A)

The asymptomatic upper bound for merge sort is O(nlog(n))

For merge sort:

T(n) = 2T(n/2) + cn

We can use master theorem to calculate the complexity of this runtime by representing this function as

AT(n/B)*n^C

Log B base A = 1 and C is also equal to 1 so we can say that both have the same dominance.

This means that Θ(n) equals to f(n)log(n) and f(n) is n so we get nlogn: from this theta we can also say that O(n) equals to nlogn.

The asymptomatic upper bound for insertion sort is O(n^2)

For Insertion sort:

T(n) = T(n-1) + cn and T(1) = c

And if we assume that T(n) can be solved in second degree polynominal time we can say that:

T(n) = a(n-1)^2 b(n-1) + c + n

T(n) = an^2 + (1-2a+b)n + (a-b+c)

Which means that

a=a

b=1-2a+b

c=a-b+c

from this we get a = ½ and b = ½

then we can say T(1) = (1^2/2) + (1/2) + c

from this we get that c = 0

so we can say that T(n) = n^2/2 + n/2 so that means that our assumption holds and

that means that O(n) = n^2

the asymptomatic upper bound for bubble sort is equal to O(n^2)

We can say this because the recurrence function for this alghoritm is T(n)=T(n-1)+n which is the same with insertion sort. So if we apply what we have applied to insertion sort we will reach the same big O value as insertion sort which is O(n^2). We can also generate a tree to solve this as a recurrence. Each our tree will look like this:

n -> n-1 -> n-2 -> n-3 … -> 1

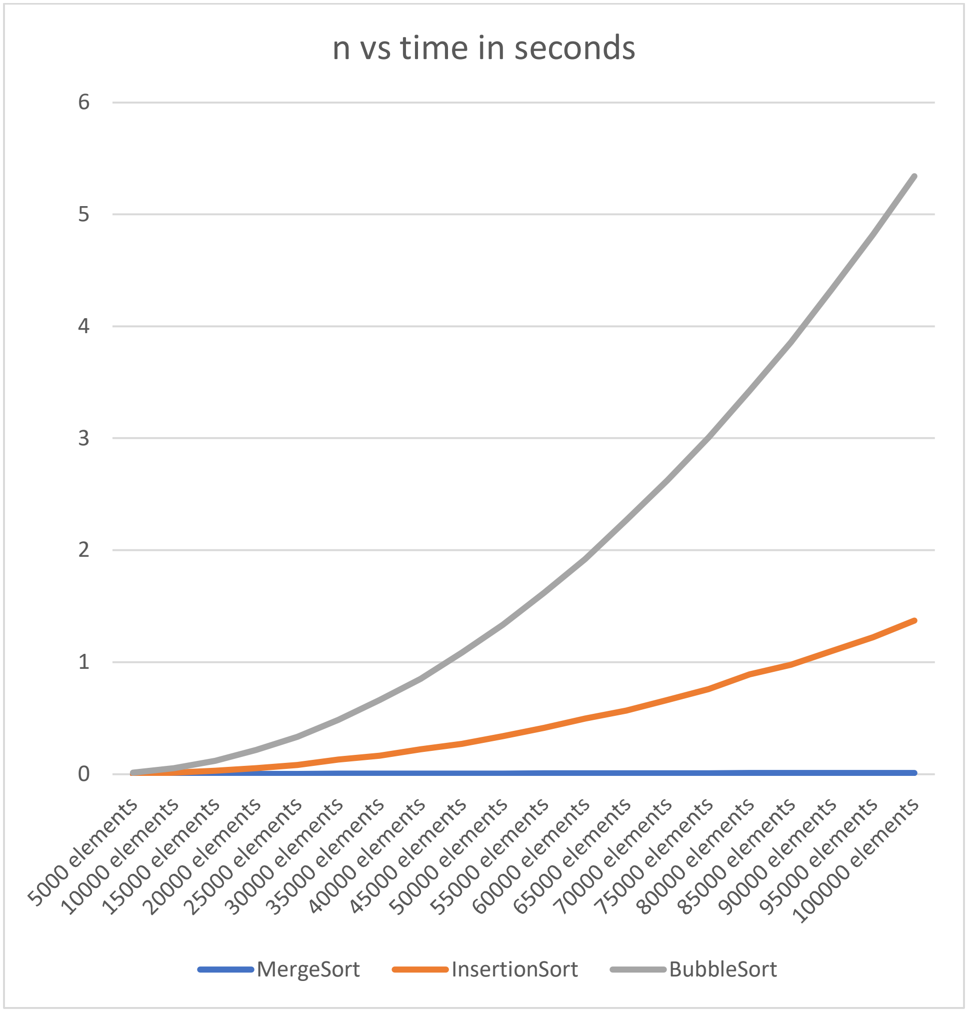
and the depth of our tree will be n. all the depths of our tree has a total value of n-c. from this information we can say that n times n-c equals to n^2+n*c since the n^2 dominates n*c we can say that O(n) = n^2.

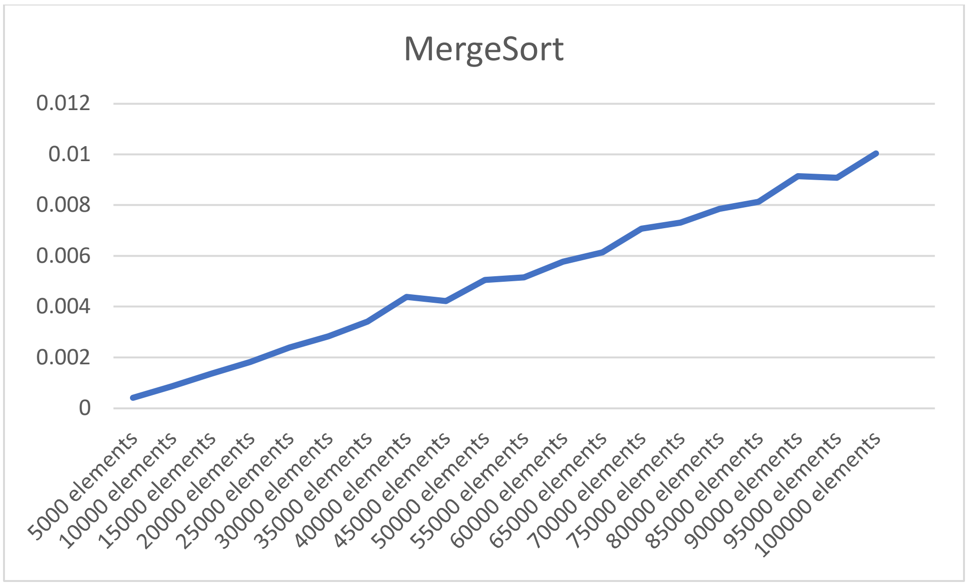
B)

	MergeSort	InsertionSort	BubbleSort
1000 elements	8.20E-05	0.000144	0.000536
10000 elements	0.000957	0.012839	0.053604
100000 elements	0.011124	1.36513	5.34218
1000000 elements	0.123892	149.456	534.913

Time is in seconds*
Compiled with -O3*
Ran with i-7 9750h*

C)





Both from the results of the question B and the plot created above we can see that calculation time of bubblesort is increasing by the square of its input size. In the chart in the B question we see that for every time we multiply the input size with 10 it also spends approximately 100 times more time which is the square of 10. We see the same trend with Insertion sort but it runs a bit faster due to its more efficient nature. For the mergesort however we that it increases by almost nlogn. The slower converging speed also is visible in the chart above where it almost looks like a straight line. In the separated mergesort chart we can see that the mergesort is growing almost lineearly because the logn is so small.