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

```
fn is_palindrome(s: &str) -> bool {
    let mut start = 0;
    let mut end = s.len() - 1;
    while start < end {
        if s[start..=start] != s[end..=end] {
            return false;
        }
        start += 1;
        end -= 1;
    }
    true
}
fn main() {
    let s1 = "rotator";
    let s2 = "Pradumn";
    println!("{} is palindrome? {}", s1, is_palindrome(s1));
    println!("{} is palindrome? {}", s2, is_palindrome(s2));
}
```

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

```
fn find_first_occurrence(arr: &[i32], num: i32) -> Option<usize> {
    let mut left = 0;
    let mut right = arr.len() - 1;
    while left <= right {
        let mid = left + (right - left) / 2;
        if arr[mid] == num && (mid == 0 || arr[mid - 1] < num) {
            return Some(mid);
        } else if arr[mid] >= num {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }
    None
}
fn main() {
    let arr = [1, 2, 2, 3, 4, 4, 4, 5, 5];
    let num = 4;
    match find_first_occurrence(&arr, num) {
        Some(idx) => println!("{} found at index {}", num, idx),
        None => println!("{} not found in array", num),
    }
}
```

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

```
fn shortest_word(s: &str) -> Option<&str> {
    let mut shortest = None;
    for word in s.split_whitespace() {
        match shortest {
            None => shortest = Some(word),
            Some(w) => {
                if word.len() < w.len() {
                    shortest = Some(word);
                }
            }
        }
    }
    shortest
}
fn main() {
    let s = "The quick brown leopard jumps over the lazy buffalo";
    match shortest_word(s) {
        Some(word) => println!("The shortest word is '{}'", word),
        None => println!("No shortest word found"),
    }
}
```

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

```
fn is_prime(n: u32) -> bool {
    if n <= 1 {
        return false;
    }
    for i in 2..=((n as f32).sqrt() as u32) {
        if n % i == 0 {
```

```
            return false;
        }
    }
    true
}
fn main() {
    let n = 23;
    if is_prime(n) {
        println!("{} is prime", n);
    } else {
        println!("{} is not prime", n);
    }
}
```

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

```
fn 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
    }
}
fn main() {
    let arr = vec![1, 2, 3, 4, 5];
    let m = median(&arr);
    println!("The median of {:?} is {}", arr, m);
}
```

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

```
        fn longest_common_prefix(strs: &[&str]) -> String {
    if strs.is_empty() {
        return String::new();
    }

    let first_str = strs[0];
    let mut longest_prefix = String::new();

    for (i, c) in first_str.chars().enumerate() {
        for str in &strs[1..] {
            if i >= str.len() || str.chars().nth(i) != Some(c) {
                return longest_prefix;
            }
        }
        longest_prefix.push(c);
    }

    longest_prefix
}
fn main() {
    let strs = vec!["flower", "flow", "flight"];
    let prefix = longest_common_prefix(&strs);
    println!("The longest common prefix of {:?} is {}", strs, prefix);
}
```

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

```
        fn kth_smallest_element(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 = vec![7, 2, 8, 1, 4, 6];
    let k = 3;
    let kth_smallest = kth_smallest_element(&arr, k);
    println!("The {}th smallest element of {:?} is {:?}", k, arr, kth_smallest);
}
```

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

```
use std::cmp;

struct TreeNode {
    val: i32,
    left: Option<Box<TreeNode>>,
    right: Option<Box<TreeNode>>,
}

impl TreeNode {
    fn new(val: i32) -> Self {
        Self {
            val,
            left: None,
            right: None,
        }
    }
}
```

```
fn max_depth(root: Option<Box<TreeNode>>) -> i32 {
    match root {
        None => 0,
        Some(node) => {
            let left_depth = max_depth(node.left);
            let right_depth = max_depth(node.right);
            cmp::max(left_depth, right_depth) + 1
        }
    }
}

fn main() {
    let mut root = TreeNode::new(3);
    root.left = Some(Box::new(TreeNode::new(9)));
    root.right = Some(Box::new(TreeNode::new(20)));
    root.right.as_mut().unwrap().left = Some(Box::new(TreeNode::new(15)));
    root.right.as_mut().unwrap().right = Some(Box::new(TreeNode::new(7)));

    println!("Max depth of binary tree: {}", max_depth(Some(Box::new(root))));
}
```

9. Reverse a string in Rust

```
fn reverse_string(s: &str) -> String {
    s.chars().rev().collect()
}

fn main() {
    let s = "hello world";
    let reversed = reverse_string(s);
    println!("{}", reversed);
}
```

10. Check if a number is prime in Rust

```
fn is_prime(n: u64) -> bool {
    if n <= 1 {
        return false;
    }

    let mut i = 2;
    while i * i <= n {
        if n % i == 0 {
            return false;
        }
        i += 1;
    }

    true
}

fn main() {
    let n = 17;
    if is_prime(n) {
        println!("{} is prime", n);
    } else {
        println!("{} is not prime", n);
    }
}
```

11. Merge two sorted arrays in Rust

```
fn merge_sorted_arrays(a: &[i32], b: &[i32]) -> Vec<i32> {
    let mut result = Vec::with_capacity(a.len() + b.len());
    let mut i = 0;
```

```
    let mut j = 0;

    while i < a.len() && j < b.len() {
        if a[i] < b[j] {
            result.push(a[i]);
            i += 1;
        } else {
            result.push(b[j]);
            j += 1;
        }
    }

    result.extend_from_slice(&a[i..]);
    result.extend_from_slice(&b[j..]);
    result
}

fn main() {
    let a = vec![1, 3, 5, 7];
    let b = vec![2, 4, 6, 8];
    let merged = merge_sorted_arrays(&a, &b);
    println!("{:?}", merged);
}
```

12. Find the maximum subarray sum in Rust

```
fn max_subarray_sum(a: &[i32]) -> i32 {
    let mut max_sum = a[0];
    let mut current_sum = a[0];

    for &x in a.iter().skip(1) {
        current_sum = current_sum.max(0) + x;
```

```
        max_sum = max_sum.max(current_sum);
    }

    max_sum
}

fn main() {
    let a = vec![-2, 1, -3, 4, -1, 2, 1, -5, 4];
    let max_sum = max_subarray_sum(&a);
    println!("{}", max_sum);
}
```