1. **Problem**

Sort strings by the number of times that string shows up.

1. **Solution Description**

Create a java program that uses Arrays to store the strings given by the user. Then that array is sorted using a selection sort.

1. **Problems Encountered**

I only struggled to find what algorithms would work and how to implement it.

1. **Is f(n) = n, O(n2)? Describe why or why not.**

No, because f(n) = n is a linear function, while O(n^2) represents a quadratic time complexity. Linear functions grow at a rate proportional to the input size, while quadratic functions grow at a rate proportional to the square of the input size. Therefore, f(n) = n cannot be O(n^2).

1. **Is f(n) = 2,000n3 + 1,000,000n2 + 200n + 4, O(n3)? Desribe why or why not.**

Yes, f(n) = 2,000n^3 + 1,000,000n^2 + 200n + 4 is O(n^3). This is because the highest order term in the function is 2,000n^3, which grows at a rate proportional to n^3. Asymptotically, this term dominates the other terms in the function, and so the overall time complexity of the function is O(n^3).

1. **Is f(n) = n!, O(2n)? Describe why or why not.**

No, f(n) = n! is not O(2^n). This is because n! grows much faster than 2^n as n increases. In fact, n! grows at a rate proportional to n^n, which is much faster than the exponential growth of 2^n. Therefore, f(n) = n! is actually O(n^n).

1. **What is the Big O time complexity for bubble sort?**

The Big O time complexity of bubble sort is O(n^2). This is because the algorithm requires two nested loops to compare and swap elements in the array, and so the time required to sort the array is proportional to n^2.

1. **What is the Big O time complexity for quick sort?**

The Big O time complexity of quick sort is O(nlog(n)). In the best case scenario, quick sort divides the input array into two equal parts at each recursive step, resulting in a logarithmic time complexity. In the worst case scenario, however, quick sort can have a time complexity of O(n^2), which occurs when the pivot element is chosen poorly.

1. **What is the Big O time complexity for merge sort?**

The Big O time complexity of merge sort is O(nlog(n)). This is because the algorithm divides the input array into halves recursively, and then merges the sorted halves back together. The time required for the recursive division and merging is proportional to log(n), while the time required for the merging itself is proportional to n.

1. **In relation to time complexity, which is considered more efficient, Selection Sort or Merge Sort? Describe why.**

Merge sort is considered more efficient than selection sort in terms of time complexity. This is because the time complexity of merge sort is O(nlog(n)), while the time complexity of selection sort is O(n^2). As the input size grows, the difference in time required by these two algorithms becomes more pronounced, with merge sort becoming exponentially faster than selection sort.