Internship Assignment by Saurabh Jaiswal

Q1:-Implement a function that checks whether a given string is a palindrome or not.

Code:-

```
fn is_palindrome(s: &str) -> bool {
    let s_lower = s.to_lowercase();
    s_lower.chars().eq(s_lower.chars().rev())
}

fn main() {
    let string1 = "Rawar";
    if is_palindrome(string1) {
        println!("'{}' is a palindrome.", string1);
    } else {
        println!("'{}' is not a palindrome.", string1);
    }
    }
```

Input :- "Rawar"
Output :- 'Rawar' is a palindrome.

Input :- "hello"
Output :- 'hello' is not a palindrome.

Q2:-Given a sorted array of integers, implement a function that returns the index of the first occurrence of a given number.

Code:-

```
fn find_first_occurrence(arr: &[i32], target: i32) -> Option<usize> {
    for (index, &element) in arr.iter().enumerate() {
        if element == target {
            return Some(index);
        }
    }
    None
}

fn main() {
    let numbers = [4, 16, 15, 16, 23, 42];
    let target_number = 0;
    if let Some(index) = find_first_occurrence(&numbers, target_number) {
        println!("Found at index: {}", index);
    } else {
        println!("Number not found.");
    }
}
```

Input :- numbers = [4, 16, 15, 16, 23, 42]
target_number = 16
Output:-Found at index: 1

Input:- numbers = [4, 16, 15, 16, 23, 42]
target_number = 0
Output:-Number not found.

Q3:-Given a string of words, implement a function that returns the shortest word in the string.

Code:-

```
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";

    if let Some(shortest) = shortest_word(sentence) {
        println!("Shortest word: {}", shortest);
    } else {
        println!("No words found.");
    }
}
```

Input :- "The quick brown fox jumps over the lazy dog"
Output :- Shortest word: The

Input :- "I am good at rust"
Output :- Shortest word: I

Q4:-Implement a function that checks whether a given number is prime or not.

Code:-

```
fn is_prime(num: u64) -> bool {
    if num <= 1 {
        return false;
    }
    for i in 2..=(num as f64).sqrt() as u64 {
        if num % i == 0 {
            return false;
        }
    }
    true
}

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

Input :- 17
Output :- 17 is prime.

Input :- 18
Output :- 18 is not prime.

Q5:-Given a sorted array of integers, implement a function that returns the median of the array.

Code:-

```
fn find_median(arr: &[i32]) -> f64 {
    let len = arr.len();
    if len % 2 == 0 {
        let mid = len / 2;
        (arr[mid - 1] + arr[mid]) as f64 / 2.0
    } else {
        arr[len / 2] as f64
    }
}

fn main() {
    let arr_even = [1, 2, 3, 4];
    println!("Median: {}", find_median(&arr_even));
}
```

Input :- [1, 2, 3, 4, 5]
Output :- Median: 3

Input :- [1, 2, 3, 4]
Output :- Median: 2.5

Q6:-Implement a function that finds the longest common prefix of a given set of strings.

Code:-

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

    let first_string = &strings[0];
    let mut prefix = String::new();

    'outer: for (i, ch) in first_string.chars().enumerate() {
        for string in &strings[1..] {
            if let Some(c) = string.chars().nth(i) {
                if c != ch {
                    break 'outer;
                }
            } else {
                break 'outer;
            }
        }
        prefix.push(ch);
    }

    prefix
}


fn main() {
    let strings = vec![
        String::from("flower"),
        String::from("flow"),
        String::from("flight"),
    ];
```

```
    println!("Longest common prefix: {}", longest_common_prefix(&strings));
}
```

Input :- ["flower","flow","flight"]
Output :- Longest common prefix: fl

Input :- ["computer","puter","put"]
Output :- Longest common prefix: put

Q7:-Implement a function that returns the kth smallest element in a given array.

Code:-

```
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, 5, 1, 3];
    let k = 3;
    if let Some(kth_smallest) = kth_smallest(&arr, k) {
        println!("The {}th smallest element is: {}", k, kth_smallest);
    } else {
        println!("Invalid input: k is out of bounds.");
    }
}
```

Input :- arr=[4, 2, 5, 1, 3]
k=3
Output :- The 3th smallest element is: 3

Input :- arr=[4, 2, 5, 1, 3]
k=2
Output :- The 2th smallest element is: 2

Q8:-Given a binary tree, implement a function that returns the maximum depth of the tree.

Code:-

```
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);
            let right_depth = max_depth(node.right);
            1 + left_depth.max(right_depth)
        }
        None => 0,
    }
}

fn main() {
    let root = Some(Box::new(TreeNode {
        val: 3,
        left: Some(Box::new(TreeNode {
            val: 9,
            left: None,
            right: None,
        })),
        right: Some(Box::new(TreeNode {
            val: 20,
            left: Some(Box::new(TreeNode {
                val: 15,
                left: None,
                right: None,
```

```
            })),
            right: Some(Box::new(TreeNode {
                val: 7,
                left: None,
                right: None,
            })),
        })),
    }));
    println!("Maximum depth of the tree: {}", max_depth(root));
}
```

Input :- Each node is inserted ,the tree look like :-

```
          3
    9         20
            15     7
```

Output :- Maximum depth of the tree: 3

Q9:-Reverse a string in Rust.

Code:-

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

fn main() {
    let original_string = "hello";
    let reversed_string = reverse_string(original_string);
    println!("Original string: {}", original_string);
    println!("Reversed string: {}", reversed_string);
}
```

Input :- "hello"
Output :- Original string: hello
Reversed string: olleh

Input :- "Saurabh"
Output :- Original string: Saurabh
Reversed string: hbaruaS

Q10:-Check if a number is prime in Rust.

Code:-

```
fn is_prime(num: u64) -> bool {
    if num <= 1 {
        return false;
    }
    for i in 2..=(num as f64).sqrt() as u64 {
        if num % i == 0 {
            return false;
        }
    }
    true
}

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

Input :- 17
Output :- 17 is prime.

Input :- 18
Output :- 18 is not prime.

Q11:-Merge two sorted arrays in Rust.

Code:-

```
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];
    let arr2 = [2, 4, 6, 8];
    let merged = merge_sorted_arrays(&arr1, &arr2);
    println!("Merged array: {:?}", merged);
}
```

Input :- arr1 = [1, 3, 5, 7]
arr2 = [2, 4, 6, 8]

Output :- Merged array: [1, 2, 3, 4, 5, 6, 7, 8]

Q12:-Find the maximum subarray sum in Rust.

Code:-

```
fn max_subarray_sum(arr: &[i32]) -> i32 {
    let mut max_ending_here = 0;
    let mut max_so_far = i32::MIN;

    for &num in arr {
        max_ending_here = max_ending_here.max(0) + num;
        max_so_far = max_so_far.max(max_ending_here);
    }
    max_so_far
}

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);
}
```

Input :- arr = [-2, 1, -3, 4, -1, 2, 1, -5, 4]
Output :- Maximum subarray sum: 6