

```

1  use std::cmp;
2
3  fn main() {
4      let initial = [66, 70, 52, 93, 44, 67, 47, 10, 11, 13, 94, 9
5      • 12];
6      let mut to_sort;
7      println!("initial:          {:?}",initial);
8
9      to_sort = initial.clone();
10     bubble_sort(&mut to_sort);
11     println!("bubble-sorted:    {:?}",to_sort);
12
13     to_sort = initial.clone();
14     sel_sort(&mut to_sort);
15     println!("selection-sorted: {:?}",to_sort);
16
17     to_sort = initial.clone();
18     insert_sort(&mut to_sort);
19     println!("insertion-sorted: {:?}",to_sort);
20
21     println!();
22     println!("unordered search:");
23     report_search(44,unordered_search(44,&initial[..]));
24     report_search(43,unordered_search(43,&initial[..]));
25
26     println!();
27     println!("binary search:");
28     report_search(44,binary_search(44,&to_sort[..]));
29     report_search(43,binary_search(43,&to_sort[..]));
30
31     println!();
32     println!("the min and max of initial are {:?}",
33     min_max(&initial[..]));
34 }
35
36 /*
37 // NOTE!! The following will not compile: It needs lifetime annotations.
38 // We'll fix this later on.
39 fn swap(x :&mut u32, y : &mut u32) {
40     let t = x;
41     x = y;
42     y = t;
43 }
44 */
45
46 fn bubble_sort(a : &mut [u32]) {
47     let len = a.len();
48     for i in 0..len {
49         for j in 0..(len-i-1) {
50             if a[j]>a[j+1] {
51                 // swap the values of a[j] and a[j+1]

```

```

51         let t = a[j];
52         a[j] = a[j+1];
53         a[j+1] = t;
54     }
55 }
56 }
57 }
58
59 fn report_search(x : u32, r : Option<usize>) {
60     print!("\t {} ",x);
61     match r {
62         None => { println!("not found"); },
63         Some(i) => { println!("found at index {}",i); },
64     }
65 }
66
67 fn unordered_search(x : u32, a : &[u32]) -> Option<usize> {
68     for i in 0..a.len() {
69         if x==a[i] { return Some(i); }
70     }
71     None
72 }
73
74
75 fn sel_sort(a : &mut [u32]) {
76     for i in 0..a.len() {
77         if i + 1 == a.len() {
78             return;
79         }
80         let mut min = i + 1;
81         for j in i..a.len() {
82             if a[min] > a[j] {
83                 min = j;
84             }
85         }
86         let temp = a[i];
87         a[i] = a[min];
88         a[min] = temp
89     }
90 }
91
92 fn insert_sort(a : &mut [u32]) {
93     let mut i = 1;
94     while i < a.len() {
95         let mut j = i;
96         while j > 0 && a[j - 1] > a[j] {
97             // swap j - 1 and j
98             let temp = a[j];
99             a[j] = a[j - 1];
100             a[j - 1] = temp;
101             j -= 1;

```

```

102     }
103     i += 1;
104 }
105 }
106
107 // https://en.wikipedia.org/wiki/Binary\_search\_algorithm
108 fn binary_search(x : u32, a : &[u32]) -> Option<usize> {
109     let mut l = 0;
110     let mut r = a.len() - 1;
111     while l <= r {
112         let m = (l + r) / 2;
113         if a[m] < x {
114             l = m + 1;
115         } else if a[m] > x {
116             r = m - 1;
117         } else {
118             return Some(m);
119         }
120     }
121     None
122 }
123
124 fn min_max(a : &[u32]) -> (u32,u32) {
125     let len = a.len();
126     assert!(len>0);
127
128     if a.len() == 1 {
129         return (a[0], a[0]);
130     } else if a.len() == 2 {
131         return (cmp::min(a[0], a[1]), cmp::max(a[0], a[1]));
132     } else {
133         let left_half = &a[0..a.len() / 2];
134         let right_half = &a[a.len() / 2..a.len()];
135         let (left_min, left_max) = min_max(left_half);
136         let (right_min, right_max) = min_max(right_half);
137         let max = cmp::max(left_max, right_max);
138         let min = cmp::min(left_min, right_min);
139         return (min, max);
140     }
141 }
142
143 // NOTE:
144 // cmp::min(a,b) returns the minimum of a and b
145 // cmp::max(a,b) returns the maximum of a and b
146

```