1. Check if a given string is a palindrome:

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

let reversed = s.chars().rev().collect::<String>();

s == reversed

}

1. Find the index of the first occurrence of a given number in a sorted array:

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

arr.iter().position(|&x| x == target)

}

1. Find the shortest word in a string:

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

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

}

1. Check if a given number is prime:

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

if n <= 1 {

return false;

}

for i in 2..=(n as f64).sqrt() as u64 {

if n % i == 0 {

return false;

}

}

true

}

1. Find the median of a sorted array:

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

let mid = arr.len() / 2;

if arr.len() % 2 == 0 {

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

} else {

arr[mid] as f64

}

}

1. Find the longest common prefix of a set of strings:

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

if strings.is\_empty() {

return String::new();

}

let first\_string = &strings[0];

let mut prefix = String::new();

for (i, c) in first\_string.chars().enumerate() {

if strings.iter().all(|s| s.chars().nth(i) == Some(c)) {

prefix.push(c);

} else {

break;

}

}

prefix

}

1. Find the kth smallest element in an array:

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

if k > arr.len() {

return None;

}

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

sorted.sort();

Some(sorted[k - 1])

}

1. Find the maximum depth of a binary tree:

// Assuming binary tree is represented using the following struct

#[derive(Debug)]

struct TreeNode {

val: i32,

left: Option<Box<TreeNode>>,

right: Option<Box<TreeNode>>,

}

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

match root {

Some(node) => {

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

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

1 + left\_depth.max(right\_depth)

}

None => 0,

}

}

1. Reverse a string:

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

s.chars().rev().collect()

}

1. Check if a number is prime:

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

if n <= 1 {

return false;

}

for i in 2..=(n as f64).sqrt() as u64 {

if n % i == 0 {

return false;

}

}

true

}

1. Merge two sorted arrays:

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

}

1. Find the maximum subarray sum:

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

let mut max\_sum = 0;

let mut current\_sum = 0;

for &num in arr {

current\_sum = current\_sum.max(0) + num;

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

}

max\_sum

}