Michael Gould
2/24/18
CS-506

Module 6: Program 1: Insertion Sort

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int COMPCOUNT = 0;

int smaller(int *A, int i, int j){
        if(A[j] < A[j-1])
                COMPCOUNT++;
        return (A[j] < A[j-1]);
}

int main(int argc, char *argv[]){

        int input[atoi(argv[1])];
        int arrayLength = atoi(argv[1]);
        int i, j, temp, init;
        srand(time(NULL));

        //WORST INPUT
        //for(init = 0; init <= arrayLength; init++)
        //      input[init] = arrayLength - init;

        //BEST INPUT
        //for(init = 0; init <= arrayLength; init++)
        //      input[init] = init;

        //RANDOM INPUT
        for(init = 0; init <= arrayLength; init++)
                input[init] = rand() % 10 + 1;

        printf("Size of sample:%d\n", arrayLength );
        printf("Array before sorting: ");
        for(i=0; i < arrayLength; i++)
                printf("%d ", input[i]);

        for(i = 1; i < arrayLength; i++){
                j = i;
                while(j > 0 && smaller(input, i, j)){
                                temp = input[j];
                                input[j] = input[j-1];
                                input[j-1] = temp;
                j--;
                }
        }
        printf("Array after sorting: " );
        for(i=0; i < arrayLength; i++)
                printf("%d ", input[i]);
        printf("Number of swaps:%d\n", COMPCOUNT );
}
```

1. Small Array Size, $n = 32$:

Worst Case:

Size of Sample: 32
Array before sorting: 32,31,30,29,28,27,26,25,24,23,22,21,20,19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1
Array after sorting: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32
Number of swaps: 496

Best Case:

Size of Sample: 32
Array before sorting: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32
Array after sorting: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32
Number of swaps: 0

Random Case:

Size of Sample: 32
Array before sorting: 4,9,1,2,4,5,5,9,10,6,8,4,2,5,8,7,10,10,7,1,9,5,3,8,7,4,3,1,3,3,5,9
Array after sorting: 1,1,1,2,2,3,3,3,3,4,4,4,4,5,5,5,5,5,6,7,7,7,8,8,8,9,9,9,9,10,10,10
Number of swaps: 236

Theoretical worst case scenario = (n^2 – n)/2, using above input size of 32 with worst case data yields (32^2 – 32)/2 = 496. The number of key comparisons does in fact equal that of the theoretical values in regards to this outlying case. In regards to the average case scenario, with the given equation (n^2 – n)/4, we find that with an input size of (32, 32^2 – 32)/4 = 248. While this number does not have the same satisfying exacting equivalence as the worst case scenario, the pseudo-random nature of the inputs provided by a pseudo-random number generator may in particular cases provide segments of "optimized" input, that would be more likened to that of the best case. This means that running the program several times will provide different answers, but all those COMPCOUNT totals will average to that of the average case equation (n^2 – n)/4.

2. Increasing Array Size, $n = 100$, $n = 1000$, $n = 10000$

n = 100:

Size of sample: 100
Number of swaps: 2616

n = 1000:

Size of sample: 1000
Number of swaps: 254195

n = 10000:

Size of sample: 10000
Number of swaps: 25010466

The number of key-comparisons does in fact show O(n^2) performance, again with varying degrees of error. This can be accounted for because of the pseudo-random nature of the inputs, but otherwise the comparison for each falls within a 3% error margin. Notably, as the sample size gets larger, the corresponding margins of error fall to drastically smaller numbers as well, this can be seen in a comparison of the 1000 and 10000 data sets, which have a 1.6% margin of error. The one constant found in all of these equations is that of the n, which represents the sample size for the sort.