Лабораторну роботу виконав: Андрусишин Орест
Опис постановки задачі та експерименту: задача полягала в тому, щоб порівняти швидкості роботи і кількості порівнянь алгоритмів сортування (Merge Sort, Insertion Sort, Shell Sort, Selection Sort). Порівняння відбувалось на 4 типах вхідних даних: випадковий масив чисел, посортований по зростанню, посортований по спаданню, масив з великою кількістю однакових елементів.
Специфікація комп'ютера на якому проводились експерименти: кількість ядер: 4, тактова частота: 2.4 - 4.1 ГГц, об'єм оперативної пам'яті: 8 Гб, Операційна система: Ubuntu 20.04.
Програмний код чотирьох алгоритмів:

```
def selection_sort(arr: list):
    """

    >>> arr = [1]
    >>> selection_sort(arr)
    [1]
    >>> arr = []
    >>> selection_sort(arr)
    []
    >>> arr = [1,1,1,1,1,1]
    >>> selection_sort(arr)
    [1, 1, 1, 1, 1, 1]
    >>> arr = [1, 2, 3]
    >>> selection_sort(arr)
    [1, 2, 3]
    >>> arr = [4,2,0,9,6,4]
    >>> selection_sort(arr)
    [0, 2, 4, 4, 6, 9]
    >>> arr = [3, 5, 1, 0, 4]
    >>> selection_sort(arr)
    [0, 1, 3, 4, 5]
    >>> arr = [3,0, 3, 4, 1]
    >>> selection_sort(arr)
    [0, 1, 3, 3, 4]
    """

    length = len(arr)

    for i in range(length):
        min_ind = i
```

```
        for j in range(i+1, length):
            if arr[j] < arr[min_ind]:
                min_ind = j

        arr[i], arr[min_ind] = arr[min_ind], arr[i]

    return arr
def insertion_sort(arr: list):
    """
    >>> arr = [1]
    >>> insertion_sort(arr)
    [1]
    >>> arr = []
    >>> insertion_sort(arr)
    []
    >>> arr = [1,1,1,1,1,1]
    >>> insertion_sort(arr)
    [1, 1, 1, 1, 1, 1]
    >>> arr = [1, 2, 3]
    >>> insertion_sort(arr)
    [1, 2, 3]
    >>> arr = [4,2,0,9,6,4]
    >>> insertion_sort(arr)
    [0, 2, 4, 4, 6, 9]
    >>> arr = [3, 5, 1, 0, 4]
    >>> insertion_sort(arr)
    [0, 1, 3, 4, 5]
    >>> arr = [3,0, 3, 4, 1]
    >>> insertion_sort(arr)
    [0, 1, 3, 3, 4]

    """

    length = len(arr)

    for i in range(1, length):
        key = arr[i]
        j = i-1
        while j >= 0 and key < arr[j] :
                arr[j + 1] = arr[j]
```

```
                j -= 1
        arr[j + 1] = key

    return arr


def merge_sort(lst: list) -> list:
    """
    Returns sorted lst using merge sort.

    >>> arr = [1]
    >>> merge_sort(arr)
    [1]
    >>> arr = []
    >>> merge_sort(arr)
    []
    >>> arr = [1,1,1,1,1,1]
    >>> merge_sort(arr)
    [1, 1, 1, 1, 1, 1]
    >>> arr = [1, 2, 3]
    >>> merge_sort(arr)
    [1, 2, 3]
    >>> arr = [4,2,0,9,6,4]
    >>> merge_sort(arr)
    [0, 2, 4, 4, 6, 9]
    >>> arr = [3, 5, 1, 0, 4]
    >>> merge_sort(arr)
    [0, 1, 3, 4, 5]
    >>> arr = [3,0, 3, 4, 1]
    >>> merge_sort(arr)
    [0, 1, 3, 3, 4]
    """

    length = len(lst)
    if length < 2:
        return lst
    middle = length // 2
    lst1 = lst[:middle]
    lst2 = lst[middle:]
    merge_sort(lst1)
```

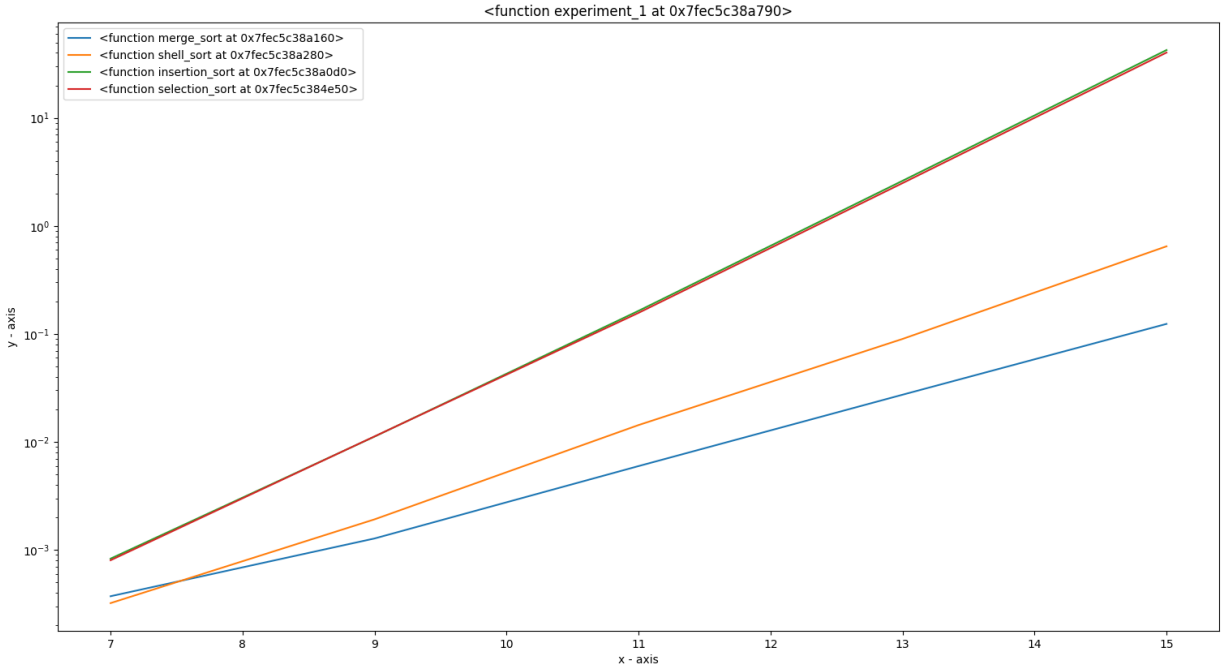
```
    merge_sort(lst2)
    merge(lst1, lst2, lst)
    return lst

def merge(lst1: list, lst2: list, lst3: list):
    """
    Merges sorted lists lst1 and lst2 in lst3.
    len(lst3) == len(lst1) + len(lst2). Returns
    nothing, just changes lst3.

    """

    in1 = in2 = in3 = 0
    len1, len2 = len(lst1), len(lst2)

    while (in1 < len1) and (in2 < len2):
        if lst1[in1] < lst2[in2]:
            lst3[in3] = lst1[in1]
            in1 += 1
        else:
            lst3[in3] = lst2[in2]
            in2 += 1
        in3 += 1

    while in1 < len1:
        lst3[in3] = lst1[in1]
        in3 += 1
        in1 += 1

    while in2 < len2:
        lst3[in3] = lst2[in2]
        in3 += 1
        in2 += 1



def shell_sort(arr):
    """
    >>> arr = [1]
    >>> shell_sort(arr)
```

```
[1]
>>> arr = []
>>> shell_sort(arr)
[]
>>> arr = [1,1,1,1,1,1]
>>> shell_sort(arr)
[1, 1, 1, 1, 1, 1]
>>> arr = [1, 2, 3]
>>> shell_sort(arr)
[1, 2, 3]
>>> arr = [4,2,0,9,6,4]
>>> shell_sort(arr)
[0, 2, 4, 4, 6, 9]
>>> arr = [3, 5, 1, 0, 4]
>>> shell_sort(arr)
[0, 1, 3, 4, 5]
>>> arr = [3,0, 3, 4, 1]
>>> shell_sort(arr)
[0, 1, 3, 3, 4]
"""

n = len(arr)
interval = n // 2
while interval > 0:
    for i in range(interval, n):
        temp = arr[i]
        j = i
        while j >= interval and arr[j - interval] > temp:
            arr[j] = arr[j - interval]
            j -= interval

        arr[j] = temp
    interval //= 2

return arr
```

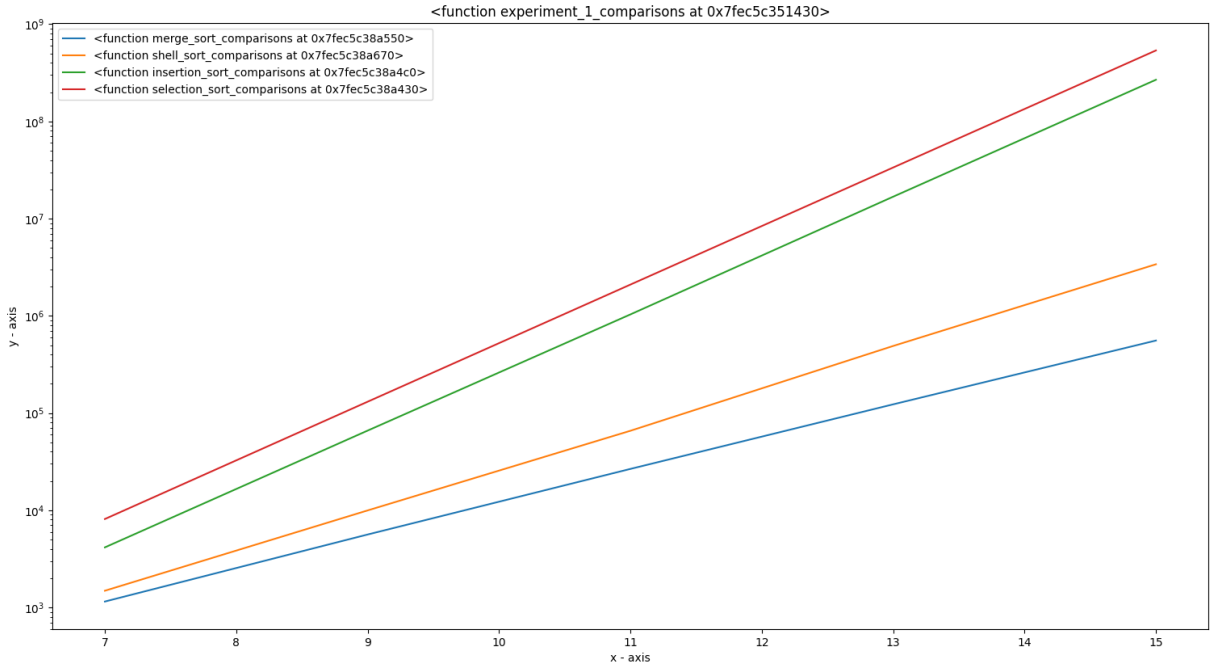
Результати експериментів:

Перший експеримент (випадковий масив):

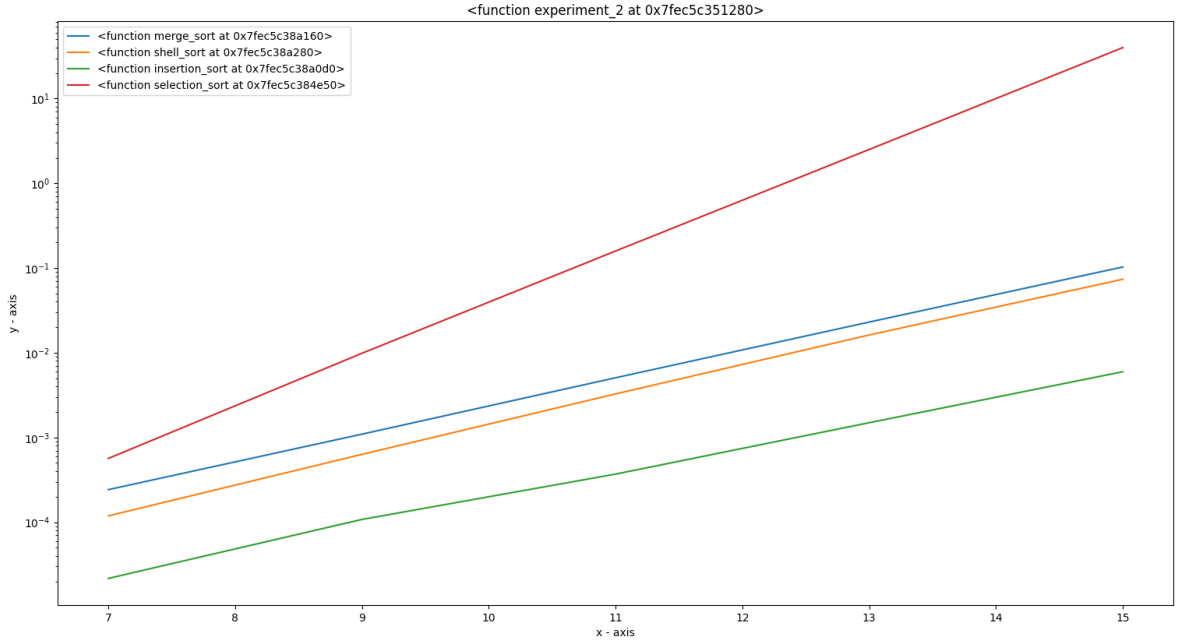
Графік логарифмічної залежності часу від розміру масиву (степеня двійки):



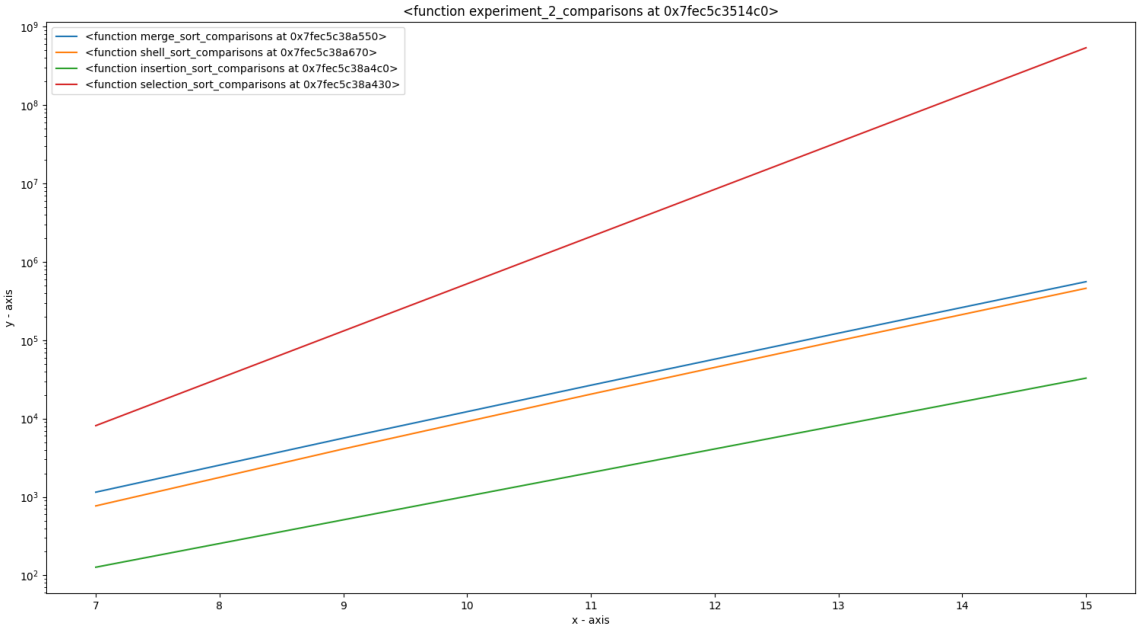
Графік логарифмічної залежності кількості порівнянь від розміру масиву:



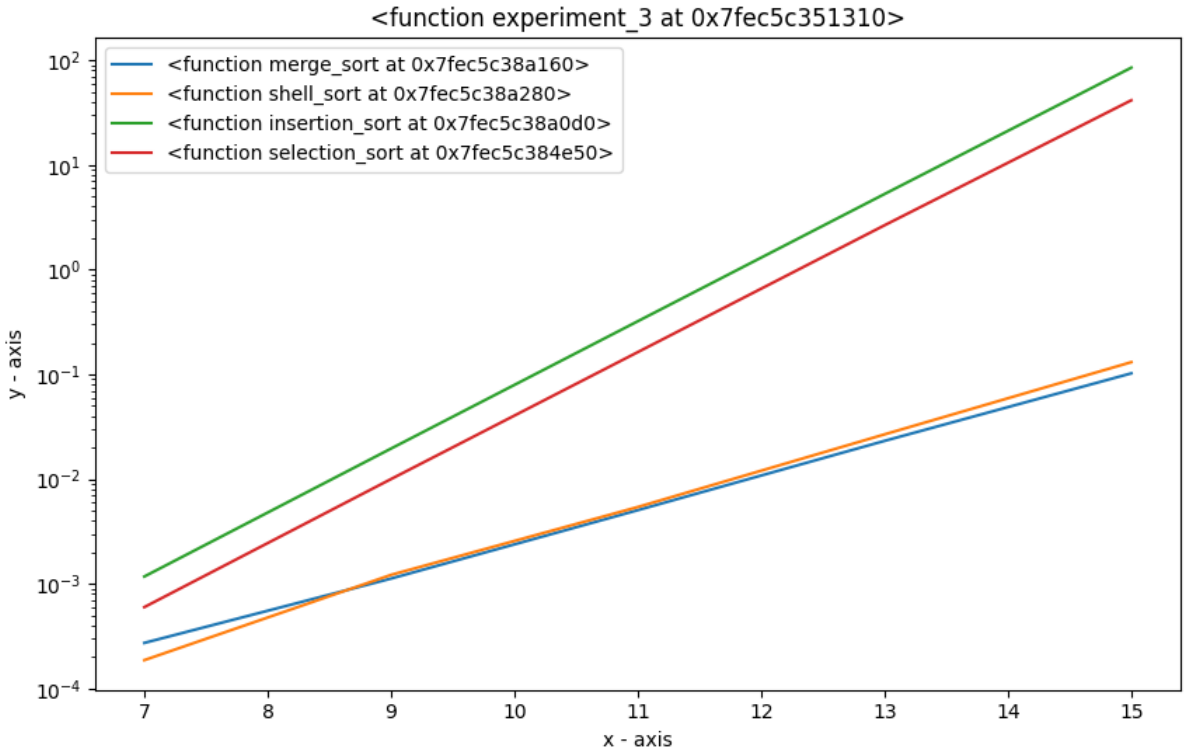
Другий експеримент (посортований в порядку зростання):
Графік логарифмічної залежності часу від розміру масиву:



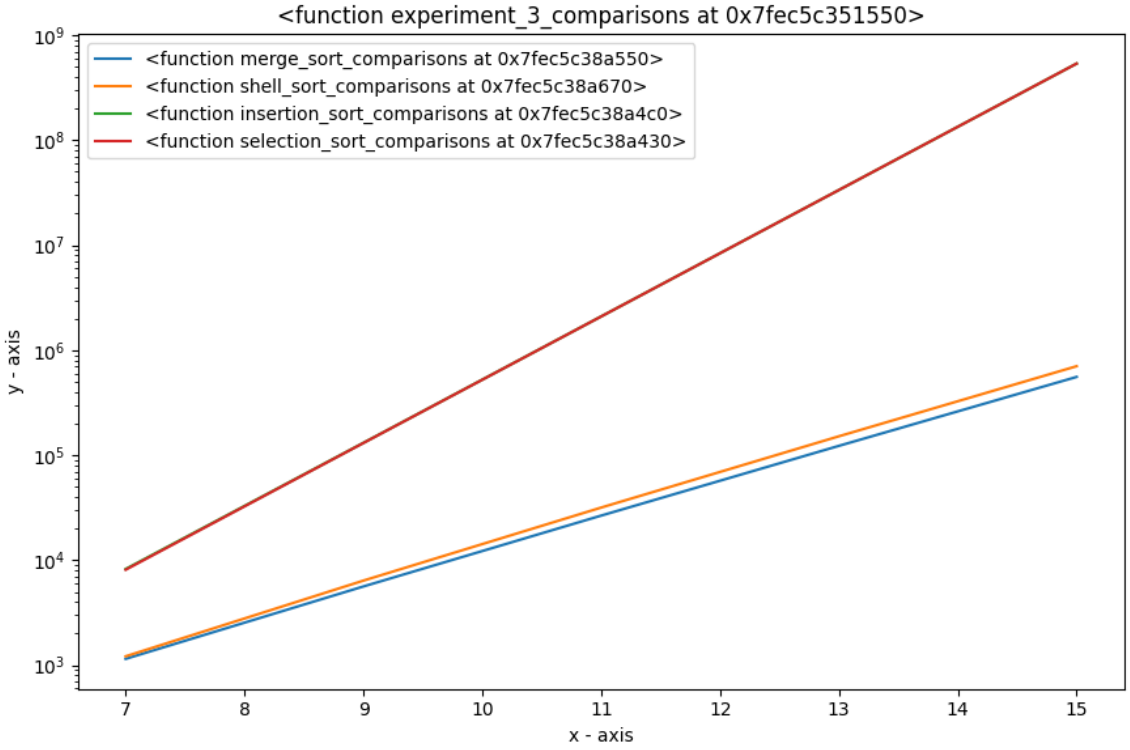
Графік логарифмічної залежності кількості порівнянь від розміру масиву:



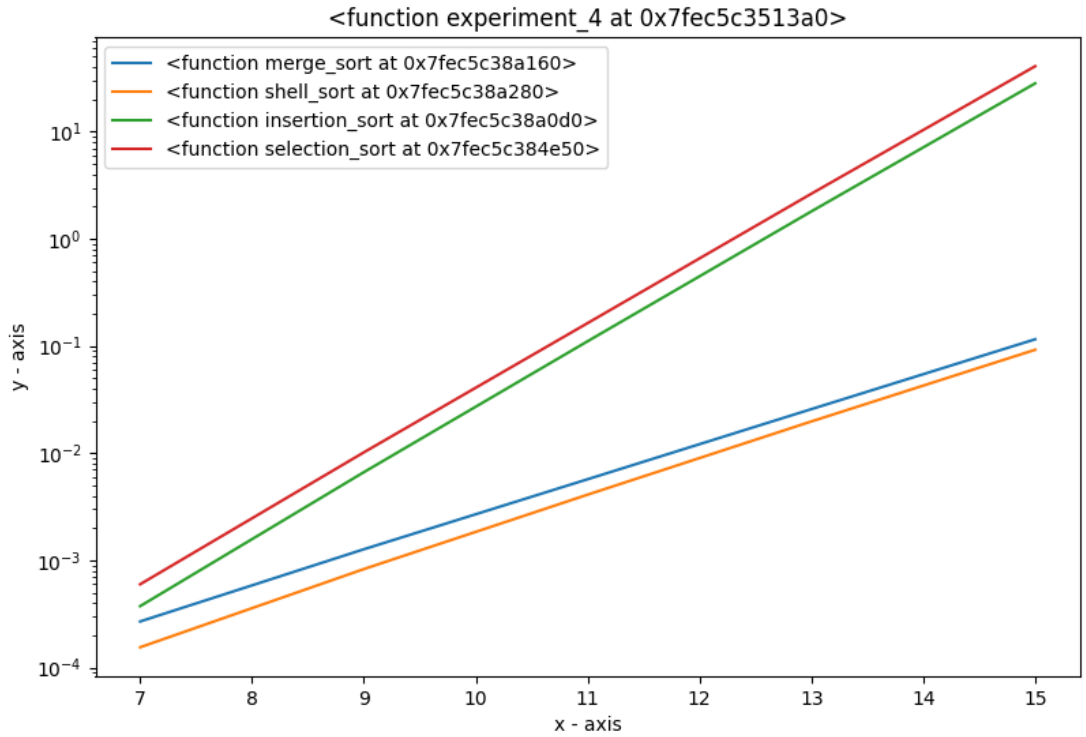
Третій експеримент (посортований в порядку спадання):
Графік логарифмічної залежності часу від розміру масиву:



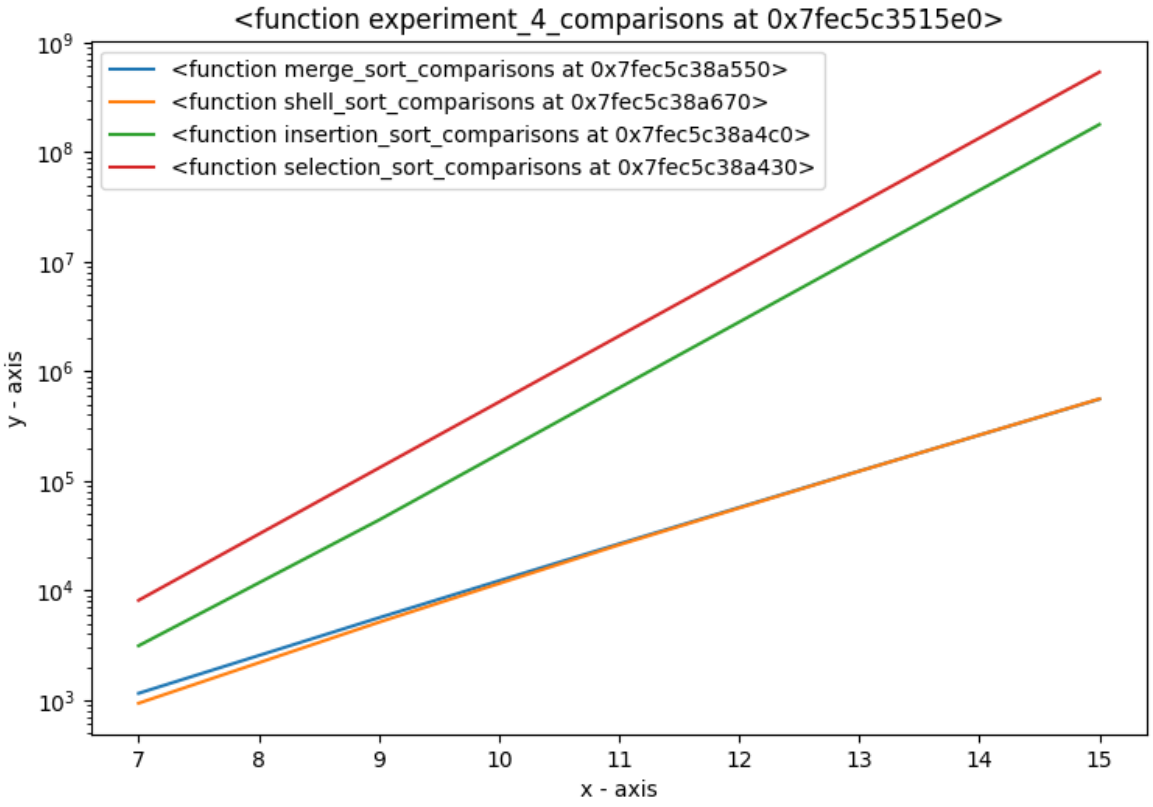
Графік логарифмічної залежності кількості порівнянь від розміру масиву:



Четвертий експеримент (багато однакових елементів):
Графік логарифмічної залежності часу від розміру масиву:



Графік логарифмічної залежності кількості порівнянь від розміру масиву:



Висновки: З отриманих графіків можна зробити висновки, що для алгоритмів merge sort і selection sort час виконання не залежить від типу вхідних даних. В усіх випадках selection sort працює повільно, а merge sort досить швидко. Це доволі закономірно оскільки складність selection sort - це n^2, а складність merge sort - цу nlogn. (в selection sort ми проходимся подвійним фором однакову кількість разів, а в merge sort процедура merge виконується завжди за час Θ(n)). Shell sort також працює доволі швидко, особливо, коли масив є посортованим. А Insertion Sort лідирує у випадку, коли масив посортований по зростанню, оскільки тоді ми фактично один раз проходимся циклом фор по ньому.

Посилання на репозиторій Github:
https://github.com/Andrusyshyn-Orest/comparing-alghorithms