# CSE222 / BİL505
# Data Structures and Algorithms
# Homework #6 – Report

Mustafa Selim Uçaral

## 1) Selection Sort

| | |
|---|---|
| **Time Analysis** | `for (i = lastIndex + 1; i < arr.length; ++i)`<br>This loop runs n−1 times for the first element n− 2 times for the second and so on summing to n*(n-1)/2. So the time complexity is the same for all scenarios, since the nature of the algorithm hasn't an checker that checks if the array is sorted. |
| **Space Analysis** | Since the algorithm uses a recursion, which the depth is n(size of the array), each recursive call uses a stack space. So the space complexity is O(n). |

## 2) Bubble Sort

| | |
|---|---|
| **Time Analysis** | The outer loop,<br>`for (int i = 0; i < arr.length - 1; ++i)`<br>runs n times where n is the length of the array.<br>The inner loop<br>`for (int j = 0; j < arr.length - i - 1; j++)`<br>also runs n times.<br>So the time complexity is O(n^2) in the worst case scenario.<br>In the best case, since we have<br>`if (!swapped)`<br>`  break;`<br>control, it detects the array is an already sorted array. So the time complexity is O(n). |
| **Space Analysis** | Since there are no additional data or array, the space complexity is O(1). This algorithm sorts the array in place. |

## 3) Quick Sort

| | |
|---|---|
| **Time Analysis** | In the best and average case the time complexity is O(nlogn).<br>`for (int j = start; j <= end - 1; j++)`<br>in this part algorithm makes swapping for all n elements. O(n)<br>`sort(start, pivot - 1);`<br>`sort(pivot + 1, end);`<br>in this part the time complexity is O(logn) since the algorithm divides the array into two halves around a pivot element. |

| | |
|---|---|
| | If we choose the pivot poorly it leads to worst case. It means that the partitions will be unbalanced. So time complexity for worst case is O(n^2) due to the depth of the recursive call stack. |
| **Space Analysis** | the space complexity of the algorithm is O(log n) on average and best cases due to recursive calls. But it can reach O(n) in the worst case with highly unbalanced partitions |

### 4) Merge Sort

| | |
|---|---|
| **Time Analysis** | This algorithm has a time complexity of O(n log n) because<br>`sort(left, middle);`<br>`sort(middle + 1, right);`<br>this part recursively divide the array in half resulting in log n levels of recursion, O(logn)<br>And<br>`while (i < n1 && j < n2)`<br>this part in the merge function performs a linear O(n) merge at each level |
| **Space Analysis** | This algorithm has a space complexity of O(n) due to the use of temporary arrays<br>`int L[] = new int[n1];`<br>`int R[] = new int[n2];`<br>during the merging process |

## General Comparison of the Algorithms

Quick sort and merge sort generally provide better efficiency for larger datasets due to their logarithmic nature, O(nlogn).

Bubble sort and selection sort are more memory efficient since they have in place sorting, they don't need additional memory.

Implementing bubble sort and selection sort are easier than implementing the quick and merge sort.