Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 4, Issue. 1, January 2015, pg.291 – 306

RESEARCH ARTICLE

PERFORMANCE ANALYSIS OF SORTING ALGORITHM

V.P.Kulalvaimozhi, M.Muthulakshmi, R.Mariselvi, G.Santhana Devi, C.Rajalakshmi & C. Durai

Department of Computer Science, Kamarajar Govt. Arts College, Surandai – 627 859

*Corresponding Author: vpkulal@yahoo.com

ABSTRACT

An algorithm is a sequence of steps that gives method of solving a problem. It creates the logic of program. As an algorithm is executed, it uses the computer's Central Processing Unit (CPU) to perform operations and its memory to hold the program and data. Sorting means arranging the data in an orderly manner. The main objective is to compare the various sorting algorithms and these sorting algorithms are implemented by using a C++ program also the asymptotic complexity of each sorting algorithm is prepared.

Keywords: Bubble sort, Insertion sort, Selection sort, Shell sort, Merge sort, Quick sort, Heap sort, Binary Tree sort, Address calculation sort, Radix sort.

1. INTRODUCTION

Performance analysis refers to the task of determining how much computing time and storage an algorithm requires. The space complexity of a program is the amount of memory it needs to run to completion. The time complexity of a program is the amount of computer time it needs to run to completion. Sorting refers to the operation of arranging data in some given order, such as increasing or decreasing with numerical data or alphabetically with character data. All sorting algorithm apply to specific kind of problems. Some sorting algorithm apply to small number of elements, some sorting algorithm suitable for floating point numbers, some are fit for specific range like (0 1). Some sorting algorithm are used for large number of data, some are used for data with duplicate values. Performance evaluation can be loosely divided into two major phases:

1. Priori estimates
2. Posteriori estimates

Simply we can refer these as performance analysis and performance measurement respectively.

The complexity of a sorting algorithm measures the running time as a function of the number of n items to be sorted. Each sorting algorithm S will be made up of the following operations, where A_1, A2,....An contain the items to be sorted and B is an auxiliary location.
1) Comparisons which test whether $A_i<A_j$ or test whether $A_i<B$.
2) Interchanges which switch the contents of A_i and A_j or A_i and B.
3) Assignments which set $B=A_i$ and then set $A_j=B$ or $A_j=A_i$.

Generally the complexity function measures only the number of comparisons, since the number of other operations is at most a constant factor of the number of other operations is at most a constant factor of the number of comparisons. Suppose space is fixed for one algorithm then only run time will be considered for obtaining the complexity of algorithm. We take 3 cases for complexity of algorithms.

A) Best case

B) Worst case

C) Average case

Best case: In this case, algorithm searches the element in first time itself. So taking this case for complexity of algorithm doesn't tell too much. Let us take a case of linear search, if it finds the element at first time itself then it behaves as best case.

Worst case: In this case, we find the element at the end or when searching of element fails. Suppose the element for which algorithm is searching is the last element of array or it's not available in array then algorithm behaves as worst case. This case is necessary in the view that algorithm will perform at least up to this efficiency and it will not go for less than this efficiency.

Average case: In this case, the average number of steps but since data can be at an place, so finding exact behavior of algorithm is difficult. As the volume of data increases average of algorithm behaves like worst case of algorithm. Analyzing the average case behavior of algorithm is little bit complex than best and worst case.

2. SORTING METHODS:

The sorting methods can be divided into two parts.

1. Internal sorting method

2. External sorting method

In an internal sorting method, data is going to be sorted will be in main memory. In an external sorting data will be on auxiliary storage like tape, floppy disk etc.,

2.1. Bubble sort:

In the bubble sort, the consecutive elements of the table are compared and if the keys of the two elements are not found in proper order, they are interchanged. It starts from the beginning of the table and continue till the end of the table. As a result of this the element with the largest key will be pushed to the last element's position. After this the second pass is made.The second pass is exactly like the first one except that this time the elements except the last are considered. After

the second pass, the next largest element will be pushed down to the next to last position. The second pass is followed by a third and so on until either (n-1) passes have been made or no interchange takes place in a pass. Here n is assumed to be the number of elements in the table.

It may be noted that the non occurrence of any interchange in a pass ensures that the table is sorted. Ie, each number slowly bubbles up to its proper position.

Ex;

Consider the following numbers are stored in an array:

Original Array: 32,51,27,85,66,23,13,57

Pass 1 : 32,27,51,66,23,13,57,85

Pass 2 : 27,33,51,23,13,57,66,85

Pass 3 : 27,33,23,13,51,57,66,85

Pass 4 : 27,23,13,33,51,57,66,85

Pass 5 : 23,13,27,33,51,57,66,85

Pass 6 : 13,23,27,33,51,57,66,85

Advantages:

1. Straightforward, simple and slow.

2. Stable.

3. Inefficient on large tables.

2.2.Insertion sort:

An insertion sort is one that sorts a set of elements by inserting elements into an existing sorted array. Insertion sort uses the least CPU time, the smallest comparions and assignments. In this sort, i^{th} pass we insert the i^{th} element A[i] into its rightful place among A[1],A[2],….A[i-1] which were previously placed in sorted order. After doing this insertion, the elements occupying A[1],……A[i] are in sorted order.

Ex;

Consider the following numbers are stored in an array:

Original Array: 77,33,44,11,88,22,66,55

Pass 1 : 33,77,44,11,88,22,66,55

Pass 2 : 33,44,77,11,88,22,66,55

Pass 3 : 11,33,44,77,88,22,66,55

Pass 4 : 11,22,33,44,77,88,66,55

Pass 5 : 11,22,33,44,55,77,88,66

Pass 6 : 11,22,33,44,55,66,77,88

Advantages:

1. Efficient for small list and mostly sorted list.
2. Sort big array slowly.
3. Save memory

2.3.Selection sort:

A selection sort is one in which successive elements are selected in order and placed into their proper sorted positions. The elements of the input may have to be preprocessed to make the ordered selection possible. This sort is used to represent the selection of an element and keeping it in sorted order.

Suppose an array A with n elements in memory. The selection sort algorithm for sorting an array A works as follows.

i) To find the smallest element in the list and put in the first position.

ii) To find the second smallest element in the list and put in the second position and so on.

Ex;

Consider the following numbers are stored in an array:

Original Array: 77,33,44,11,88,22,66,55

Pass 1 :11,33,44,77,88,22,66,55

Pass 2 :11,22,44,77,88,33,66,55

Pass 3 :11,22,33,77,88,44,66,55

Pass 4 :11,22,33,44,88,77,66,55

Pass 5 :11,22,33,44,55,77,66,88

Pass 6 :11,22,33,44,55,66,77,88

Advantages:

1. Improves the performance of bubble sort and also slow.
2. Unstable but can be implemented as a stable sort.
3. Quite slow for large amount of data.

2.4. Shell sort:

Shell sort is also called **Diminishing increment** sort. D.L. Shell proposed an improvement on insertion sort in 1959 named after him as shell sort. This method sorts separate sub files of the original file. These sub files contain every k^{th} element of the original file. The value of k is called an increment.

Ex;

Consider the following numbers are stored in an array:

Original Array: 25,57,48,37,12,92,86,33

Pass 1 : 25,57,33,37,12,92,86,48

Pass 2 : 25,12,33,37,48,92,86,57

Pass 3 : 12,25,33,37,48,57,86,92

Advantages:

1. Efficient for large list.
2. It requires relative small amount of memory, extension of insertion sort.
3. Fastest algorithm for small list of elements. 4. More constraints, not stable.

2.5. Merge sort:

Merging is the process of combining two or more sorted arrays into a third sorted array. Merge sort is a sorting algorithm that has the nice property elements are to be sorted in non-decreasing order. Given a sequence of n elements a[1],a[2]….a[n], the general idea is to split them in two sets resulting sorted sequences are merged to produce a single sorted sequence of n elements. This is an ideal example of the divide-and-conquer strategy in which the splitting is into two equal-sized sets and the combining operation is the merging of two sorted sets into one.

Ex;

Consider the following numbers are stored in an array:
Original Array: 310,285,179,652,351,423,861,254,450,520
Pass1: 310,285,179,652,351|423,861,254,450,520
Pass 2:
285,310,179|652,351|423,861,254,450,520
Pass 3:
179,285,310|652,351|423,861,250,450,520
Pass 4:
179,285,310|351,652|423,861,250,450,520
Pass 5:
179,285,310,351,652|423,861,250,450,520
Pass 6:
179,285,310,351,652|423,861,250|450,520

Pass 7:
179,285,310,351,652|250,423,861|450,520
Pass 8:
179,285,310,351,652|250,423,861|450,520
Pass 9:
179,285,310,351,652|250,423,450,520,861
Pass 10:
179,285,250,310,351,423,450,520,652,861

Advantages:

1. Well for very large list, stable sort.
2. A fast recursive sorting.
3. Both useful for internal and external sorting.
4. It requires an auxiliary array that is as large as the original array to be sorted.

2.6.Quick sort:

Quick sort is a sorting algorithm that is based on the fact that it is faster and easier to sort two small lists than one larger one. This algorithm is used to sort a list of data elements significantly faster than any of the common simple sorts. The basic strategy of Quick sort is divide and conquer. In this sort, we divide the original list into two sub lists.

Ex;

Consider the following numbers are stored in an array:

Original array	:	25,57,48,37,12,92,86,33
Pass 1	:	12,25,57,48,37,92,86,33
Pass 2	:	12,25,48,37,33,57,92,86
Pass 3	:	12,25,37,33,48,57,92,86
Pass 4	:	12,25,33,37,48,57,92,86
Pass 5	:	12,25,33,37,48,57,86,92

Advantages:

1. Fastest sorting algorithm in practice.
2. Available in many standard libraries.
3. O (log n) space usage.
4. Unstable sort and complex for choosing a good pivot element.

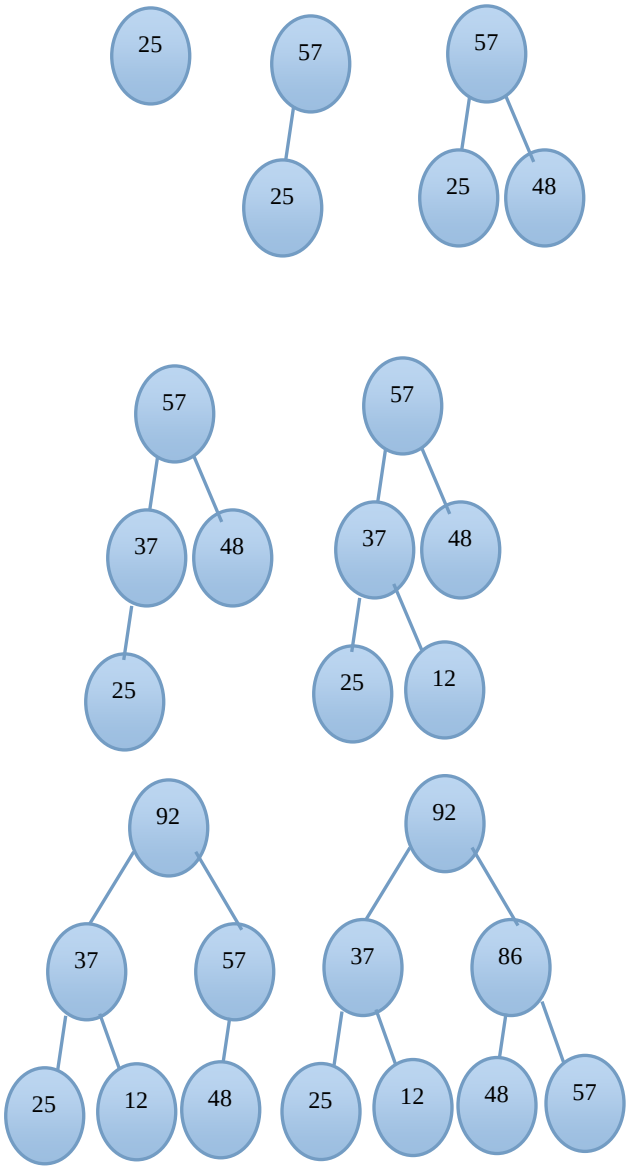
2.7.Heap sort:

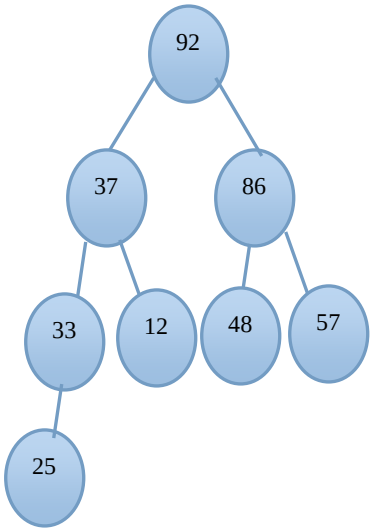
The heap sort sorting algorithm sorts an array of values that represent a tree with a special property: the heap property. A heap is a complete binary tree and is implemented in array as

sequential representation rather than linked representation. A heap is called max heap or descending heap if every node has a value greater than or equal to the value of every child of that node. So in max heap root will keep the highest value.
Ex;

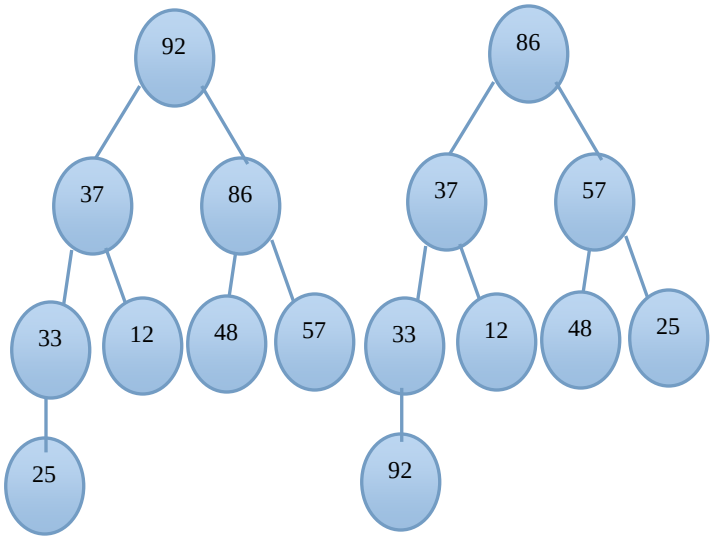
Consider the following numbers are stored in an array:

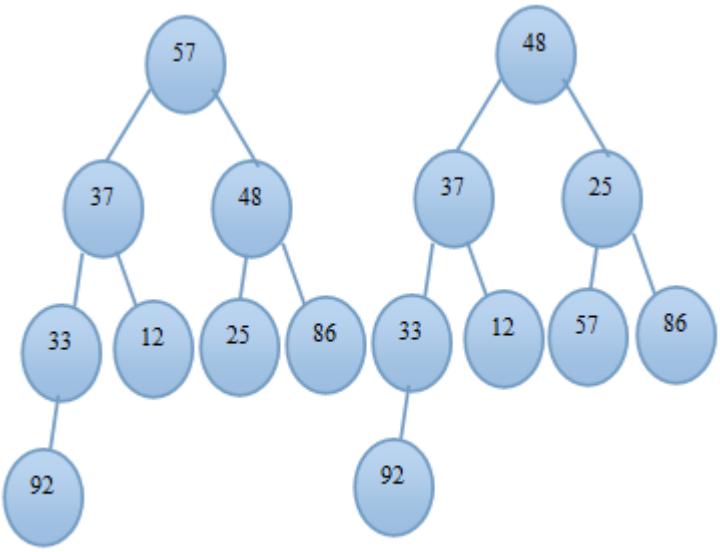
Original array : 25,57,48,37,12,92,86,33

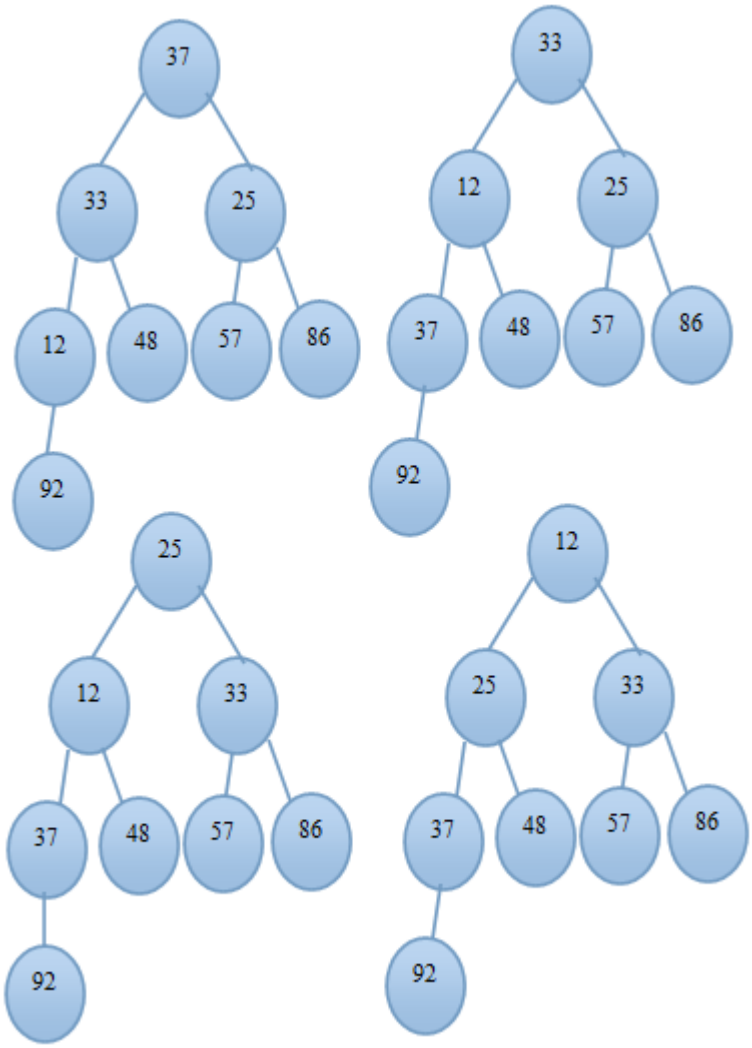




Similarly a heap is called min heap or ascending heap if every node of heap has a value less than or equal to the value of every child node of that node.




57
37
48
33
12
25
86
92
48
37
25
33
12
57
86
92


37
33
25
12
48
57
86
92
33
12
25
37
48
57
86
92
25
12
33
37
48
57
86
92
12
25
33
37
48
57
86
92

Advantages:

1. More efficient version of selection sort.
2. No need extra buffer.
3. It does not require recursion.
4. Slower than Quick and Merge sorts.

2.8.Binary tree sort:

A binary tree is a finite set of nodes. It consists root node with 2 disjoint binary trees called left subtree and right subtree. Each element of the tree is known as node.

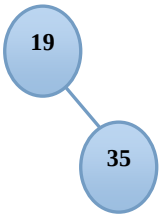
Ex;

Consider the following list of elements:

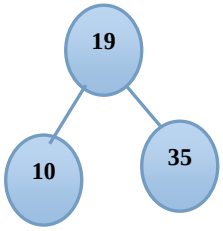
19,35,10,12,46,6,40,3,8,90

Step 1:

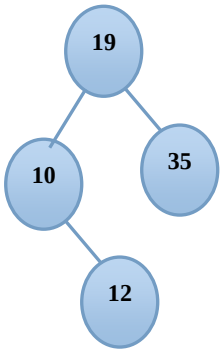
Step 2:



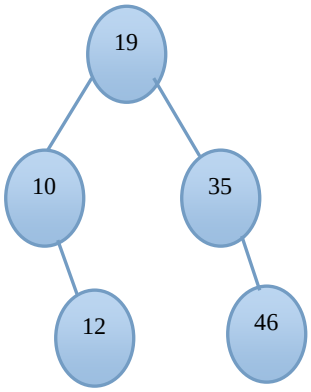
Step 3:



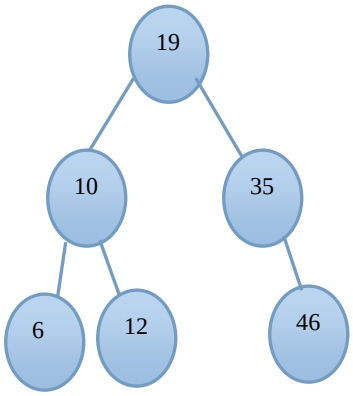
Step 4:



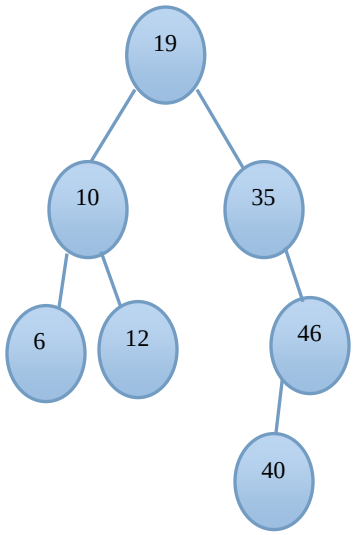
Step 5:



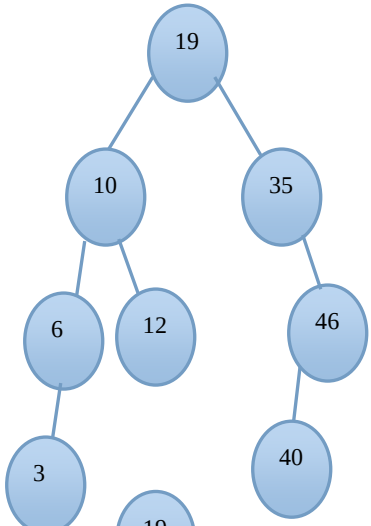
Step 6:



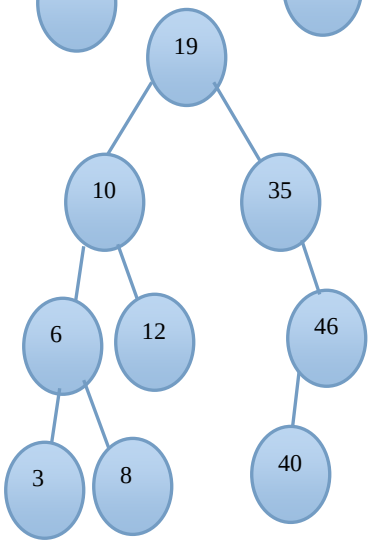
Step 7:



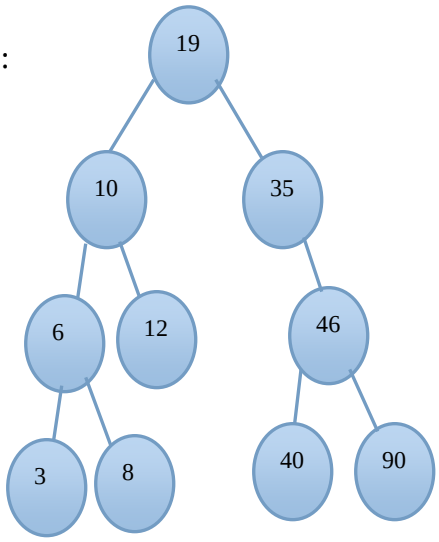
Step 8:



Step 9:



Step 10:



Advantages:

1. Stable
2. It requires extra space for its gaps.

2.9. Address calculation sort:

In this method, sorting by hashing technique is used. Here we will use some non decreasing function. Based on this function we will get the key of particular element. If x<y then this non decreasing function will give value f(x) < f(y). Now we will keep the element in sorted linked list corresponding to that particular key. So this sorting is basically based on the address calculated by this function.

Ex;

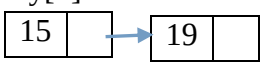
Consider the following list of elements:

19,24,49,15,45,33,89,66

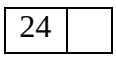
Now we have to create the list. Lets create the list based on the function which gives the key as first digit of element. So there will be 10 lists. Here lists are sorted linked lists.

Array[0] : null

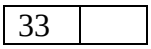
Array[1] :



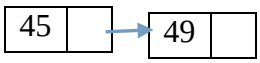
Array[2]:



Array[3]:

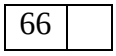


Array[4]:



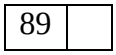
Array[5]→null

Array[6]:



Array[7]→null

Array[8]:



Array[9]→null

Advantages:

1. Stable sort.
2. Variation sort.

2.10. Radix sort:

The radix sort is based on the values of the actual digits in the positional representation of the numbers being sorted.

Ex;

Consider the following list of numbers:
19,24,49,15,45,33,89,66

First distribution: (separate based on last digit)	Merge
0)	33
1)	24
2)	15
3) 33	45
4) 24	66
5) 15,45	19
6) 66	49
7)	89
8)	
9) 19,49,89	

Second distribution: (separate based on first digit)	Merge
0)	15
1) 15,19	19
2) 24	24
3) 33	33
4) 45,49	45
5)	49
6) 66	66
7)	89
8) 89	
9)	

Advantages:

1. Stable, fast.
2. Used in special cases when the key can be used to calculate the address of buckets.

3. Behavior of sorting techniques:

Sorting techniques	Average case	Worst case
Bubble	$O(n^2)$	$O(n^2)$
Insertion	$O(n^2)$	$O(n^2)$
Selection	$O(n^2)$	$O(n^2)$
Shell	O(n logn)	O(n log2n)
Merge	O(n logn)	O(n logn)
Quick	O(n logn)	$O(n^2)$
Heap	O(n logn)	O(n logn)
Binary tree	O(n logn)	$O(n^2)$
Address calculation	O(n logn)	$O(n^2)$
Radix	O(n logn)	$O(n^2)$

4. Analysis and discussion:

Basically, the number of comparisons is more important. It directly determine the efficiency of a sorting algorithm. The number of comparisons plays a more crucial role in sorting speed. Quicksort is a preferred sorting methods in applications that require a sorting algorithm that is usually very fast, but on occasion can have a longer running time. Research efforts have been made to enhance this algorithm for the worst case scenarios by improving the way the algorithm chooses its pivot element for partitioning. But the hidden constant associated with this improved version of Quicksort is so large that it results in an algorithm worse than Heapsort in every case [Brassard and Bratley, 1996].

5. Conclusion:

Every sorting algorithm has some advantages and disadvantages. The "Performance analysis of sorting algorithms" deals and analyze the most commonly used internal sorting algorithms and evaluate their performance. To sort a list of elements, First of all we analyzed the given problem i.e. the given problem is of which type (small numbers, large values). The time complexity may vary depending upon the sorting algorithm used. Each sorting algorithm follows a unique method to sort an array of numbers either by ascending or descending order. The ultimate goal of this study is to compare the various sorting algorithms and finding out the asymptotic complexity of each sorting algorithm. This study proposed a methodology for the users to select an efficient sorting algorithm. Finally, the reader with a particular problem in mind can choose the best sorting algorithm.

REFERENCES

[1] Knuth, D. The Art of Computer Programming Sorting and Searching, 2nd edition, 3. Addison-Wesley, (1998).
[2] Wirth, N., 1976. Algorithms + Data Structures = Programs: Prentice-Hall, Inc. Englewood Cliffs, N.J.K.Mehlhorn. Sorting and Searching. Springer Verlag, Berlin, (1984).
[3] LaMarca and R. Ladner. The Influence of Caches on the Performance of Sorting. In Proceeding of the ACM/SIAM Symposium on Discrete Algorithms, (January 1997), 370–379.
[4] Hoare, C.A.R. Algorithm 64: Quicksort. Comm. ACM 4, 7 (July 1961), 321. 242-250, 242-250.
[5] .lores, I. Analysis of Internal Computer Sorting. J.ACM 7, 4 (Oct. 1960), 389-409.
[6] .lores, I. Analysis of Internal Computer Sorting. J.ACM 8, (Jan. 1961), 41-80.
[7] Andersson, A. and Nilsson, S. 1994. A New Efficient Radix Sort. In Proceeding of the 35th Annual IEEE Symposium on foundation of Computer Science (1994), 714-721.
[8] V. Estivill-Castro and D.Wood. A Survey of Adaptive Sorting Algorithms. Computing Surveys, 24, (1992),441-476.
[9]T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms. MIT Press,Cambridge, MA, 2nd edition, 2001.
[10] Deependra Kr. Dwivedi *et. al* / VSRD International Journal of CS & IT Vol. 1 (4), 2011, 264-267
[11] RobertLafore"Object oriented programming in turbo C ++ "
[12] Ellis Horowitz, Sartaj Sahni, Dinesh Mehta "Fundamentals of Data structures in C+"
[13] Baase, S., Computer Algorithms, second edition, Addison-Wesley Publishing, 1996
[14] Brassard, G., and P. Bratley, Fundamentals of algorithmics, Prentice-Hall, Englewood Cliffs, NJ, 1996.
[15] Data Structures by Seymour Lipschutz and G A Vijayalakshmi Pai (Tata McGraw Hill companies), Indian adapted edition-2006,7 west patel nagar,New Delhi-110063
[16] Introduction to Algorithms by Thomas H. Cormen

