```rust
use std::cmp;

fn main() {
    let initial = [66, 70, 52, 93, 44, 67, 47, 10, 11, 13, 94, 9
    12];
    let mut to_sort;
    println!("initial:          {:?}",initial);

    to_sort  = initial.clone();
    bubble_sort(&mut to_sort);
    println!("bubble-sorted:    {:?}",to_sort);

    to_sort  = initial.clone();
    sel_sort(&mut to_sort);
    println!("selection-sorted: {:?}",to_sort);

    to_sort  = initial.clone();
    insert_sort(&mut to_sort);
    println!("insertion-sorted: {:?}",to_sort);

    println!();
    println!("unordered search:");
    report_search(44,unordered_search(44,&initial[..]));
    report_search(43,unordered_search(43,&initial[..]));

    println!();
    println!("binary search:");
    report_search(44,binary_search(44,&to_sort[..]));
    report_search(43,binary_search(43,&to_sort[..]));

    println!();
    println!("the min and max of initial are {:?}",
            min_max(&initial[..]));
}

/*
// NOTE!! The following will not compile: It needs lifetime annotations.
// We'll fix this later on.
fn swap(x :&mut u32, y : &mut u32) {
    let t = x;
    x = y;
    y = t;
}
*/

fn bubble_sort(a : &mut [u32]) {
    let len = a.len();
    for i in 0..len {
        for j in 0..(len-i-1) {
            if a[j]>a[j+1] {
                // swap the values of a[j] and a[j+1]
```

```
                let t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }
        }
    }
}

fn report_search(x : u32, r : Option<usize>) {
    print!("\t {} ",x);
    match r {
        None    => { println!("not found"); },
        Some(i) => { println!("found at index {}",i); },
    }
}

fn unordered_search(x : u32, a : &[u32]) -> Option<usize> {
    for i in 0..a.len() {
        if x==a[i] { return Some(i); }
    }
    None
}


fn sel_sort(a : &mut [u32]) {
    for i in 0..a.len() {
        if i + 1 == a.len() {
            return;
        }
        let mut min = i + 1;
        for j in i..a.len() {
            if a[min] > a[j] {
                min = j;
            }
        }
        let temp = a[i];
        a[i] = a[min];
        a[min] = temp
    }
}

fn insert_sort(a : &mut [u32]) {
    let mut i = 1;
    while i < a.len() {
        let mut j = i;
        while j > 0 && a[j - 1] > a[j] {
            // swap j - 1 and j
            let temp = a[j];
            a[j] = a[j - 1];
            a[j - 1] = temp;
            j -= 1;
```

```rust
        }
        i += 1;
    }
}

// https://en.wikipedia.org/wiki/Binary_search_algorithm
fn binary_search(x : u32, a : &[u32]) -> Option<usize> {
    let mut l = 0;
    let mut r = a.len() - 1;
    while l <= r {
        let m = (l + r) / 2;
        if a[m] < x {
            l = m + 1;
        } else if a[m] > x {
            r = m - 1;
        } else {
            return Some(m);
        }
    }
    None
}

fn min_max(a : &[u32]) -> (u32,u32) {
    let len = a.len();
    assert!(len>0);

    if a.len() == 1 {
        return (a[0], a[0]);
    } else if a.len() == 2 {
        return (cmp::min(a[0], a[1]), cmp::max(a[0], a[1]));
    } else {
        let left_half = &a[0..a.len() / 2];
        let right_half = &a[a.len() / 2..a.len()];
        let (left_min, left_max) = min_max(left_half);
        let (right_min, right_max) = min_max(right_half);
        let max = cmp::max(left_max, right_max);
        let min = cmp::min(left_min, right_min);
        return (min, max);
    }
}

// NOTE:
// cmp::min(a,b) returns the minimum of a and b
// cmp::max(a,b) returns the maximum of a and b

```