

# Практичне завдання №1. Порівняння роботи Selection sort, Insertion sort, Merge sort, Shell Sort

Звіт

Косик Дарини  
2-й курс ІТ та БА

## Мета:

Порівняти ефективність роботи алгоритмів Selection sort, Insertion sort, Merge sort, та Shellsort для різних масивів розмірами від  $2^7$  і до  $2^{15}$ , з кроком  $\times 2$ . Серед масивів були: випадковим чином згенерований масив, масив, відсортований у порядку зростання; масив, відсортований у порядку спадання; масив, який містить лише елементи з множини  $\{1, 2, 3\}$ .

## Специфікація комп'ютера:

Кількість ядер: 8

Тактова частота: 2 ГГц

Оперативна пам'ять: 16 ГБ

ОС: Windows 10

### Selection sort:

```
def selection_sort(num_lst):  
    for i in range(len(num_lst)):  
        min_idx = i  
        for j in range(i+1, len(num_lst)):  
            if num_lst[min_idx] > num_lst[j]:  
                min_idx = j  
            global comparisons  
            comparisons += 1  
        num_lst[i], num_lst[min_idx] = num_lst[min_idx], num_lst[i]  
  
    return num_lst
```

## Insertion sort:

```
def insertion_sort(num_lst):  
    for i in range(1, len(num_lst)):  
        minimal_idx = num_lst[i]  
        j = i-1  
        while j >= 0 and minimal_idx < num_lst[j]:  
            num_lst[j + 1] = num_lst[j]  
            j -= 1  
            global comparisons  
            comparisons += 1  
        num_lst[j + 1] = minimal_idx  
  
        comparisons += 1  
  
    return num_lst
```

## Merge sort:

```
def merge_sort(lst):  
    lst_length = len(lst)  
  
    if lst_length == 1:  
        return lst  
  
    middle = lst_length // 2  
  
    left_lst = merge_sort(lst[:middle])  
    right_lst = merge_sort(lst[middle:])  
  
    return merge(left_lst, right_lst)  
  
def merge(left_lst, right_lst):  
    final_lst = []  
    i = j = 0  
  
    while i < len(left_lst) and j < len(right_lst):  
        if left_lst[i] < right_lst[j]:  
            final_lst.append(left_lst[i])  
            i += 1  
        else:  
            final_lst.append(right_lst[j])  
            j += 1  
  
        global comparisons  
        comparisons += 1  
  
    final_lst.extend(left_lst[i:])  
    final_lst.extend(right_lst[j:])  
  
    return final_lst
```

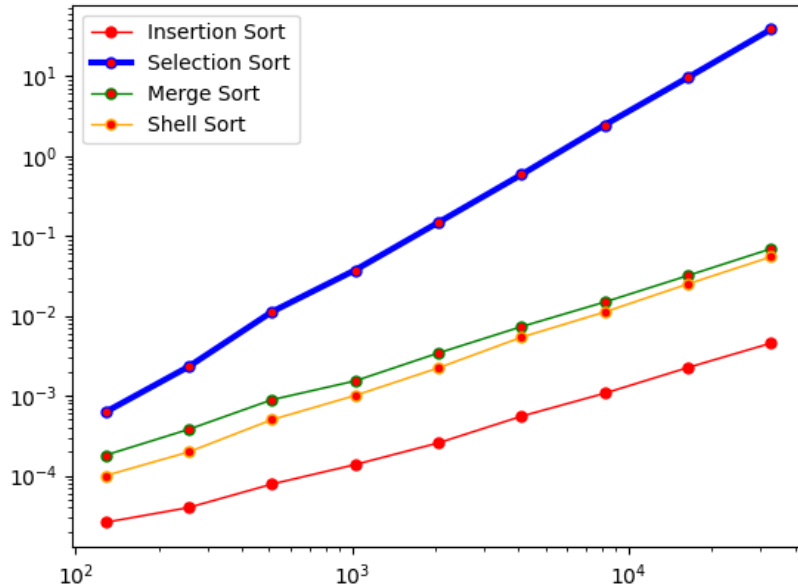
## Shell sort:

```
def shell_sort(num_list):  
    n = len(num_list)  
    gap = n//2  
  
    while gap > 0:  
        for i in range(gap, n):  
            current = num_list[i]  
            j = i  
            while j >= gap and num_list[j-gap] > current:  
                num_list[j] = num_list[j-gap]  
                j -= gap  
                global comparisons  
                comparisons += 1  
  
            num_list[j] = current  
            comparisons += 1  
        gap //= 2  
  
    return num_list
```

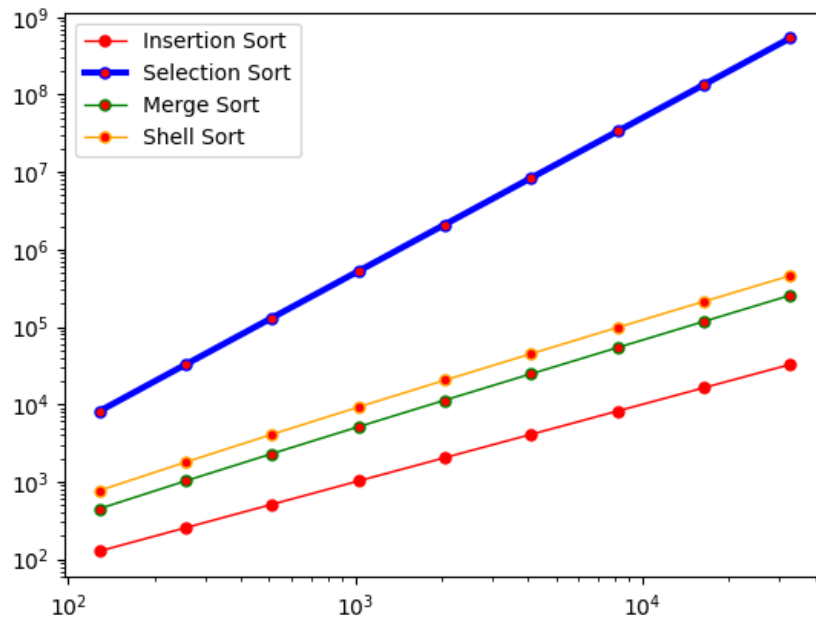
## Графіки

### 1. Масив, згенерований випадковим чином:

#### Час



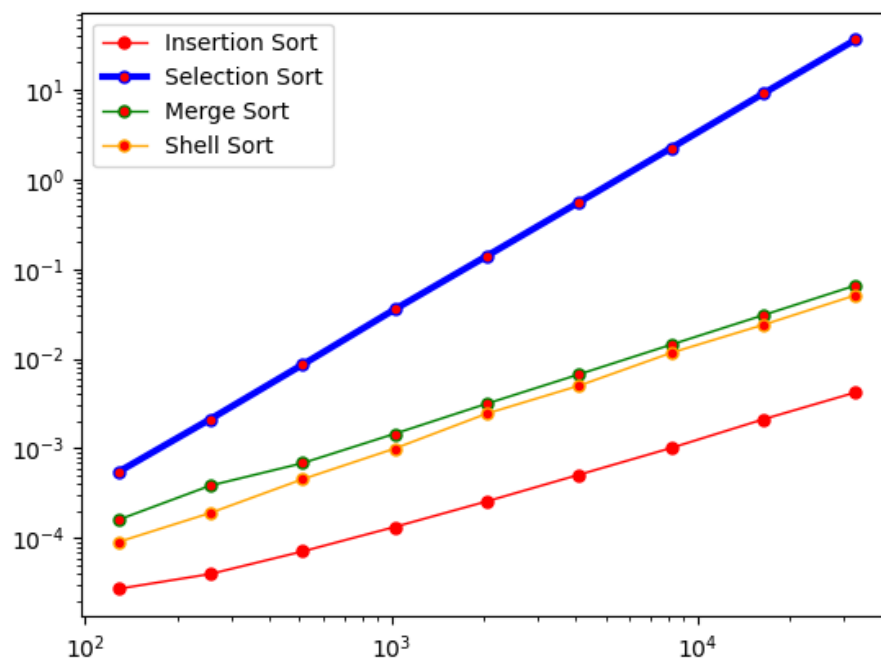
#### Порівняння



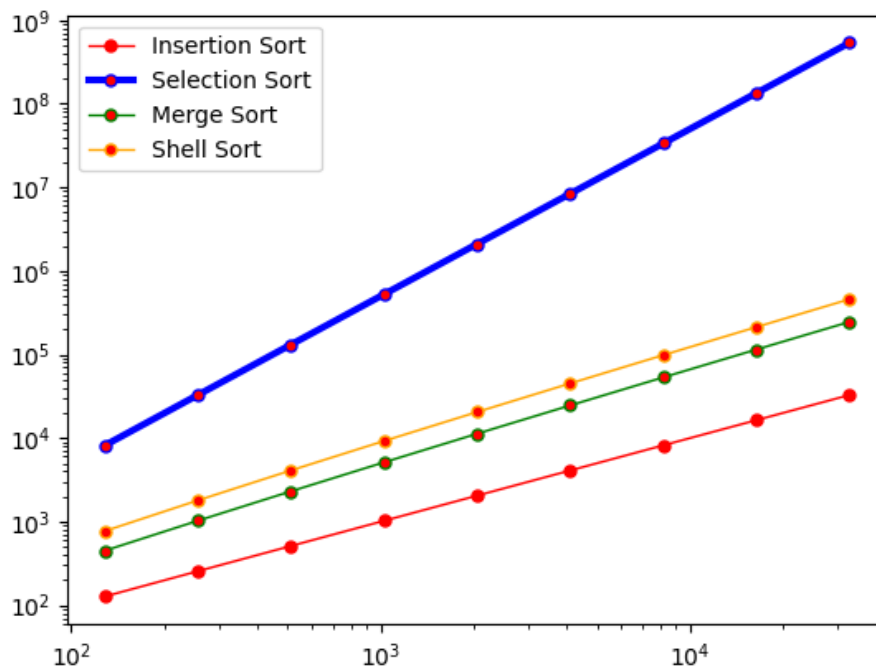
Після проведення експерименту на масиві, згенерованому випадковим чином, можна зробити висновок, що алгоритм Selection Sort працює найменш ефективно. Insertion Sort під час цього експерименту працює найефективніше як стосовно часу роботи, так і стосовно кількості порівнянь. Merge Sort та Shell Sort працюють приблизно однаково.

## 2. Масив, посортований у порядку зростання:

Час



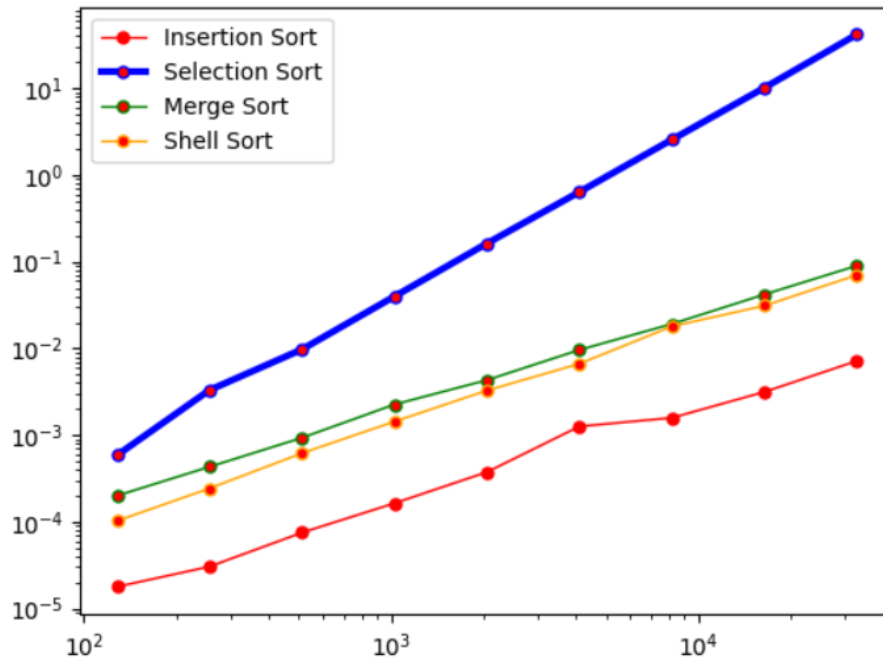
Порівняння



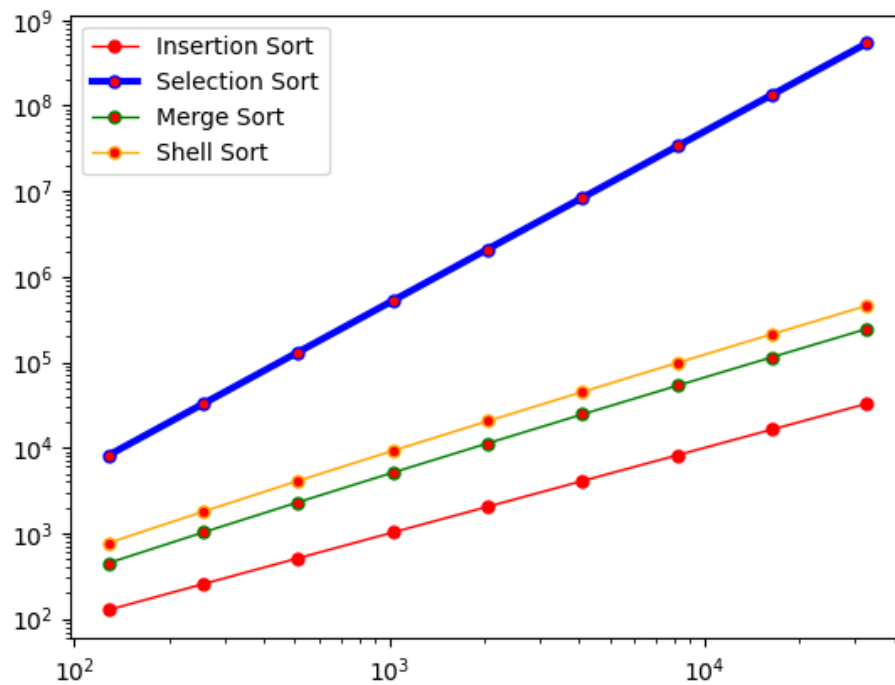
Динаміка роботи алгоритмів практично не змінилась, усі вони зростають майже лінійно.

### 3. Масив, посортований у порядку спадання:

Час



Порівняння

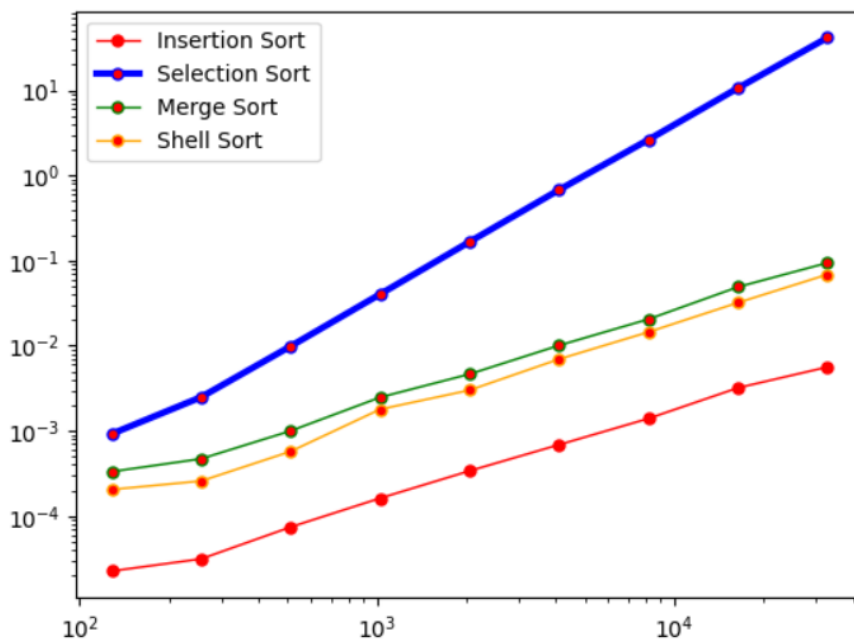


Робота алгоритмів майже нічим не відрізняється, найефективнішим залишається Insertion Sort.

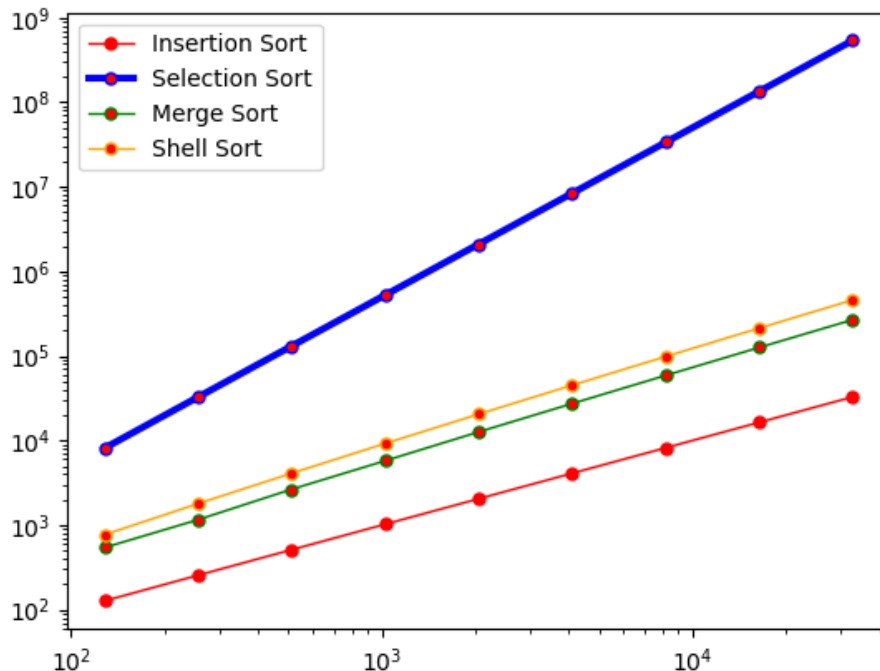


4, Масив, який складається з елементів з множини  $\{1, 2, 3\}$

Час



Порівняння



Робота алгоритмів залишається майже незмінною, Merge Sort та Shell Sort працюють майже за однаковий час та мають приблизно ту саму кількість порівнянь, проте Shell Sort переважає за часом, а Merge Sort має меншу кількість порівнянь.

## Висновок

Під час усіх експериментів алгоритми Selection sort, Insertion sort, Merge sort та Shell Sort показували схожу динаміку роботи. Алгоритм Selection sort залишається найменш ефективним з усіх алгоритмів, адже він має найбільші час роботи та кількість порівнянь. Найшвидшим та з найменшою кількістю порівнянь є алгоритм Insertion Sort. Алгоритми Merge sort та Shell Sort працюють приблизно зі схожою ефективністю, проте Merge sort має меншу кількість порівнянь, а Shell Sort - коротший час роботи. Усі графіки зростають майже лінійно, їх ефективність мало залежить від особливостей вхідного масиву.