CO322 - Lab 4: Simple Sorting Algorithms

E/14/317

1. Bubble sort

   Starting on the left, compare adjacent items and keep "bubbling" the larger one to the right (it's in its final place). Bubble sort the remaining N -1 items.

```
static void bubble_sort(int[] data)
{
       int length = data.length;
        for (int i = 0; i < length-1; i++)
        {
             for (int j = 0; j < length-1; j++)
             {
                 if (data[j] > data[j+1])
                 {
                     int temp = data[j];
                     data[j] = data[j+1];
                     data[j+1] = temp;
                 }
             }
        }
}
```

   Complexity

   - Best Case - O(n)
   - Worst Case – O(n^2)

Comparison

- Bubble sort can be more time consuming in a case where all the elements are in reverse order. Since algorithm has to go through all the elements 'Bubbling', the number of swaps are higher.
- Worst case for the bubble sort can be where the last element of the list is also the smallest. Thus it has to pass down the list all through each element.
- In the aspect of the CPU hardware, Bubble sort is considered to be less efficient since its cache misses are twice as high as insertion sort.
- In Overall, Bubble sort performance is considered to be the worst of all three.

2. Selection Sort

Scan all items and find the smallest. Swap it into position as the first item. Repeat the selection sort on the remaining N-1 items.

```
static void selection_sort(int[] data)
{
        int len = data.length;
        for(int i=0;i<len;i++)
        {
                for(int j=0;j<len;j++)
                {
                        if(data[i]<data[j])
                        {
                                int temp = data[i];
                                data[i] = data[j];
                                data[j] = temp;
                        }
                }
        }
}
```

Complexity

- Best Case – O(n^2)
- Worst Case – O(n^2)

Comparison

- Selections sort efficiency is independent of the data being sorted, which is an advantage in real-time applications.
- This perform well for complex algorithms where the auxiliary memory is low. Because of its divide and conquer nature.
- Selections sort perform twice as high comparisons when compared to the insertion sort.

3. Insertion Sort

```
static void insertion_sort(int[] data)
{
        int len = data.length;
        int j=0;
        int node=0;
        for(int i=0;i<len;i++)
        {
                node = data[i];
                j = i-1;
                while (j>=0 && data[j] > node)
                {
                   data[j+1] = data[j];
                   j-- ;
                }
                data[j+1] = node;
        }
}
```

Complexity

- Best Case – O(n)
- Worst Case – O(n^2)

Comparison

- Worst case is where all the items are in reverse order.
- Number of comparisons are lower when compared to the selection sort.
- The primary advantage of insertion sort over selection sort is that selection sort must always scan all remaining elements to find the absolute smallest element in the unsorted portion of the list, while insertion sort requires only a single comparison when the *k*+1th element is greater than the *k*th element.
- A disadvantage of insertion sort over selection sort is that it requires more writes due to the fact that, on each iteration, inserting the *k*+1th element into the sorted portion of the array requires many element swaps to shift all of the following elements, while only a single swap is required for each iteration in selection sort.

Conclusion

Thus regarding all the facts insertion sort out perform other algorithms in complex situations.

But regarding trivial data sets it is easy to implement bubble sort since in such situations the performances does not vary that much.