Министерство цифрового развития, связи и массовых коммуникаций

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Отчет по лабораторной работе №1

по дисциплине «Структура и алгоритмы обработки данных»

по теме «Методы сортировки»

Выполнил: студент группы

БВТ1902

Подпоркин В.С.

Проверил:

Москва

2021 г.

Оглавление

Цель работы

Разобрать принцип работы методов сортировки, используя генератор матриц с параметрами.

Задания

1. Написать генератор случайных матриц(многомерных), который принимает опциональные параметры m, n, min_limit, max_limit, где m и n указывают размер матрицы, а min_lim и max_lim - минимальное и максимальное значение для генерируемого числа. По умолчанию при отсутствии параметров принимать следующие значения:

   m = 50

   n = 50

   min_limit = -50

   max_limit = 50

2. Реализовать методы сортировки строк числовой матрицы в соответствии с заданием. Оценить время работы каждого алгоритма сортировки и сравнить его со временем стандартной функции сортировки. Испытания проводить на сгенерированных матрицах.

Методы:

Выбором	Вставкой	Обменом	Шелла	Быстрая сортировка	Пирамидальная

Код программы

Программа выполнена на языке Rust.

```rust
pub fn sort_matrix<F: FnMut(&mut [i32])>(matrix: &mut Vec<Vec<i32>>, mut sorter:
F) {
    matrix.into_iter().for_each(|x| sorter(x.as_mut_slice()))
}

/// Сортировка выбором
pub fn select_sort<T: Ord>(arr: &mut [T]) {
    for i in 0..(arr.len() - 1) {
        let min_idx = {
            let mut min_idx = i;
            let mut min_val = &arr[i];
            for i in (i + 1)..arr.len() {
                let x = &arr[i];
                if x < min_val {
                    min_idx = i;
                    min_val = x;
                }
```

```
            }
            min_idx
        };
        arr.swap(i, min_idx)
    }
}

/// Сортировка вставками
pub fn insert_sort<T: Ord>(arr: &mut [T]) {
    for i in 0..arr.len() {
        let mut n = i;
        while n > 0 && arr[n] < arr[n - 1] {
            arr.swap(n, n - 1);
            n -= 1;
        }
    }
}

/// Сортировка Шелла
pub fn shell_sort<T: Ord>(arr: &mut [T]) {
    let len = arr.len();
    let mut s = len / 2;
    while s > 0 {
        for i in 0..len {
            for j in ((i + s)..len).step_by(s) {
                if arr[i] > arr[j] {
                    arr.swap(i, j);
                }
            }
        }
        s /= 2;
    }
}

/// Сортировка пузырьковая
pub fn bubble_sort<T: Ord>(arr: &mut [T]) {
    for lim in (0..arr.len()).rev() {
        for i in 0..lim {
            if arr[i] > arr[i + 1] {
                arr.swap(i, i + 1)
            }
        }
    }
}

// Heapsort
pub fn heap_sort<T: Ord>(arr: &mut [T]) {
    // Преобразуем массив в сортирующее дерево
    let end = arr.len();
    // Пропускаем последние end/2 элементов (листы дерева не имеет смысла
перемещать между собой)
    for start in (0..end / 2).rev() {
        sift_down(arr, start, end - 1);
```

```rust
    }

    // Сортировка сортирующего дерева
    for end in (1..arr.len()).rev() {
        // Самый большой элемент считаем отсортированным. Перемещаем его в конец
        // и исключаем из дерева
        arr.swap(end, 0);
        // "всплывает" следующий наибольший элемент
        sift_down(arr, 0, end - 1);
    }
}

// "поднимаем" наибольший элемент из дочерних в позицию start
fn sift_down<T: Ord>(arr: &mut [T], start: usize, end: usize) {
    let mut root = start;
    loop {
        let mut child = root * 2 + 1; // Получаем левого ребенка
        if child > end {
            break;
        }
        if child < end && arr[child] < arr[child + 1] {
            // Если правый ребенок существует и больший
            child += 1;
        }

        if arr[root] < arr[child] {
            // Если ребенок не меньше корня, меняем их
            arr.swap(root, child);
            root = child;
        } else {
            break;
        }
    }
}

/// Сортировка турнирная
pub fn turn_sort<T: Ord>(arr: &mut [T]) {
    unimplemented!()
}

/// Сортировка быстрая
pub fn quick_sort<T: Ord + Copy>(arr: &mut [T]) {
    let vec = arr.iter().map(|x| *x).collect::<Vec<_>>();
    let result = quick_sort_inner(vec);
    for i in 0..result.len() {
        arr[i] = result[i]
    }
}

pub fn quick_sort_inner<T: Ord + Copy>(arr: Vec<T>) -> Vec<T> {
    if arr.len() <= 1 {
        return arr;
    }
```

```rust
    let mut less = Vec::new();
    let mut equal = Vec::new();
    let mut great = Vec::new();
    let cmpr = arr[arr.len() / 2];

    for x in arr {
        if x < cmpr {
            less.push(x);
        }
        if x == cmpr {
            equal.push(x);
        }
        if x > cmpr {
            great.push(x);
        }
    }

    let mut less = quick_sort_inner(less);
    let mut great = quick_sort_inner(great);
    less.append(&mut equal);
    less.append(&mut great);
    less
}

pub fn gen_matrix(m: usize, n: usize, min: i32, max: i32) -> Vec<Vec<i32>> {
    let mut rng = Rng::new(
        std::time::SystemTime::now()
            .duration_since(UNIX_EPOCH)
            .map(|dur| dur.as_secs())
            .unwrap_or(173),
    );

    (0..m)
        .map(|_| (0..n).map(|_| rng.gen_i32_in(min, max)).collect())
        .collect()
}

pub fn print_matrix(matrix: &Vec<Vec<i32>>) {
    matrix.into_iter().for_each(|line| {
        line.into_iter().for_each(|x| print!("{} ", x));
        println!()
    });
    println!()
}

fn main() {
    let a = 2;
    let b = 12;

    let print_matrix = false;

    let mut matrix = n2::gen_matrix(200, 5000, -50000, 50000);
```

```rust
println!("Исходная матрица");
if print_matrix {
    n2::print_matrix(&matrix);
}

let start = std::time::Instant::now();
n3::sort_matrix(&mut matrix, |arr| arr.sort());
println!("Сортировка стандартной библиотеки");
println!(
    "{} ms",
    std::time::Instant::now().duration_since(start).as_millis()
);
if print_matrix {
    n2::print_matrix(&matrix);
}

let start = std::time::Instant::now();
n3::sort_matrix(&mut matrix, n3::select_sort);
println!("Сортировка выбором");
println!(
    "{} ms",
    std::time::Instant::now().duration_since(start).as_millis()
);
if print_matrix {
    n2::print_matrix(&matrix);
}

let start = std::time::Instant::now();
n3::sort_matrix(&mut matrix, n3::insert_sort);
println!("Сортировка вставками");
println!(
    "{} ms",
    std::time::Instant::now().duration_since(start).as_millis()
);
if print_matrix {
    n2::print_matrix(&matrix);
}

let start = std::time::Instant::now();
n3::sort_matrix(&mut matrix, n3::bubble_sort);
println!("Сортировка пузырьковая");
println!(
    "{} ms",
    std::time::Instant::now().duration_since(start).as_millis()
);
if print_matrix {
    n2::print_matrix(&matrix);
}

let start = std::time::Instant::now();
n3::sort_matrix(&mut matrix, n3::shell_sort);
println!("Сортировка Шелла");
println!(
```

```
        "{} ms",
        std::time::Instant::now().duration_since(start).as_millis()
    );
    if print_matrix {
        n2::print_matrix(&matrix);
    }

    let start = std::time::Instant::now();
    n3::sort_matrix(&mut matrix, n3::heap_sort);
    println!("Сортировка пирамидальная");
    println!(
        "{} ms",
        std::time::Instant::now().duration_since(start).as_millis()
    );
    if print_matrix {
        n2::print_matrix(&matrix);
    }

    let start = std::time::Instant::now();
    n3::sort_matrix(&mut matrix, n3::quick_sort);
    println!("Сортировка быстрая");
    println!(
        "{} ms",
        std::time::Instant::now().duration_since(start).as_millis()
    );
    if print_matrix {
        n2::print_matrix(&matrix);
    }
}
```

Снимки экрана работы программы

```
Сортировка стандартной библиотеки
0 ms
-43 -43 -43 -43 -42 -39 -35 -31 -30 -29 -26 -26 -26 -24 -24 -14 -12 -12 -9 -9 -6 -6 -5 0 3 5 6 7 7 9 10 10 12 13 15 16 19 21 22 29 31 31 34 34 35 36 37 39 48 48
-50 -48 -41 -40 -39 -36 -35 -34 -29 -26 -25 -25 -15 -14 -11 -10 -9 -9 -5 -4 -1 -1 0 2 6 6 6 8 8 11 12 14 18 19 20 25 26 26 31 31 32 35 35 38 38 45 45 46 49 49
-46 -38 -34 -31 -28 -28 -26 -26 -25 -25 -23 -21 -21 -20 -20 -15 -10 -8 -7 -7 -7 -7 -6 -4 -3 -2 -2 0 1 6 8 12 15 18 20 24 25 25 27 28 29 33 33 35 41 42 43 43 46 47
-50 -48 -42 -42 -41 -41 -36 -33 -31 -30 -30 -26 -22 -21 -20 -16 -15 -12 -10 -10 -10 -10 -5 -3 2 4 7 7 9 10 12 13 15 15 16 19 22 27 29 29 31 33 33 34 35 37 37 40 41 45 46
-49 -48 -47 -42 -31 -31 -31 -30 -23 -19 -18 -18 -16 -14 -12 -12 -11 -9 -8 -6 -4 -1 0 0 16 17 18 19 20 20 22 23 24 24 28 28 34 35 39 39 40 40 41 42 43 44 47 48 50 50
-49 -44 -43 -37 -37 -32 -27 -25 -25 -24 -20 -17 -15 -15 -14 -13 -13 -11 -11 -10 -6 3 7 9 11 14 15 16 17 19 21 22 22 23 25 27 27 29 30 32 34 35 36 36 38 40 42 44 47 50
-48 -47 -46 -43 -40 -38 -35 -33 -31 -30 -28 -27 -25 -24 -20 -14 -14 -11 -11 -10 -7 -6 -6 -6 -5 -2 -1 0 0 3 5 10 11 11 14 15 21 24 26 29 29 29 29 34 36 41 41 41 44 48
-47 -47 -37 -36 -32 -31 -28 -26 -25 -24 -22 -21 -18 -16 -13 -10 -10 -8 -2 -2 -2 2 3 3 4 4 7 9 13 15 18 21 23 23 26 26 27 27 28 31 32 33 34 35 37 40 40 43 45 46
-47 -46 -40 -40 -39 -39 -37 -34 -29 -27 -26 -23 -20 -19 -14 -12 -11 -10 -8 -7 -6 -6 -5 1 3 5 7 8 8 9 11 12 13 15 17 19 21 21 21 26 27 30 31 31 35 37 38 39 46 47
-50 -50 -50 -49 -49 -46 -46 -42 -41 -40 -39 -35 -33 -31 -26 -19 -18 -18 -16 -13 -9 -9 -8 -6 -5 -4 -2 3 4 4 4 8 14 16 18 19 20 20 20 22 23 23 25 25 31 33 34 42 47 50
-50 -49 -47 -47 -47 -46 -45 -42 -40 -37 -37 -35 -31 -29 -29 -24 -23 -23 -22 -19 -18 -18 -17 -14 -13 -12 -10 -7 -5 -3 -1 1 10 15 17 18 19 19 22 24 32 33 35 35 41 45 46 47 49 50
-42 -42 -39 -39 -36 -35 -34 -34 -34 -33 -32 -31 -28 -27 -26 -25 -24 -21 -18 -17 -15 -13 -5 -4 -3 -2 -1 -1 4 5 7 12 12 14 15 16 21 23 24 25 29 38 38 38 39 44 46 47 48 49
-49 -48 -46 -45 -45 -43 -42 -37 -36 -30 -29 -28 -24 -22 -21 -20 -19 -17 -16 -14 -8 -6 -5 -5 -4 -4 -2 -1 2 4 4 7 16 17 22 31 31 32 33 34 35 38 40 44 44 46 46 47 48 50
-50 -50 -50 -48 -47 -44 -43 -38 -38 -38 -30 -27 -26 -25 -22 -21 -20 -19 -16 -13 -9 -4 0 1 2 6 9 10 15 20 20 21 23 25 25 26 28 31 32 32 35 36 38 41 42 44 47 47 48 50
-49 -43 -43 -41 -41 -40 -40 -40 -38 -36 -35 -32 -26 -26 -24 -24 -13 -13 -13 -11 -10 -9 -9 -6 -6 -2 -1 0 4 6 7 7 8 10 12 20 20 28 32 34 34 38 40 40 43 44 44 46 46 48
-49 -46 -46 -44 -43 -41 -37 -35 -35 -34 -34 -28 -27 -27 -25 -25 -23 -23 -20 -16 -10 -9 -8 -5 2 3 3 7 9 9 9 10 13 14 17 20 21 25 28 28 35 35 36 38 43 43 43 49 49 50
-47 -45 -45 -43 -39 -38 -37 -32 -31 -31 -28 -26 -25 -24 -22 -15 -14 -14 -12 -7 -6 -5 -4 -4 -2 2 2 7 8 9 11 15 17 20 22 22 27 27 30 30 31 31 34 35 38 43 45 46 46 50
-48 -46 -45 -43 -41 -34 -32 -28 -25 -23 -21 -19 -16 -13 -12 -9 -3 -3 -1 1 1 2 4 4 5 6 7 9 10 12 14 18 21 23 24 25 26 26 29 30 32 35 37 38 41 43 44 45 47 50
-49 -46 -43 -41 -39 -36 -35 -31 -31 -31 -28 -26 -25 -24 -23 -21 -20 -20 -20 -18 -17 -17 -16 -11 -9 -5 -3 -3 -2 3 5 6 11 13 13 15 17 18 26 32 32 33 38 38 39 41 45 47 47 50
-48 -41 -40 -39 -39 -38 -34 -32 -30 -30 -29 -24 -24 -24 -21 -20 -15 -14 -11 -10 -9 -8 -6 -4 -3 -1 1 6 9 11 13 19 19 20 22 27 28 29 31 36 36 39 44 46 46 46 48 48 48 49
-50 -50 -50 -44 -43 -41 -40 -39 -38 -34 -34 -32 -29 -29 -29 -27 -26 -25 -25 -24 -22 -22 -21 -18 -18 -14 -11 -10 -8 -5 3 7 8 16 20 21 26 26 28 32 33 33 36 37 37 37 39 45 47 47
-48 -46 -42 -42 -41 -40 -35 -33 -31 -28 -23 -22 -22 -17 -15 -13 -12 -10 -6 -5 -3 3 6 7 7 11 12 12 12 13 18 18 24 24 27 28 29 32 35 36 36 36 37 39 41 45 46 47 48 50
-49 -47 -46 -43 -39 -39 -38 -38 -37 -37 -36 -33 -33 -31 -30 -28 -26 -25 -24 -24 -23 -20 -18 -17 -17 -17 -17 -13 -11 -9 -8 -8 -7 1 3 4 8 9 9 11 11 12 17 17 25 25 30 31 36 39
-49 -46 -44 -42 -40 -39 -39 -37 -34 -34 -33 -30 -30 -24 -19 -18 -16 -14 -10 -5 -4 -4 -3 -2 -1 1 6 12 14 14 14 17 17 18 20 21 21 22 24 25 33 34 36 37 38 40 42 43 43 46
-47 -44 -42 -37 -37 -37 -36 -36 -34 -33 -33 -32 -29 -29 -28 -26 -24 -24 -23 -20 -11 -11 -7 -5 1 4 4 5 12 17 20 24 27 29 29 31 31 32 35 36 38 39 39 39 40 43 44 45 47 48
-47 -46 -42 -42 -41 -38 -38 -37 -36 -34 -32 -26 -25 -25 -23 -21 -20 -17 -16 -15 -13 -11 -9 -6 -5 -4 -4 2 6 8 11 11 16 18 21 22 24 26 26 26 28 31 35 37 41 42 44 45 49 49
-47 -47 -46 -45 -44 -43 -42 -32 -31 -29 -28 -27 -27 -26 -26 -24 -21 -19 -15 -13 -13 -10 -9 -9 -7 -3 -2 0 3 15 17 20 28 28 28 31 32 33 33 35 40 40 43 43 44 45 46 47 48 48
-50 -48 -47 -46 -42 -41 -39 -37 -36 -35 -35 -34 -33 -31 -29 -28 -24 -21 -20 -18 -16 -15 -13 -8 -8 -5 0 2 2 3 5 5 5 16 20 22 24 25 32 34 35 35 38 39 40 40 44 44 47 48
-47 -45 -38 -37 -34 -33 -32 -32 -26 -23 -22 -22 -21 -21 -21 -20 -17 -15 -14 -13 -11 -10 -10 -5 -2 -1 3 4 8 14 14 17 18 23 23 24 25 28 30 35 35 36 36 37 38 39 43 45 48 49
-46 -41 -41 -39 -38 -35 -34 -34 -33 -31 -31 -30 -28 -21 -20 -19 -17 -15 -14 -10 -10 -9 -9 -6 -5 -2 -2 0 6 7 7 8 9 11 19 20 27 27 30 35 38 38 42 42 43 45 47 49 49 50
-50 -50 -41 -40 -39 -36 -36 -27 -24 -23 -20 -16 -15 -12 -10 -7 0 0 1 3 3 6 9 9 10 11 11 15 17 17 17 19 22 31 33 35 35 36 37 40 42 42 43 43 45 45 46 50 50 50
-48 -45 -43 -41 -40 -38 -34 -30 -29 -29 -28 -25 -21 -21 -21 -19 -19 -18 -14 -14 -13 -13 -13 -11 -9 -7 -5 2 3 3 3 5 8 9 14 16 22 22 24 25 28 29 31 31 31 31 40 46 49 50
-49 -49 -47 -47 -42 -40 -36 -35 -29 -25 -22 -22 -20 -18 -18 -17 -13 -13 -12 -6 -6 0 1 2 4 6 14 15 17 21 22 22 26 26 27 29 29 30 34 36 35 38 38 39 39 42 43 45 49 50
```

Рисунок 1 – Пример вывода стандартной сортировки.

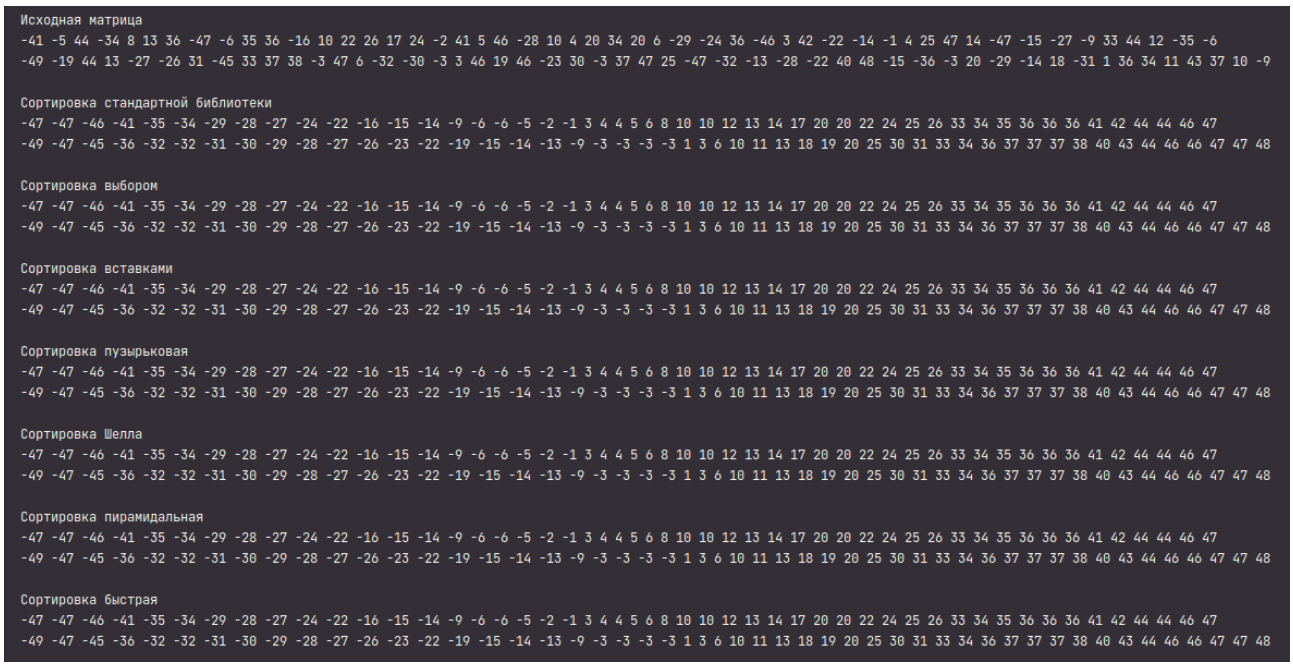

Рисунок 2 – Демонстрация корректности всех видов сортировок на небольшой выборке

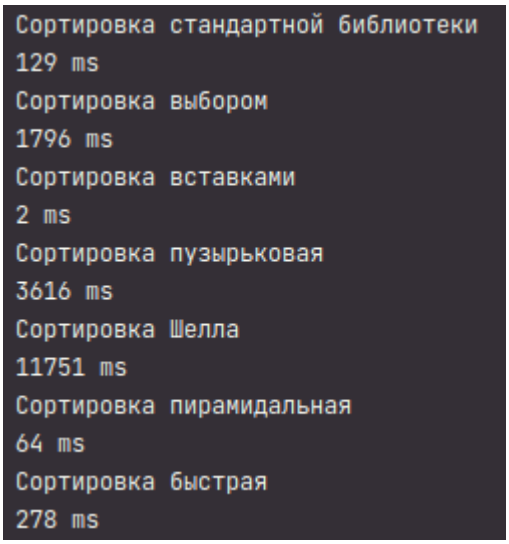

Рисунок 3 – Замер работы всех алгоритмов на большом наборе данных

Вывод

Я получил представление о методах сортировки данны с генерацией матриц с параметрами на языке Rust.