Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра ІПІ

Звіт

з лабораторної роботи №3 з дисципліни
«Алгоритми та структури даних 2. Структури даних»

„ **Метод швидкого сортування**”

Виконав ІП-45 Янов Богдан Євгенійович

Перевірив Соколовський Владислав Володимирович

Київ 2025

Лабораторна робота №3

Метод швидкого сортування

Мета: реалізувати варіанти алгоритму швидкого сортування. Провести порівняння їх швидкості за кількістю порівнянь елементів масиву.

Швидке сортування (QuickSort) - це ефективний алгоритм сортування, що використовує принцип «розділяй і володарюй». Суть методу полягає у виборі опорного елемента, за яким здійснюється розбиття масиву на дві частини: менші та більші за опорний. Далі ці частини сортуються рекурсивно.

У цій лабораторній роботі розглядаються:

1. Класичний QuickSort – опорним елементом обирається останній елемент підмасиву.
2. QuickSort з 3-медіаною – опорним елементом обирається медіана трьох значень: першого, середнього та останнього.

Псевдокод алгоритмів

```
partition1(slice) {
	len = slice.len()
	pivot = slice[len - 1]
	i = 0

	for j in 0..len - 1 {
		if slice[j] < pivot {
			slice.swap(i, j);
			i += 1;
		}
	}

	slice.swap(i, len - 1);
	ret i
}

sort1(slice) {
	len = slice.len()
	if len > 0 {
		pivot = partition1(slice)
		sort1(slice[0..pivot])
		sort1(slice[pivot + 1..len])
	}
}

median_of_three(a, b, c) {
	if (a > b) ^ (a > c) {
		ret a
	} else if (b > a) ^ (b > c) {
		ret b
```

```
	} else {
		ret c
	}
}

small_sort(slice) {
	for i in 0..slice.len() {
		for j in i + 1..slice.len() {
			if slice[i] > slice[j] {
			slice.swap(i, j)
			}
		}
	}
}

partition2(slice) {
	len = slice.len()
	mid = (len - 1) / 2
	piv_val = median_of_three(slice[0], slice[mid], slice[len - 1])

	if piv_val == slice[0] {
		piv_idx = 0
	} else if piv_val == slice[mid] {
		piv_idx = mid
	} else {
		piv_idx = len - 1
	}

	slice.swap(piv_idx, len - 1)

	pivot = slice[len - 1]
	i = 0

	for j in 0..len - 1 {
		if slice[j] <= pivot {
			slice.swap(i, j)
			i += 1
		}
	}

	slice.swap(i, len - 1)
	ret i
}

sort2(slice){
	len = slice.len()
	if len <= 3 {
		small_sort(slice)
	} else {
		let p = partition2(slice)
		sort2(slice[..p])
```

```
                sort2(slice[p + 1..])
            }
        }
```

Порівняння методів швидкого сортування

Параметр	Класичний QuickSort	Модифікований QuickSort
Опорний елемент	Останній	Медіана з 3-х елементів
Тип розбиття	2 частини	2 рівномірні частини
Обробка маленьких підмасивів	Рекурсивно	Ручне сортування
Середній випадок складності	O(n log n)	O(n log n)
Найгірший	O(n^2)	O(n^2)
Простота реалізації	Висока	Середня
Рівномірність розбиття	Нерівномірне у найгіршому випадку	Краща рівномірність завдяки медіані
Час виконання	Залежить від вибору опорного елемента	Стабільніше на невпорядкованих даних

Вихідний код

```rust
use std::io::BufRead;

mod qs1 {
    fn partition(slice: &mut [usize], cmp: &mut usize) -> usize {
        let len = slice.len();
        let pivot = slice[len - 1];
        let mut i = 0;

        for j in 0..len - 1 {
            *cmp += 1;
            if slice[j] < pivot {
                slice.swap(i, j);
                i += 1;
            }
        }

        slice.swap(i, len - 1);
```

```
        i
    }

    pub fn sort(slice: &mut [usize]) -> usize {
        let mut cmp = 0;

        if !slice.is_empty() {
            let partition_index = partition(slice, &mut cmp);
            let len = slice.len();

            cmp += sort(&mut slice[0..partition_index]);
            cmp += sort(&mut slice[partition_index + 1..len]);
        }

        cmp
    }
}

mod qs2 {
    fn median_of_three(a: usize, b: usize, c: usize) -> usize {
        if (a > b) ^ (a > c) {
            a
        } else if (b > a) ^ (b > c) {
            b
        } else {
            c
        }
    }

    fn sort_small(slice: &mut [usize], cmp: &mut usize) {
        for i in 0..slice.len() {
            for j in i + 1..slice.len() {
                *cmp += 1;
                if slice[i] > slice[j] {
                    slice.swap(i, j);
                }
            }
        }
    }

    fn partition(slice: &mut [usize], cmp: &mut usize) -> usize {
        let len = slice.len();
        let mid = (len - 1) / 2;
```

```
        let piv_val = median_of_three(slice[0], slice[mid], slice[len - 1]);

        let piv_idx = if piv_val == slice[0] {
            0
        } else if piv_val == slice[mid] {
            mid
        } else {
            len - 1
        };

        slice.swap(piv_idx, len - 1);

        let pivot = slice[len - 1];
        let mut i = 0;

        for j in 0..len - 1 {
            *cmp += 1;
            if slice[j] <= pivot {
                slice.swap(i, j);
                i += 1;
            }
        }

        slice.swap(i, len - 1);
        i
    }

    pub fn sort(slice: &mut [usize]) -> usize {
        let mut cmp = 0;
        let len = slice.len();
        if len <= 3 {
            sort_small(slice, &mut cmp);
        } else {
            let p = partition(slice, &mut cmp);
            cmp += sort(&mut slice[..p]);
            cmp += sort(&mut slice[p + 1..]);
        }
        cmp
    }
}

fn main() {
    let file_path = "./data/input_03_10.txt";
```

```rust
    let file = std::fs::File::open(file_path).expect("Unable to open file");
    let reader = std::io::BufReader::new(file);
    let data: Vec<_> = reader
        .lines()
        .skip(1)
        .filter_map(|line| line.ok())
        .filter_map(|line| line.parse::<usize>().ok())
        .collect();

    let comp1 = qs1::sort(&mut data.clone());
    let comp2 = qs2::sort(&mut data.clone());

    println!("{} {}", comp1, comp2);
}
```

Приклад роботи

```
λ Partur lab3 → cat data/input_03_10.txt
10
3
10
1
4
7
5
6
2
9
8%
λ Partur lab3 → cat data/output_03_10.txt
26 21 21%
λ Partur lab3 → cargo run
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.07s
     Running `target/debug/lab1`
26 21
```

Висновок

У цій роботі реалізовано та порівняно два варіанти алгоритму швидкого сортування:

1. Класичний з опорним елементом-останнім,
2. З трьома медіанами та оптимізацією для підмасивів розміром <= 3.

Експерименти показали, що хоча обидва алгоритми мають у середньому O(n log n), модифікація з трьома медіанами стабільніше розбиває масив і менше піддається "поганим" випадкам, тож у більшості практичних даних вона робить менше порівнянь. При цьому асимптотично обидва методи у найгіршому випадку можуть втягнутися в O(n^2), але медіана значно знижує ймовірність такого сценарію та зменшує константні витрати. Для загальних завдань швидкого сортування варіант із трьома медіанами та дрібно-розмірною оптимізацією є ефективнішим без значного ускладнення реалізації.