**PROJECT PROPOSAL 2012-13**

**Project Name :-** Comparative Study Over Sorting Algorithms

**Group No. :-** One (1)

**Members :-** Seven (7)

**Names :-** MD Rameez Raja

Manish Kumar Gupta

Arup Shngha

MD Imran Ansari

Sujoya Das

Swarna Basu

Sarita Chaurasia

**Lead By :-** MD Rameez Raja

**Under By :-** Toufique Sir

**Front End :-** C Language.

**Back End :-** No such data base ; Random input from the users.

**Project type :-** Analytical Project.

**Aim :-**

* **To study different sorting algorithms**
* **To develop different sorting programs**
* **To measure different sorting programs complexity**
* **To select the best sorting algorithm among them, and**
* **Try to develop another sorting algorithm better than the best.**

**CONTENT**

**\*Acknowledgement**

**\*Preface**

**\*Classification**

**\* Why we choose this project**

**\* Types of algorithm**

**\* Comparision of algorithum**

**\* Asymptotic notation**

**\* Conclusion**

**ACKNOWLEDGEMENT :~**

**The analysis of the project work wishes to express my gratitude to Mr. Tofique Ahmad Gazi for allowing the degree attitude and providing effective guidance in development of this project work. His conscription of the topic and all the helpful hints, he provided, contributed greatly to successful development of this work, without being pedagogic and overbearing influence.**

**I also express my sincere gratitude to Mr. Tofique Ahmad Gazi .**

**Finally, I take this opportunity to thank to Dr. Arun Kumar Dey Principal of Elite Polytechnic Institude for giving us the scope of carrying out the project work.**

**PREFACE :-**

**In computer science, a sorting algorithm is an algorithm that puts elements of a list in a certain order. The most-used orders are numerical order and lexicographical order. Efficient sorting is important for optimizing the use of other algorithms (such as search and merge algorithms) that require sorted lists to work correctly; it is also often useful for canonicalizing data and for producing human-readable output. More formally, the output must satisfy two conditions:**

**1. The output is in non decreasing order (each element is no smaller than the previous element according to the desired total order);**

**2. The output is a permutation (reordering) of the input.**

**Since the dawn of computing, the sorting problem has attracted a great deal of research, perhaps due to the complexity of solving it efficiently despite its simple, familiar statement. For example, bubble sort was analyzed as early as 1956.[1] Although many consider it a solved problem, useful new sorting algorithms are still being invented (for example, library sort was first published in 2006). Sorting algorithms are prevalent in introductory computer science classes, where the abundance of algorithms for the problem provides a gentle introduction to a variety of core algorithm concepts, such as big O notation, divide and conquer algorithms, data structures, randomized algorithms, best, worst and average case analysis, time-s** **pace tradeoffs, and lower bounds.**

CLASSIFICATION:-

Sorting algorithms used in computer science are often classified by:

* Computational complexity (worst, average and best behavior) of element comparisons in terms of the size of the list (*n*). For typical sorting algorithms good behavior is O(*n* log *n*) and bad behavior is O(*n*2). (See [Big O notation](http://en.wikipedia.org/wiki/Big_O_notation).) Ideal behavior for a sort is O(*n*), but this is not possible in the average case. [Comparison-based sorting algorithms](http://en.wikipedia.org/wiki/Comparison_sort), which evaluate the elements of the list via an abstract key comparison operation, need at least O(*n* log *n*) comparisons for most inputs.
* [Computational complexity](http://en.wikipedia.org/wiki/Computational_complexity_theory) of swaps (for "in place" algorithms).
* Memory usage (and use of other computer resources). In particular, some sorting algorithms are "[in place](http://en.wikipedia.org/wiki/In-place_algorithm)". Strictly, an in place sort needs only O(1) memory beyond the items being sorted; sometimes O(log(*n*)) additional memory is considered "in place".
* Recursion. Some algorithms are either recursive or non-recursive, while others may be both (e.g., merge sort).
* Stability: [stable sorting algorithms](http://en.wikipedia.org/wiki/Sorting_algorithm#Stability) maintain the relative order of records with equal keys (i.e., values).
* Whether or not they are a [comparison sort](http://en.wikipedia.org/wiki/Comparison_sort). A comparison sort examines the data only by comparing two elements with a comparison operator.
* General method: insertion, exchange, selection, merging, *etc.*. Exchange sorts include bubble sort and quicksort. Selection sorts include shaker sort and heapsort.
* Adaptability: Whether or not the presortedness of the input affects the running time. Algorithms that take this into account are known to be [adaptive](http://en.wikipedia.org/wiki/Adaptive_sort).

**WHY WE CHOOSE THIS PROJECT ?**

**\* An *internal sort* requires that the collection of data fit entirely in the computer’s main memory.**

***\* Sorting* is a process that organizes a collection of data into either ascending or descending order.**

**\* We can use an *external sort* when the collection of data cannot fit in the computer’s main memory all at once but must reside in secondary storage such as on a disk.**

**\* We will analyze only internal sorting algorithms.**

**\* Sorting also has indirect uses. An initial sort of the data can significantly enhance the performance of an algorithm.**

**\* Majority of programming projects use a sort somewhere, and in many cases, the sorting cost determines the running time.**

**\* A comparison-based sorting algorithm makes ordering decisions only on the basis of comparisons.**

**Among different types of sorting algorithm each has its advantages and disadvantages and to determine which algorithm is better and faster for sorting a set of data we have done a comparative study of different types of sorting algorithms.**

**TYPES OF SORTING ALGORITHMS :~**

**1.American flag sort 11. Flash sort**

**2.Bead sort 12.Heap sort**

**3.Bogo sort 13.Insertion sort**

**4.Bubble sort 14.Intro sort**

**5.Bucket sort 15.Merge sort**

**6.Cartesian sort 16.Odd even sort**

**7.Cocktail sort 17.Quick sort**

**8.Comb sort 18.Radix sort**

**9.Counting sort 19.Selection sort**

**10.Cycle sort 20.Shell sort**

**Comparison of algorithms**

In this table, *n* is the number of records to be sorted. The columns "Average" and "Worst" give the time complexity in each case, under the assumption that the length of each key is constant, and that therefore all comparisons, swaps, and other needed operations can proceed in constant time. "Memory" denotes the amount of auxiliary storage needed beyond that used by the list itself, under the same assumption. These are all [comparison sorts](http://en.wikipedia.org/wiki/Comparison_sort). The run time and the memory of algorithms could be measured using various notations like theta, omega, Big-O, small-o, etc. The memory and the run times below are applicable for all the 5 notations.

In future we try to write a chart to compares every type of sorting algorithm. Including in the chat that best case , average case ,worst case ,memory ,stable ,method of an algorithms.

BUBBLE SORT

Best case: \mathcal{} n

Average case: \mathcal{} n^2

Worst case: \mathcal{} n^2

Memory: \mathcal{} {1}

Stable: Yes

Method: Exchanging

BUBBLE SORT PROGRAM :

#include "stdio.h"

#include "conio.h"

void main( )

{

int arr[5] = { 25, 17, 31, 13, 2 } ;

int i, j, temp ;

clrscr();

printf ( "\n\nArray before sorting:\n") ;

for ( i = 0 ; i <= 4 ; i++ )

printf ( "%d\t", arr[i] ) ;

for ( i = 0 ; i <= 3 ; i++ )

{

for ( j = 0 ; j <= 3 - i ; j++ )

{

if ( arr[j] > arr[j + 1] )

{

temp = arr[j] ;

arr[j] = arr[j + 1] ;

arr[j + 1] = temp ;

}

}

}

printf ( "\n\nArray after sorting:\n") ;

for ( i = 0 ; i <= 4 ; i++ )

printf ( "%d\t", arr[i] ) ;

getch();

}

ASYMPTOTIC NOTATION

\* For given function g(n),we denote by o(g(n)) the set of functions that are different than g(n) by a constant Asymptotic upper bound.

\* Here, we assume that we should compare the algorithm in an independent machine, i.e; if there is different types of processor, then their system may vary the execution time depending on the instruction format for execution each and every operations. If we consider two different machine for any algorithm to measure the complexity then we should prefer ***ASYMPTOTIC NOTATION.***

CONCLUSION:-

By analyzing an algorithm, we mean to study the performance of an algorithm including the assertion of its correctness and a determination of the cost of its execution. Although a given algorithm is often analyzed in a particular way that is most suitable for such an algorithm, we are more interested in general procedures and techniques that can be used to study the performance of classes of algorithms. To be able to talk about general analysis techniques will not only add to our understanding of the behavior of a class of algorithms but will also, in many cases, lead to useful synthesis procedures. A good example illustrating these points is the various techniques that can be used to analyze a class of sorting algorithms which can be modelled as networks made up of comparator modules. In this paper, we discuss several approaches to such an analysis problem. Moreover, synthesis procedures suggested by these analysis techniques will also be presented.