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 3](#__RefHeading___Toc6883_2819935426)

[Задания 3](#__RefHeading___Toc6885_2819935426)

[Код программы 3](#__RefHeading___Toc6887_2819935426)

[Снимки экрана работы программы 9](#__RefHeading___Toc6889_2819935426)

[Вывод 10](#__RefHeading___Toc6891_2819935426)

## 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

1. 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

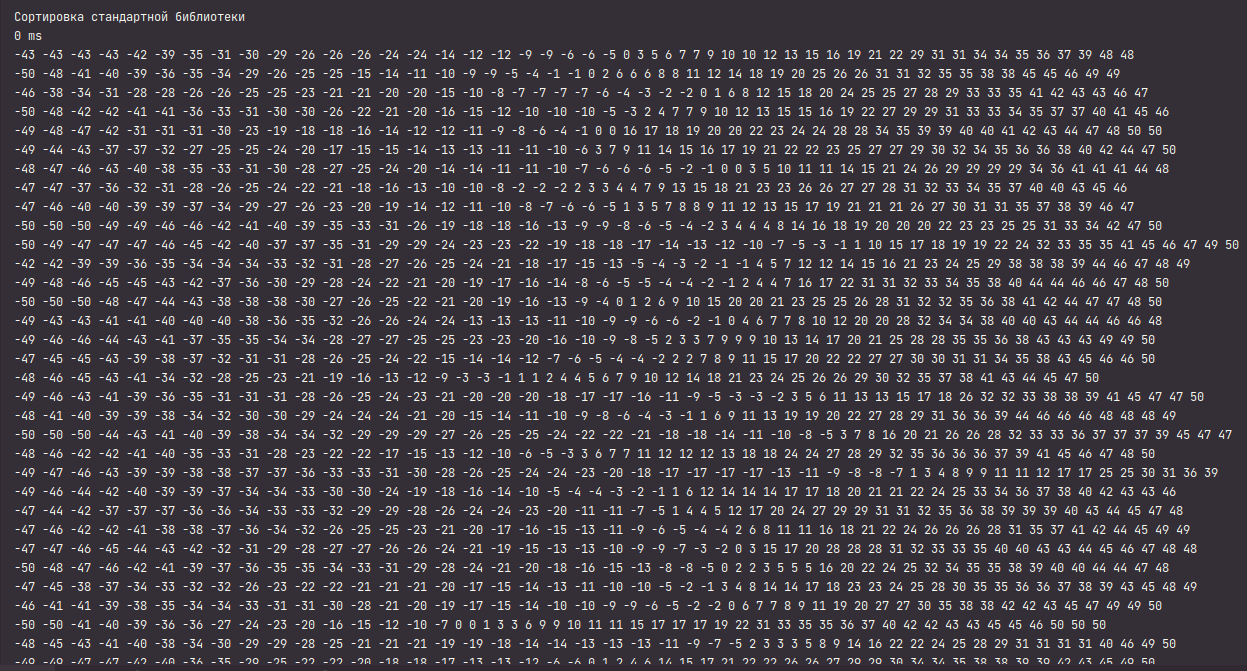
Программа выполнена на языке Rust.

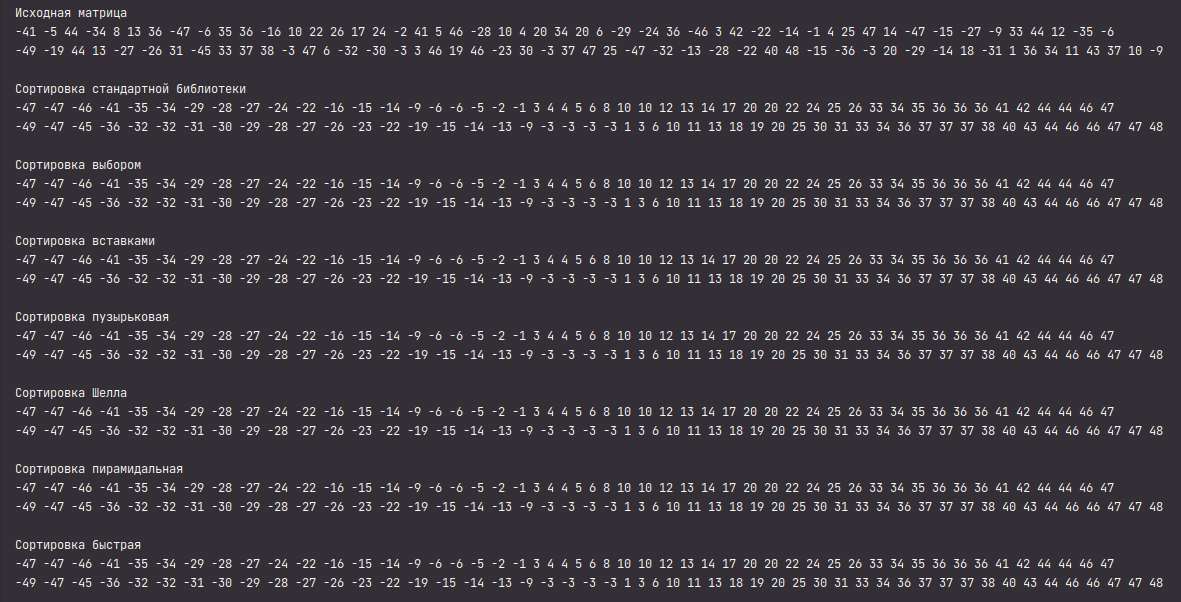
pub fn sort\_matrix<F: FnMut(&mut [i32])>(matrix: &mut Vec<Vec<i32>>, mut sorter: F) {  
 matrix.into\_iter().for\_each(|x| sorter(x.as\_mut\_slice()))  
}  
  
*/// Сортировка выбором*  
pub fn select\_sort<T: Ord>(arr: &mut [T]) {  
 for i in 0..(arr.len() - 1) {  
 let min\_idx = {  
 let mut min\_idx = 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);  
 }  
  
 *// 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;  
 }  
  
 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  
}

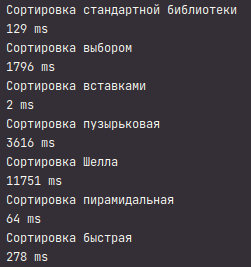
pub fn gen\_matrix(m: usize, n: usize, min: i32, max: i32) -> Vec<Vec<i32>> {  
 let mut rng = Rng::*new*(  
 std::time::SystemTime::*now*()  
 .duration\_since(*UNIX\_EPOCH*)  
 .map(|dur| dur.as\_secs())  
 .unwrap\_or(173),  
 );  
  
 (0..m)  
 .map(|\_| (0..n).map(|\_| rng.gen\_i32\_in(min, max)).collect())  
 .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);  
 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

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.