2Jake Tantorski

1.
The reason that MergeSort is recommended over the quicksort is that the memory allocation is different within a linked list. The nodes are not always next to each other in memory therefore it is more useful to use a merge sort. It is easier to add nodes into the middle of a linked list due to the fact that it takes O(1) time to insert it whereas the time it takes a quick sort is O(nlogn). In a linked list you have to travel through each of the nodes starting at the head therefore it would be more efficient to use the merge sort due to the fact that quicksort uses more memory. Overall, you want to be able to access the linked list more quickly, therefore, you should use a merge sort.

2a
The value of p after the first invocation of Partition would be 11 due to the fact that in a partition the first index becomes the pivot and the numbers lower than it goes to the left and the numbers higher than it goes to the right. The two recursive calls would be the quick sort at the top of the list due to the fact that after a partition it needs to sort the numbers.

2b.
```
median(a, b, c) {
        if (a < b && a < c) {
                if (b < c) {
                        return b
                }
                else
                        return c
        } else if (b < a && b < c)
                if(a < c) {
                        return a
                }
                else
                        return c
        } else if  (c < a && c < b) {
                if (a < b) {
                        return a
                }
                else
                        return b
        }
}

        Partition(A, s, e)
                P1 = Rand(s,e)
                P2 = Rand(s,e)
                P3 = Rand(s,e)
                Index = Median(p1,p2,p3)\
                pivot = A[index], i = s+1, j = e; // Let the leftmost element be the pivot
                while i<=j // Rearrange elements
                        while i < e & A[i] < pivot,
                            i = i + 1
                        end while
                        while j > s & A[j] >= pivot,
                            j = j - 1
                        end while
                        if i >= j
                            break
                        end if
                        swap A[i] nd A[j]
                end while
                swap A[s] nd A[j]
                return j; // Return the index of pivot after the partition
```

2c.

```
Partition(A)
    Index = rand(s,e)
        pivot = A[index], i = s, j = e; // Let the leftmost element be the pivot
        while i<=j // Rearrange elements
            while i < e & A[i] < pivot,
                i = i + 1
            end while
            while j > s & A[j] >= pivot,
                j = j - 1
            end while
            if i >= j
                break
            end if
            swap A[i] nd A[j]
        end while
        swap A[s] nd A[j]
        return j; // Return the index of pivot after the partition
```

3b. Psuedocode for Probelm 3

```
partition(a[], low, high)

    pivot = a[high];                //1
    i = low-1;                      //1
    for (int j=low; j<high; j++)  //n

      if (a[j] < pivot)
```

```
            I++;                        //1
            int temp = a[i];            //1
            a[i] = a[j];                //1
            a[j] = temp;                //1
        End if
End for
        temp = a[i+1];                  //1
        a[i+1] = a[high];               //1
        a[high] = temp;                 //1

        return i+1;

    threeway( a[], low, high)

        if (low < high)
          pivot = partition(a, low, high);    //1
          threeway(a, low, pivot-1);                //T(n/2)
          threeway(a, pivot+1, high);               //T(n/2)
        End if
```

2T(n/2) + 0(n) =

A = 2, b = 2, d= 1

= O(nlogn) using Master theorem

The idea behind this algorithm is to use the partition function in order to find the value in which to use it as a pivot. From there the threeway function will move the pivot along each side of the array thus shrinking the size of the array. Also long as the value for low is less than the value for high then the loop should occur causing this to happen. After the algorithm completes the array should be sorted on both sides of the selected pivot.