CMPT 202 - Selection and Insertion Sorts

1. What is the Big-oh notation of the sequential search? $O(\quad)$?

2. What is the Big-oh notation of the binary search? $O(\quad)$?

3. In terms of efficiency, which algorithm is better?

4. What requirement does the binary search have that the sequential search does not require?

The **Selection Sort** operates by continually selecting the next smallest element in the list, and placing it in its appropriate position in the list. This process is repeated until all N elements are in their correct position.

Consider the following list of numeric values:

82 79 76 65 78 68

The first *iteration* compares each element in the list, and finds the smallest element (65). Since this is the smallest element in the list, it exchanges this element with the current element at the first position in the list: (A bold-faced value indicates it is in its correct position in the list.)

65 79 76 82 78 68

5. How many comparisons were necessary to determine the smallest element in the list?

It then continues to find the next smallest (68)

65 **68** 76 82 78 79

6. How many comparisons were necessary to find the next smallest item?

7. Complete the algorithm to sort all remaining elements, noting the required number of comparisons to place each element in its correct location in the list:

						Number of Comparisons
82	79	76	65	78	68	(initial list)
65	79	76	82	78	68	5
65	**68**	76	82	78	79	

The **Insertion Sort** operates by dividing the list of elements into *sorted* and *unsorted* regions. (The sorted region is underlined)

$\underline{5}$ 2 4 6 1 3

It then inserts the first element in the unsorted region into its correct position in the sorted region:

$\underline{2}$ $\underline{5}$ 4 6 1 3

1. How many comparisons were necessary to insert 2 into its correct position in the list?

The subsequent pass inserts the next element from the unsorted region into its correct position in the sorted region:

$\underline{2}$ $\underline{5}$ 4 6 1 3

$\underline{2}$ $\underline{4}$ $\underline{5}$ 6 1 3

2. Complete the algorithm to sort all remaining elements, noting the required number of comparisons to place each element in its correct location in the list:

	Number of Comparisons
$\underline{5}$ 2 4 6 1 3	(initial list)
$\underline{2}$ $\underline{5}$ 4 6 1 3	1
$\underline{2}$ $\underline{4}$ $\underline{5}$ 6 1 3	

3. Provide an example of a list which would give the *best* performance (fewest number of comparisons) of the Insertion sort.

4. Provide an example of a list which would give the *worst* performance (most number of comparisons) of the Insertion sort.