Summary previous lecture

1- Bubble Sort
2- Selection Sort
3- Insertion Sort
Example: sort in ascending order

Bubble Sort (main step):

- Start from the left

- Compare neighbors: “swap” and iterate left to right

O(N^2) comparisons and swaps

```
public void bubbleSort()
{
    int in, out;
    int temp;
    for (out=N-1; out>0; out--)
        // outer loop (backward)
        for (in=0; in<out; in++)
            // inner loop (forward)
            if (array[in] > array[in+1])
                // out of order? swap them
                { temp = array[in];
                  array[in] = array[in+1];
                  array[in+1] = temp;
                }
} // end bubbleSort()
```
Example: sort in ascending order

```java
public void selectionSort() {
    int in, out, min;
    int temp; // temp variable
    for (out = 0; out < N - 1; out++) { // outer loop
        // find the minimum item between [out+1, N-1]
        min = out; // initialize minimum index
        for (in = out + 1; in < N; in++) { // inner loop
            if (array[in] < array[min]) min = in; // update minimum index
        } // end inner loop
        // swap item "array[min]" with item "array[out]"
        temp = array[out];
        array[out] = array[min];
        array[min] = temp
    } // end outer loop
} // end selectionSort()
```

Selection Sort (main step):

- Find smallest item
- Swap it with item at 1st position

**O(N²) comparisons; O(N) swaps**
1- Bubble Sort
2- Selection Sort
3- Insertion Sort
3- Insertion Sort- overview

Easier to understand if we start in the middle of the process

- Players are partially sorted at the left of the marked player

- The marked player is removed from the list. The players from the left that are taller than the marked player, shift up.

- The marked player is inserted into the empty spot on the left. A new marked player is selected. And so on...
2- Insertion Sort - overview

- Example: sort in ascending order

All Steps for 5 items

- Select 2\textsuperscript{nd} item (key)..........................

- Swap it with the 1\textsuperscript{st} item if not in order..........

- Select 3\textsuperscript{rd} item (key)..........................

- Insert it inside the ordered array on the left........

- Select 4\textsuperscript{th} item (key)..........................

- Insert it inside the ordered array on the left........

- Select 5\textsuperscript{th} item (key)..........................

- Insert it inside the ordered array on the left........

- The end.........................................................
3- Insertion Sort- Examples

- For fun: https://www.youtube.com/watch?v=ROalU379l3U

- To do: Test Java applet InsertSort.html
The basic insertionSort method is few lines long (example below uses array of integer for simplicity)

```java
public void insertionSort()
{
    int in, out;
    int temp;       // temp variable
    for(out=1; out<N; out++)   // outer loop – select key
    {
        temp = array[out];   // save in memory select key item
        in = out;             // start shifting at out
        while(in>0 && array[in-1]>=temp) // shift until key-item in position
        {
            array[in] = array[in-1];   // shift up
            in--;                      // go down one position
        }
        array[in] = temp;           // insert select key item
    }  // end outer loop
} // end insertionSort()
```
3- Insertion Sort- Complexity Analysis

- Complexity analysis: (two loops) so it is still a $O(N^2)$

- Max number of comparisons
  
  $1+2+3+\ldots+(N-1)=\frac{N\times(N-1)}{2}$

  - However, only half this number in average $\frac{N\times(N-1)}{4}$
  - Half the time of BubbleSort

- Number of shifts (copies) is also equal in average to $\frac{N\times(N-1)}{4}$

  - However, a shift is not as time consuming as a swap
3- Insertion Sort- Complexity Analysis

- For random data
  - Insertion Sort should run twice faster than Bubble Sort
  - Insertion Sort should also run faster than Selection Sort
- For data arranged in inverse order
  - Every possible comparisons and shifts take place
  - No faster than Bubble Sort
- For data that is already sorted or almost sorted
  - Insertion sort runs in $O(N)$ (the while loop is never true)
  - Efficient way to order arrays that are slightly out of order
  - Often use as the final stage of more sophisticated algorithm such as quicksort
Simple Sorting

- **In theory**, Bubble Sort, Selection Sort and Insertion Sort are all $O(N^2)$; they are also all 'in-place' memory efficient algorithms.

- **In practice**, Insertion sort is the best bet of the three in most situations.

- It runs in $O(N)$ for 'almost sorted data'; for 'random data', efficiency may be improved using a binary search to insert the key.
**Enhanced Insertion Sort**

- Number of comparisons is $O(N \log N)$, shift is still $O(N^2)$
Complement: 'Unsorting' algorithm

- Need to randomly shuffle items of an 'ordered' array?
  Fisher-Yates/Knuth shuffle algorithm to generate random permutations

**All Steps for 5 items**

- Random select one item in the [0-4] index range
- Swap it with the 5\(^{th}\) item.................................
- Random select one item in the [0-3] index range
- Swap it with the 4\(^{th}\) item.................................
- Random select one item in the [0-2] index range
- Swap it with the 3\(^{rd}\) item.................................
- Random select one item in the [0-1] index range
- Swap it with the 2\(^{nd}\) item.................................
- The end..............................................................
Complement: 'Unsorting' algorithm

- Durstenfeld modern implementation of the Fisher-Yates algorithm

```java
public void shuffleArray()
{
    Random rnd= new Random();
    int out,index;
    int temp;
    for(out=N-1; out>0; out--)
    {
        index=rnd.nextInt(out+1); // select random number in [0:out]
        // simple swap
        temp = array[index];
        array[index] = array[out];
        array[out] = temp;
    }
} // end shuffleArray()
```

- Complexity analysis: O(N)
- Work 'in-place' (no extra copy of the array is needed, only one temp variable)