Notes – Sorting

Sorted information is usually more useful than unsorted information. Imagine if you wanted to look up a word in a dictionary and the words weren't in order. That would be silly. Think about when you go to the bowling alley and people didn't put their bowling balls on the correct weight shelf; that's annoying. You have to look all over to find the correct ball. Sorting makes life easier (and, more importantly, makes searching easier).

Selection Sort

The selection sort works by selecting the smallest unsorted item remaining in the list, and then swapping it with the item in the next position to be filled. The selection sort has a complexity of $O(n^2)$.

The code for the algorithm is as follows:

```
// array is an array of size == n
for (i = 0; i < (n - 1); i++)
{
  min_index = i;
  for (j = (i + 1); j < n; j++)
  {
      if (array[j] <= array[min_index])
       {
            min_index = j;
       }
  }
  int temp = array[i];
  array[i] = array[min_index];
  array[min_index] = temp;
}
```

For example, the following shows the array after each "pass" (iteration of the for-i loop):

9	4	8	6	3

3 is the smallest, so 3 and 9 switch

3	4	8	6	9

4 is the smallest, so it remains where it is

3	4	8	6	9

6 is the smallest, so 6 and 8 switch

3	4	6	8	9

8 is the smallest, so it remains where it is

3	4	6	8	9

Pros: Simple and easy to implement.
Cons: Inefficient for large lists, so similar to the more efficient insertion sort that the insertion sort should be used in its place.

Insertion Sort

The insertion sort works just like its name suggests - it inserts each item into its proper place in the final list. The simplest implementation of this requires two list structures - the source list and the list into which sorted items are inserted. To save memory, most implementations use an in-place sort that works by moving the current item past the already sorted items and repeatedly swapping it with the preceding item until it is in place. The insertion sort has a complexity of $O(n^2)$.

The code for the algorithm is as follows:

```
      for (i = 1; i < n; i++)
      {
            inserted = false;
            j = i;
            while ((j >= 1) && (inserted == false))
            {
                  if (array[j] < array[j-1])
                  {
                        int temp = array[j];
                        array[j] = array[j-1];
                        array[j-1] = temp;
                  }
                  else
                        inserted = true;
                  j--;
            }
    }
```

For example, the following shows the array after each “pass” (iteration of the for-i loop):

9	4	8	6	3

4 belongs before 9, so move it there

4	9	8	6	3

8 belongs between 4 and 9, so move it there

4	8	9	6	3

6 belongs between 4 and 8, so move it there

4	6	8	9	3

3 belongs before 4, so move it there

3	4	6	8	9

Pros: Relatively simple and easy to implement.

Cons: Inefficient for large lists.

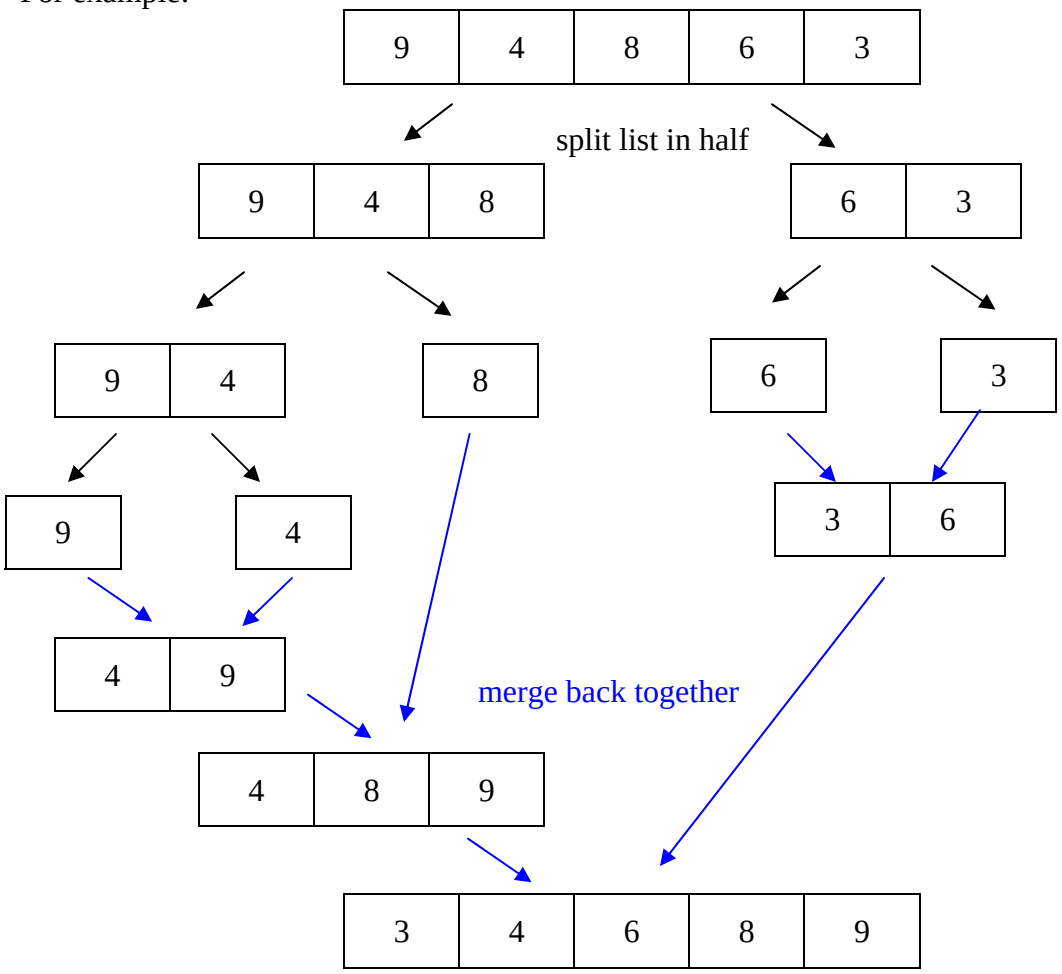
Merge Sort

The merge sort splits the list to be sorted into two equal halves, and places them in separate arrays. Each array is recursively sorted, and then merged back together to form the final sorted list. Like most recursive sorts, the merge sort has an algorithmic complexity of $O(n \log n)$.

The algorithm for mergesort is as follows:

1. If the size of the array is larger than 1
    a. Split the list in half.
    b. MergeSort each half of the list.
    c. Merge the sorted halves back together in order.
2. Otherwise, if the size of the array is not larger than 1
    a. Assume an array of size 1 or 0 is in order.

For example:



Pros: Marginally faster than the heap sort for larger sets.

Cons: At least twice the memory requirements of the other sorts; recursive (calls itself).

Comprehension Questions:

1. **Why is sorting important?**
2. **How does selection sort work?**
3. **How does insertion sort work?**
4. **How does mergesort work?**
5. **Draw each pass of selection sort on the following numbers: 1 4 5 2 3**
6. **Draw each pass of insertion sort on the following numbers: 5 4 3 2 1**
7. **Draw a diagram showing how mergesort works on the numbers: 5 2 3 4 1**