

Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики

Кафедра програмування

Лабораторна робота №1
Визначення часової ефективності алгоритму
з курсу “Теорія алгоритмів”

Виконав:
студент групи ПМІ-12
Козій Дем'ян Назарович

Львів – 2025

Метод 1: Послідовне оновлення

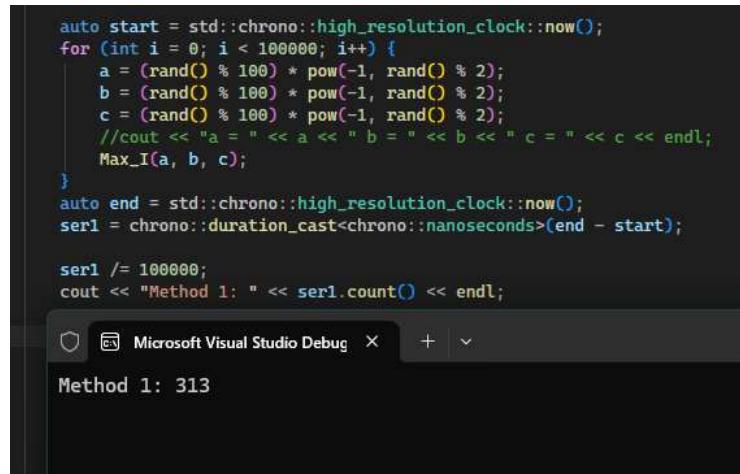
Алгоритм:

1. Приймаємо перше число (a) за поточний максимум (max)
2. Порівнюємо поточний max з другим числом (b)
 - a. Якщо b > max, оновлюємо max = b
3. Порівнюємо поточний max з третім числом (c)
 - a. Якщо c > max, оновлюємо max = c
4. Повертаємо max

Реалізація:

```
int Max_I(int a, int b, int c) {  
    int max;  
    max = a;  
    if (max < b) max = b;  
    if (max < c) max = c;  
    return(max);  
}
```

Приклад роботи:



```
auto start = std::chrono::high_resolution_clock::now();  
for (int i = 0; i < 100000; i++) {  
    a = (rand() % 100) * pow(-1, rand() % 2);  
    b = (rand() % 100) * pow(-1, rand() % 2);  
    c = (rand() % 100) * pow(-1, rand() % 2);  
    //cout << "a = " << a << " b = " << b << " c = " << c << endl;  
    Max_I(a, b, c);  
}  
auto end = std::chrono::high_resolution_clock::now();  
ser1 = chrono::duration_cast<chrono::nanoseconds>(end - start);  
  
ser1 /= 100000;  
cout << "Method 1: " << ser1.count() << endl;
```

Method 1: 313

Метод 2: Проміжний максимум

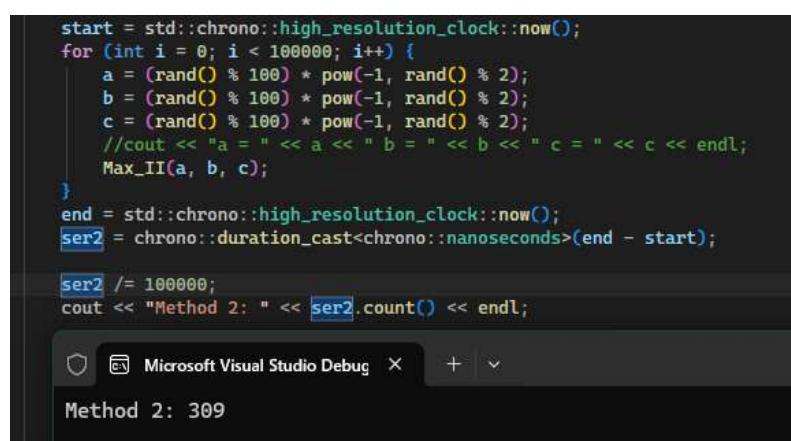
Алгоритм:

1. Порівнюємо перші два числа (a і b)
 - a. Якщо a > b, проміжний результат max = a
 - b. Інакше max = b
2. Порівнюємо отриманий проміжний max з третім числом (c)
 - a. Якщо max < c, оновлюємо max = c
3. Повертаємо max

Реалізація:

```
int Max_II(int a, int b, int c) {
    int max;
    if (a > b) max = a;
    else max = b;
    if (max < c) max = c;
    return(max);
}
```

Приклад роботи:



```
start = std::chrono::high_resolution_clock::now();
for (int i = 0; i < 100000; i++) {
    a = (rand() % 100) * pow(-1, rand() % 2);
    b = (rand() % 100) * pow(-1, rand() % 2);
    c = (rand() % 100) * pow(-1, rand() % 2);
    //cout << "a = " << a << " b = " << b << " c = " << c << endl;
    Max_II(a, b, c);
}
end = std::chrono::high_resolution_clock::now();
ser2 = chrono::duration_cast<chrono::nanoseconds>(end - start);

ser2 /= 100000;
cout << "Method 2: " << ser2.count() << endl;
```

Method 2: 309

Метод 3: Дерево рішень

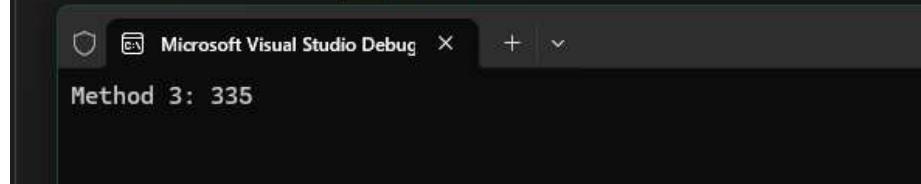
Алгоритм:

1. Порівнюємо a і b
2. Якщо $a > b$:
 - a. Порівнюємо переможця (a) з c
 - i. Якщо $a > c$, то максимум — a . Інакше — c
3. Якщо $a \leq b$:
 - a. Порівнюємо переможця (b) з c
 - i. Якщо $b > c$, то максимум — b . Інакше — c

Реалізація:

```
int Max_III(int a, int b, int c) {
    int max;
    if (a > b) {
        if (a > c) max = a;
        else max = c;
    }
    else {
        if (b > c) max = b;
        else max = c;
    }
    return(max);
}
```

Приклад роботи:



```
start = std::chrono::high_resolution_clock::now();
for (int i = 0; i < 100000; i++) {
    a = (rand() % 100) * pow(-1, rand() % 2);
    b = (rand() % 100) * pow(-1, rand() % 2);
    c = (rand() % 100) * pow(-1, rand() % 2);
    //cout << "a = " << a << " b = " << b << " c = " << c << endl;
    Max_III(a, b, c);
}
end = std::chrono::high_resolution_clock::now();
ser3 = chrono::duration_cast<chrono::nanoseconds>(end - start);

ser3 /= 100000;
cout << "Method 3: " << ser3.count() << endl;
```

Результати тестування продуктивності

Для оцінки ефективності було проведено 1 000 000 ітерацій для кожного методу з генерацією випадкових чисел. Час виконання вимірювався у наносекундах.



```
Method 1: 316 Method 2: 306 Method 3: 305
```

Оскільки тіло циклу включає генерацію випадкових чисел, яка є значно важчою операцією за просте порівняння цілих чисел, загальний час виконання для всіх трьох методів є майже однаковим. Різниця в часі знаходитьться в межах статистичної похибки. Це доводить, що всі три методи є однаково ефективні.

Посилання на репозиторій з програмою:

<https://github.com/DemkaO/Univer/tree/master/algo/Homework/HM1>