Name: Ingho Choi **ID:** 1801787.

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.

① 16 99 3 19 13 0 13 12 6

② 16 3 99 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
16	3	13	0	13	12	6	19	99

① 16 99 3 19 13 0 13 12 6

16 3 99 19 13 0 13 12 6

16 3 19 99 13 0 13 12 6

16 3 19 13 99 0 13 12 6

16 3 19 13 0 99 13 12 6

16 3 19 13 0 13 99 12 6

16 3 19 13 0 13 12 99 6

16 3 19 13 0 13 12 6 99

② 16 3 19 13 0 13 12 6 99

16 3 13 19 0 13 12 6 99

16 3 13 0 19 13 12 6 99

16 3 13 0 13 19 12 6 99

16 3 13 0 13 12 19 6 99

16 3 13 0 13 12 6 19 99

16 3 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:** 1801198.

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.

```
int main() {
    int N = 8;
    int a[N] = {8, 99, 2, 41, 7, 77, 1, 25};   // pivot.
    int low = 0;
    int high = 7;
    int pivot = arr[0]; , int pi = low;

    do {
        while ( low <= high && a[low] <= pivot )
            low++;
        while ( a[high] > pivot )
            high--;
        if( low < high )
            swap(&a[low], &a[high]) // ① swap a[low]=99 and a[high]=1
                                    //   [8|1|2|41|7|77|99|25]
    }
    while(low < high);
    swap(&a[pi], &a[high]); // swap a[pi]=8 and a[high]=7.

}
```

② swap a[low]=41 and a[high]=7

[8|1|2|7|41|77|99|25]

low high.

[7|1|2|8|41|77|99|25]

```
void swap (int *a, int *b) {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

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

⇒ $O(n \log_2 n)$.

Name: Inho Choi **ID:** 1801181

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.

1. If array has 1 item, then return.
2. Split array in two equal sections.
3. Call MergeSort on the left half.
4. Call MergeSort on the right half.
5. Merge the halves back together.

99	16	3	19	13	0	13	12

① No 1 item.
② Split array
99 16 3 19 | 13 0 13 12
③ Call mergeSort on the left half.
99 16 3 19
④ No 1 item
⑤ Split array.
99 16 | 3 19
⑥ Call merge sort on the left half
99 16
⑦ No 1 item
⑧ Split array
99 | 16
⑨ Call MergeSort on the left
99
⑩ 1 item return
⑪ Call mergeSort on the right.
16
⑫ 1 item return
⑬ Merge the halves back together
16 99
⑭ Call MergeSort on the right.
16 99 | 3 19.
⑮ No 1 item.
⑯ Split array
3 | 19.
⑰ Call MergeSort on the left half
3
⑱ 1 item return.
⑲ Call MergeSort on the right half
19
⑳ Merge the halves back together
3 19.
㉑ Merge together
3 16 19 99
㉒ Call MergeSort on the right.
Store cut.
13 0 | 13 12.
recursive split → 13 0 | 13 12 ← recursive/split
recursive split → 13 | 0 | 13 | 12 ← recursive/split.
merge → 13 0 | 12 13 ← merge.
0 12 13 13 ← merge
㉓ Merge lastly.
3 16 19 99 | 0 12 13 13
0 3 12 13 13 19 99 ← merge

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

$O(n \log_2 n)$

Name: Jinho choi **ID:** 1801780.

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	N/A	N/A	N/A	$O(n^2)$ for already / mostly reversed ordered arrays or array with the same value repeated many times.	N/A
Best Complexity	$O(n^2)$.	$O(n)$.	$O(n^2)$	$O(n\log_2 n)$	$O(n\log_2 n)$
Condition for Best	N/A	$O(n)$ for already or nearly-ordered arrays.	$O(n)$ for already or nearly-ordered arrays.	N/A	N/A.
Stable?	unstable	Stable.	stable	unstable	Stable.

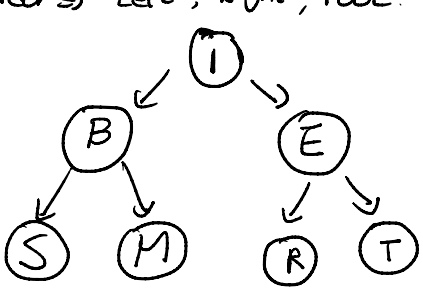
Problem 5.

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 binary tree of height 3 whose post-order traversal is S, M, B, R, T, E, I.

*Binary Tree ⇒ every node has at most two children nodes.

* post Order ⇒ Left, Right, root.



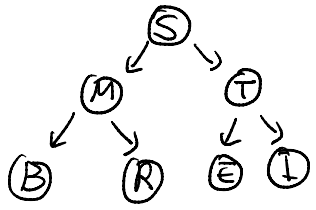
Name: Intho Choi **ID:** 1801787.

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.

* Pre-order ⇒ Root, Left, Right.



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

* In-order ⇒ Left, Root, Right.

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