Chapter 6 – Arrays, Part C – Sorting Arrays

Sorting

Sorting an array means putting the elements in some kind of order. For now, we will just talk about numerical order. In other words, you may have an array of test scores and you want to sort it so that the scores are listed from smallest to largest.

Selection Sort Idea

There are many sorting algorithms. We will study two here and you will study others in other classes. One of the simplest sorting algorithms is the *Selection Sort* algorithm. It works as follows. First, find the largest element in the array and put it in the last position. Next, find the next largest value and put it in the next to last position, *etc*.

Selection Sort Idea, Example

Find max in positions 0-6,

0	1	2	3	4	5	6
2	9	5	4	8	1	6

swap max with value in position 6

						sorted
0	1	2	3	4	5	6
2	6	5	4	8	1	9

Find max in positions 0-5,

						sorted
0	1	2	3	4	5	6
2	6	5	4	8	1	9

swap max with value in position 5

					sorted	sorted
0	1	2	3	4	5	6
2	6	5	4	1	8	9

Find max in positions 0-4,

					sorted	sorted
0	1	2	3	4	5	6
2	6	5	4	1	8	9

swap max with value in position 4

				sorted	sorted	sorted
0	1	2	3	4	5	6
2	1	5	4	6	8	9

Find max in positions 0-3, swap max with value in position 3

0	1	2	3	sorted 4	sorted 5	sorted 6
2	1	5	4	6	8	9

0	1	2	sorted 3	sorted 4	sorted 5	sorted 6
2	1	4	5	6	8	9

Find max in positions 0-2, swap max with value in position 2 (no swap necessary)

0	1	2	sorted 3	sorted 4	sorted 5	sorted 6
2	1	4	5	6	8	9

Find max in positions 0-1,

0	1	sorted 2	sorted 3	sorted 4	sorted 5	sorted 6
2	1	4	5	6	8	9

swap max with value in position 1

0	sorted 1	sorted 2	sorted 3	sorted 4	sorted 5	sorted 6
1	2	4	5	6	8	9

All done

sorted 0	sorted 1	sorted 2	sorted 3	sorted 4	sorted 5	sorted 6
1	2	4	5	6	8	9

Selection Sort, Algorithm

a. First pass:

```
FOR j=last element down to 2nd element
        FOR i=0 up to j
                IF list[i] > max
                        max = list[i]
                        i_max = i
        Swap values in i_max and j
```

b. Some observations:

For the inner *for* loop, we can start it at 1 by initializing max=list[0].

If i_max=j, this means that the maximum value is already in the correct position. Thus, we only need to swap when i_max is not equal to j.

c. Second pass:

```
FOR j=last element down to 2nd element
        max = list[0]
        i_max = 0
        FOR i=1 up to j
                IF list[i] > max
                        max = list[i]
                        i_max = i
        IF i_max != j
                Swap values in i_max and j
```

d. Third pass:

```
for( j=list.length-1; j>=1; j--)
        max = list[0]
        i_max = 0
        for( i=1; i<=j, i++)
                if( list[i] > max )
                        max = list[i]
                        i_max = i
        if( i_max != j )
                list[i_max] = list[j]
                list[j] = max
```

Selection Sort, Example

```
for( int j=list.length-1; j>=1; j-- )          // for(j=6; j>=1; j--)
                                               // j=6
```

						j
0	1	2	3	4	5	6
2	9	5	4	8	1	6

```
     currentMax = list[0]                      // currentMax = 2
     currentMaxIndex = 0                       // currentMaxIndex = 0

     for( int i=1; i<=j; i++)                  // for(i=1; i<=6; i++)
     {
          if ( list[i] > currentMax )          // i=1, true, 9<2
          {
               currentMax = list[i]            // currentMax = 9
               currentMaxIndex = i             // currentMaxIndex = 1
          }
     }

                                               // i=2, false
                                               // i=3, false
                                               // i=4, false
                                               // i=5, false
                                               // i=6, false
```

	max					j
0	1	2	3	4	5	6
2	9	5	4	8	1	6

```
     if ( currentMaxIndex != j )               // true, 1 != 6
     {
          list[currentMaxIndex] = list[j]      // list[1] = list[6]
          list[j] = currentMax;                // list[6] = 9
     }
```

						j
0	1	2	3	4	5	6
2	6	5	4	8	1	9

```
for( int j=list.length-1; j>=1; j-- )          // for(j=6; j>=1; j--)
                                               // j=5
```

					j	sorted
0	1	2	3	4	5	6
2	6	5	4	8	1	9

```
     currentMax = list[0]                      // currentMax = 2
     currentMaxIndex = 0                       // currentMaxIndex = 0

     for( int i=1; i<=j; i++)                  // for(i=1; i<=5; i++)
     {
          if ( currentMax < list[i] )          // i=1, true, 2<6
          {
               currentMax = list[i]            // currentMax = 6
               currentMaxIndex = i             // currentMaxIndex = 1
          }
     }

                                               // i=2, false
                                               // i=3, false
                                               // i=4, true, 6<8
                                               // currentMax = 8
                                               // currentMaxIndex = 4
                                               // i=5, false
```

				max	j	sorted
0	1	2	3	4	5	6
2	6	5	4	8	1	9

```
     if ( currentMaxIndex != j )               // true, 4 != 5
     {
          list[currentMaxIndex] = list[j]      // list[4] = list[5]
          list[j] = currentMax;                // list[5] = 8
     }
```

					j	sorted
0	1	2	3	4	5	6
2	6	5	4	1	8	9

```
for( int j=list.length-1; j>=1; j-- )          // for(j=6; j>=1; j--)
                                               // j=4
```

				j	sorted	sorted
0	1	2	3	4	5	6
2	6	5	4	1	8	9

```
     currentMax = list[0]                      // currentMax = 2
     currentMaxIndex = 0                       // currentMaxIndex = 0

     for( int i=1; i<=j; i++)                  // for(i=1; i<=4; i++)
     {
          if ( currentMax < list[i] )          // i=1, true, 2<6
          {
               currentMax = list[i]            // currentMax = 6
               currentMaxIndex = i             // currentMaxIndex = 1
          }
     }

                                               // i=2, false
                                               // i=3, false
                                               // i=4, false
```

				j	sorted	sorted
0	1	2	3	4	5	6
2	6	5	4	1	8	9

```
     if ( currentMaxIndex != j )               // true, 1 != 4
     {
          list[currentMaxIndex] = list[j]      // list[1] = list[4]
          list[j] = currentMax;                // list[4] = 6
     }
```

				j	sorted	sorted
0	1	2	3	4	5	6
2	1	5	4	6	8	9

```
for( int j=list.length-1; j>=1; j-- )          // for(j=6; j>=1; j--)
                                               // j=3
```

			j	sorted	sorted	sorted
0	1	2	3	4	5	6
2	1	5	4	6	8	9

```
     currentMax = list[0]                      // currentMax = 2
     currentMaxIndex = 0                       // currentMaxIndex = 0

     for( int i=1; i<=j; i++)                  // for(i=1; i<=3; i++)
     {
          if ( currentMax < list[i] )          // i=1, false
          {
               currentMax = list[i]
               currentMaxIndex = i
          }
     }

                                               // i=2, true, 2<5
                                               // currentMax = 5
                                               // currentMaxIndex = 2
                                               // i=3, false
```

		max	j	sorted	sorted	sorted
0	1	2	3	4	5	6
2	1	5	4	6	8	9

```
     if ( currentMaxIndex != j )               // true, 2 != 3
     {
          list[currentMaxIndex] = list[j]      // list[2] = list[3]
          list[j] = currentMax;                // list[3] = 5
     }
```

			j	sorted	sorted	sorted
0	1	2	3	4	5	6
2	1	4	5	6	8	9

```
for( int j=list.length-1; j>=1; j-- )          // for(j=6; j>=1; j--)
                                               // j=2
```

		j	sorted	sorted	sorted	sorted
0	1	2	3	4	5	6
2	1	4	5	6	8	9

```
    currentMax = list[0]                       // currentMax = 2
    currentMaxIndex = 0                        // currentMaxIndex = 0

    for( int i=1; i<=j; i++)                   // for(i=1; i<=2; i++)
    {
        if ( currentMax < list[i] )            // i=1, false
        {
            currentMax = list[i]
            currentMaxIndex = i
        }
    }
                                               // i=2, true, 2<4
                                               // currentMax = 4
                                               // currentMaxIndex = 2
```

		j, max	sorted	sorted	sorted	sorted
0	1	2	3	4	5	6
2	1	4	5	6	8	9

```
    if ( currentMaxIndex != j )                // false, 2==2
    {                                          // thus, no swap
        list[currentMaxIndex] = list[j]
        list[j] = currentMax;
    }
```

		j	sorted	sorted	sorted	sorted
0	1	2	3	4	5	6
2	1	4	5	6	8	9

```
for( int j=list.length-1; j>=1; j-- )          // for(j=6; j>=1; j--)
                                               // j=1
```

	j	sorted	sorted	sorted	sorted	sorted
0	1	2	3	4	5	6
2	1	4	5	6	8	9

```
     currentMax = list[0]                      // currentMax = 2
     currentMaxIndex = 0                       // currentMaxIndex = 0

     for( int i=1; i<=j; i++)                  // for(i=1; i<=1; i++)
     {
          if ( currentMax < list[i] )          // i=1, false
          {
               currentMax = list[i]
               currentMaxIndex = i
          }
     }
```

max	j	sorted	sorted	sorted	sorted	sorted
0	1	2	3	4	5	6
2	1	4	5	6	8	9

```
     if ( currentMaxIndex != j )               // true, 0 != 1
     {
          list[currentMaxIndex] = list[j]      // list[0] = list[1]
          list[j] = currentMax;                // list[1] = 2
     }
```

	j	sorted	sorted	sorted	sorted	sorted
0	1	2	3	4	5	6
1	2	4	5	6	8	9

All done!

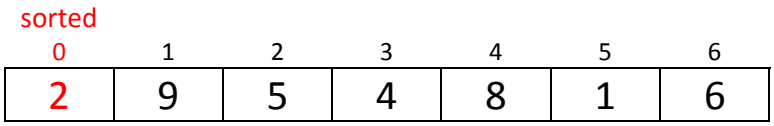
Insertion Sort Idea

The idea is that if a sub-list is in order, it is easy to put a new element in order. We just start at the end of the sub-list (the largest element in the sub-list), and check to see if the new value belongs in front of this element. If it is, we move the element over one to the right. Then we repeat this process, asking if the new value belongs in front of the next largest value in the sub-list. Eventually, when we answer no, we have found where the new element belongs.

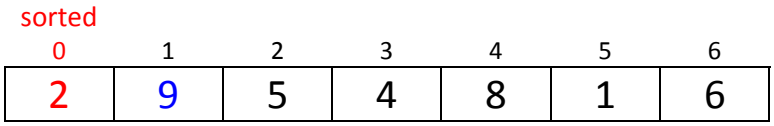
The algorithm starts with a sorted list that consists of only the first element. The algorithm will then do list.length-1 iterations. During each iteration, it takes the next, non-sorted number and then successively asks if it belongs in front a sorted element as we work backwards through the sorted list. If the new (non-sorted) number **does** belong in front of a sorted value, we shuffle the sorted value one place to the right. We stop when we can do no more shuffling (*e.g.* when the non-sorted value is larger than a sorted value, or we reach the beginning of the list. Finally, in each iteration, we put the non-sorted value in the correct position, the one to the right of the last sorted position checked.

Insertion Sort Idea, Example

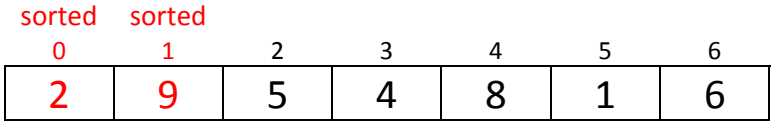
Initialize sorted list.



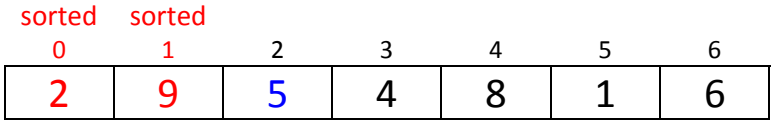
Get next val, val=9. Belongs in front of list[0]? No.



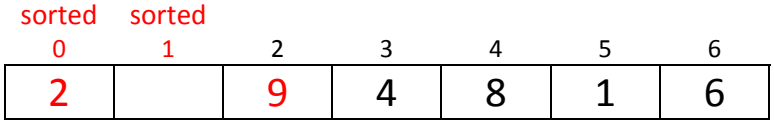
Leave in position. Increase length of sorted list.



Get next val, val=5. Belongs in front of list[1]? Yes.



Move list[1] to list[2]. val belongs in front of list[0]? No.



Set list[1] = val. Increase length of sorted list.

sorted 0	sorted 1	sorted 2	3	4	5	6
2	5	9	4	8	1	6

Get next val, val=4. val belongs in front of list[2]? Yes.

sorted 0	sorted 1	sorted 2	3	4	5	6
2	5	9	4	8	1	6

Move list[2] to list[3]. val belongs in front of list[1]? Yes.

sorted 0	sorted 1	sorted 2	3	4	5	6
2	5		9	8	1	6

Move list[1] to list[2]. val Belongs in front of list[0]? No.

sorted 0	sorted 1	sorted 2	3	4	5	6
2		5	9	8	1	6

Set list[1] = val. Increase length of sorted list.

sorted 0	sorted 1	sorted 2	sorted 3	4	5	6
2	4	5	9	8	1	6

Get next val, val=8. val belongs in front of list[3]? Yes.

sorted 0	sorted 1	sorted 2	sorted 3	4	5	6
2	4	5	9	8	1	6

Copy list[3] to list[4]. val Belongs in front of list[2]? No.

sorted 0	sorted 1	sorted 2	sorted 3	4	5	6
2	4	5		9	1	6

Set list[3] = val. Increase length of sorted list.

sorted 0	sorted 1	sorted 2	sorted 3	sorted 4	5	6
2	4	5	8	9	1	6

Get next val, val=1. val belongs in front of list[4]? Yes

sorted 0	sorted 1	sorted 2	sorted 3	sorted 4	5	6
2	4	5	8	9	1	6

Move list[4] to list[5]. val belongs in front of list[3]? Yes

sorted 0	sorted 1	sorted 2	sorted 3	sorted 4	5	6
2	4	5	8		9	6

Move list[3] to list[4]. val belongs in front of list[2]? Yes.

sorted 0	sorted 1	sorted 2	sorted 3	sorted 4	5	6
2	4	5		8	9	6

Move list[2] to list[3]. val belongs in front of list[1]? Yes.

sorted 0	sorted 1	sorted 2	sorted 3	sorted 4	5	6
2	4		5	8	9	6

Move list[1] to list[2]. val belongs in front of list[0]? Yes.

sorted 0	sorted 1	sorted 2	sorted 3	sorted 4	5	6
2		4	5	8	9	6

Move list[0] to list[1]. val belongs in front of list[-1]? Stop.

sorted 0	sorted 1	sorted 2	sorted 3	sorted 4	5	6
	2	4	5	8	9	6

Set list[0] = val. Increase length of sorted list. Get next val.

sorted 0	sorted 1	sorted 2	sorted 3	sorted 4	sorted 5	6
1	2	4	5	8	9	6

Move to proper position

sorted 0	sorted 1	sorted 2	sorted 3	sorted 4	sorted 5	sorted 6
1	2	4	5	6	8	9

Insertion Sort, Algorithm

a. First pass:

```
Initialize index for end of sorted list, end=0
FOR i=1 to end of list
        Get current value, val = list[i]
        FOR j=end DOWNTO 0
                IF val < list[j]
                        Move list[j] over to the right
                ELSE
                        Break
        Put val in correct position
        Update index for end of sorted list
```

b. Comments:

What value does j have when we exit the *for* loop?

c. Second pass:

```
Initialize index for end of sorted list, end=0
for( i=1; i<list.length; i++)
        val = list[i]
        for( j=end; j>=0; j-- )
                if( val < list[j] )
                        list[j+1] = list[j]
                else
                        break
        list[j+1] = val
        end++
```