3.1 Algorithms

Searching Algorithms

Linear Search

Linear search finds a element in an list by checking every element in the list one by one until it is found.

Procedure *LinearSearch*(x : integer, $a_1, a_2, \ldots, a_n$: distinct integers)

1: $i := 1$
2: **while** $i \leq n$ and $x \neq a_i$ **do**
3: $\quad i := i + 1$
4: **end while**
5: **if** $i \leq n$ **then**
6: $\quad location := i$
7: **else**
8: $\quad location := 0$
9: **end if**
10: **return** $location$ {$location$ is the subscript of the term that equals x, or is 0 if x is not found}

Binary Search

If a list is in sorted order, a binary search can be performed to quickly find an element in the list. To find an element in the list, the algorithm will always narrow its search space by a half every time a comparison is made until the element is found.

Procedure *BinarySearch*(x : integer, $a_1, a_2, \ldots, a_n$: increasing integers)

1: $i := 1$ {i is left endpoint of search interval}
2: $j := n$ {j is right endpoint of search interval}
3: **while** $i < j$ **do**
4: $\quad m := \lfloor (i + j)/2 \rfloor$
5: $\quad$ **if** $x > a_m$ **then**
6: $\quad\quad i := m + 1$
7: $\quad$ **else**
8: $\quad\quad j := m$
9: $\quad$ **end if**
10: **end while**
11: **if** $x = a_i$ **then**
12: $\quad location := i$
13: **else**
14: $\quad location := 0$
15: **end if**
16: **return** $location$ {$location$ is the subscript i of the term a_i that equals x, or is 0 if x is not found}

Sorting Algorithms

Bubble Sort

A sorting algorithm where the smaller elements will "bubble" (or "float") to the beginning of the list while bigger elements will "sink" to the end of the list.

Procedure *BubbleSort*($a_1, \ldots, a_n$: real numbers with $n \geq 2$)

```
1: for i := 1 to n − 1 do
2:    for j := 1 to n − i do
3:       if a_j > a_{j+1} then
4:          interchange a_j and a_{j+1}
5:       end if
6:    end for
7: end for
8: {a_1, ..., a_n is in increasing order}
```

Insertion Sort

A sorting algorithm where each element becomes "inserted" into the correct position at every iteration.

Procedure *InsertionSort*($a_1, \ldots, a_n$: real numbers with $n \geq 2$)

```
1: for j := 2 to n do
2:    i := 1
3:    while a_j > a_i do
4:       i := i + 1
5:    end while
6:    m := a_j
7:    for k := 0 to j − i − 1 do
8:       a_{j−k} := a_{j−k−1}
9:    end for
10:   a_i := m
11: end for
12: {a_1, ..., a_n is in increasing order}
```

3.1 pg 202 # 13

List all the steps used to search for 9 in the sequence 1, 3, 4, 5, 6, 8, 9, 11 using

a) a linear search.

Note that there are eight numbers, so $n = 8$.
$i = 1$, $1 \neq 9$
$i = 2$, $3 \neq 9$
$i = 3$, $4 \neq 9$

$i = 4$, $5 \neq 9$
$i = 5$, $6 \neq 9$
$i = 6$, $8 \neq 9$
$i = 7$, $9 = 9$
Stop because 9 was found at $i = 7$

b) a binary search.

Start with 1, 3, 4, 5, 6, 8, 9, 11 ($i = 1, j = 8, m = 4$)
Compare the m-th element in the list with 9. In this case $m = 4$, so compare 9 with 5.

Since $5 < 9$, take the second half 6, 8, 9, 11 ($i = 5, j = 8, m = 6$)
Compare the m-th element in the list with 9. In this case $m = 6$, so compare 9 with 8.

Since $8 < 9$, take the second half 9, 11 ($i = 7, j = 8, m = 7$)
Compare the m-th element in the list with 9. In this case $m = 7$, so compare 9 with 9.

Since 9 is not less than 9, take the first half by setting $j = 7 (= m)$.
At this moment the algorithm exits the while loop because i is not less than j.
After exiting the loop, compare 9 with the ith element. Since $9 = 9$, the algorithms returns i, which is 7.

3.1 pg 203 # 35

Use the bubble sort to sort 3, 1, 5, 7, 4, showing the lists obtained at each step.

Note that we have 5 numbers, so $n = 5$
$i = 1$
$(\mathbf{3, 1}, 5, 7, 4) \rightarrow (\mathbf{1, 3}, 5, 7, 4)$
$(1, \mathbf{3, 5}, 7, 4) \rightarrow (1, \mathbf{3, 5}, 7, 4)$
$(1, 3, \mathbf{5, 7}, 4) \rightarrow (1, 3, \mathbf{5, 7}, 4)$
$(1, 3, 5, \mathbf{7, 4}) \rightarrow (1, 3, 5, \mathbf{4, 7})$

$i = 2$
$(\mathbf{1, 3}, 5, 4, 7) \rightarrow (\mathbf{1, 3}, 5, 4, 7)$
$(1, \mathbf{3, 5}, 4, 7) \rightarrow (1, \mathbf{3, 5}, 4, 7)$
$(1, 3, \mathbf{5, 4}, 7) \rightarrow (1, 3, \mathbf{4, 5}, 7)$

While we can see that the list is in sorted order, the algorithm does not know that it is correctly sorted. Hence we keep going until the i-for loop can terminate.

$i = 3$
$(\mathbf{1, 3}, 4, 5, 7) \rightarrow (\mathbf{1, 3}, 4, 5, 7)$
$(1, \mathbf{3, 4}, 5, 7) \rightarrow (1, \mathbf{3, 4}, 5, 7)$

$i = 4$
(**1, 3**, 4, 5, 7) $\rightarrow$ (**1, 3**, 4, 5, 7)

3.1 pg 203 # 39

Use the insertion sort to sort 3, 1, 5, 7, 4, showing the lists obtained at each step.

Note that the bolded numbers are in sorted order.
3, 1, 5, 7, 4
3, 1, 5, 7, 4
1, 3, 5, 7, 4
1, 3, 5, 7, 4
1, 3, 5, 7, 4
1, 3, 4, 5, 7