KIGALI INDEPENDENT UNIVERSITY (ULK)

POSTGRADUATE PROGRAM

MASTER OF SCIENCE IN INTERNET SYSTEMS

COMPUTER STRUCTURE MODULE

DATA STRUCTURE ASSIGNMENT 1

ACADEMIC YEAR: 2021-2022

DATE: 2th September, 2021

**Roll Number: 202110036**

**Names: Hodard Hazwinayo**

**Data Structures Assignment**

1. a program that solves the 8-puzzle problem using A\* or Greedy best first search

**Answer:**

**References: ** ** **

[**Github\_Hyperlink full assignment**](https://github.com/HodardHazwinayo/Data-Structure-)

2) Explaining the complexity of one of the two algorithms.

(Insertion Sort & Quick Sort):

**Answer:**

1. Insertion Sort is an easy-to-implement, stable sorting algorithm with time complexity of O(n²) in the average and worst case, and O(n) in the best case.

For very small n, Insertion Sort is faster than more efficient algorithms such as Quicksort or Merge Sort.

1. Overall time complexity of Quick Sort is **O(nLogn)**. In the worst case, it makes O(n2) comparisons, though this behavior is rare. The space complexity of Quick Sort is O(nLogn). It is an in-place sort (i.e. it doesn't require any extra storage)
2. Hence, Insertion Sort is an easy-to-implement, stable sorting algorithm with time complexity of **O(n²) in the average and worst case**, and O(n) in the best case. For very small n, Insertion Sort is faster than more efficient algorithms such as Quicksort or Merge Sort.

**In practical, let’s check on each algorithm below:**

1. **Insertion Sort**

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

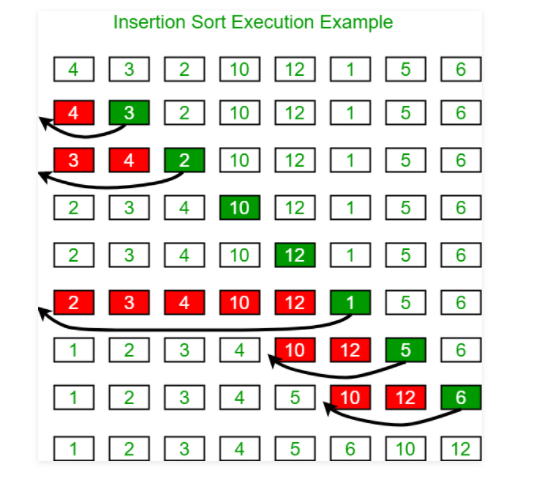
**Algorithm**

To sort an array of size n in ascending order:

1: Iterate from arr[1] to arr[n] over the array.

2: Compare the current element (key) to its predecessor.

3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

Example: 

1. **Quicksort**