1-Implement a function that checks whether a given string is a palindrome or not.

fn is\_palindrome(s: &str) -> bool {

let s = s.trim().to\_lowercase();

let len = s.len();

for i in 0..len / 2 {

if s.chars().nth(i) != s.chars().nth(len - i - 1) {

return false;

}

}

true

}

fn main() {

let test\_string1 = "radar";

let test\_string2 = "hello";

println!("Is '{}' a palindrome? {}", test\_string1, is\_palindrome(test\_string1));

println!("Is '{}' a palindrome? {}", test\_string2, is\_palindrome(test\_string2));

}

2-Given a sorted array of integers, implement a function that returns the index of the first occurrence of a given number.

fn first\_occurrence(arr: &[i32], target: i32) -> Option<usize> {

for (index, &num) in arr.iter().enumerate() {

if num == target {

return Some(index);

}

}

None

}

fn main() {

let arr = vec![1, 2, 3, 3, 4, 5, 6, 6, 7];

let target = 3;

match first\_occurrence(&arr, target) {

Some(index) => println!("The first occurrence of {} is at index {}", target, index),

None => println!("{} not found in the array", target),

}

}

3-Given a string of words, implement a function that returns the shortest word in the string

fn shortest\_word(s: &str) -> Option<&str> {

s.split\_whitespace().min\_by\_key(|word| word.len())

}

fn main() {

let sentence = "The quick brown fox jumps over the lazy dog";

match shortest\_word(sentence) {

Some(shortest) => println!("The shortest word is: {}", shortest),

None => println!("No words found"),

}

}

4-Implement a function that checks whether a given number is prime or not.

fn is\_prime(n: u64) -> bool {

if n <= 1 {

return false; // 0 and 1 are not prime numbers

}

let sqrt\_n = (n as f64).sqrt() as u64;

for i in 2..=sqrt\_n {

if n % i == 0 {

return false; // If n is divisible by any number from 2 to sqrt(n), it's not prime

}

}

true // If no divisor is found, n is prime

}

fn main() {

let num1 = 17;

let num2 = 21;

println!("Is {} prime? {}", num1, is\_prime(num1));

println!("Is {} prime? {}", num2, is\_prime(num2));

}

5-Given a sorted array of integers, implement a function that returns the median of the array.

fn find\_median(arr: &[i32]) -> Option<f64> {

let len = arr.len();

if len == 0 {

return None;

}

let mid = len / 2;

if len % 2 == 0

{

Some((arr[mid - 1] + arr[mid]) as f64 / 2.0)

} else {

// If the number of elements is odd, return the middle element

Some(arr[mid] as f64)

}

}

fn main() {

let arr1 = vec![1, 2, 3, 4, 5];

let arr2 = vec![1, 2, 3, 4, 5, 6];

println!("Median of {:?}: {:?}", arr1, find\_median(&arr1));

println!("Median of {:?}: {:?}", arr2, find\_median(&arr2));

}

6-Implement a function that finds the longest common prefix of a given set of strings.

fn longest\_common\_prefix(strings: &[String]) -> String {

if strings.is\_empty() {

return String::new();

}

let first\_string = &strings[0];

for (index, char) in first\_string.chars().enumerate() {

for string in strings.iter().skip(1) {

if index >= string.len() || string.chars().nth(index) != Some(char) {

return first\_string[..index].to\_string();

}

}

}

first\_string.to\_string()

}

fn main() {

let strings = vec![

String::from("flower"),

String::from("flow"),

String::from("flight"),

];

println!("Longest common prefix: {}", longest\_common\_prefix(&strings));

}

7-Implement a function that returns the kth smallest element in a given array.

fn kth\_smallest(arr: &[i32], k: usize) -> Option<i32> {

if k > arr.len() {

return None;

}

let mut sorted\_arr = arr.to\_vec();

sorted\_arr.sort();

Some(sorted\_arr[k - 1])

}

fn main() {

let arr = [4, 2, 7, 1, 9, 5];

let k = 3;

match kth\_smallest(&arr, k) {

Some(smallest) => println!("The {}th smallest element is: {}", k, smallest),

None => println!("Array is empty or k is out of bounds"),

}

}

8-Given a binary tree, implement a function that returns the maximum depth of the tree.

use std::cmp;

// Definition for a binary tree node.

#[derive(Debug)]

struct TreeNode {

val: i32,

left: Option<Box<TreeNode>>,

right: Option<Box<TreeNode>>,

}

impl TreeNode {

fn new(val: i32) -> Self {

TreeNode { val, left: None, right: None }

}

}

fn max\_depth(root: Option<Box<TreeNode>>) -> i32 {

match root {

Some(node) => {

let left\_depth = max\_depth(node.left);

let right\_depth = max\_depth(node.right);

cmp::max(left\_depth, right\_depth)+1

}

None => 0,

}

}

fn main() {

// Construct a sample binary tree

let mut root = TreeNode::new(3);

let mut left\_child = TreeNode::new(9);

let mut right\_child = TreeNode::new(20);

let right\_left\_child = TreeNode::new(15);

let right\_right\_child = TreeNode::new(7);

right\_child.left = Some(Box::new(right\_left\_child));

right\_child.right = Some(Box::new(right\_right\_child));

root.left = Some(Box::new(left\_child));

root.right = Some(Box::new(right\_child));

let depth = max\_depth(Some(Box::new(root)));

println!("Maximum depth of the binary tree: {}", depth);

}

9-Reverse a string in Rust

fn reverse\_string(input: &str) -> String {

let mut chars: Vec<char> = input.chars().collect();

chars.reverse();

chars.into\_iter().collect()

}

fn main() {

let original = "Hello, world!";

let reversed = reverse\_string(original);

println!("Original: {}", original);

println!("Reversed: {}", reversed);

}

10-Check if a number is prime in Rust

fn is\_prime(n: u64) -> bool {

if n <= 1 {

return false; //

}

let sqrt\_n = (n as f64).sqrt() as u64;

for i in 2..=sqrt\_n {

if n % i == 0 {

return false;

}

}

true // If no divisor is found, n is prime

}

fn main() {

let num1 = 17;

let num2 = 21;

println!("Is {} prime? {}", num1, is\_prime(num1));

println!("Is {} prime? {}", num2, is\_prime(num2));

}

11-Merge two sorted arrays in Rust

fn merge\_sorted\_arrays(arr1: &[i32], arr2: &[i32]) -> Vec<i32> {

let mut merged = Vec::with\_capacity(arr1.len() + arr2.len());

let (mut i, mut j) = (0, 0);

while i < arr1.len() && j < arr2.len() {

if arr1[i] < arr2[j] {

merged.push(arr1[i]);

i += 1;

} else {

merged.push(arr2[j]);

j += 1;

}

}

merged.extend\_from\_slice(&arr1[i..]);

merged.extend\_from\_slice(&arr2[j..]);

merged

}

fn main() {

let arr1 = [1, 3, 5, 7, 9];

let arr2 = [2, 4, 6, 8, 10];

let merged = merge\_sorted\_arrays(&arr1, &arr2);

println!("Merged array: {:?}", merged);

}

12-Find the maximum subarray sum in Rust

fn max\_subarray\_sum(arr: &[i32]) -> i32 {

let mut max\_sum = arr[0];

let mut current\_sum = arr[0];

for &num in arr.iter().skip(1) {

current\_sum = num.max(current\_sum + num);

max\_sum = max\_sum.max(current\_sum);

}

max\_sum

}

fn main() {

let arr = [-2, 1, -3, 4, -1, 2, 1, -5, 4];

let max\_sum = max\_subarray\_sum(&arr);

println!("Maximum subarray sum: {}", max\_sum);

}