні вирази зручно подавати у формі потокового графу алгоритму, оскільки в них обо відсутні залежності керування за даними, або їх легко трансформувати в додаткові обчислення для вибору результату. Розглянемо потоковий граф алгоритму розв'язання квадратного рівняння *(рис. 2.1.)* для двократного розпаралелення при одночасному виконанні тільки однотипних інструкцій(відповідно до концепції SSE2).

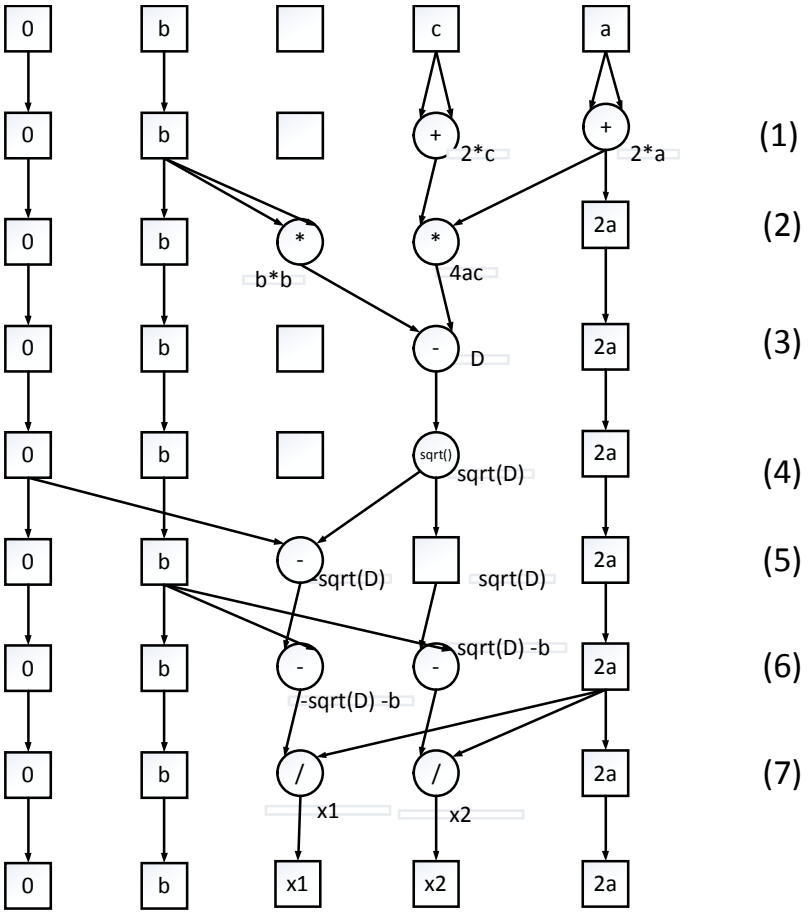


Рис. 2.1. ПГА для розв'язання квадратного рівняння

3. Приклад програми.

Лістинг 3.1

```
// don't forget to use compilation key for Linux: -lm

//-fno-tree-vectorize
//gcc -O3 -no-tree-vectorize
//gcc -O3 -ftree-vectorizer-verbose=6 -msse4.1 -ffast-math
//__attribute__((optimize("no-tree-vectorize")))
//extern "C" __attribute__ ((optimize("no-tree-vectorize")))


#include <stdio.h>
#include <stdlib.h>
//#include <x86intrin.h> // Linux
#include <intrin.h> // Windows
#include <math.h>
#include <time.h>

#define A  0.33333333
#define B  0.
#define C -3.

#pragma GCC push_options
#pragma GCC optimize ("no-unroll-loops")

#define REPEAT_COUNT 1000000
#define REPEATOR(count, code) \
for (unsigned int indexIteration = (count); indexIteration--;){ code; }
#define TWO_VALUES_SELECTOR(variable, firstValue, secondValue) \
	(variable) = indexIteration % 2 ? (firstValue) : (secondValue);

double getCurrentTime(){
	clock_t time = clock();
	if (time != (clock_t)-1) {
		return ((double)time / (double)CLOCKS_PER_SEC);
	}
	return 0.; // else
}

#pragma GCC push_options
#pragma GCC target ("no-sse2")
//__attribute__((__target__("no-sse2")))
void run_native(double * const dArr){
	double * const dAC = dArr;
	double * const dA = &dAC[0];
	double * const dC = &dAC[1];
	double * const dB = &dArr[2];
	double * const dResult = &dArr[4];
	double * const dX1 = &dResult[1];
	double * const dX2 = &dResult[0];

	REPEATOR(REPEAT_COUNT,
		TWO_VALUES_SELECTOR(*dA, 4., A);
		TWO_VALUES_SELECTOR(*dB, 3., B);
		TWO_VALUES_SELECTOR(*dC, 1., C);
		double vD = sqrt((*dB)*(*dB) - 4.*(*dA)*(*dC));
		(*dX1) = (-(*dB) + vD) / (2.*(*dA));
		(*dX2) = (-(*dB) - vD) / (2.*(*dA));
	)
}
#pragma GCC pop_options

void run_SSE2(double * const dArr){
	double * const dAC = dArr;
```

```
	double * const dA = &dAC[0];
	double * const dC = &dAC[1];
	double * const dB = &dArr[2];
	double * const dResult = &dArr[4];
	double * const dX1 = &dResult[1];
	double * const dX2 = &dResult[0];

	__m128d r__zero_zero, r__c_a, r__uORb_b, r__2cORbOR2a_2a,
		r__zero_bb, r__sqrtDiscriminant_zero, r_result;

	r__zero_zero = _mm_set_pd(0., 0.); // init

	REPEATOR(REPEAT_COUNT,
		TWO_VALUES_SELECTOR(*dA, 4., A);
		TWO_VALUES_SELECTOR(*dB, 3., B);
		TWO_VALUES_SELECTOR(*dC, 1., C);
		r__c_a = _mm_load_pd(dAC);
		// r__uORb_b = _mm_load_pd1(dB);
		r__uORb_b = _mm_load1_pd(dB);
		// b b
		r__uORb_b = _mm_unpacklo_pd(r__uORb_b, r__uORb_b);
		// (etap 1)
		r__2cORbOR2a_2a = _mm_add_pd(r__c_a, r__c_a);
		// b 2c
		r_result = _mm_unpackhi_pd(r__2cORbOR2a_2a, r__uORb_b);
		// b 2a
		r__2cORbOR2a_2a = _mm_unpacklo_pd(r__2cORbOR2a_2a, r__uORb_b);
		// bb 4ac (etap 2)
		r_result = _mm_mul_pd(r_result, r__2cORbOR2a_2a);
		r__zero_bb = _mm_unpackhi_pd(r_result, r__zero_zero);
		// zero Discriminant (etap 3)
		r_result = _mm_sub_sd(r__zero_bb, r_result);
		// zero sqrtDiscriminant (etap 4)
		r_result = _mm_sqrt_sd(r_result, r_result);
		r__sqrtDiscriminant_zero = _mm_shuffle_pd(r_result, r_result, 1);
		// sqrtDiscriminant -sqrtDiscriminant (etap 5)
		r_result = _mm_sub_sd(r__sqrtDiscriminant_zero, r_result);
		// (etap 6)
		r_result = _mm_sub_pd(r_result, r__uORb_b);
		// 2a 2a
		r__2cORbOR2a_2a = _mm_unpacklo_pd(r__2cORbOR2a_2a, r__2cORbOR2a_2a);
		// (etap 7)
		r_result = _mm_div_pd(r_result, r__2cORbOR2a_2a);
		_mm_store_pd(dResult, r_result);
	)
}

void printResult(char * const title, double * const dArr, unsigned int runTime){
	double * const dAC = dArr;
	double * const dA = &dAC[0];
	double * const dC = &dAC[1];
	double * const dB = &dArr[2];
	double * const dResult = &dArr[4];
	double * const dX1 = &dResult[1];
	double * const dX2 = &dResult[0];

	printf("%s:\r\n", title);
	printf("%fx^2 + %fx + %f = 0;\r\n", *dA, *dB, *dC);
	printf("x1 = %1.0f; x2 = %1.0f;\r\n", *dX1, *dX2);
	printf("run time: %dns\r\n\r\n", runTime);
}

int main() {
	double * const dArr = (double *)_mm_malloc(6 * sizeof(double), 16);
```

```
        double * const dAC = dArr;
        double * const dA = &dAC[0];
        double * const dC = &dAC[1];
        double * const dB = &dArr[2];
        double * const dResult = &dArr[4];
        double * const dX1 = &dResult[1];
        double * const dX2 = &dResult[0];

        double startTime, endTime;

        // native (only x86, if auto vectorization by compiler is off)
        startTime = getCurrentTime();
        run_native(dArr);
        endTime = getCurrentTime();
        printResult("x86",
               dArr,
               (unsigned int)((endTime - startTime) * (1000000000 / REPEAT_COUNT)));

        // SSE2
        startTime = getCurrentTime();
        run_SSE2(dArr);
        endTime = getCurrentTime();
        printResult("SSE2",
               dArr,
               (unsigned int)((endTime - startTime) * (1000000000 / REPEAT_COUNT)));

        _mm_free(dArr);

        printf("Press any key to continue . . .");
        getchar();
        return 0;
}

#pragma GCC pop_options
```

4. Спрощене завдання.

Скласти програму (C/C++), яка дозволяє провести порівняння двох реалізацій розв'язання квадратного рівняння(звичайний послідовний код та код на основі інструкцій SSE2, що відображає потоковий граф алгоритму) за характеристикою часової складності для таких вхідних даних:

n	**Вхідні дані**		
	c	**b**	**a**
1	111	112	107
2	145	146	141
3	165	166	161
4	185	186	181
5	205	206	201
6	225	226	221
7	245	246	241
8	265	266	261
9	285	286	281
10	305	306	301
11	325	326	321
12	345	346	341
13	365	366	361
14	385	386	381
15	405	406	401
16	425	426	421

17	445	446	441
18	465	466	461
19	485	486	481
20	505	506	501
21	525	526	521
22	545	546	541
23	565	566	561
24	585	586	581
25	605	606	601
26	625	626	621
27	645	646	641
28	665	666	661
29	685	686	681
30	705	706,	701
n – порядковий номер у журналі			

**** Для отримання 50% балів за лабораторну роботу можна використати наявний програмний код з лістингу 3.1. Для отримання 100% балів за лабораторну роботу потрібно написати власний код.***

5. Завдання базового рівня.

Скласти програму (C/C++), яка дозволяє провести порівняння двох реалізацій обчислення виразу(звичайний послідовний код та код на основі інструкцій SSE2, що відображає потоковий граф алгоритму) за характеристикою часової складності для вхідних даних, що вводяться під час роботи програми.

Варіант	**Вираз**
1	y = (a + b) + (c + d) + (e + f) + (g + h)
2	y = (a + b) - (c + d) + (e + f) - (g + h)
3	y = (a + b) * (c + d) + (e + f) * (g + h)
4	y = (a + b) / (c + d) + (e + f) / (g + h)
5	y = (a + b) + (c + d) + (e + f) + (g + h)
6	y = (a + b) - (c + d) + (e + f) - (g + h)
7	y = (a + b) * (c + d) + (e + f) * (g + h)
8	y = (a + b) / (c + d) + (e + f) / (g + h)
9	y = (a + b) + (c + d) + (e - f) + (g - h)
10	y = (a + b) - (c + d) + (e - f) - (g - h)
11	y = (a + b) * (c + d) + (e - f) * (g - h)
12	y = (a + b) / (c + d) + (e - f) / (g - h)
13	y = (a + b) + (c + d) + (e - f) + (g - h)
14	y = (a + b) - (c + d) + (e - f) - (g - h)
15	y = (a + b) * (c + d) + (e - f) * (g - h)
16	y = (a + b) / (c + d) + (e - f) / (g - h)
17	y = (a - b) + (c - d) + (e + f) + (g + h)
18	y = (a - b) - (c - d) + (e + f) - (g + h)
19	y = (a - b) * (c - d) + (e + f) * (g + h)
20	y = (a - b) / (c - d) + (e + f) / (g + h)
21	y = (a - b) + (c - d) + (e + f) + (g + h)
22	y = (a - b) - (c - d) + (e + f) - (g + h)
23	y = (a - b) * (c - d) + (e + f) * (g + h)
24	y = (a - b) / (c - d) + (e + f) / (g + h)
25	y = (a - b) + (c - d) + (e - f) + (g - h)
26	y = (a - b) - (c - d) + (e - f) - (g - h)
27	y = (a - b) * (c - d) + (e - f) * (g - h)
28	y = (a - b) / (c - d) + (e - f) / (g - h)
29	y = (a - b) + (c - d) + (e - f) + (g - h)
30	y = (a - b) - (c - d) + (e - f) - (g + h)
n – порядковий номер у журналі	

6. Зміст звіту

- Титульний лист;
- Завдання;
- Алгоритм рішення завдання;
- Код програми;
- Екранна форма з результатами роботи програми;
- Висновки.