Name: Inho Choi **ID:** 1801087

Problem 1:

Given the following array show the result after Two iterations of the each of the sorting algorithms indicated. One iteration being one full execution of the algorithm's outer most for/while loop.

a. Insertion Sort:

99	16	3	19	13	0	13	12	6
16	99	3	19	13	0	13	12	6
3	16	99	19	13	0	13	12	6

b. Bubble Sort

99	16	3	19	13	0	13	12	6
16	3	19	13	0	13	12	6	99
3	16	13	0	13	12	6	19	99

c. Selection Sort:

99	16	3	19	13	0	13	12	6
0	16	3	19	13	99	13	12	6
0	3	16	19	13	99	13	12	6

Name: Inho Choi **ID:** 160178

Problem 2.

Recall the implementation of partition discussed in class which consisted of two inner loops and one outer do-while loop.

a. Give a specific example of an integer array with some reasonable number of elements, N, where:

- **i.** both inner low and high loops execute only once per outer loop iteration, and the outer loop executes in total roughly N/2 times
- **ii.** the inner low loop executes once, the inner high loop executes roughly N times, and the outer loop executes only once.

I. | 30 | 31 | 42 | 56 | 66 | 87 | 40 | 4 | 8 | 21 | 15 | 3 |

II. | 10 | 31 | 42 | 56 | 66 | 37 | 40 | 44 | 26 | 21 | 15 | 32 |

b. What is the Big-O of the partition function we discussed for quicksort?

Case 1: low and high move one step per iteration. So each of the low and high loops reduce to just constant time, low and high will meet in the middle meaning the outer do-while loop went roughly N/2 step.

Case 2: where low moves all the way down to the end of the array. which means the low loop goes N steps, the high loop happens once, and the outer do-while happens only.

Case 3: where high moves all the way up to the front of the array. the "low" loop happens once, the high loop were N steps, the outer do-while happens only once.

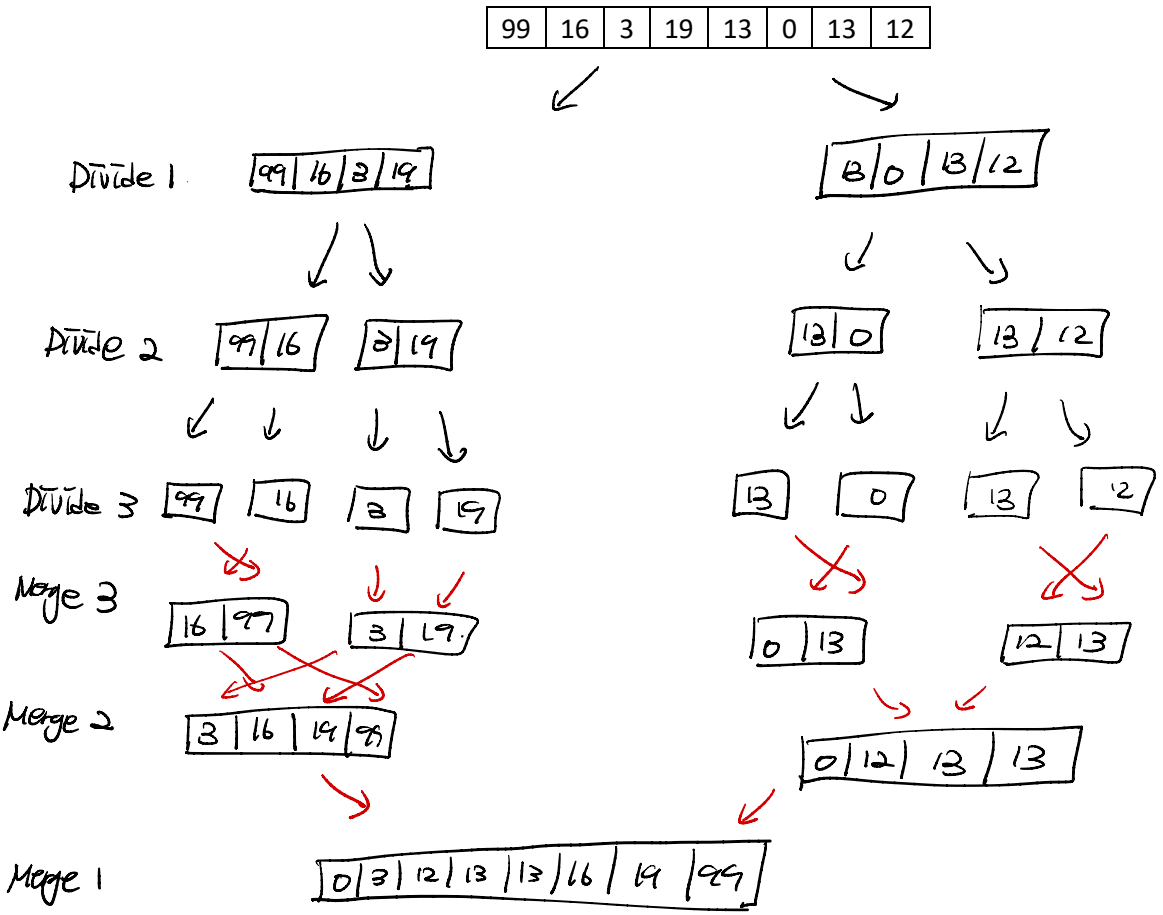
```
int partition(int a[], int low, int high) {
    int pi = low;
    int pivot = a[low];
    do {
        while (low <= high && a[low] <= pivot)
                low++;
        while (a[high] > pivot)
                high--;
        if (low < high)
                swap(a[low], a[high]);
    } while (low < high);
    swap(a[pi], a[high]);
    pi = high;
    return(pi);
}
```

Case 1:	Case 2:	Case 3:
O(1)	O(N)	O(1)
O(1)	O(1)	O(N)
O(N)	O(1)	O(1)

Name: Inho Choi **ID:** 1801180.

Problem 3.

a. Sort the following array using the Mergesort algorithm. Similar to what we discussed in class with books, draw out each recursive step illustrating the movement of the elements, including the merge.



b. What is the Big-O of the merge function we discussed for mergesort?

```
void merge(int data[], int n1, int n2) {
   int i = 0, j = 0, k = 0;
   int *temp = new int[n1 + n2];
   int *sechalf = data + n1;

   while (i < n1 || j < n2) {
       if (i == n1)
              temp[k++] = sechalf[j++];
       else if (j == n2)
              temp[k++] = data[i++];
       else if (data[i] < sechalf[j])
              temp[k++] = data[i++];
       else
              temp[k++] = sechalf[j++];
   }
   for (i = 0; i<n1 + n2; i++)
       data[i] = temp[i];
   delete[] temp;
}
```

We're only grabbing one book either from the first half **or** the second half each iteration. So we will iterated n1+n2 times to move that many books. Inside the loop just amounts to just constant time operations. so this loop in total is O(n1+n2); in the case of mergesort n1 roughly equals n2.

O(n1+n2)

Name: Inho Choi **ID:** 180in&n

Problem 4: Sorting Summary

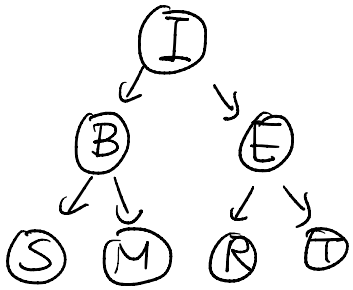
Fill out the following table or sorting properties, if there is no special condition for a particular case then leave it blank:

Sorting Algorithm	Selection	Insertion	Bubble	Quick	Merge
Average Complexity	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \log_2 n)$	$O(n \log_2 n)$
Worse Complexity	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n \log_2 n)$
Condition for Worse				mostly sorted or reserved	
Best Complexity	$O(n^2)$	$O(n)$	$O(n)$	$O(n \log_2 n)$	$O(n \log_2 n)$
Condition for Best		mostly sorted	mostly sorted		
Stable?	No	Yes	Yes	No	Yes

Problem 5.

Suppose we define the height of any tree is the number of nodes between the root node and the deepest leaf and that an empty tree has height 0.

a. Draw a <u>binary tree</u> of height 3 whose post-order traversal is S, M, B, R, T, E, I.

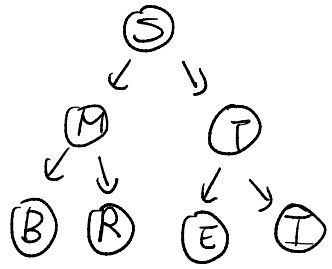


Name: Inho Choi **ID:** 1801181

b. What is the level-order traversal of the tree you created above?

I B E S M R T

c. Draw a binary tree of height 3 whose pre-order traversal is S, M, B, R, T, E, I.



d. What is the in-order traversal of the tree you created above?

B M R S E T I

Problem 6.

It is possible to construct a unique binary tree when given both its pre-order and in-order traversals (The same is true for post/in, but not pre/post). Given the following traversals, draw the binary tree:

Pre-order: A B D E F C G H J L K
In-order: D B F E A G C L J H K