Problem1.docx

**void** **mergeSort**(**int** a[], **int** low, **int** high)

{

**int** mid;

**if**(low < high)

{

mid = (high + low) / 2;

mergeSort(a, low, mid);

mergeSort(a, mid + 1, high);

merge(a, low, high, mid);

}

}

1. I chose to use Merge Sort algorithm. It is stable since the array is partitioned but it will stay in the same order and at the end of the sort, elements that hold the same value will be in the same order any time they are sorted.

2. The best case time complexity is N log N as it runs N times for the number of elements but does not run across the entire array again. It splits it up into pieces and compares the separate elements afterwards and merges them together.

3. The worst case time complexity is also N log N due to the fact that the Merge Sorting algorithm does the same thing every single time by splitting up the array into smaller parts and comparing them together afterwards and putting them in sorted order into the main array.