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:**

**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 we sort playing cards in our 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.

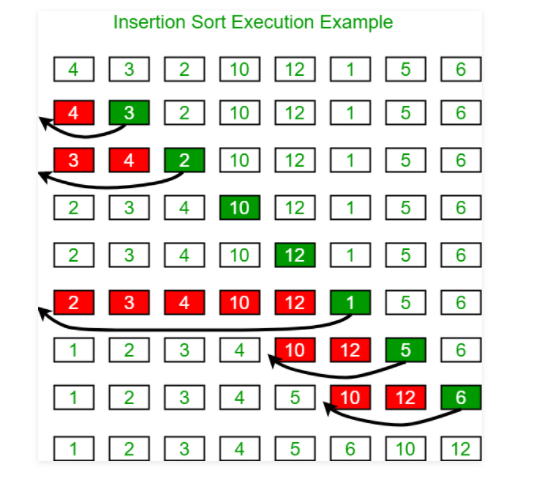
**Algorithm**

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: 

Example of the insertion sort program on the above image: 

**Time Complexity:** O(n^2)

**Auxiliary Space:** O(1)

**Boundary Cases**: Insertion sort takes maximum time to sort if elements are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.

**Algorithmic Paradigm:** Incremental Approach

**Sorting In Place**: Yes

**Stable**: Yes

**Online:** Yes

**Uses:** Insertion sort is used when number of **elements is small**. It can also be useful when input array is almost sorted, only few elements are misplaced in complete **big array**.

1. **Quicksort**

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Pseudo Code for recursive QuickSort function:

/\* low --> Starting index, high --> Ending index \*/

quickSort(arr[], low, high)

{

if (low < high)

{

/\* pi is partitioning index, arr[p] is now

at right place \*/

pi = partition(arr, low, high);

quickSort(arr, low, pi - 1); // Before pi

quickSort(arr, pi + 1, high); // After pi

}

}

The time complexity of the quicksort is dependent upon:

1. What we choose to be our partition, and
2. How sorted the list already is

* The average runtime for an unsorted list, and a partition close to the median is: 0(nlogn).
* The average runtime for sorted (or nearly-sorted) list, or a partition that is far from the median is 0(n2).

Note: We are not recommended to use quicksort for mostly-sorted lists.

Meaning that: For arrays that are nearly or completely sorted, quicksort operates at its worst. In other words, the average runtime for a sorted or a nearly-sorted list is quicksort’s worst case runtime: O(n²).

Example program of the quicksort: 

**……………………………………………………………………………………………………………………………………………..**

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.