# **Requirements**

## **Introduction**

Sorting techniques have a wide variety of applications. Computer-Aided Engineering systems often use sorting algorithms to help reason about geometric objects, process numerical data, rearrange lists, etc. In general, therefore, we will be interested in sorting a set of records containing keys, so that the keys are ordered according to some well defined ordering rule, such as numerical or alphabetical order. Often, the keys will form only a small part of the record. In such cases, it will usually be more efficient to sort a list of keys without physically rearranging the records.

**Objective of the Project**

This deals with the presentation of information or data in an appreciable form and also to minimize the time used in searching for items. However, it can be seen that a well-sorted or organized file enhances easy searching of data while an unsorted one will pose little or great problems to locate a given item in a large list. However, the comparison process will be based on the following:

(i) Memory space used by methods

(ii) Different techniques used for sorting

(iii) Number of comparison made during sorting process

(iv) Coding based on the sorting algorithm

**Signification of the Project**

Sorting algorithm was designed to enable the people and the society to be acquainted with the arrangement of data and items. Above topic will let us know the organization of data in the memory location and also make proper use and utilization of the computer time.

**Benefits of Sorting Techniques**

The advantage of an algorithm is that it can provide a method to follow when attempting to solve a real problem.

The other common form of problem-solving is "trial and error", which is often much more inefficient than

studying the problem,

looking for an algorithm that can solve that problem,

then use the algorithm to build a concrete solution.

**Performance Criteria**

There are several criteria to be used in evaluating a sorting algorithm:

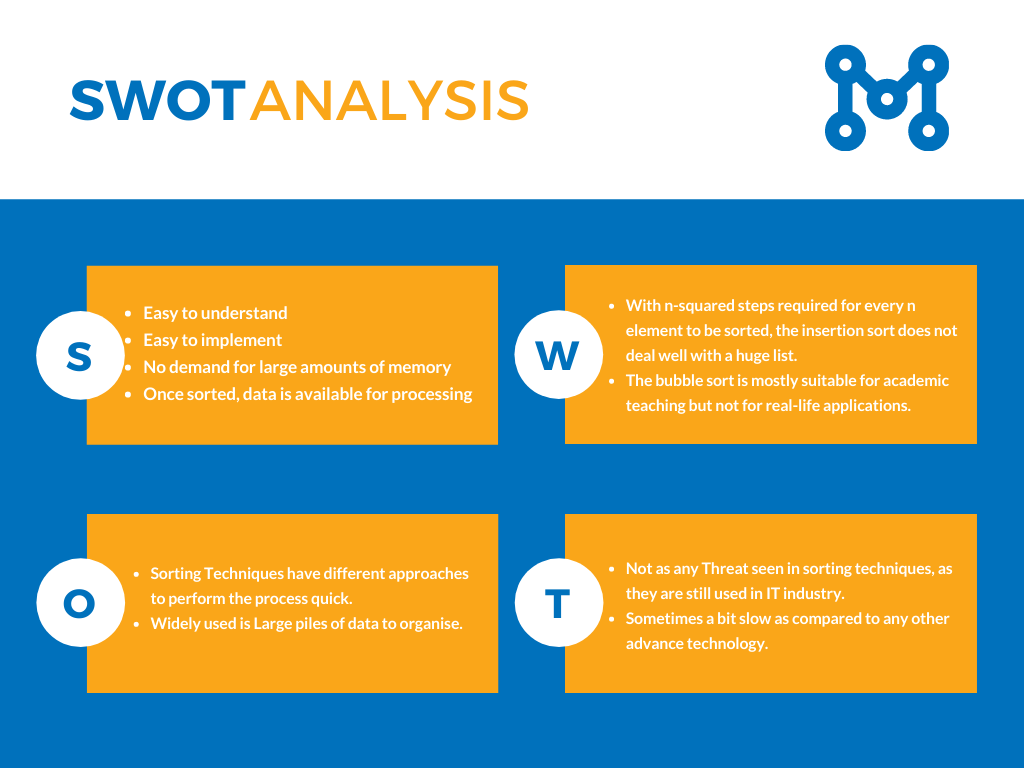
Running time: - Typically, an elementary sorting algorithm requires O(N2) steps to sort N randomly arranged items.  More sophisticated sorting algorithms require O(N log N) steps on average. Algorithms differ in the constant that appears in front of the N2 or N log N.  Furthermore, some sorting algorithms are more sensitive to the nature of the input than others.  Quicksort, for example, requires O(N log N) time in the average case, but requires O(N2) time in the worst case.

Memory requirements: - The amount of extra memory required by a sorting algorithm is also an important consideration.  In place sorting algorithms are the most memory efficient, since they require practically no additional memory.  Linked list representations require an additional N words of memory for a list of pointers. Still other algorithms require sufficient memory for another copy of the input array. These are the most inefficient in terms of memory usage.

Stability: - This is the ability of a sorting algorithm to preserve the relative order of equal keys in a file.

**SWOT Analysis**

Below is the SWOT Analysis of different sorting algorithms:

****

**Detail requirements**

#### **High Level Requirements:**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Description** | **Category** | **Status** |
| H\_SA01 | User should be able to give input of their choice | Technical | To Be Implemented |
| H\_SA02 | User should be able to input array of their choice | Technical | To Be Implemented |
| H\_SA03 | User should be able to perform bubble sort | Technical | To Be Implemented |
| H\_SA04 | User should be able to perform insertion sort | Technical | To Be Implemented |
| H\_SA05 | User should be able to perform quick sort | Technical | To Be Implemented |
| H\_SA06 | User should be able to perform merge sort | Technical | To Be Implemented |

#### **Low Level Requirements:**

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Description** | **HLR ID** | **Status** |
| L\_SA01 | User should be able to give array input for the bubble sort technique | H\_SA02,      H\_SA03 | To Be Implemented |
| L\_SA02 | User should be able to give array input for the insertion sort technique | H\_SA02,      H\_SA04 | To Be Implemented |
| L\_SA03 | User should be able to give array input for the quick sort technique | H\_SA02,      H\_SA05 | To Be Implemented |
| L\_SA04 | User should be able to give array input for the merge sort technique | H\_SA02,      H\_SA06 | To Be Implemented |

# 

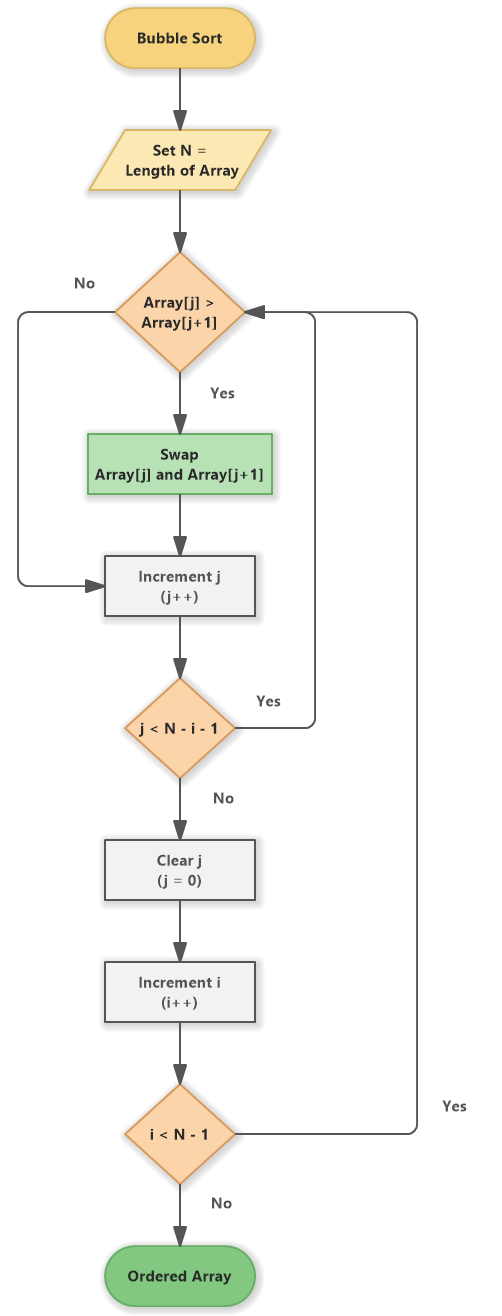
# 

# **Design**

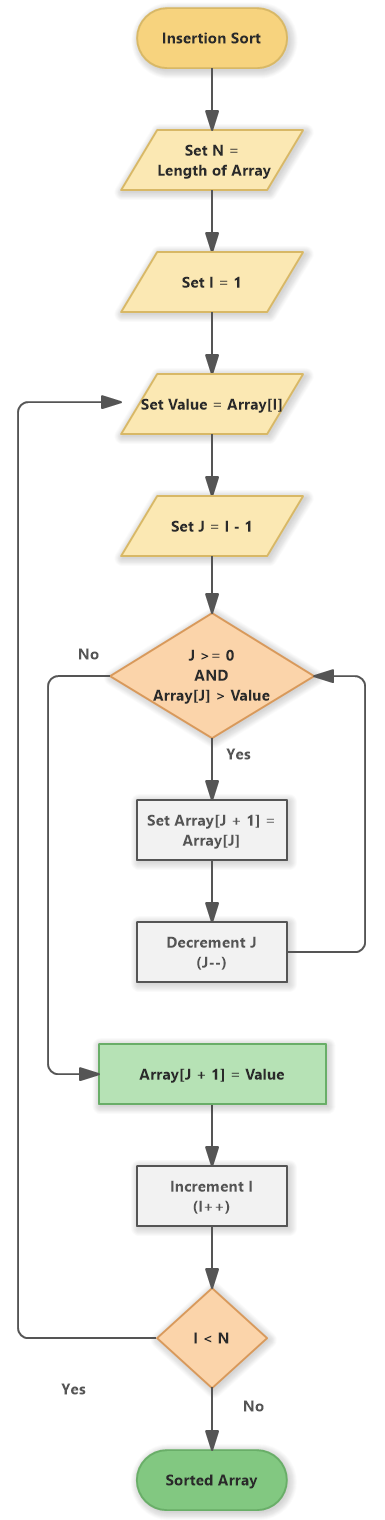
## **High Level Design**

### **Architecture Design :**IMG_257

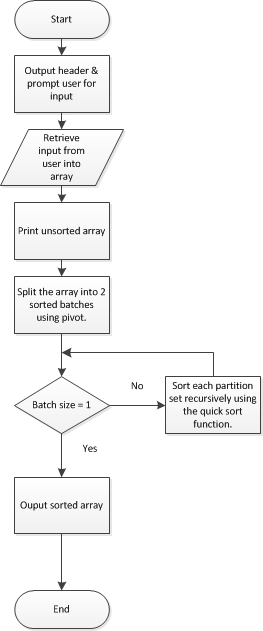
#### **Bubble Sort :**

****

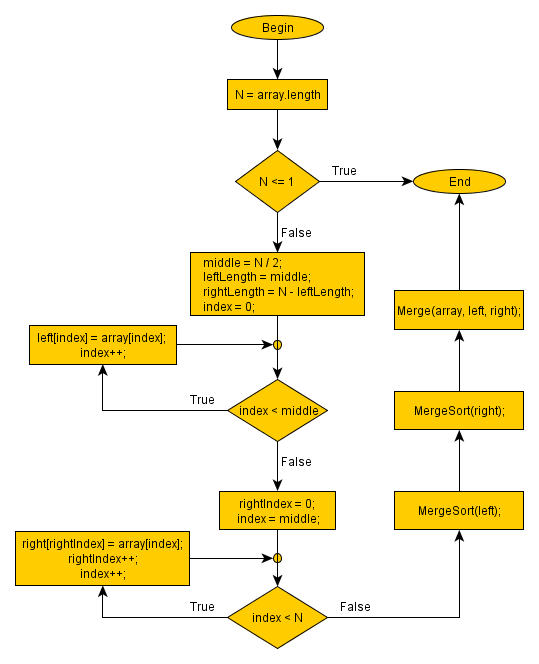
#### **Insertion Sort :**

****

#### **Quick Sort :**

****

#### **Merge Sort :**

****

# 

# **TEST PLAN:**

## **Table no: High level test plan**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Exp I/P** | **Exp O/P** | **Actual Out** | **Type Of Test** |
| H\_001 |  |  |  |  | Requirement based |
| H\_002 |  |  |  |  |  |
| H\_003 |  |  |  |  |  |
| H\_004 |  |  |  |  |  |
| H\_005 |  |  |  |  |  |
| H\_006 |  |  |  |  |  |

## **Table no: Low level test plan**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Description** | **Exp I/P** | **Exp O/P** | **Actual Out** | **Type Of Test** |
| L\_001 |  |  |  |  |  |
| L\_002 |  |  |  |  |  |
| L\_003 |  |  |  |  |  |