PROBLEM 2

//Recursive merge sort with extra parameter, y begins below. For integer types, meaning only integers are used in an array, merge sort can be made inplace using some mathematics.
//That means storing the value of two elements at a single index and can be extracted using modulus and division. In order to do this properly,
//first we have to find a value greater than all the elements of the array. Now we can store the original value as modulus and the second value as division.
//Since this process is used it becomes a O(1) instead of an O(n) thereby meaning that the time can be concluded to be NlogN as this is what mergesort provides
//according to the chart.

```
void mergeSortRec(int arr[], int beg, int end, int y)
{
  if (beg < end) {
     int mid = (beg + end) / 2;
     mergeSortRec(arr, beg, mid, y);
     mergeSortRec(arr, mid + 1, end, y);
     mergeSort(arr, beg, mid, end, y);
  }
}
```

// This functions finds the max element and calls recursive and then the merge sort process begins. Some bottlenecking may occur as it requires multiple mergesorts to complete the process, however it should still be time efficient.

```
void mergeSort(int arr[], int n)
{
  int maxele = *max_element(arr, arr+n) + 1;
  mergeSortRec(arr, 0, n-1, maxele);
}
```

//This mergesort is stable as can be seen in the starting portions of the code. This is because when merging two halves,
//the left and right values are taken into accountance after being split meaning that when another mergesort is performed
//the previous sorting is remembered. It can be seen here were multiple mergesorts are used:

```
// if (beg < end) {
//       int mid = (beg + end) / 2;
//       mergeSortRec(arr, beg, mid, y);
```

```
//      mergeSortRec(arr, mid + 1, end, y);
//      mergeSort(arr, beg, mid, end, y);
//  }
```