1. **Introduction**

The sorting process is defined as the operation of arranging data in ascending or descending order numerically or alphabetically. It is an important operation in computer programming. The sorted data   is beneficial for searching, insertion and deletion.

There are many sorting algorithms to arrange data. All sorting algorithms are problem specific. Generally, the particular algorithm is chosen according to depending on properties of data and operations performed on data. Accordingly, we will want to know the complexity of each algorithm, that is, to know which algorithm is the most efficient by time and space.  These complexities will help us to compare each algorithm.

In this experiment, we will design to compare different algorithms for finding the median of an unsorted list of n numbers. There are following candidate methods:

1. Sort all elements by Insertion-sort and return the⌈𝑛/2⌉’th element in the list,
2. Sort all elements by Merge-sort and return the ⌈𝑛/2⌉’th element in the list,
3. Store all elements in a max-heap and apply ⌊𝑛/2⌋ times max removal. Return the max element in the root,
4. Not sort the list, but apply a quick select algorithm, which is based on array partitioning, as described in the class. While partitioning, choose the pivot element as the first element in an array,
5. Apply quick select algorithm, but this time use median-of-three pivot selection1,
6. Apply quick select algorithm with median-of-medians pivot selection

In addition, these methods will be applied on input lists which have various characteristics following:

* An array with completely random integer
* An array with reverse order integer
* An array with at least %25 of sorted
* An array with at most %25 of unsorted

Also, the input size is varied from 10 to 100K.

1. **Sorting Algorithms**
   1. Insertion Sort Algorithm

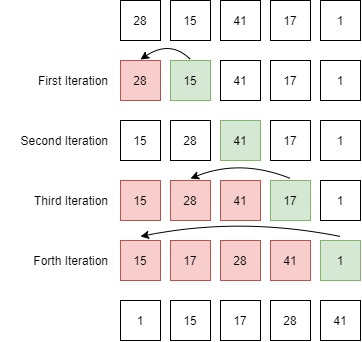
Insertion sort is the method most card players use to sort their cards. They keep the cards dealt so far in sorted order, and as each new card arrives, they insert it into its proper relative position.

Insertion sort works as follows:

* Firstly, the element and its adjacent are compared.
* And if at every comparison, the particular position where inserted of the element is found, space is created for it by shifting the other elements one position to the right and inserting the element suitable position.
* The above procedure is repeated until all the element in the array is at their appropriate position.

Let us now understand working with the following example:

Consider the following array: 28, 15, 41, 17, 1



**First Iteration:** Compare 28 with 15. Swap 15 and 28, because the comparison shows 15 < 28

**Second Iteration**: Compare 41 with 28. There is no swapping, due to 41 > 28

**Third Iteration:** Compare 17 with 41, 28, 15. Swap the 17, 48 and 17,28 respectively, but not swap 17, 15 because 17 > 15.

**Fourth Iteration:** Compare 1 with 41, 28, 17, 15. Swap the place of 1 and places of 41, 28, 17, 15 respectively.

**Pseudocode:**

Algorithm insertion\_sort(A[0...n-1])

for i = 1 to n-1

key ← i - 1

while j >= 0 and A[j] > key

A[j+1] ← A[j]

j ← j – 1

end while

A[j+1] ← key

end for

**Time Complexity:**

* + - * **The Best Case:** The best case of Insertion Sort Algorithm is that the given input sequence is already sorted. It gives the linear running time complexity O(n). During each iteration, only one comparison is done.
      * **The Worst Case:** The worst case of Insertion Sort Algorithm is that the given input array is sorted by descending order. The set of all worst-case inputs consists of all arrays where each element is the smallest or second-smallest of the elements before it. In these cases, every iteration of the inner loop will scan and shift the entire sorted subsection of the array before inserting the next element. This gives insertion sort a quadratic running time (i.e., O(n2)).
  1. Merge Sort Algorithm