

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут КНІТ  
Кафедра ПЗ

**ЗВІТ**

До лабораторної роботи № 4  
На тему: "Метод швидкого сортування"  
З дисципліни: "Алгоритми та структури даних"

**Лектор:**  
доцент кафедри ПЗ  
Коротєєва Т.О.

**Виконав:**  
студент групи ПЗ-22  
Коваленко Д.М.

**Прийняв:**  
асистент кафедри ПЗ  
Франко А.В.

«\_\_\_\_\_» \_\_\_\_\_ 2022 р.  
 $\Sigma$  = \_\_\_\_\_

**Тема.** Метод швидкого сортування.

**Мета.** Вивчити алгоритм швидкого сортування. Здійснити програмну реалізацію алгоритму швидкого сортування. Дослідити швидкодію алгоритму швидкого сортування.

## Лабораторне завдання

Створити віконний проект та написати програму, яка реалізує алгоритм сортування Шелла.

5. Задано одновірний масив дійсних чисел. Виключити з нього моду (елемент, який повторюється найчастіше). Отриманий масив посортувати в порядку зростання

## Теоретичні відомості

Швидке сортування - алгоритм сортування, добре відомий, як алгоритм розроблений Чарльзом Хоаром, який не потребує додаткової пам'яті і виконує у середньому  $O(n \cdot \log(n))$  операцій. Оскільки алгоритм використовує дуже прості цикли і операції, він працює швидше інших алгоритмів, що мають таку ж асимптотичну оцінку складності.

В основі алгоритму лежить принцип «розділяй та володарюй» (англійською «Divide and Conquer»). Ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої. Впорядкування кожної з частин відбувається рекурсивно. Алгоритм швидкого сортування може бути реалізований як на масиві, так і на двобічному списку.

Швидке сортування є алгоритмом на основі порівнянь, і не є стабільним. Алгоритм швидкого сортування було розроблено Чарльзом Хоаром у 1962 році під час роботи у маленькій британській компанії Elliott Brothers. В класичному варіанті, запропонованому Хоаром, з масиву обирався один елемент, і весь масив розбивався на дві частини по принципу: в першій частині - ті що не більші даного елемента, в другій частині - ті що не менші даного елемента.

Час роботи алгоритму сортування залежить від збалансованості, що характеризує розбиття. Збалансованість, у свою чергу залежить від того, який елемент обрано як опорний (відносно якого елемента виконується розбиття). Якщо розбиття збалансоване, то асимптотично алгоритм працює так само швидко як і алгоритм сортування злиттям.

Математичне очікування часу роботи алгоритму на всіх можливих вхідних масивах є  $O(n \cdot \log(n))$ , тобто середній випадок ближчий до найкращого.

В середньому алгоритм працює дуже швидко, але на практиці, не всі можливі вхідні масиви мають однакову імовірність. Тоді, шляхом додавання рандомізації вдається отримати середній час роботи в будь-якому випадку. В рандомізованому алгоритмі, при кожному розбитті в якості опорного обирається випадковий елемент

## Покроковий опис роботи алгоритму швидкого сортування

**Алгоритм S - швидке сортування**

**S1** Якщо  $n < 2$ , то перейти до кроку 7.

**S2** Визначити базовий елемент  $P$ .

**S3** Присвоїти змінні  $i = 1$ ,  $j = n$ .

**S4** Цикл. Повторювати крок 5 при  $i < j$ .

**S5** Якщо  $R_i < P$ , то  $i = i + 1$ , якщо  $R_j > P$ , то  $j = j - 1$ , інакше переставити місцями елементи  $R_i$  та  $R_j$ , та присвоїти значення змінним  $i = i + 1$ ,  $j = j - 1$ .

**S6** Рекурсивне сортування частин. Повторити заданий алгоритм для лівої частини  $R_1, \dots, R_{i-1}$  та правої частини  $R_i, \dots, R_n$ .

**S7** Вихід.

## Хід роботи

### Файл sort.rs

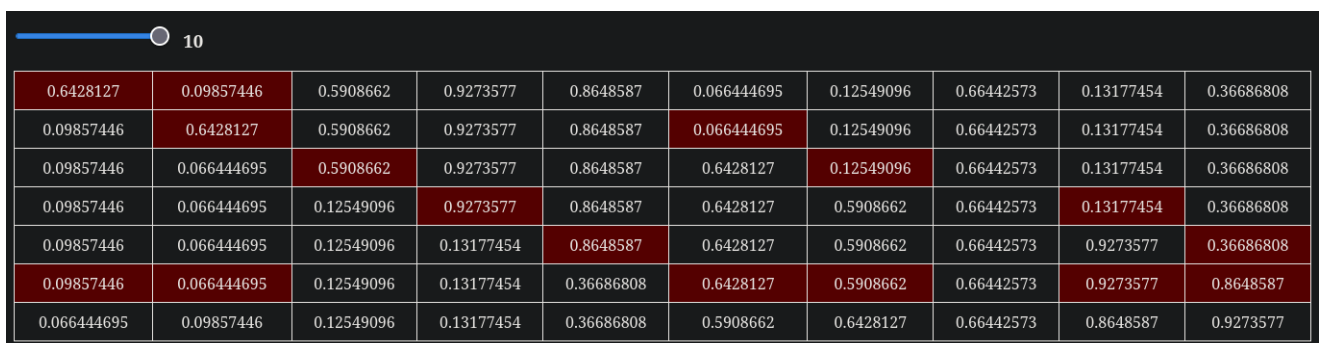
```
use crate::data::Data;

pub fn sort(input: &mut [Data]) -> Vec<Vec<Data>> {
    let mut res = vec![input.to_vec()];
    if input.len() > 1 {
        let pivot = lomuto_partition(&mut input[..], &mut res);
        sort(&mut input[..pivot]);
        sort(&mut input[pivot + 1..]);
    }
    res.push(input.to_vec());
    res
}

fn lomuto_partition(input: &mut [Data], res: &mut Vec<Vec<Data>>) -> usize {
    let pivot = input.len() - 1;
    let mut swap = 0;
    for i in 0..pivot {
        if input[i] < input[pivot] {
            if swap != i {
                input.swap(swap, i);
                res.push(input.to_vec());
            }
            swap += 1;
        }
    }

    if swap != pivot {
        input.swap(swap, pivot);
        res.push(input.to_vec());
    }
    swap
}
```

## Результат роботи



|             |             |            |            |            |             |            |            |            |            |
|-------------|-------------|------------|------------|------------|-------------|------------|------------|------------|------------|
| 0.6428127   | 0.09857446  | 0.5908662  | 0.9273577  | 0.8648587  | 0.066444695 | 0.12549096 | 0.66442573 | 0.13177454 | 0.36686808 |
| 0.09857446  | 0.6428127   | 0.5908662  | 0.9273577  | 0.8648587  | 0.066444695 | 0.12549096 | 0.66442573 | 0.13177454 | 0.36686808 |
| 0.09857446  | 0.066444695 | 0.5908662  | 0.9273577  | 0.8648587  | 0.6428127   | 0.12549096 | 0.66442573 | 0.13177454 | 0.36686808 |
| 0.09857446  | 0.066444695 | 0.12549096 | 0.9273577  | 0.8648587  | 0.6428127   | 0.5908662  | 0.66442573 | 0.13177454 | 0.36686808 |
| 0.09857446  | 0.066444695 | 0.12549096 | 0.13177454 | 0.8648587  | 0.6428127   | 0.5908662  | 0.66442573 | 0.9273577  | 0.36686808 |
| 0.09857446  | 0.066444695 | 0.12549096 | 0.13177454 | 0.36686808 | 0.6428127   | 0.5908662  | 0.66442573 | 0.9273577  | 0.8648587  |
| 0.066444695 | 0.09857446  | 0.12549096 | 0.13177454 | 0.36686808 | 0.5908662   | 0.6428127  | 0.66442573 | 0.8648587  | 0.9273577  |

Рис. 1: Виконання програми

## Висновок

Під час виконання лабораторної роботи я вивчив алгоритм швидкого сортування. Здійснив програмну реалізацію алгоритму швидкого сортування. Дослідив швидкодію алгоритму швидкого сортування.