НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

## Факультет інформатики та обчислювальної техніки

## Кафедра інформатики та програмної інженерії

## Звіт по лабораторній роботі №4

## «Моделювання систем»

## «Оцінка точності та складності алгоритму імітації»

Студент: Галько М.В. і

Група: ІП-01 і

## Київ, 2023

# ЗМІСТ

# 

[**Завдання**](#_f8k273fnsa66) **2**

[**1. Побудова послідовної моделі**](#_13xnwxeuc3ts) **2**

[**2. Результати експериментів послідовної моделі**](#_hziafn1ymrr) **2**

[2.1 Програма](#_hymnksro12gv) 2

[2.2 Результати](#_s816sp3yra5p) 3

[**3. Теоретична оцінка складності**](#_jjz1wudl8xnf) **4**

[**4. Приклад на розгалуженої моделі**](#_kym0v9sywrba) **4**

[4.1 Код розгалуженої моделі](#_lcjo403tfxpb) 4

[4.1 Результати](#_v0j45uyu4yvi) 4

[**Висновок**](#_9csizy3yplbd) **6**

# 

# 

# Завдання

1. Розробити модель масового обслуговування, яка складається з N систем масового обслуговування. Число N є параметром моделі. Кількість подій в моделі оцінюється числом N+1. **20 балів.**
2. Виконати експериментальну оцінку складності алгоритму імітації мережі масового обслуговування. Для цього виконайте серію експериментів, в якій спостерігається збільшення часу обчислення алгоритму імітації при збільшенні кількості подій в моделі. **40 балів.**
3. Виконати теоретичну оцінку складності побудованого алгоритму імітації. **30 балів.**
4. Повторіть експеримент при зміні структури мережі масового обслуговування. **10 балів**

# Побудова послідовної моделі

Розробимо статичний клас ModelHelper із методом CreateSequential. В нього необхідно передавати кількість процесів яка має бути у фінальній моделі:

public static Model CreateSequential(int processCount)

{

List<Element> elements = new() { new Create() };

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

{

elements.Add(new Process());

elements[^2].NextElementsContainer = new NextElementContainer(elements[^1]);

}

return new Model(elements);

}

# Результати експериментів послідовної моделі

## 2.1 Програма

Для оцінки складності алгоритму будемо використовувати готовий інструмент BenchmarkDotNet. Виконаємо тести продуктивності для моделі з 50-ти послідовних процесів із часом симуляції від 1000 до 10000 із кроком 1000. Отже виконаємо 10 тестів. Для цього сформуємо наступний код:

BenchmarkRunner.*Run*<Lab4Benchmark>();

public class Lab4Benchmark

{

[Params(1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000)]

public int ModelTime;

private Model? \_sequentialModel;

[IterationSetup]

public void CreateModel()

{

\_sequentialModel = ModelHelper.*CreateSequential*(50);

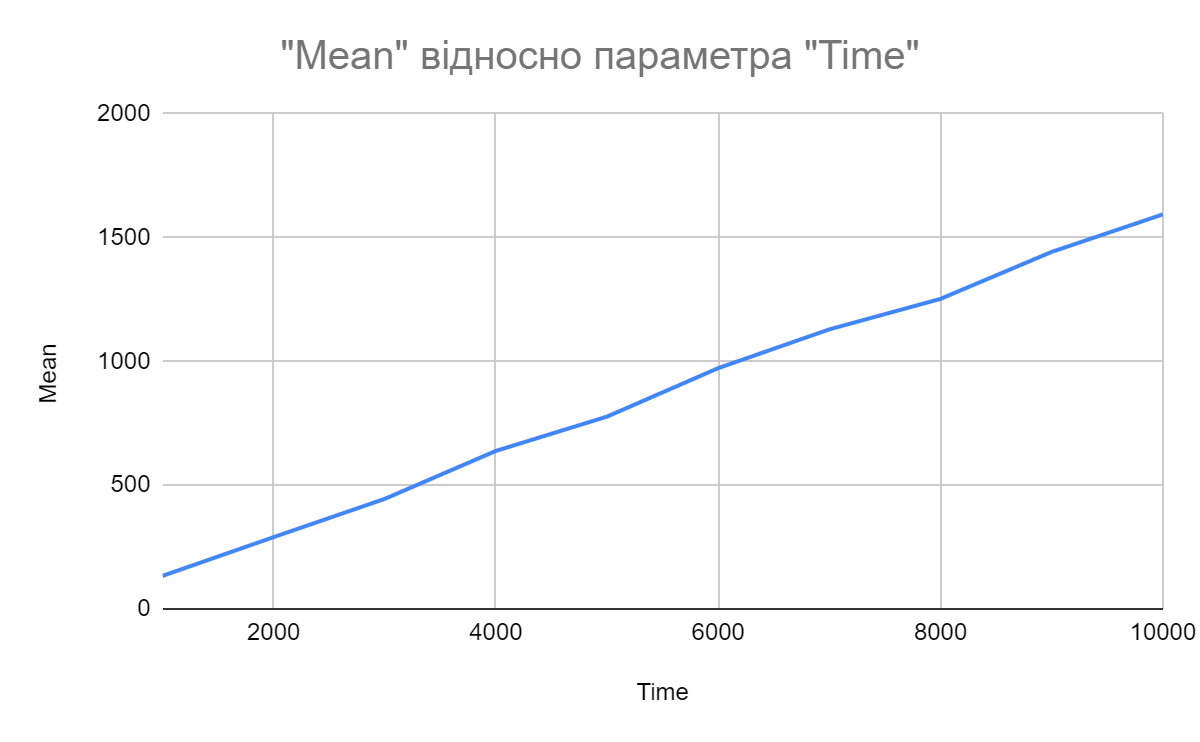
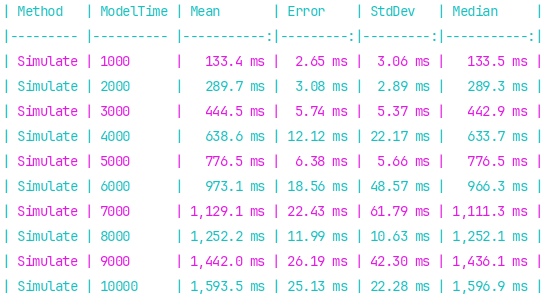
}

[Benchmark]

public void SimulateSequential() => \_sequentialModel.Simulate(time: ModelTime, printResult: false);

}

## 2.2 Результати



# Теоретична оцінка складності

Теоретична оцінка складності виконується за формулою О(v\*timeMod\*k), де v – інтенсивність подій (delay для елемента Create = 1), timeMode – час моделювання (ModelTime), k – середня кількість елементарних операцій для обробки 1-єї події (кількість елементів Process = 50).

Отже, наша формула змінює вигляд на O(1 \* ModelTime \* N) = O(N), що є лінійною залежністю. Бачимо, що результативний графік відповідає цій умові.

# Приклад на розгалуженої моделі

## 4.1 Код розгалуженої моделі

public static Model CreateBranching2Levels(int processCount, int nextProcessCount)

{

List<Element> allElements = new() { new Create() };

NextElementsContainerByProbability createContainer = new();

allElements[^1].NextElementsContainer = createContainer;

for (int n = 0; n < processCount; n++) {

allElements.Add(new Process());

createContainer.AddNextElement(allElements[^1], 1.0 / processCount);

NextElementsContainerByProbability processContainer = new();

allElements[^1].NextElementsContainer = processContainer;

for (int m = 0; m < nextProcessCount; m++){

allElements.Add(new Process(name: " PROCESS\_" + (n+1) + "\_" + m));

processContainer.AddNextElement(allElements[^1], 1.0 / nextProcessCount);

}

}

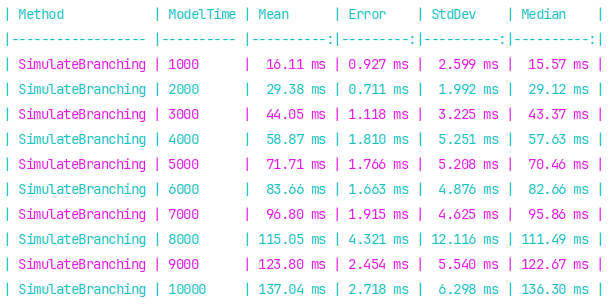
return new Model(allElements);

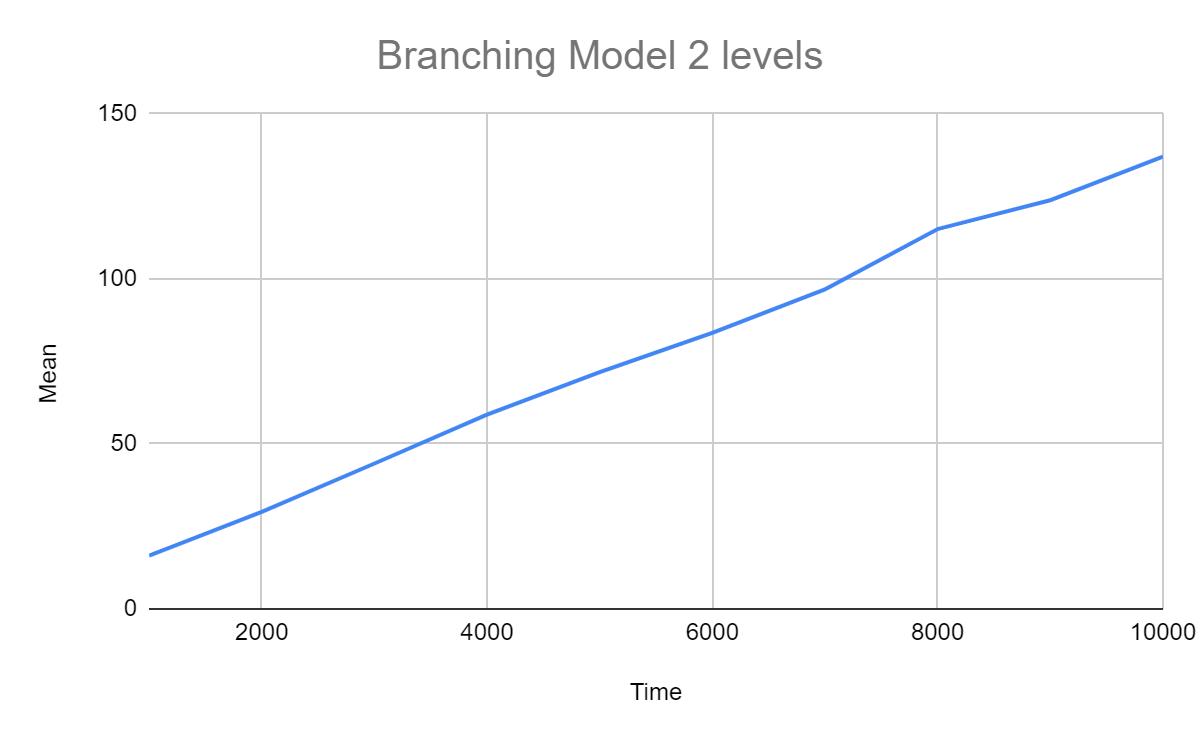
}

## 

## 4.1 Результати

Виконавши ідентичне застосування інструмента BenchmarkDotNet для розгалуженої моделі (структура: 1 Create, 10 Process та від кожного Process йде 5 Process) отримуємо наступний результат:





# 

# Висновок

В ході виконання лабораторної роботи було створено статичний клас ModelHelper який реалізує 2 методи CreateSequential та CreateBranch2Levels, які повертають послідовну та розгалужену глубини 2 моделі.

За допомогою інструмента BenchmarkDotNet були протестовані 2 моделі із різним часом симуляції від 1000 до 10000 із кроком 1000 та підрахована їх теоретична оцінка складності. Проаналізувавши функцію оцінки складності і її параметрів отримали лінійну закономірність О(N), де N – кількість елементів типу Process. Перевівши табличні результати BenchmarkDotNet у графічний вигляд, продемонстрували відношення Mean до ModelTime. З результатів графіків видна лінійна закономірність для обох моделей.

Для побудови моделей були використані авторські бібліотеки сформовані під час виконання 1-3 лабораторних робіт:

[Код бібліотек](https://github.com/MilaHalko/S7_SystemsModelingBasics/tree/main/libraries/MassServiceModeling/MassServiceModeling)