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

Звіт

Сушко Степан
Комп’ютерні Науки
2 курс

Мета:

Провести порівняльний аналіз роботи алгоритмів Selection sort, Insertion sort, Merge sort, та Shellsort на прикладі сортування масивів з розмірами від 2^7 і до 2^15, а також різною специфікою наповнення числами (рандомно згенерований масив, масиви, відсортовані у порядку зростання та спадання, масив, який містить лише елементи з множини {1, 2, 3}).

Специфікація комп’ютера:

Кількість ядер: 6
Тактова частота: 2.6 ГГц
Оперативна пам’ять: 16 ГБ
ОС: Arch Linux

Selection sort:

```python
def selectionSort(arr):
    """ Sort the array with merge method """
    counter = 0
    for i in range(len(arr)):
        min_pos = i

        for j in range(i+1, len(arr)):
            counter += 1
            if arr[min_pos] > arr[j]:
                min_pos = j

        counter += 1
        temp = arr[i]
        arr[i] = arr[min_pos]
        arr[min_pos] = temp

    return arr, counter
```

Insertion sort:

```python
def insertionSort(arr):
    counter = 0
    for i in range(len(arr)):
        temp = arr[i]
        hole_pos = i

        while(hole_pos > 0 and arr[hole_pos - 1] > temp):
            counter += 1
            arr[hole_pos] = arr[hole_pos - 1]
            hole_pos -= 1

        counter += 1
        arr[hole_pos] = temp

    return arr,counter
```

Merge sort:

```python
mergecounter = 0

def mergeSort(a):
    global mergecounter
    width = 1
    n = len(a)
    while (width < n):
        l=0;
        while (l < n):
            r = min(l+(width*2-1), n-1)
            m = (l+r)//2
            if (width>n//2):
                m = r-(n%width)
                mergecounter += 1
            merge(a, l, m, r)
            l += width*2
        width *= 2
    return a, mergecounter
```

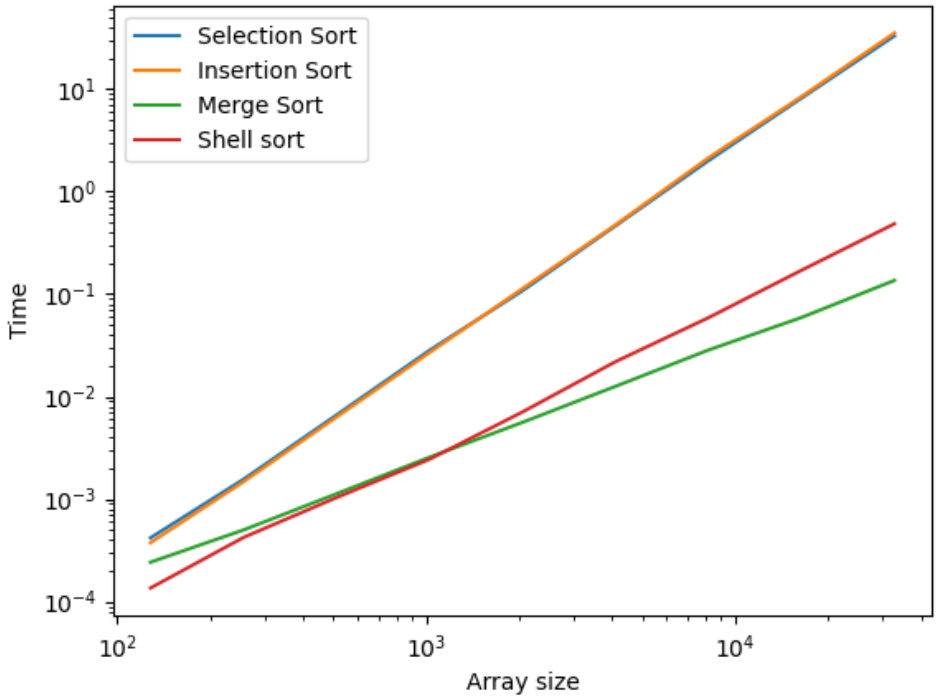
```python
def merge(a, l, m, r):

    global mergecounter

    n1 = m - l + 1
    n2 = r - m
    L = [0] * n1
    R = [0] * n2
    for i in range(0, n1):
        L[i] = a[l + i]
    for i in range(0, n2):
        R[i] = a[m + i + 1]

    i, j, k = 0, 0, l
    while i < n1 and j < n2:
        mergecounter += 1
        if L[i] > R[j]:
            a[k] = R[j]
            j += 1
        else:
            a[k] = L[i]
            i += 1
        k += 1

    while i < n1:
        a[k] = L[i]
        i += 1
        k += 1

    while j < n2:
        a[k] = R[j]
        j += 1
        k += 1
```

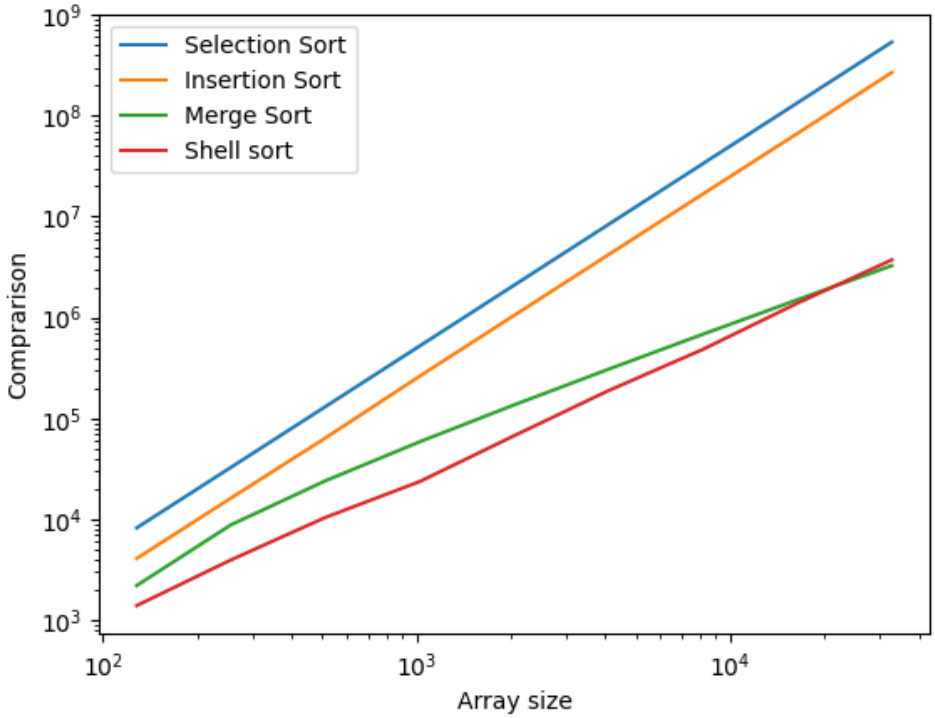
Shell sort:

```python
def shellSort(arr):
    counter = 0
    n = len(arr)
    gap = int(n/2)

    while gap > 0:

        for i in range(gap, n):
            elem = arr[i]
            j = i

            while j >= gap and arr[j - gap] > elem:
                counter += 1
                arr[j] = arr[j - gap]
                j -= gap

            counter += 1
            arr[j] = elem

        gap //= 2

    return arr, counter
```

Графіки

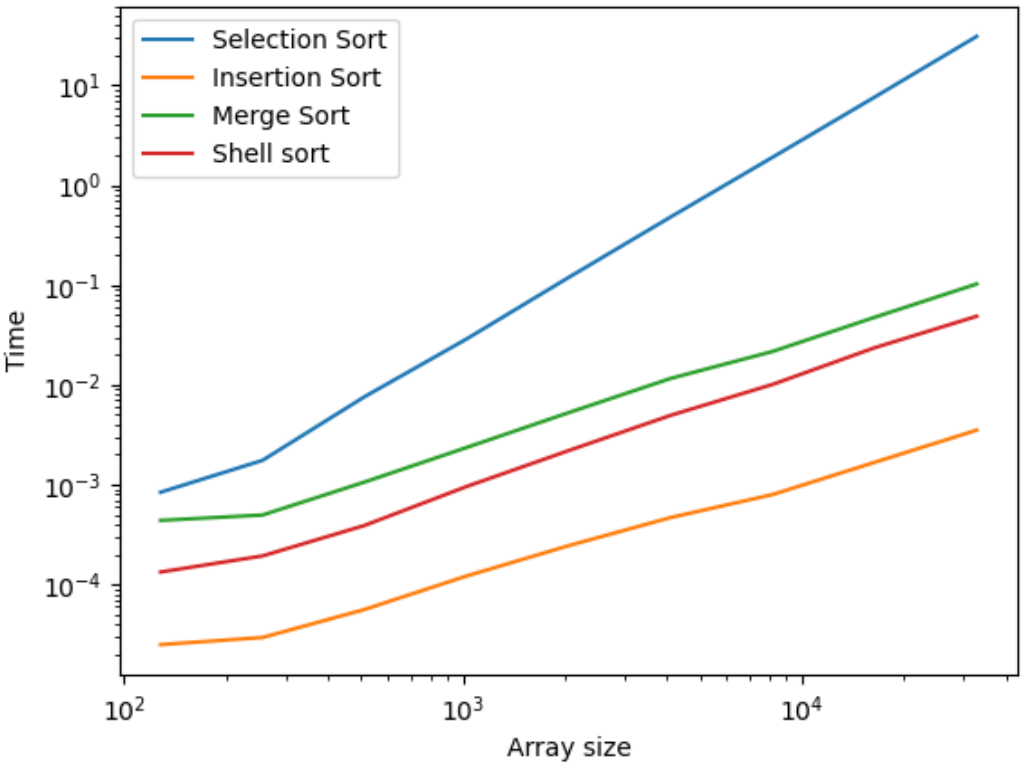
1. Масив, згенерований випадковим чином:

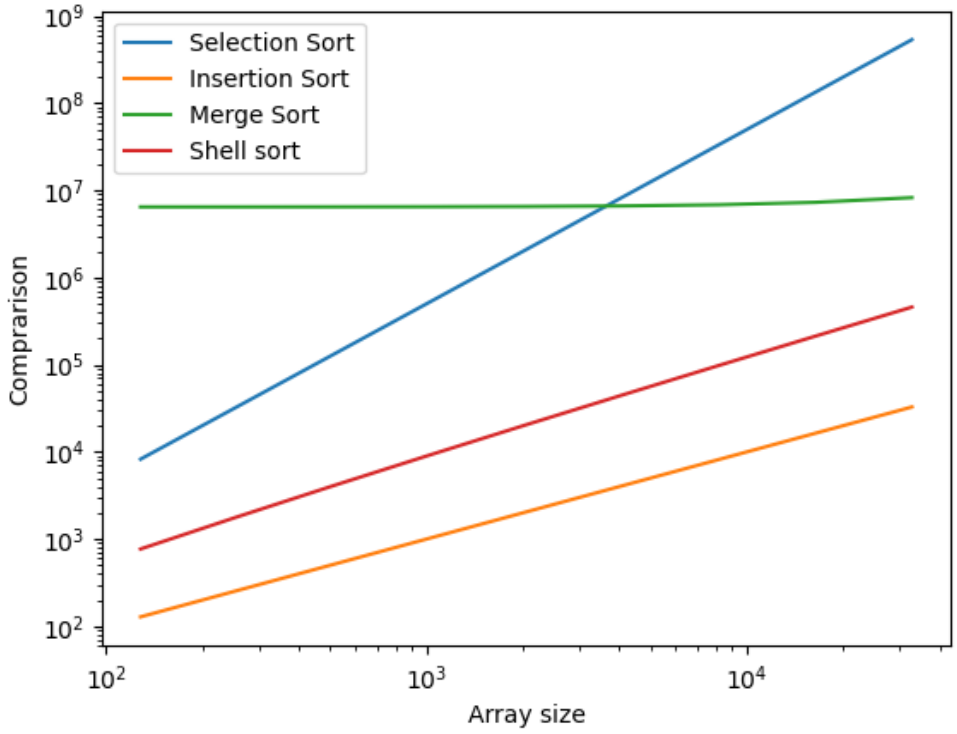




Алгоритми Selection Sort та Insertion Sort працюють найменш ефективно стосовно часу (аналогічно) та кількості порівнянь. Алгоритм Merge Sort на масивах великого розміру потребує найменше часу для сортування, щодо кількості порівнянь - Merge Sort та Shell Sort прямують до однакової ефективності.

2. Масив, посортований у порядку зростання:

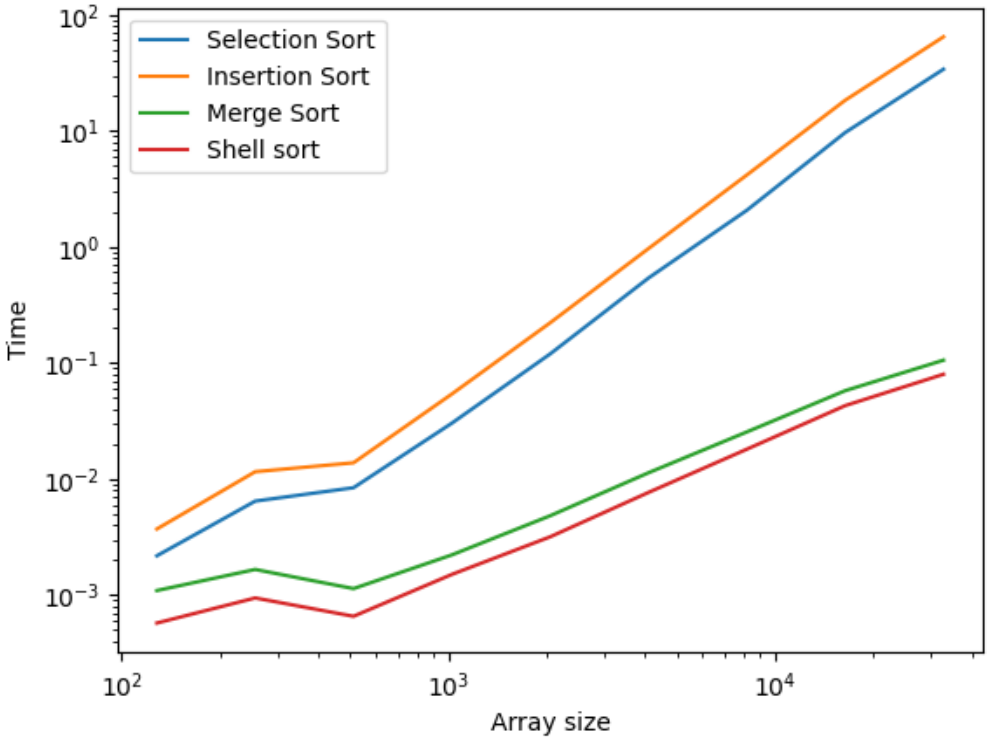


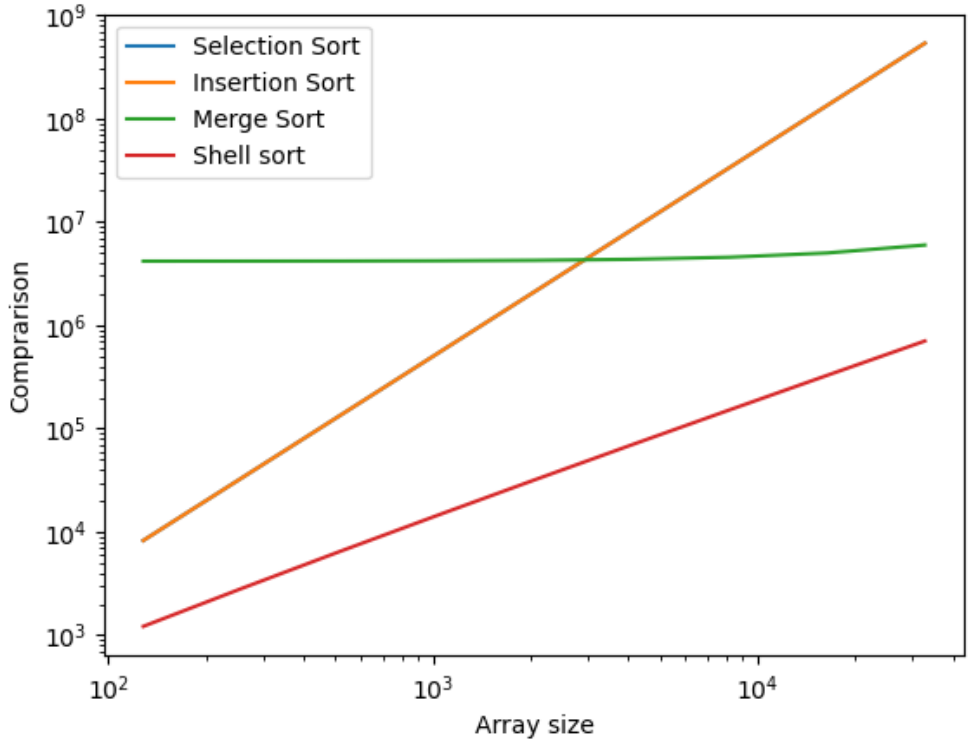


Для алгоритму, посортованого у порядку зростання, Shell Sort є найменш ефективним щодо часу для масивів будь-якого розміру. Щодо кількості порівнянь - після певного розміру

масиву Shell Sort теж стає наймеш ефективним, проте для невеликих масивів найгірше працює Merge Sort. Найефективнішим щодо часу та кількості порівнянь є Insertion Sort.

3. Масив, посортований у порядку спадання:

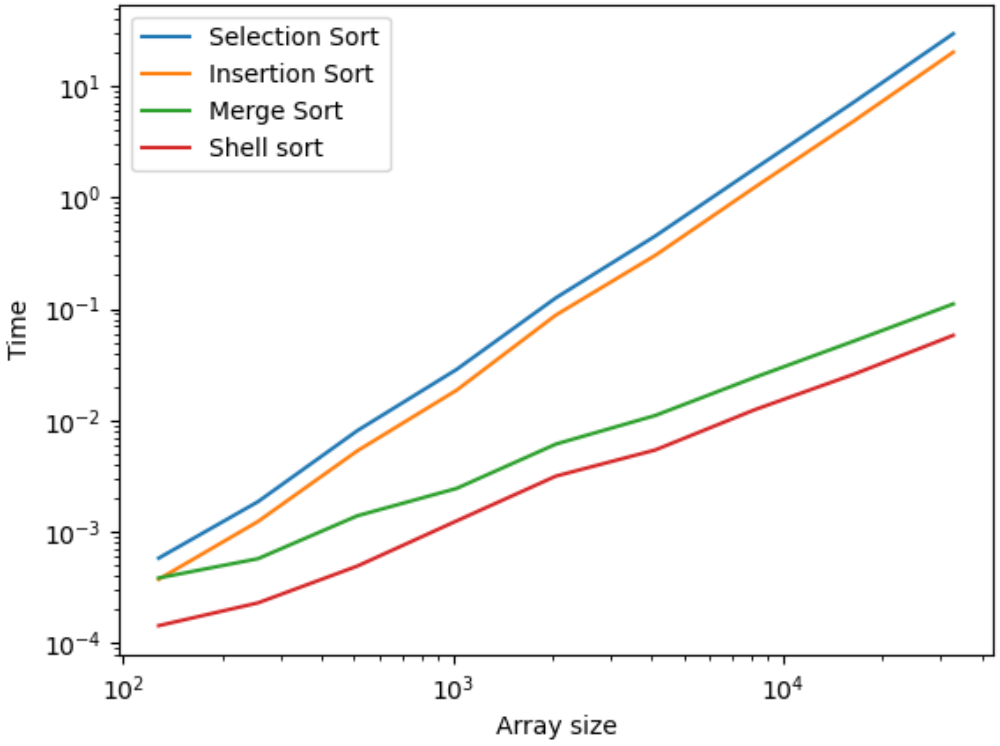


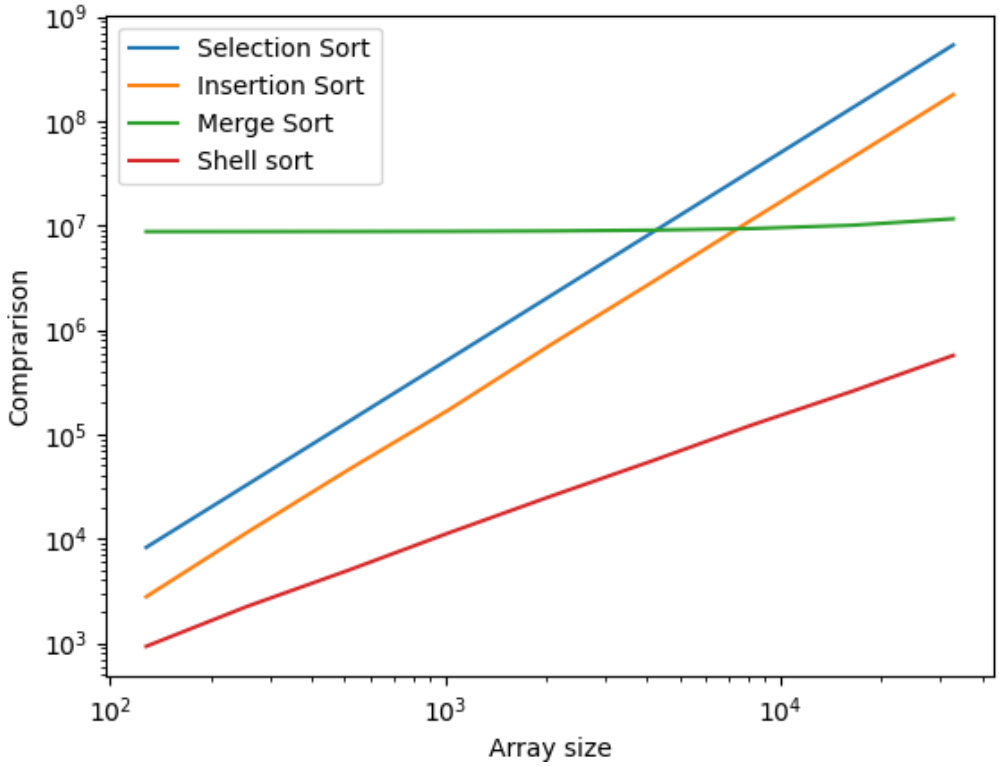


Для масиву, посортованого у порядку спадання, Insertion Sort працює найгірше для масиві великого розміру. Для невеликих масивів стосовно часу Insertion Sort залишається найменш ефективним, проте для масивів невеликого розміру Merge Sort працює найгірше стосовно

кількості порівнянь під час сортування. Найкращі показники має Shell Sort. На графіку кількості порівнянь криві алгоритмів Insertion Sort та Selection Sort збігаються.

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





Під час сортування масиву, який складається з елементів з множини {1, 2, 3}, Найкраще працює алгоритм Shell Sort, а найгірше - Selection Sort, та Insertion Sort. Щодо кількості порівнянь - алгоритм Shell Sort працює найефективніше. Для масивів невеликого розміру

найбільшу кількість порівнянь має Merge Sort, проте для великих масивів найбільше порівнянь здійснюють Selection Sort та Insertion Sort.

Висновок

Під час здійснення усіх експериментів алгоритми показували різну динаміку роботи та ефективність, яка залежала від розмірів масиву та його характеристик. У деяких випадках неможливо було визначити алгоритми, які працювали б найкраще чи найгірше для масиву з певними характеристиками, адже ts алгоритми, які потребували менше часу для здійснення сортування, могли мати високі показники кількості порівнянь і навпаки. Після проведеного аналізу можна зробити висновок, що алгоритм Merge Sort є малоефективним стосовно кількості порівнянь на масивах невеликого розміру. Алгоритм Selection Sort є не дуже ефективним у всіх експериментах, стосовно часу та кількості порівнянь. Для масиву, посортованого у порядку зростання, алгоритм Insertion Sort працює значно ефективніше за всі інші. Для інших масивів майже у всіх випадках найефективніше працює Shell Sort.