Problem 1

1: $X \leftarrow N$
2: **repeat**
3: $\quad Y \leftarrow N$
4: $\quad$ **while** $Y > 0$ **do**
5: $\quad\quad \ldots$
6: $\quad\quad Y \leftarrow Y - 1$
7: $\quad$ **end while**
8: $\quad X \leftarrow X + X$
9: **until** $X > N \times N$

The innermost statements are trivially $\mathcal{O}(1)$. The innermost while loop is linear, as it goes form n to 0 ($\mathcal{O}(n)$). The outhermost loop is $\mathcal{O}(\log_2(2n^2))$. This makes the final complexity $\mathcal{O}(n \times \log_2(2n^2))$ or $\mathcal{O}(n \log n)$. Substituting n for 1000 gives 80000 itterations. Each itteration takes a total of 4 microseconds plus 2 milliseconds which is 2.004 ms. $80000(2.004) = 160320$. This means that thiss operation takes 160320 ms when $n = 1000$.

Problem 2

Table 1: Toy Data (0-9)

	Best Case		Average Case		Worst Case	
Algorithm	Exchanges	Comparisons	Exchanges	Comparisons	Exchanges	Comparisons
Bubble Sort	0	9	20	43	45	45
Selection Sort	9	45	9	45	9	45
Insertion Sort	0	10	20	30	45	55
Merge Sort	24	40	32	56	28	48
Radix Sort	40	0	40	0	40	0

Table 2: Test Data (0-1999)

	Best Case		Average Case		Worst Case	
Algorithm	Exchanges	Comparisons	Exchanges	Comparisons	Exchanges	Comparisons
Bubble Sort	0	1999	1004333	1997649	1999000	1999000
Selection Sort	1999	1999000	1999	1999000	1999	1999000
Insertion Sort	0	2000	1004333	1006333	1999000	2001000
Merge Sort	12863	23728	21430	40862	13087	24176
Radix Sort	20000	0	20000	0	20000	0

Radix sort can be modified to work with string objects. All that is required for radix sort is that it's items be able to be interpreted as ordinals. One could, for example, convert each unique string into a unique ordinal by using its Base64 representation and them performing the same radix sort with a radix of 64 instead of 10. This would create a large amount of "buckets" in memory, but would be functionally equivilant to the standard integer radix sort.

An analysis of merge sort reveals a complexity of $\mathcal{O}(n \log n)$ for worst case, average case, and best case. This should perform significantly better than the $\mathcal{O}(n^2)$ class sorting algorithms like selection and insertion sort. However, due to implenetation specific details (like using a tree of recursive calls and array copies), this implementation requires a significant number of operations. It should also be noted that the "worst case" file does not represent the worst case for merge sort, as reverse order will not maximize the number of exchanges performed by the algorithm. Radix sort is a rather interesting algorithm as it is not sensitive to its input. It has no true "worst case" and operates without using any comparisions by sorting elements into "buckets" by their digits. It has a complexity of $\mathcal{O}(Kn)$, where K is the number of digits needed to represent the largest element in the array. In this case it would be $\log L$, where L is the largest element. This means that radix sort will take the same number of steps, regardless of input. This is reflected by the data.