Петрівська Маріанна Практична №1

Опис постановки задачі та експерименту.

Метою дослідження є порівняння 3-ох алгоритмів сортування: insertion sort, selection sort та shell sort. Для проведення більш наочного аналізу взято як вибірку масиви, довжини яких відрізняються удвічі, починаючи з 2 у 7 степені і закінчуючи 2 у 17. Ефективність алгоритму вимірюється величинами часу та кількістю операцій порівняння.

Усього для кожного з алгоритмів сортування поставлено 4 експерименти:

1)Для 10 випадковим чином згенерованих масивів кожної довжини знаходимо середні величини.

2)Для масивів різних довжин, відсортованих у порядку зростання.

3)Для масивів різних довжин, відсортованих у порядку спадання.

4)Для 10 випадковим чином згенерованих масивів, які складаються з елементів множини {1, 2, 3} для кожної довжини знаходимо середні величини.

Примітка. Для більшої точності час у цих експериментах вимірюється у наносекундах.

Специфікація комп’ютера

Операційна система – Microsoft Windows 10 Home Single Language.

Процесор – Intel® Core™ i5-8250U CPU

Кількість ядер – 4

Кількість логічних процесорів – 8

Тактова частота – 1.60 GHz, 1800 Mhz

Пам’ять – 8.0 GB

Алгоритми сортування

1)Insertion sort

```
def insertion_sort(arr):
    num_of_comparisons = 0
    array = arr.copy()
    for j in range(1, len(arr)):
        key = arr[j]
        i = j - 1
        num_of_comparisons += 1
        while i >= 0 and arr[i] > key:
            arr[i + 1] = arr[i]
            i -= 1
            num_of_comparisons += 1
        arr[i + 1] = key
    return arr, num_of_comparisons
```

2)Selection sort

```
def selection_sort(arr):
    num_of_comparisons = 0
    array = arr.copy()
    for i in range(len(array)):
        min_idx = i
        for j in range(i + 1, len(array)):
            num_of_comparisons+=1
            if array[min_idx] > array[j]:
                min_idx = j
        array[i], array[min_idx] = array[min_idx], array[i]
    return array,num_of_comparisons
```

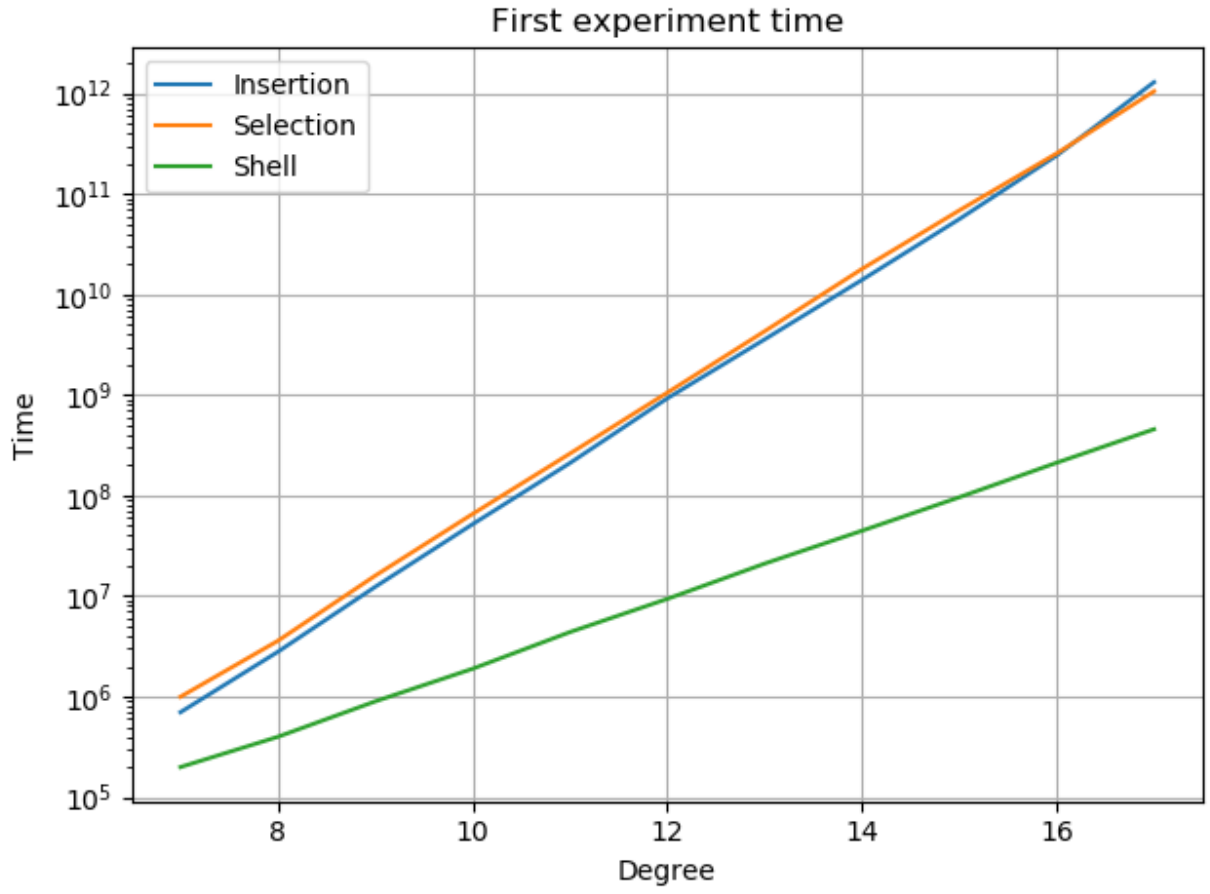
3)Shell sort

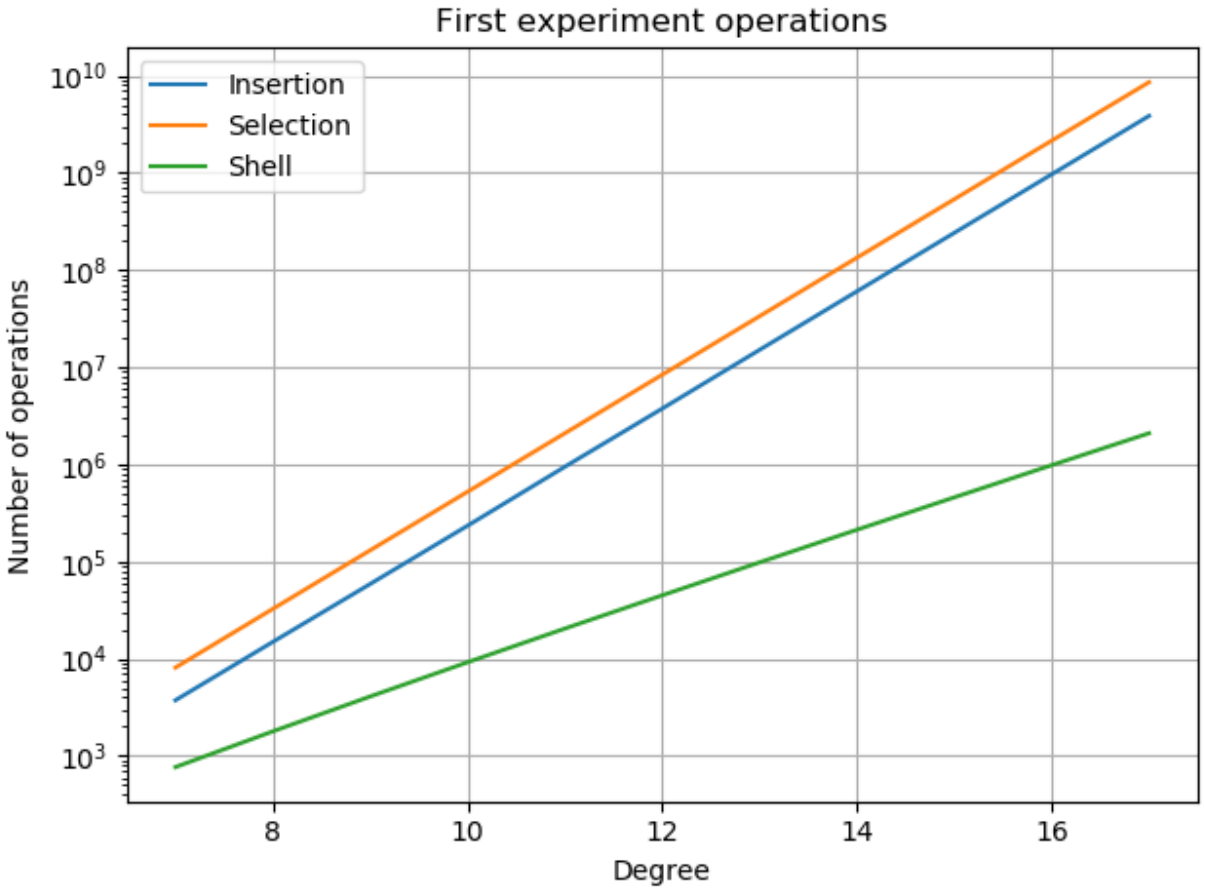
```
def shell_sort(arr):
    num_of_comparisons=0
    array = arr.copy()
    n = len(array)
    gap = n // 2
    while gap > 0:
        for i in range(gap, n):
            temp = array[i]
            j = i
            num_of_comparisons += 1
            while j >= gap and array[j - gap] > temp:
                array[j] = array[j - gap]
                j -= gap
```

```
                num_of_comparisons += 1
            array[j] = temp
        gap //= 2
    return array,num_of_comparisons
```

Перший експеримент

У випадку обчислення середнього часу та кількості операцій для 10 згенерованих масивів від довжини 2^7 до 2^17 простежується така тенденція:
1)Shell sort виконується помітно швидше та вимагає меншу кількість операцій порівняння, ніж insertion та selection для будь-якої довжини масиву.
2)Selection і insertion виконуються за приблизно однаковий час на усіх довжинах масивів з невеликим відхиленням: selection sort виконується спочатку трохи довше, ніж insertion, до значення довжини масиву 2^16, а далі повільніше. Натомість операцій порівняння для selection sort потрібно більше, ніж для insertion.





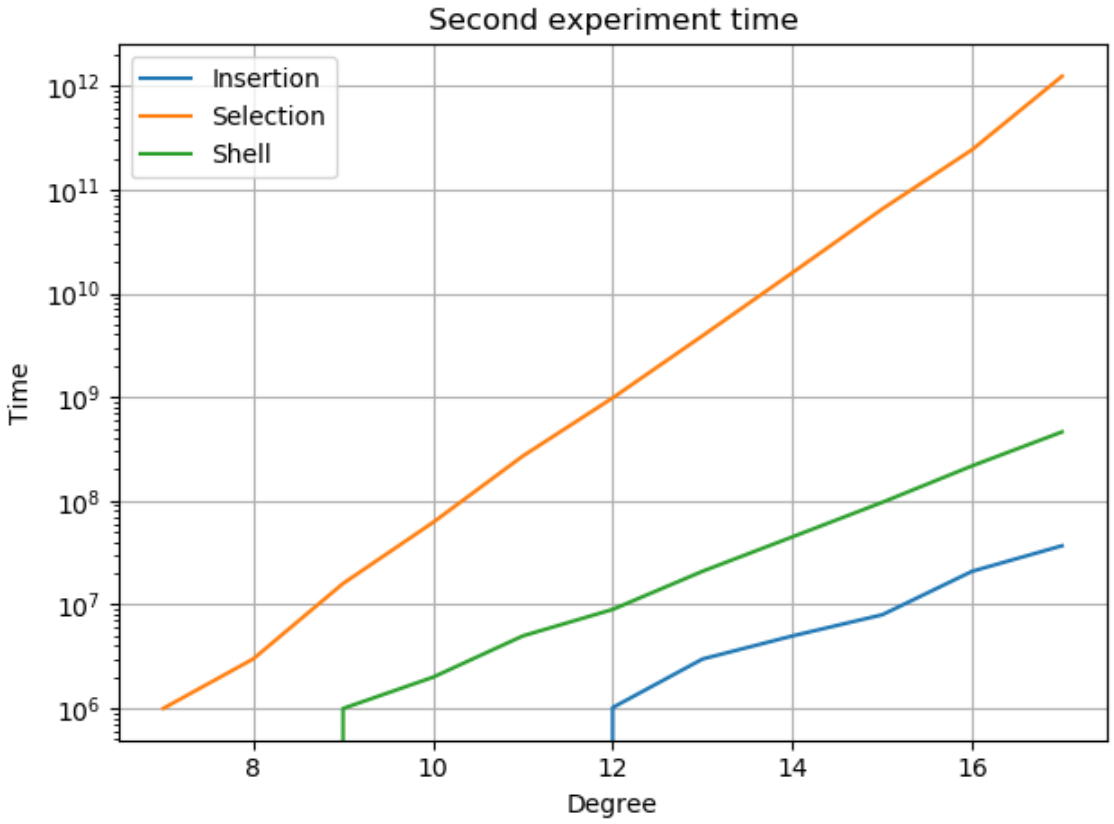
Другий експеримент

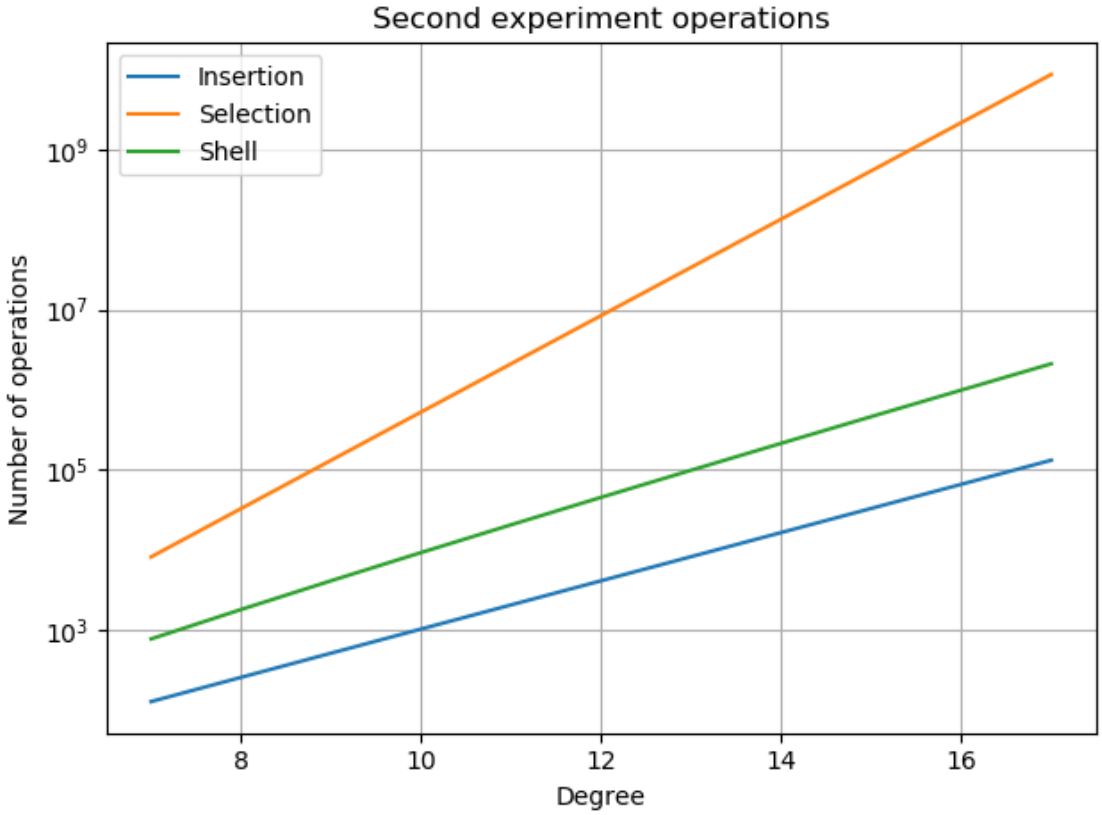
Дослідження проведені на масиві відсортованому по зростанню показують:

1.Selection sort буде виконуватись найдовше та за найбільшу кількість операцій порівняння, адже час та кількість операцій порівняння для цього сортування не залежать від вхідних даних. Тобто відсортований масив, чи ні – не впливає ні на час, ні на кількість операцій порівняння.

2.Натомість найефективніше у цьому випадку звісно працюватиме insertion sort, адже відбудеться просто проходження по усіх числах і перевірка чи число зліва від даного є меншим за дане число.

3.Shell sort, який є оптимізованою версією insertion буде більш ефективним, ніж selection, але менш ефективним, ніж insertion.


Second experiment time
Insertion
Selection
Shell
Time
Degree

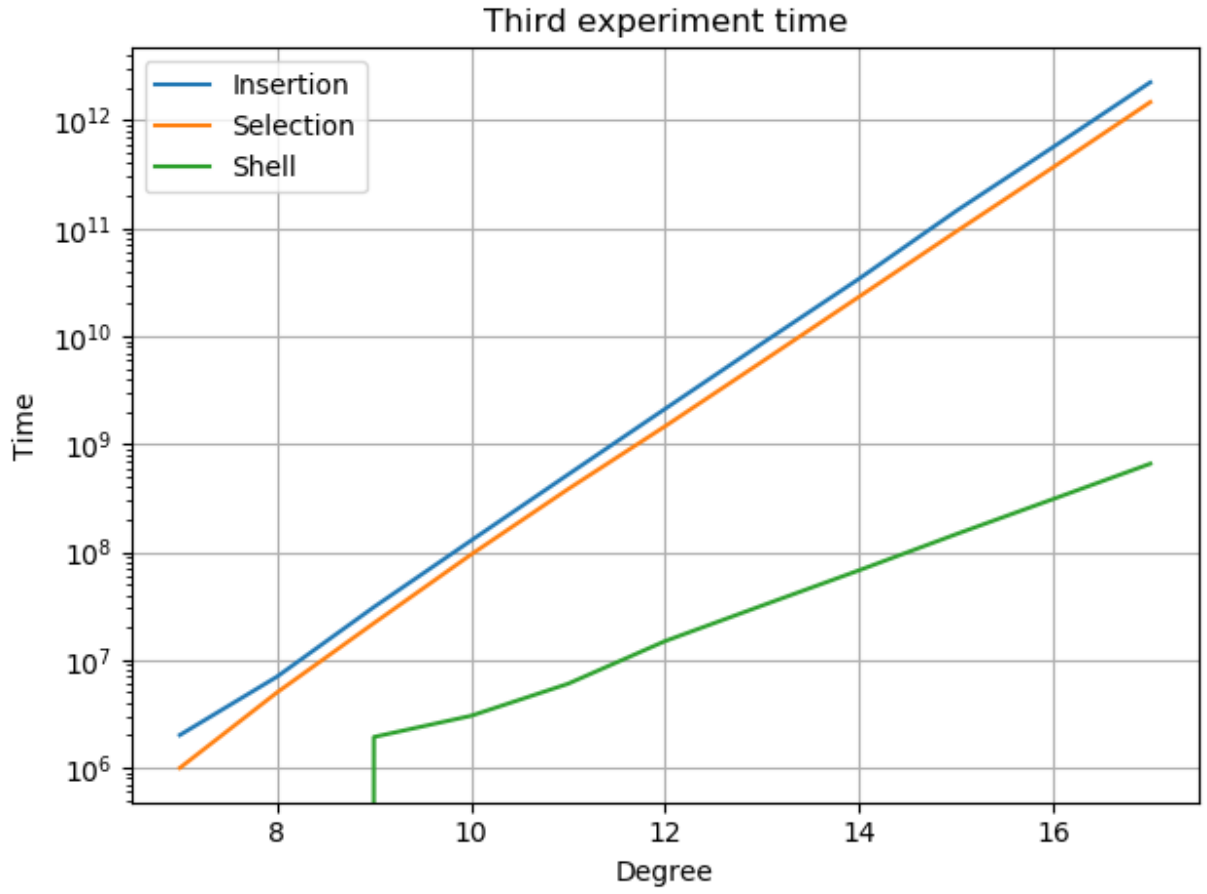

Second experiment operations
Insertion
Selection
Shell
Number of operations
Degree

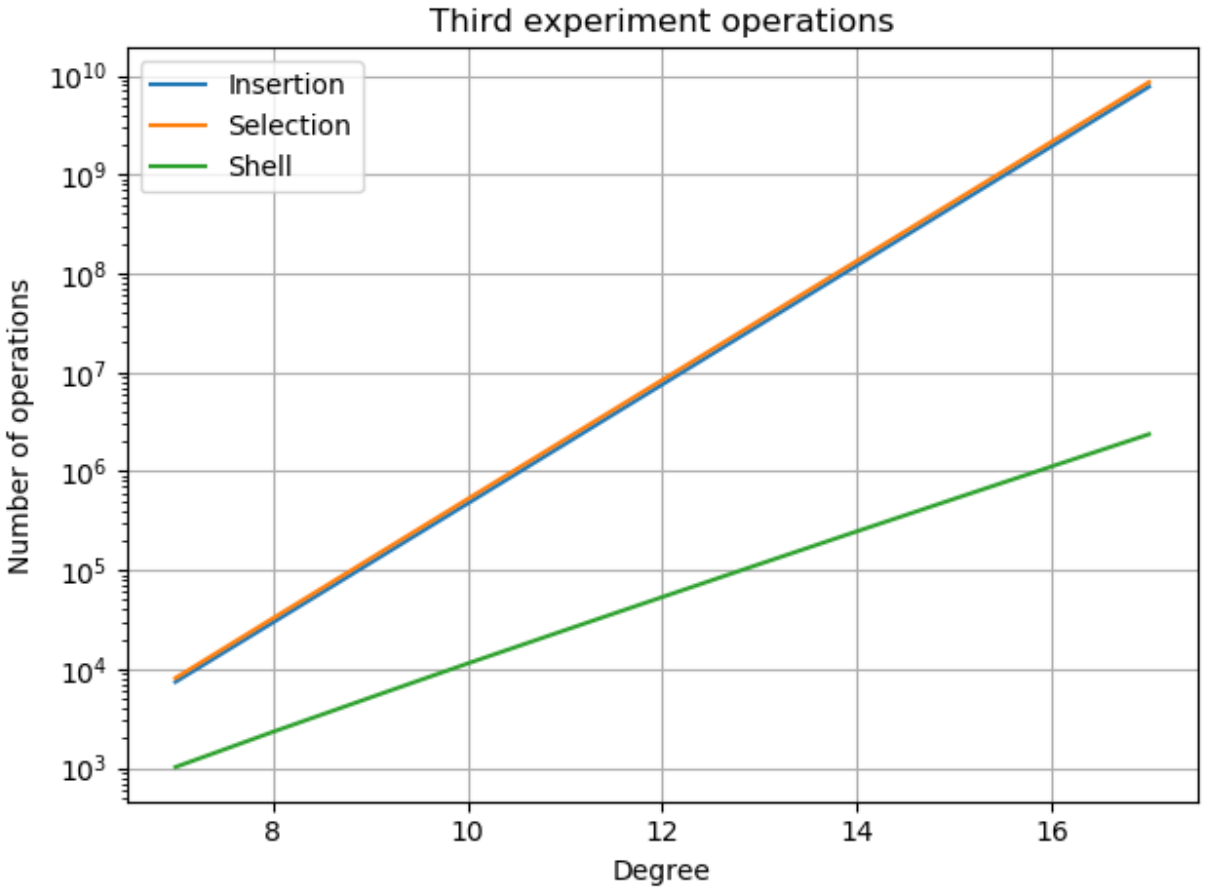
Третій експеримент

Дослідження, проведені на масиві, відсортованому по спаданню показали, що:

1.Найдовше у цьому експерименті буде працювати insertion sort, адже спадний масив – це найгірший для цього типу сортування випадок, коли переміщувати доведеться кожен елемент масиву, трохи швидше selection і найшвидше shell.

2.Кількість операцій порівняння для insertion та selection буде однаковою для будь-якої довжини, а для shell sort буде меншою.



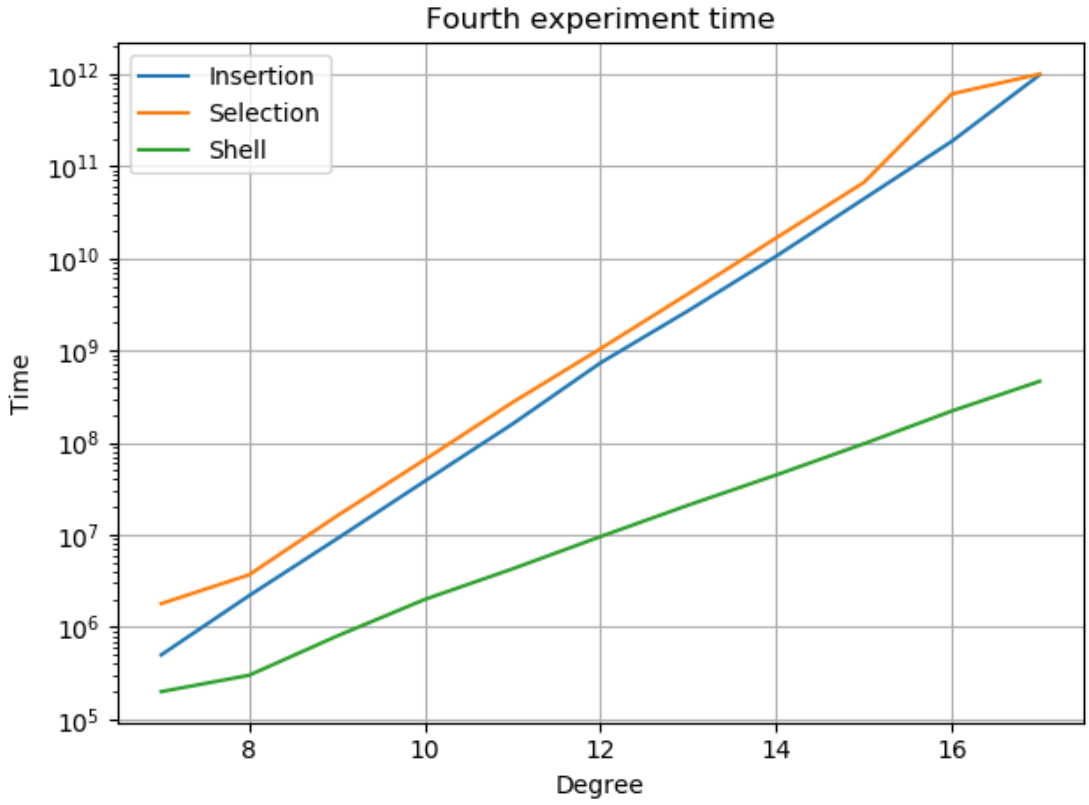


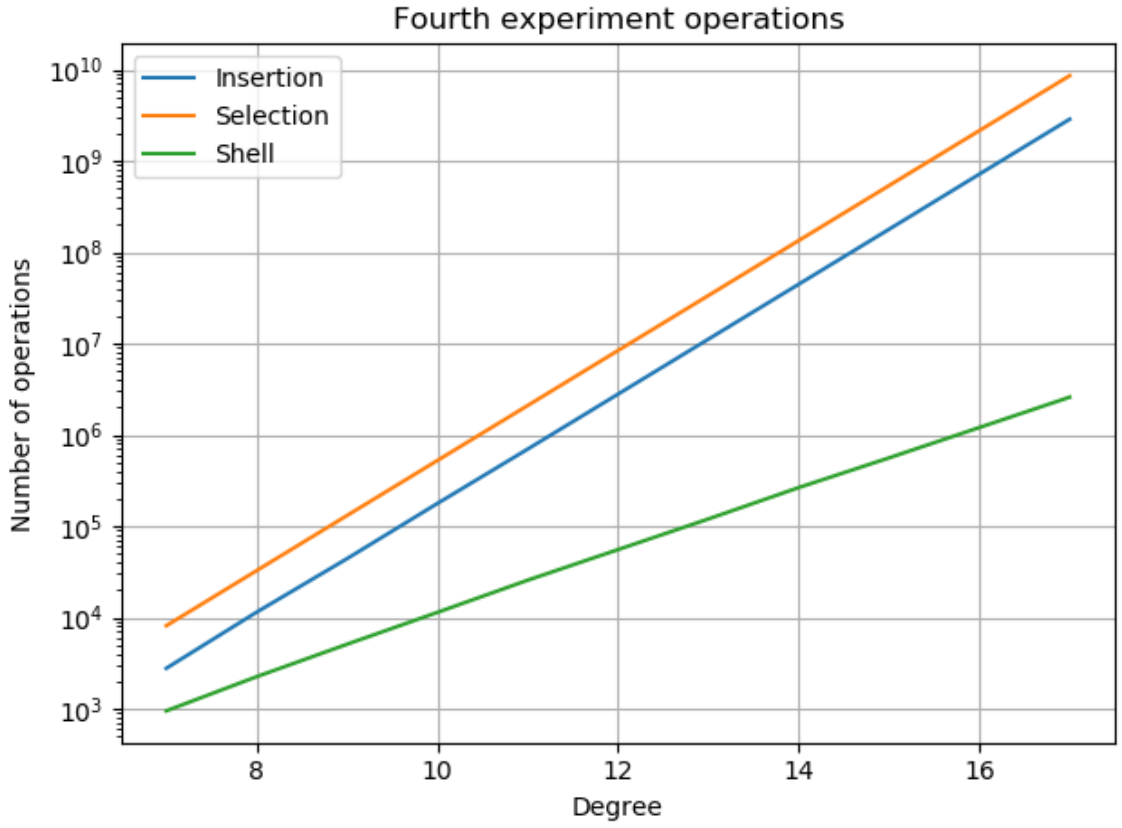
Четвертий експеримент

Цікаво, що коли зменшити числа у масиві до множини {1,2,3} , тобто коли більша кількість повторів одних і тих самих чисел, то спостерігається інша ситуація, зокрема:

1)Insertion працює швидше порівняно з експериментом 1 через більшу кількість повторів. Найшвидше працює shell і найдовше selection.

2)Кількість операцій порівняння найменша для shell, середня для insertion і найбільша для selection.


Fourth experiment time
Insertion
Selection
Shell
Time
Degree
10^{12}
10^{11}
10^{10}
10^{9}
10^{8}
10^{7}
10^{6}
10^{5}
8
10
12
14
16


Fourth experiment operations
Insertion
Selection
Shell
Number of operations
Degree
10^{10}
10^{9}
10^{8}
10^{7}
10^{6}
10^{5}
10^{4}
10^{3}
8
10
12
14
16

Висновок

Отож, у більшості випадків найшвидшим і найменш затратним у плані операцій порівняння є алгоритм shell sort, за винятком сортування зростаючого масиву(бо тоді найвигіднішим є insertion sort).

За графіками можна наочно переконатись в особливості алгоритму selection sort, для якого формат вхідних даних не впливає на величини часу та операцій порівняння. Тоді як для insertion sort кардинально змінюється ефективність у залежності від масиву на вході. Якщо він зростаючий, то insertion працює найкраще, якщо спадний – то найгірше з трьох. Також, якщо значення у масиві повторюються частіше, це також впливає на ефективність insertion sort у позитивну сторону. Під час експериментів також було встановлено, що у загальному випадку час виконання insertion та selection приблизно рівний, а у випадку спадного масиву у них є однакова кількість операцій порівняння.

Отже, selection sort є дуже загальним алгоритмом, який трохи вигідніше використовувати замість insertion лише у випадку, коли вхідна інформація не сприятлива, наприклад найгірший випадок - спадний масив. Тоді як shell sort - це удосконалена версія insertion, і працює ефективніше, ніж інші 2 алгоритми за винятком найсприятливішого випадку для insertion sort – зростаючого масиву.