

Algoritmu salīdzinājums

Rihards Irbe,
Patriks Noass Ercums,
Aleksandrs Ilarionovs,
Daniels Bērziņš

Sākotnējie mērķi

- Apskatīt 3 kārtošanas algoritmus (1 no tiem pašu izveidots)
- Katram no tiem likt kārtot 10 dažādus masīvus un fiksēt to izpildes laikus
- Aprēķināt vidējos kārtošanas laikus
- Grafikos attēlot visu trīs algoritmu vidējos izpildes laikus
- Lietotājam draudzīgs UI

Izmantotie algoritmi

1. Pašu izdomātais algoritms
2. Quick Sort algoritms
3. Python iebūvētais algoritms

Algoritmi

Pašu izdomātais

```
def self_made(arr: list) -> list:
    unsorted_arr = arr
    for i in range(0, (len(unsorted_arr) - 1)):
        for i in range(0, (len(unsorted_arr) - 1)):
            if(unsorted_arr[i + 1] < unsorted_arr[i]):
                temp_elm = unsorted_arr[i]
                unsorted_arr[i] = unsorted_arr[i + 1]
                unsorted_arr[i + 1] = temp_elm
    return unsorted_arr
```

Quick Sort

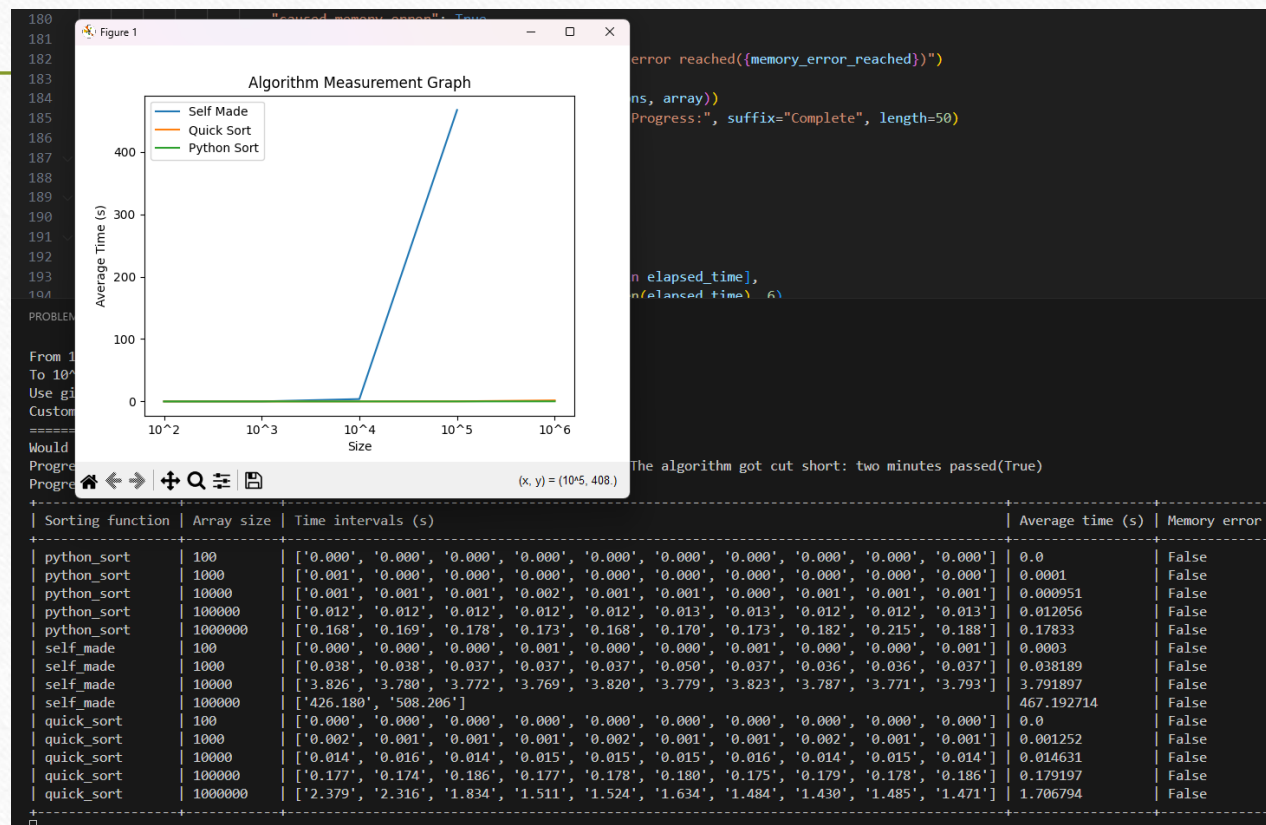
```
def quick_sort(array_file): # Atsauce: https://www.geeksforgeeks.org/python-program-for-quicksort/
    def partition(arr, low, high):
        pivot_index = random.randint(low, high)
        arr[pivot_index], arr[high] = arr[high], arr[pivot_index]
        pivot = arr[high]
        i = low - 1
        for j in range(low, high):
            if arr[j] <= pivot:
                i += 1
                arr[i], arr[j] = arr[j], arr[i]
        arr[i + 1], arr[high] = arr[high], arr[i + 1]
        return i + 1

    def quicksort_helper(arr, low, high):
        while low < high:
            pi = partition(arr, low, high)
            if pi - low < high - pi:
                quicksort_helper(arr, low, pi - 1)
                low = pi + 1
            else:
                quicksort_helper(arr, pi + 1, high)
                high = pi - 1

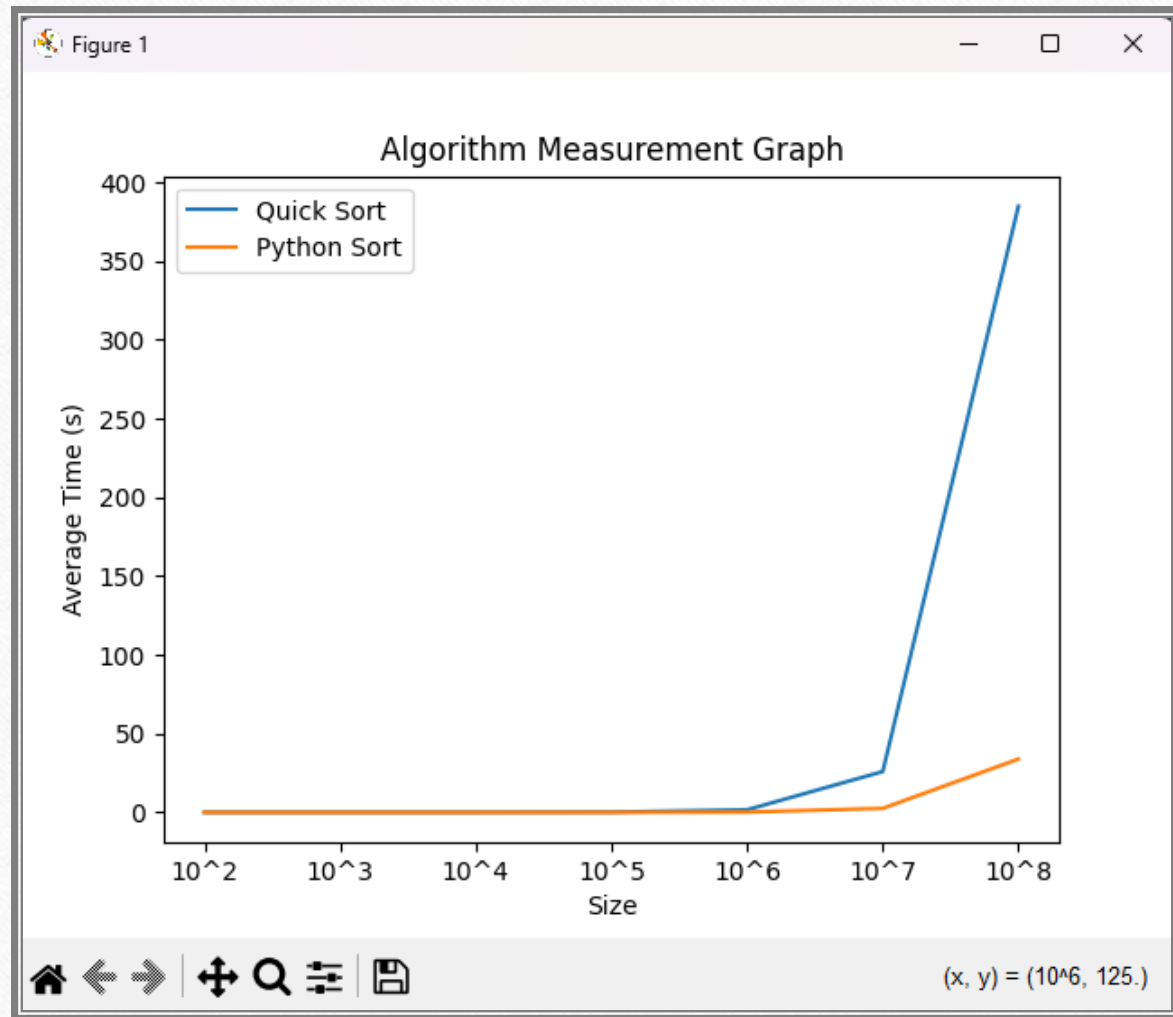
    quicksort_helper(array_file, 0, len(array_file) - 1)
```

Algoritmu ātrumi

1. Python iebūvētais
2. Quick Sort
3. Pašizveidotais



Quick sort pret Python sort











Problēmas ar ko saskārāties

- Nevarējām izveidot/saglabāt masīvu virs 10^9 elementu izmēru.
- Saprast vai pašizveidotais kārtošanas algoritms pareizi rēķina rezultātu.

Secinājumi

- Pašizveidogais kārtošanas algoritms bija daudz lēnāks par Quick Sort un Python iebūvēto kārtošanas algoritmu
- Ģenerēto jaukto masīvu failu, ar elementu skaitu lielāku par 10^9 , izmērs palielinājas dramatiski un to pat nevar izdarīt ar 64 GB RAM

 <code>unsorted_array_10^2.npy</code>	17/12/2024 19:07	NPY File	1 KB
 <code>unsorted_array_10^3.npy</code>	17/12/2024 19:10	NPY File	8 KB
 <code>unsorted_array_10^4.npy</code>	17/12/2024 19:10	NPY File	79 KB
 <code>unsorted_array_10^5.npy</code>	17/12/2024 19:57	NPY File	782 KB
 <code>unsorted_array_10^6.npy</code>	17/12/2024 19:57	NPY File	7,813 KB
 <code>unsorted_array_10^7.npy</code>	17/12/2024 20:15	NPY File	78,126 KB
 <code>unsorted_array_10^8.npy</code>	18/12/2024 15:58	NPY File	781,251 KB
 <code>unsorted_array_10^9.npy</code>	18/12/2024 16:16	NPY File	7,812,501 KB

Vai mēs paveicām, ko gribējām?

- ✓ • Apskatīt 3 kārtošanas algoritmus (1 no tiem pašu izveidotais)
- ✓ • Katram no tiem likt kārtot 10 dažādus masīvus un fiksēt to izpildes laikus
- ✓ • Aprēķināt vidējos kārtošanas laikus
- ✓ • Grafikos attēlot visu trīs algoritmu vidējos izpildes laikus
- ✓ • Lietotājam draudzīgs UI

Paldies par uzmanību!