EX-1:

- I will not plagiarize.
- I understand and will follow the submission rules.
- I have to submit my homework in two separate ways, The Report and the code by Monday midnight.
- My report should tell what approach I used, what are the results and what are the conclusions.
- Code must be provided for all coding tasks.
- Late, incomplete submissions, false formats or naming conventions, no graphical representations etc. will result in deduction of points.
- I am allowed to discuss the tasks but solution must be done by myself.
- I will be representing my homework multiple times, where I will explain the tasks, the approach, results and take questions from TA and students. It won't a graded presentation but will help to polish my presentation skills.

EX-2:

I chose insertion sort algorithm for my analysis because most of the programming languages implements built-in hybrid quick or merge sort with insertion sort. The code is written in C#.

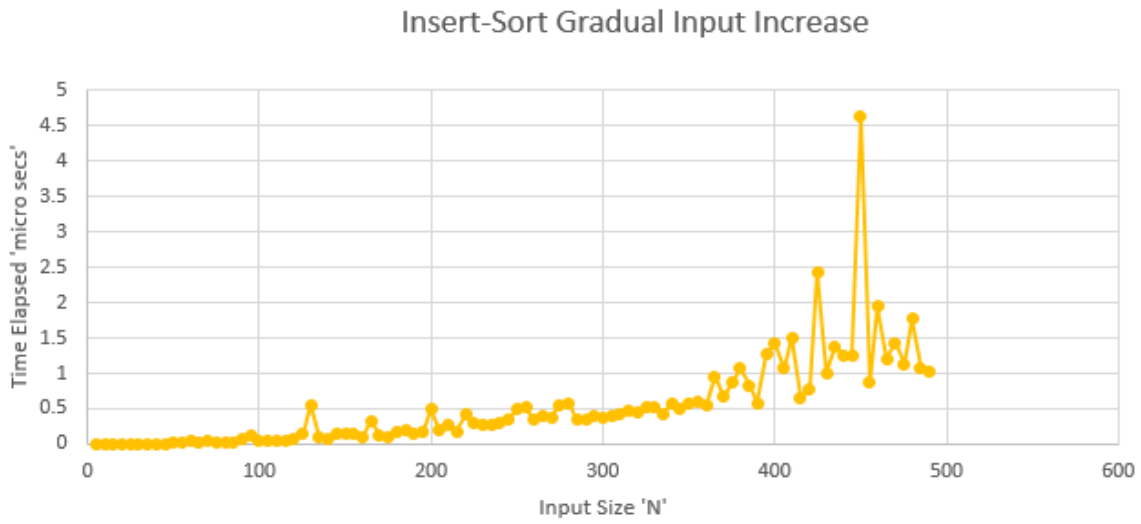
My program generates Random numbers from 0 to int.MaxValue i.e +2147483647 in C#. After taking desired number of input from user the program first makes an array with the size of given input and populates it with random values.

```
void Populate_Random_Data(ref int[] A)
{
    Random rand = new Random(int.MaxValue);
    Console.WriteLine("\n--------< Random Array >--------\n");
    for (int i=0; i< A.Length;i++)
    {
        A[i] = rand.Next();
        Console.WriteLine("Data_Array [" + i + "] = " + A[i]);
    }
}
```

Just after populating values randomly Custom_Sort(ref int[] A) method is called which takes the reference of underlying data array and runs insertion sort on the array.

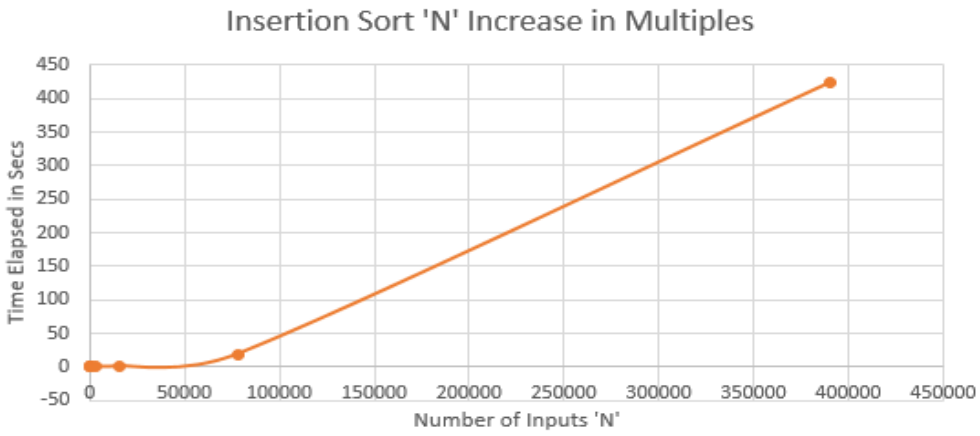
```
void Custom_Sort(ref int[] A)
{
    var watch = System.Diagnostics.Stopwatch.StartNew();

    for (int i =0;i< A.Length-1;i++)
    {
        for(int j=i+1;j>0 ;j--)
        {
            if (A[j - 1] > A[j])
            {
                int temp = A[j];
                A[j] = A[j - 1];
                A[j - 1] = temp;
            }
        }
    }

    watch.Stop();
    var elapsedMs = watch.Elapsed;
```

The running time is calculated by System.Diagnostics.Stopwatch and is displayed in the output. Firstly I started from an array of 5 elements and increased the inputs by 5 every time and gathered results of 100 Cycles. The Analysis is given below.



The Graphs shows gradual increase with spikes. In my point of view these spikes are because of randomly generated data because insertion sort running may vary depending upon the array entries. If I used the same generated integers every time then there wouldn't be such spikes in the running time.

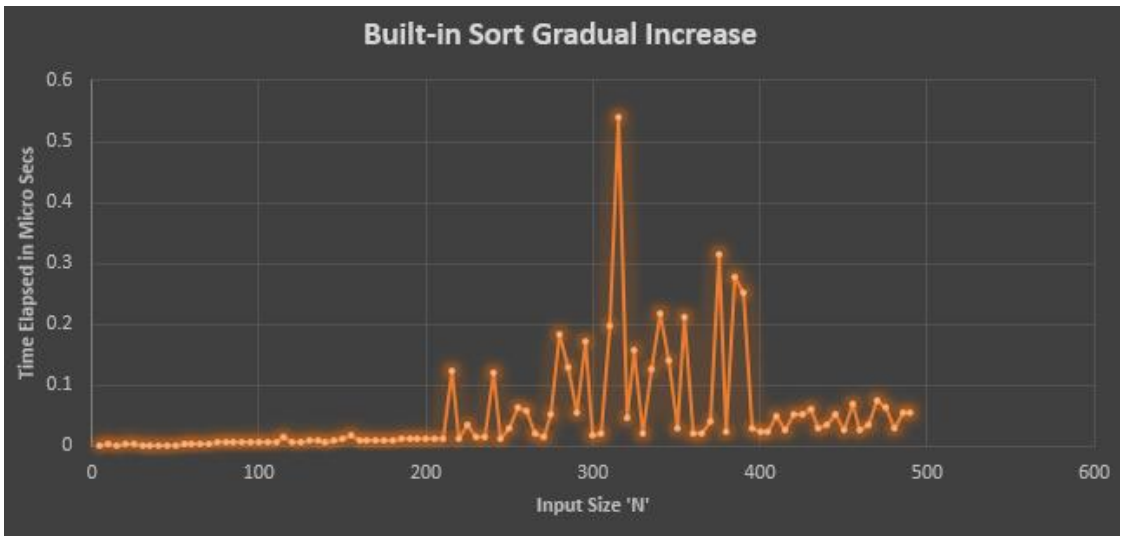
For next Phase of testing I increased the size of input data with a multiple of 5 every time.



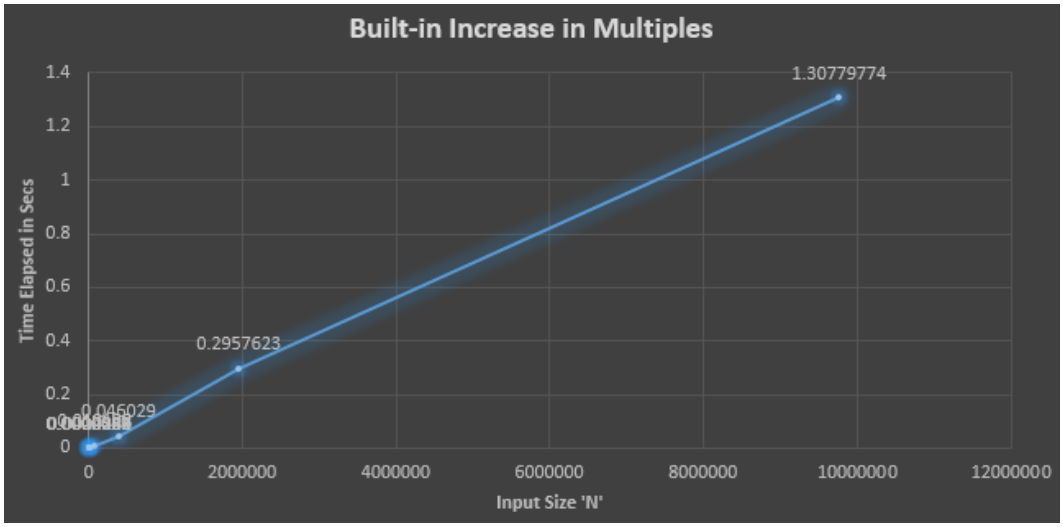
This time there was a prominent increase by every cycle. The time went above 7 minutes after 6 cycles. Concluding from both graph analysis it can be seen clearly for large inputs this algorithm will fail badly but in case of small inputs with some randomization it is quite effective and this is probably one of the biggest reasons why it is used in hybrid sorting algorithms if the input size is less.

EX-3:

The built-in C# sorting is categorized based on the number of inputs. If the input size is less than 16 elements, it will use insertion sort. If the number size exceeds 2*Log^N, where N is range of input array, it will use Heapsort otherwise it's going to use Quicksort for sorting. (MSDN Array.Sort()) The worst case of sorting in this case f(n)= n log n.. Here are some time analysis on same data used previously.



In case of input increase in a multiple of ‘N’. The Built-in sort shows following results.



According to this it seems that it’s not following the worst case scenario here because the graph is somewhat similar to linear graph i.e. f(x) =x.

EX-4:

The custom written sorting algorithm roughly sorts 150,000 elements in 1 minute while the built-in sorting algorithm sorts about 400,000,000 elements in 1 minute.

If following input size of both algorithms is increased by hundred then following are the time stats using Extrapolation (Lagrange polynomial)

Custom Sort (Insertion Sort)		***Built-in Sort***	
Input Size ‘N’	Time Taken	Input Size ‘N’	Time Taken
150	0.0000932 sec	40000	0.0039925 sec
15000	0.6262119 sec	4000000	0.0039925 sec
150000	100.6 sec	400000000	100.5 sec
15000000	***Approx. 12 days***	40000000000	***Approx. 6 days***

EX-5:

Assume that $f(n) \in \theta(g(n))$ i.e. there exists some positive constants c_1, c_2 and n_o such that:

$c_1 . g(n) \leq f(n) \leq c_2 . g(n)$, $\forall\, n \geq n_0$

So,

$$c_1.n^3 \leq 4n^3 + 0.5n \leq c_2n^3$$

- $f(n) \leq c_2 . g(n)$

$4n^3 + 0.5n \leq c_2n^3$

Let $c_2 = 4.5$, $n_o = 1$

$4n^3 + 0.5n \leq 4.5n^3$

$0.5n \leq 4.5n^3 - 4n^3$

$0.5n \leq 0.5n^3$ Hence, $f(n) \leq c_2 . g(n)$, $\forall\, n \geq n_0$

Now,

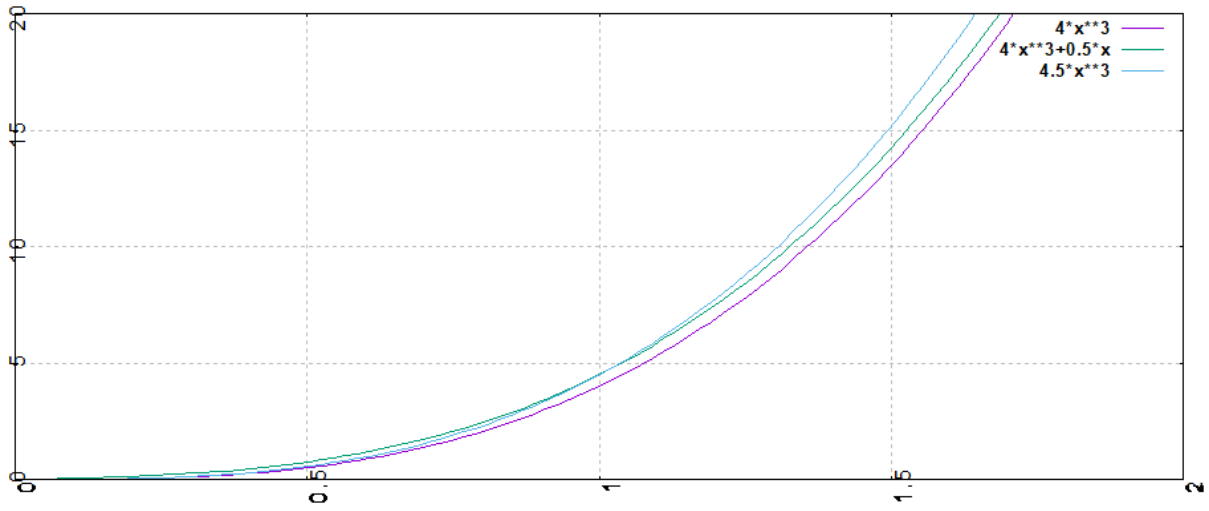
- $c_1 . g(n) \leq f(n)$

$c_1n^3 \leq 4n^3 + 0.5n$

Let $c_1 = 4$, $n_o = 1$

$4n^3 \leq 4n^3 + 0.5n$ Hence, $c_1 . g(n) \leq f(n)$, $\forall\, n \geq n_0$

Hence Proved $f(n) \in \theta(g(n))$


4*x**3
4*x**3+0.5*x
4.5*x**3
0
5
10
15
20
0
0.5
1
1.5
2