COMPARATIVE ANALYSIS OF SORTING AND SEARCHING ALGORITHMS

Muqarrab Qureshi
Dept. of ECS, Shah and Anchor Kutchhi Engg College, Mumbai, India

Danish Surve
Dept. of ECS, Shah and Anchor Kutchhi Engg College, Mumbai, India

Piyush Waghela
Dept. of ECS, Shah and Anchor Kutchhi Engg College, Mumbai, India

Nidhi Sakpal
Dept. of ECS, Shah and Anchor Kutchhi Engg College, Mumbai, India

ABSTRACT

Various algorithms on sorting and searching algorithms are presented. The analysis shows the advantages and disadvantages of various sorting and searching algorithms along with examples. Various sorting techniques are analyzed based on time complexity and space complexity. On analysis, it is found that quick sort is productive for large data items and insertion sort is efficient for small lists. Also, it is found that binary search is suitable for mid-sized data items and is applicable in arrays and in linked lists, whereas hash search is best for large data items, and exponential search can be used for an infinite set of elements. The research evaluates the merits and limitations of several algorithms based on time complexity, space complexity, stability, adaptability, and scalability. This study sheds light on algorithmic efficiency and optimization strategies through empirical tests and statistical analysis, enabling algorithm selection for practical applications in data processing, analysis, and retrieval. We measure and compare several key metrics, including execution time, memory usage, algorithmic complexity, and stability. By analyzing these metrics, we gain insights into the efficiency and suitability of each algorithm for different dataset sizes and characteristics. We also discuss the implications of the findings in practical applications. Our results reveal important trade-offs among the sorting algorithms. While some algorithms excel in certain scenarios, others demonstrate better scalability or memory efficiency. We identify the best-performing algorithms for specific dataset characteristics and highlight their strengths and limitations. This research can assist developers and practitioners in selecting appropriate sorting algorithms based on their specific requirements and dataset characteristics. In conclusion, this comparative analysis provides a valuable contribution to the understanding of sorting algorithm performance.

Keyword: Algorithm, Searching, Sorting, Space Complexity, Time Complexity

1. INTRODUCTION

A process or set of rules or well defined list of steps to be followed in calculations or other problem-solving operations, especially by a computer is called as an algorithm. The analysis of an algorithm provides information that gives us a general idea of how long an algorithm will take to solve a problem.

Searching for an element and arranging the elements in a particular order are everyday jobs. Also in computer programming searching and sorting are common. As these tasks are common, different algorithms are developed. If the data is kept in some sorted order then searching becomes very easy. After looking at each of the algorithms meticulously and experiencing the illustrations, we can decide on the execution of every algorithm, considering how rapidly it completes the assigned task.

According to a linear relationship between the data members sorting is defined as arranging the data in ascending or descending order. In this data is organized in an order that allows information to be found easier

Adaptive and Non Adaptive Sorting

An algorithm that tries not to reorder the already sorted elements is a adaptive sorting algorithm. In contrast, non adaptive sorting algorithm is one which changes the sorted data forcefully.

In-Place and Not In Place Sorting Algorithms

The algorithm which does not requires temporary memory and additional space for sorting are called in-place sorting algorithms. And not in-place sorting algorithm is one which requires additional temporary memory and space.

Searching is the process of locating an element (record, file or a table) from a given set of elements. A search algorithm is one that accepts a key and tries to locate it in the given set. A search can be unsuccessful if the key does not exist in the given set of elements.

Time Complexity

Time complexity deals with finding out how the computational time of an algorithm changes with the change in size of the input. The number of basic operations performed by an algorithm is used to calculate time complexity, where each operation takes a pre-set amount of time. Hence, the number of basic operations and the time required by an algorithm differ by a constant factor.

Since an algorithm's performance time may vary with different inputs of the same size, one commonly uses the worst-case time complexity of an algorithm, denoted as T(n), which is defined as the maximum amount of time taken on any input of size n. Less common, and usually specified explicitly, is the measure of average-case complexity. Time complexities are classified by the nature of the function T (n). For instance, an algorithm with $T(n) = O(n)$ is called a linear time algorithm, and an algorithm with $\log T(n)=\Omega(n^a)$ for some constant $\alpha \geq 1$ is said to be an exponential time algorithm.

Space Complexity

Total amount of computer memory required by an algorithm to complete its execution is called as space complexity of that algorithm. Space complexity deals with finding out how much (extra) space would be required by the algorithm with change in the input size. To calculate the space complexity, we must know the memory required to store different data type values. If any algorithm requires a fixed amount of space for all input values, then that space complexity is said to be Constant Space Complexity.

Big O notation is used in Computer Science to describe the performance or complexity of an algorithm. Big O specifically describes the worst-case scenario and can be used to describe the execution time required or the space used (e.g. in memory or on disk) by an algorithm.

- O(1) describes an algorithm that will always execute in the same time (or space) regardless of the size of the input data set.
- O(N) describes an algorithm whose performance will grow linearly and in direct proportion to the size of the input data set.
- $O(N^2)$ represents an algorithm whose performance is directly proportional to the square of the size of the input data set. This is common with algorithms that involve nested iterations over the data set. Deeper nested iterations will result in $O(N^3)$, $O(N^4)$ etc.
- $O(2^N)$ denotes an algorithm whose growth doubles with each addition to the input data set. The growth curve of an $O(2^N)$ function is exponential - starting off very shallow, then rising meteorically

This method efficiently searches and retrieves keys from a given list. This article discusses a search algorithm that considers time and space complexity. Sorting and searching procedures, as well as their implementation, vary depending on the context. Sorting algorithms typically rely on two parameters: execution time and execution speed/space. Here are some other sorting techniques:

- Quick sort.
- Selection
- Bubble
- Insertion
- Merge
- Heap sorting methods are available.

To evaluate the performance of these algorithms, we utilize datasets with varying sizes and characteristics. Our dataset selection includes well-known datasets from the machine learning domain, such as the Iris dataset and Wine dataset, as well as synthetic datasets generated with different distributions, including Uniform, Normal, Exponential, and Bimodal distributions. By incorporating a wide range of dataset characteristics, we can assess how the sorting algorithms handle different data distributions and dataset sizes.

2. RELEVANT WORK

Ajay Kumar, Bharat Kumar, Chirag Dawar and Dinesh Bajaj explained about different sorting techniques their behavior for different inputs. The research reveals that the Insertion sort is best for small data items and, Merge sort and quick sort is used for large data sets [1]. Rekhadwivedi and Dr. Dinesh C. Jain discussed about sorting, sorting algorithm, types of sorting algorithm and comparison on basis of time complexity. The research observed that quick sort is the best based on execution time [2]. Ramesh Chand Pandey explains about the various sorting algorithms, their advantages and disadvantages. It compared sorting algorithms based on time complexity, memory required and stability. It proposed a new sorting algorithm which works on priority basis. The new algorithm gives the specific data first then general data if required [3].

V.P.Kulalvaimozhi, M.Muthulakshmi, R.Mariselvi, G.Santhana Devi, C.Rajalakshmi and C. Durai's research dealt with most commonly used internal sorting algorithms and their performance. It compared various sorting algorithms and found out asymptotic complexity of each sorting algorithm. The study proposed a methodology for the users to select an efficient sorting algorithm [4]. Pankaj Sareen's review paper used one program and compared all the other algorithms with it. It calculated the average running time for each algorithm and displayed the result with the help of chart and concluded that Quick sort is the most efficient

algorithm [5]. Kamlesh Kumar Pandey and Narendra Pradhan discussed searching algorithms, its advantages and disadvantages. They analyzed the different searching algorithms and found that Hash search is faster for large lists when compared to other searching algorithms and binary search is efficient for mid-sized lists [6].

Thomas Niemann in his research work on sorting and searching algorithms stated about internal and external sort, compared different sorting and searching algorithms on basics of time, space and simplicity [7]. Brad Miller and David Ranumanalysed different sorting and searching techniques. He also discussed about Hash functions [8]. Debadrita Roy and Arnab Kundu in their review summarized three searching algorithmsand also included searching concept, algorithm, coding & time complexity [9]. Amy CsizmarDalal gave a brief description on sorting and searching techniques. He explained about different searching and sorting techniques, algorithms and implementation of the techniques with examples [10].

Khalid Suleiman Al-Kharabsheh, Ibrahim Mahmoud AlTurani, Abdallah Mahmoud Ibrahim AlTurani&NabeelImhammedZanoon described briefly on sorting techniques. They compared the different sorts and gave the run time for different number of elements [11].Vimal P.Parmar explained about bubble and selection sort with arrays and linked list. He compared

array and linked list. He explained the possibilities of implementation of bubble and selection sort with arrays and linked list. On analysis, he found that binary search is efficient when compared to linear search as it can be implemented in both arrays and linked list by making necessary modifications [12].Yuvraj Singh Chandrawat, Abhijeet Vajpayee and AayushPathakintroduced the types of searching techniques (internal and external search) and the working procedure, time complexity, advantages and disadvantages of linear and binary search. Finally, they concluded that binary search is the efficient one [13].

JehadHammad introduced different strategies for algorithms. He explains about bubble sort, gnome sort and selection sort, compared their performances, and produced graphs as results. He says that gnome sort algorithm is efficient for sorted data whereas selection sort is quicker than gnome and bubble in unsorted data [14].FahriyeGemciFurat explained and compared insertion and selection sort. Implementation and running time of them is also explained in a detailed manner. He concluded that based on running time selection sort is efficient [15].Clifford A. Shaffer in his book explained about the different sorting mechanisms (internal and external sort). He divided the sorts on basics different methods to approach\solve an algorithm. The book majorly explains about three sorts (internal sort, bubble sort, selection sort) and other internal and external sorts. It demonstrates some of the searching techniques [16].

The website provides the basic information about algorithms. It gives the detailed information on different sorting and searching techniques. It provides the working procedure, time complexity and pseudocode of searching and sorting algorithms. Hash table and operations on it are explained briefly [17]. SmitaPaira, Sourabh Chandra, SkSafikulAlam and Subhendu Sekhar Patra introduced a searching technique called Bi Linear search. The paper explained about the searching technique using pseudocode and program in C. The paper has proved Bi- linear Search to be highly efficient for unsorted list of data. It takes comparatively less time tosearch an item from a large collection of records than a normal Linear Search. Even in a sorted list, it sometimes beats the Binary Search, especially when the element is present in the last position of the array. The Bi-linear Search has a worst case time complexity of O(1) [18].

The website describes about Selection sort and binary search. It provides complexity analysis of binary search and selection sort. It gives the detailed working procedure of binary search and selection sort algorithms [19]. Muhammad Usman, Zaman Bajwa and MudassarAfza discussed and analyzed the performance of different searching algorithms. The analysis is based on time. In all experiments, the results are different on the same size file. They analyzed the execution time for searching a pattern in a sample text file using linear, binary and brute force [20].Nitin Arora, GarimaBhasin and Neha Sharma introduced an algorithm called two-way Linear Search Algorithm. They compared two-way linear search algorithm with linear search and provided the results with the help of algorithm and comparison table. They concluded that in Linear Search it is not compulsory to arrange an array in any order (Ascending or Descending) as in the case of binary search orderly [21].Manpreet Singh Bajwa, Arun Prakash Agarwal and SumatiManchanda presented new algorithm for searching that is Ternary search while comparing it with Binary search algorithm. The time complexity of ternary search is reduced when compared to binary search. Hence, they concluded that it is efficient [22].

3. VARIOUS SORTING ALGORITHMS

The sorting techniques can be divided into two categories:

Internal Sort: In internal sorting, all the data to sort is stored in memory at all times while sorting is in progress.

External Sort: In external sorting data is stored outside memory (like on disk) and only loaded into memory in small chunks. External sorting is usually applied in cases when data can't fit into memory entirely.

BUBBLE SORT

Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted, shown in Fig. 1.

Algorithm

1. Compare each pair of adjacent elements from the beginning of an array and, if they are in reversed order, swap them.
2. If at least one swap has been done, repeat step 1.

Illustration

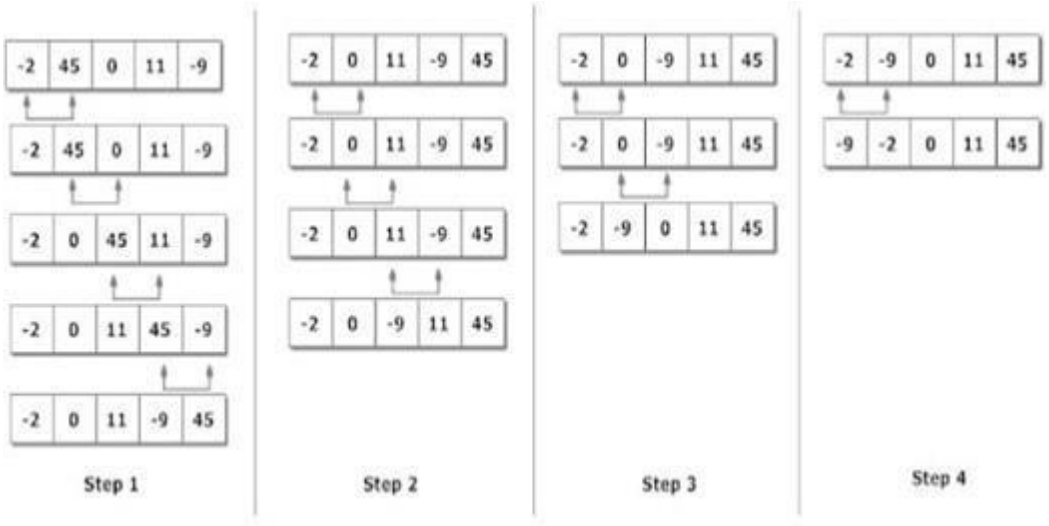

Figure 1 Illustration of Bubble Sort

Advantages

- It is efficient for small data set
- Easy to implement
- Last element is known after first iteration

Disadvantages

- It is slow
- The loop continues to run even if the array is sorted, if the code is not optimised

COCKTAIL SORT

Cocktail Sort is a variation of Bubble sort. The Bubble sort algorithm always traverses elements from left and moves the largest element to its correct position in first iteration and second largest in second iteration and so on. Cocktail Sort traverses through a given array in both directions alternatively, shown in Fig. 2.

Algorithm

Each iteration of the algorithm is broken up into 2 stages:

1. The first stage loops through the array from left to right, just like the Bubble Sort. During the loop, adjacent items are compared and if value on the left is greater than the value on the right, then values are swapped. At the end of first iteration, largest number will reside at the end of the array.
2. The second stage loops through the array in opposite direction- starting from the item just before the most recently sorted item, and moving back to the start of the array. Here also, adjacent items are compared and are swapped if required.

Illustration

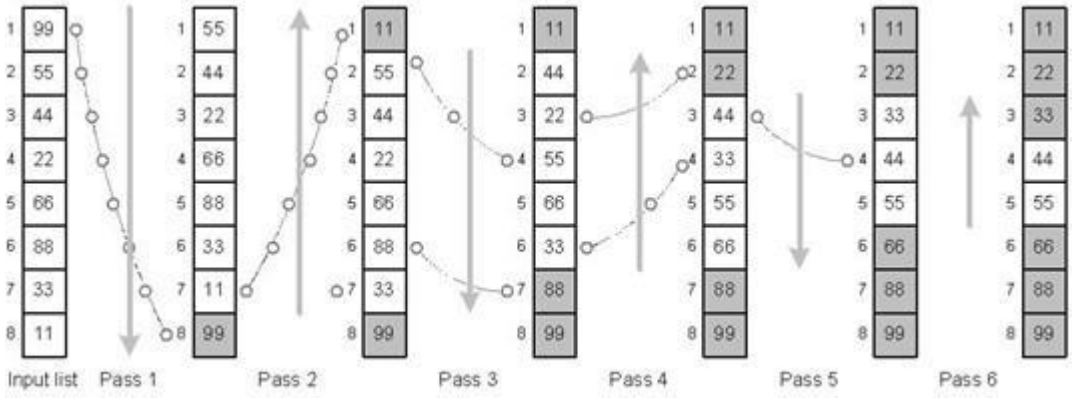

Figure 2 Illustration of Cocktail Sort

Advantages

- It is a stable sort.
- Easy to implement
- Last element is known after first iteration

Disadvantages

- It is slow
- The loop continues to run even if the array is sorted, if the code is not optimised

COMB SORT

Comb sort is a relatively simple sorting algorithm originally designed by WłodzimierzDobosiewicz in 1980. Later it was rediscovered by Stephen Lacey and Richard Box in 1991. Comb sort improves on bubble sort, shown in Fig 3.

Algorithm

1. Create variables gap and swapped and constant SHRINK_FACTOR and initialize as below:
gap=size of the arra
swapped=false

SHRINK_FACTOR=1.3 'swapped' is used to check whether any 2 elements have been swapped at the end of an iteration, like it is used in Bubble Sort algorithm for optimization.
2. Set swapped = false
3. Set gap = (int) (gap/SHRINK_FACTOR).
4. Iterate over the array from i = 0 to i< n - gap:
a. If array[i] > array[i + gap]
i. swap the elements array
[i] and array[i + gap], to arrange in sorted order
ii. set swapped = true
5. Repeat steps 2-4 while gap != 1 and swapped = true

Illustration

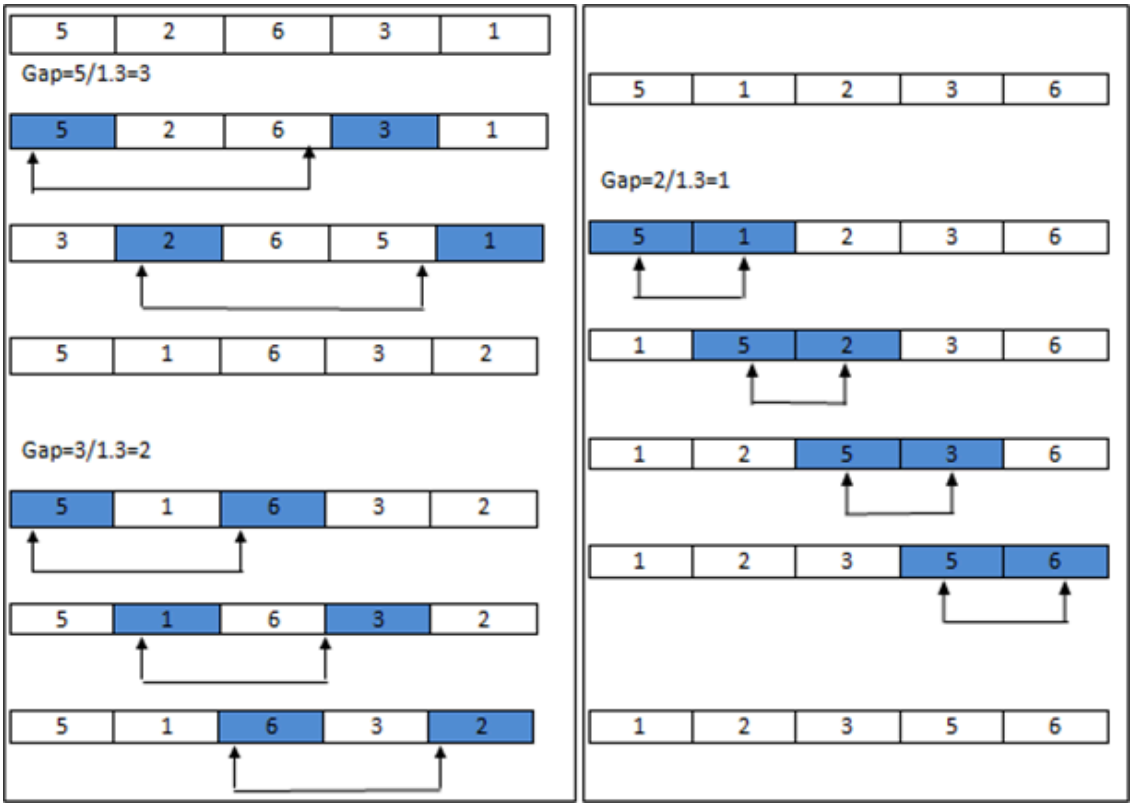

Figure 3 Illustration of Comb Sort

Advantages

- Is proper for data sets composed of either numbers or strings;
- Time complexity very good, could be compared to quick sort.
- No recursive function-calls
- In-place-sorting, no extra memory needed
- No worst-case-situation like in Quick sort.

Disadvantages

- must resize the gap with a division by 1.3, which is a fraction;

GNOME SORT

Gnome sort algorithm is almost related to insertion sort. In this sort an element is moved to an appropriate place by a series of swaps which is similar to bubble sort. As it does not require any nested loops, it is simple. The time complexity is $O(n^2)$, if some elements of the list are sorted initially the time complexity tends to O(n), shown in Fig 4.

Algorithm

1. If you are at the start of the array then go to the right element (from arr[0] to arr[1]).
2. If the current array element is larger or equal to the previous array element then go one step right

```
if (arr[i] >= arr[i-1])
i++;
```

3. If the current array element is smaller than the previous array element then swap these two elements and go one step backwards

```
if (arr[i] <arr[i-1])
{
swap(arr[i],
arr[i-1]); i--;
}
```

4. Repeat steps 2) and 3) till 'i' reaches the end of the array (i.e- 'n-1')
5. If end of the array is reached then stop and the array is sorted.

Illustration

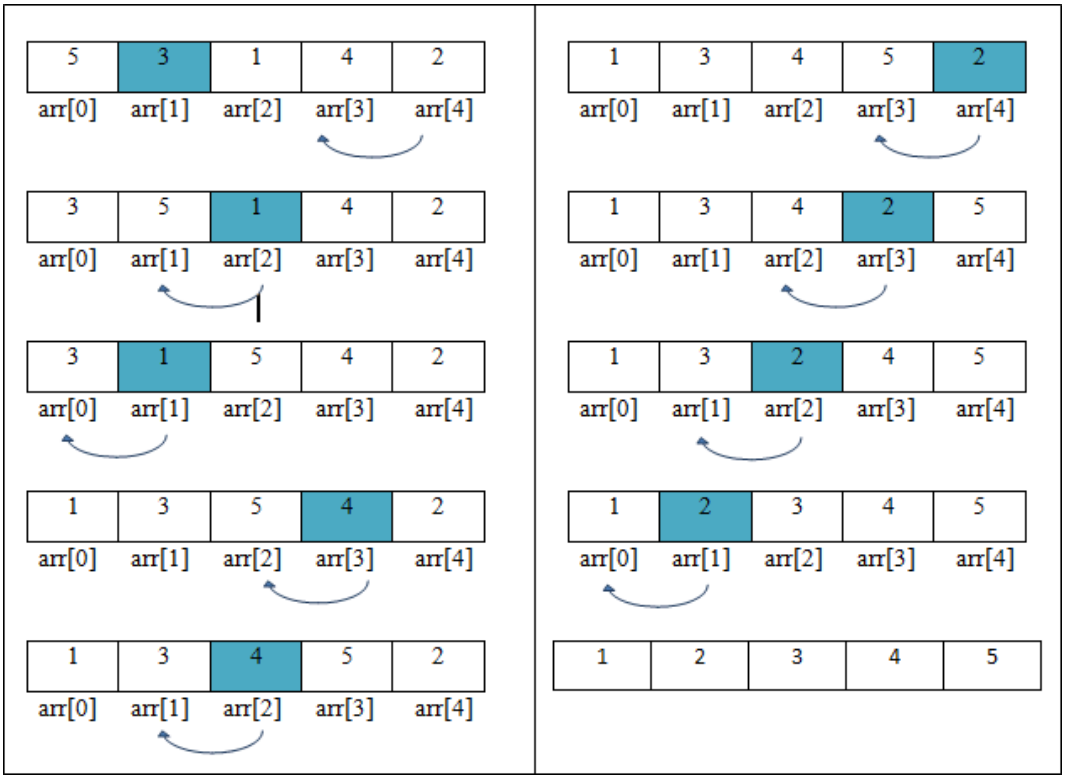

Fig 4: Illustration of Gnome Sort

Advantages

- Is a very simple algorithm to learn;

Disadvantages

- The complexity is O(n^2);
- Not good on large data sets;

SELECTION SORT

This is a sorting algorithm, specifically an in-place comparison sort. It has O(n2) time complexity, it is inefficient on large data item, and performs worse than insertion sort. It is simple, and has high performance over complicated algorithms in some situations specially when secondary memory is limited, shown in Fig. 5.

Algorithm

1. Set MIN to location 0
2. Search for the least element in the given list
3. Swap with the value at the location MIN
4. Increment MIN so that it points to the next element
5. Repeat until list is sorted

Illustration

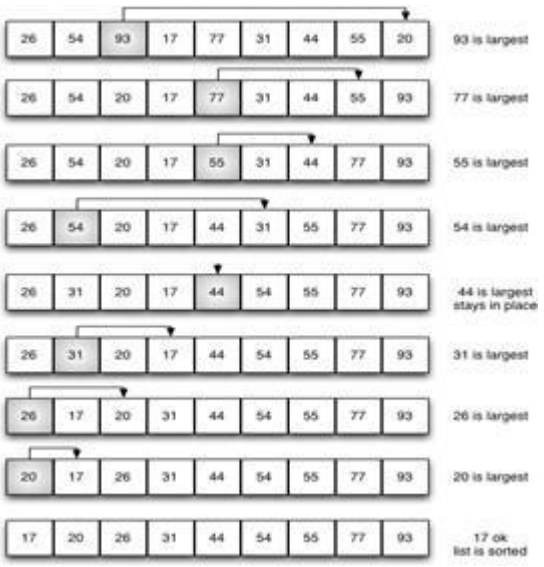

Figure 5 Illustration of Selection Sort

Advantages

- The major advantage of the selection sort is that it performs efficiently on a small list.
- As it is in-place sorting algorithm, no additional impermanent storage is required apart from what is required to hold the given list.
- It's performance is easily influenced by the initial ordering of the data before sorting.

Disadvantages

- The major disadvantage of this sort is its less efficiency when dealing with a large list of data items.
- Quick Sort is more efficient when compared to selection sort

INSERTION SORT

Insertion sort is an in-place sorting algorithm which is based on comparison. Here, a small part of the list is maintained which is already sorted. For instance, the lower part of an array is taken which is sorted. A data item that is to be inserted in this sorted sub-list must find its appropriate place and then it is to be inserted. Hence it is known as insertion sort. The given array is searched linearly and unsorted data items are moved and inserted into the sorted sub-list in the array. This algorithm is not efficient for large data sets as its average and worst case complexity are of O (n^2), where n is number of items, shown in Fig .6.

Algorithm

1. If the first element is already sorted, Return 1;
2. Take the next element
3. Compare the element with all other elements in sorted sub-list
4. Move all the elements in the sorted sub-list which is bigger than the Value to be sorted
5. Insert the value
6. Repeat until list is sorted

Illustration

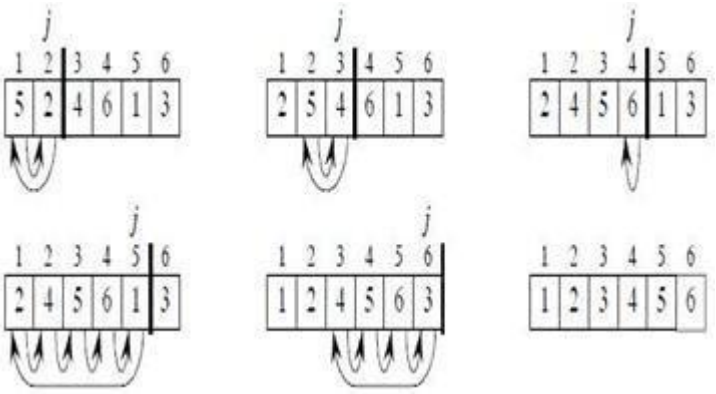

Figure 6 Illustration of Insertion Sort

Advantages

- Simple implementation
- Efficient for small data sets
- It is stable

Disadvantages

- Less efficient for the list containing more elements
- It needs large number of element shifts

SHELL SORT

Shell sort is an efficient step by step procedure and depends on insertion sort algorithm. This algorithm avoids large shifts as in case of insertion sort, if the least value is to the right end and has to be moved to the left end. This algorithm uses insertion sort on broadly spread elements, first to sort them and then sorts the less widely spaced elements. This spacing is termed as interval, shown in Fig. 7.

Algorithm

- Initialize the value of h
- Divide the given list into smaller sub-list of equal intervals of h
- Sort the sub-lists using the insertion sort
- Repeat until complete list is sorted

Illustration

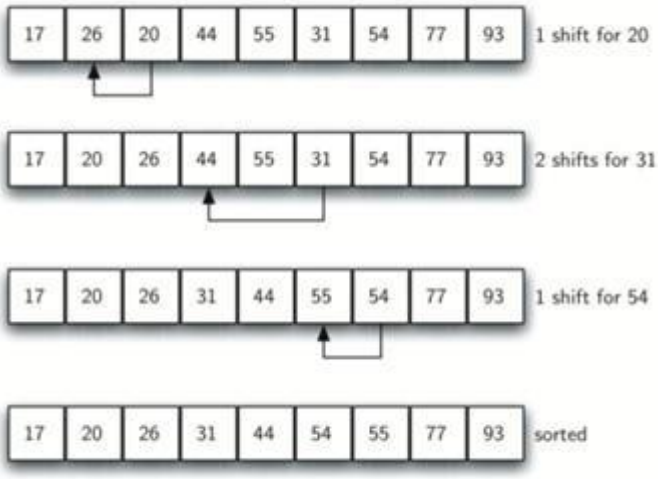

Figure 7 Illustration of Shell Sort

Advantages

- Faster than the ordinary insertion sort
- More efficient exchange of elements that are far from their proper position
- The fastest of the non-recursive sorts shown here
- Easy to get working

Disadvantages

- Running time is slightly sensitive to the data (but far less so than binary tree sort or Quicksort)
- Not quite as fast as ultra-fast sorts like Quicksort
- If you mess up the sequence, you get a sort that will still work but is slow.

BINARY TREE SORT

A tree sort is a sort algorithm that builds a binary search tree from the elements to be sorted, and then traverses the tree (in-order) so that the elements come out in sorted order. Its typical use is sorting elements adaptively: after each insertion, the set of elements seen so far is available in sorted order, shown in Fig 8.

Algorithm

1. Take the elements input in an array.
2. Create a Binary search tree by inserting data items from the array into the Binary search tree.
3. Perform in-order traversal on the tree to get the elements in sorted order.

Advantages

- If you've already implemented binary search trees, then this sort is quite easy.

Disadvantages

- Hogs memory for the pointers
- Not very pleasant to implement or debug
- Not very fast

Illustration

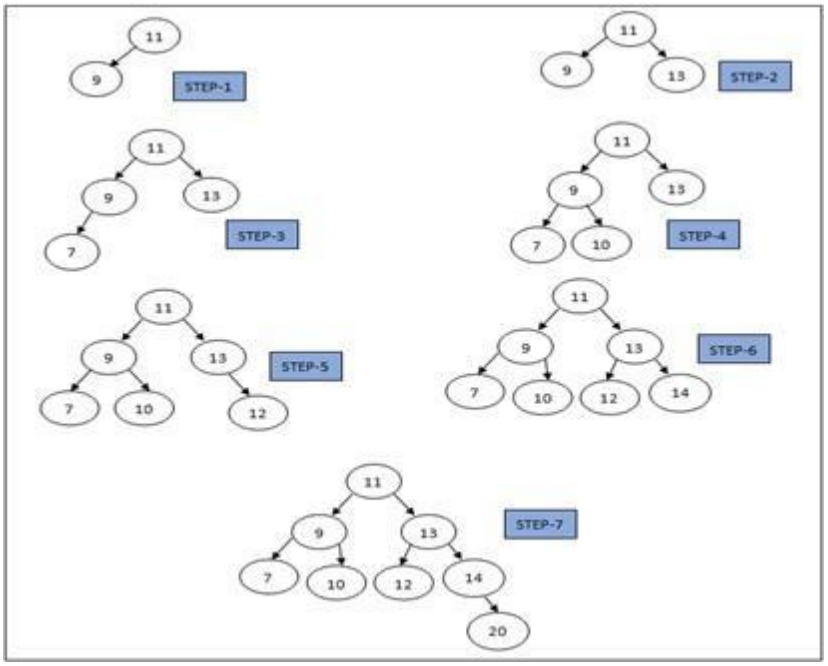

Figure 8 Illustration of Binary Tree Sort

QUICK SORT

Quick sort (sometimes called partition-exchange sort) is an efficient sorting algorithm, serving as a systematic method for placing the elements of an array in order. Developed by Tony Hoare in 1959 and published in 1961, it is still a commonly used algorithm for sorting. When implemented well, it can be about two or three times faster than its main competitors, merge sort and heap sort, shown in Fig. 9.

Algorithm

1. Choose the highest index value has pivot
2. Take two variables to point left and right of the list excluding pivot
3. left points to the low index
4. right points to the high
5. while value at left is less than pivot move right

6. while value at right is greater than pivot move left

1. if both step 5 and step 6 does not match swap left and right
2. if left ≥ right, the point where they met is new pivot

Illustration

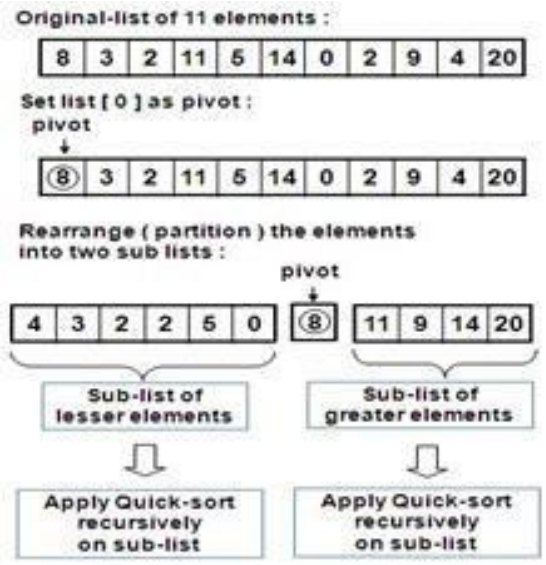

Figure 9 Illustration of Quick Sort

Advantages

- Advantages include the efficient average case compared to any for mentioned sort algorithm, as well as the elegant recursive definition, and the popularity due to its high efficiency

Disadvantages

- Disadvantages include the difficulty of implementing the partitioning algorithm and the average efficiency for the worst-case scenario, which is not offset by the difficult implementation

MERGE SORT

Merge sort is a sorting technique which is based on the divide and conquers technique. Though the worst-case time complexity being O(n log n), it is one of the most respected algorithms. It first divides the given array into two equal parts and then joins them in a sorted manner, shown in Fig. 10.

Algorithm

1. If the first element is already sorted, return 1.
2. The list is divided recursively into two equal parts until it cannot be divided.
3. Combine the smaller list of data items into the new list which has sorted elements.

Illustration

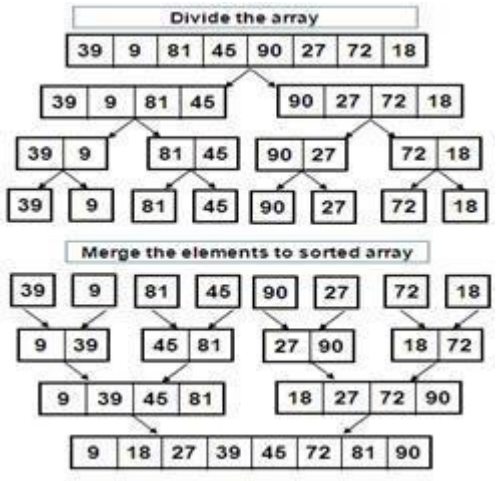

Figure 10 Illustration of Merge Sort

Advantages

- Merge sort can be useful to files of any size.
- It is fast and stable

Disadvantages

- Merge Sort takes more space when compared to other sorts.
- Merge sort is less efficient than other sort

HEAP SORT

It is the best in-place sorting algorithm with no quadratic worst-case situations. This algorithm is divided into two basic parts: Creating a Heap of the unsorted list and Then a sorted array is generated by continuously removing the greatest\smallest data item from the heap, and placing the data item into the array. The heap is rebuilt after each removal, shown in Fig. 11.

Algorithm

1. Build a max heap from the input data.
2. At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.
3. Repeat above steps while size of heap is greater than 1.

Advantages

- Because of its efficiency, this algorithm is widely used.
- As it is an in-place sorting algorithm its memory usage is nominal.

Disadvantages

- More space is required for sorting
- Quick sort is more efficient when compared to Heap in most of cases

A tree of sorting elements is made in Heap sort.

Illustration

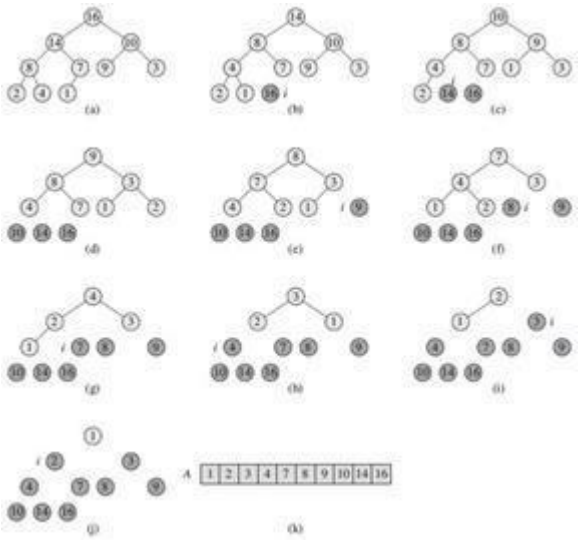

Figure 11 Illustration of Heap Sort

4. ANALYSIS OF SORTING ALGORITHMS

The comparison of various sorting algorithms is depicted in Table 1:

TABLE 1 Comparison of sorting algorithms

Name	Average	Worst	Memory	Stable	Method	Other notes
Bubble Sort	-	$O(n^2)$	O(1)	Yes	Exchanging	Oldest sort
Cocktail Sort	-	$O(n^2)$	O(1)	Yes	Exchanging	Variation of bubble sort
Comb Sort	-	-	O(1)	No	Exchanging	Small code size
Gnome Sort	-	$O(n^2)$	O(1)	Yes	Exchanging	Tiny code size
Selection Sort	$O(n^2)$	$O(n^2)$	O(1)	No	Selection sort	Can be implemented as a stable sort
Insertion Sort	$O(n^2)$	$O(n^2)$	O(1)	Yes	Selection sort	Average case in O(n+d), where d is the number of inversion
Shell Sort	-	$O(n \log^2 n)$	O(1)	No	Insertion	No extra memory required
Binary Tree Sort	O(n log n)	O(n log n)	O(n)	No	Insertion	When using a self-balancing binary search tree
Merge Sort	O(n log n)	O(n log n)	O(n)	Yes	Merging	Recursive nature, extra memory required
Heap Sort	O(n log n)	O(n log n)	O(1)	No	Selection	Recursive, extra memory required
Quick Sort	O(n log n)	$O(n^2)$	O(log n)	No	Partitioning	Recursive, based on divide conquer technique

5. SEARCHING ALGORITHMS

Searching technique is used to find the location of any data item in given list of data items. The searching techniques can be categorized into two parts:

Internal Searches: if file or table is kept in primary memory this type of searching is called internal search.

External Searches: if file or table is stored in secondary memory (hard disk, floppy, tape etc.) this type of searching is called External Search.

LINEAR SEARCH

It is a simple search algorithm. In this a sequential search is made over all data items one after other. Every data item is scanned and if the match is found then that particular data item is sent back, else the search continues till the list ends, shown in Fig. 12.

- Worst Case: Θ(n)Θ(n); search key not present or last element
- Best Case: Θ(1)Θ(1) ; first element
- No. of comparisons: Θ(n)Θ(n) in worst case & 11 in best case

Algorithm

Linear Search (Array B, Value y)

1. Assign 1 to i
2. if i> n, move to step 7
3. if B[i] = y, move to step 6
4. now assign i + 1 to i
5. move to Step-2
6. print the value of y found at location I and move to step 8
7. Print element not found
8. Exit

Illustration

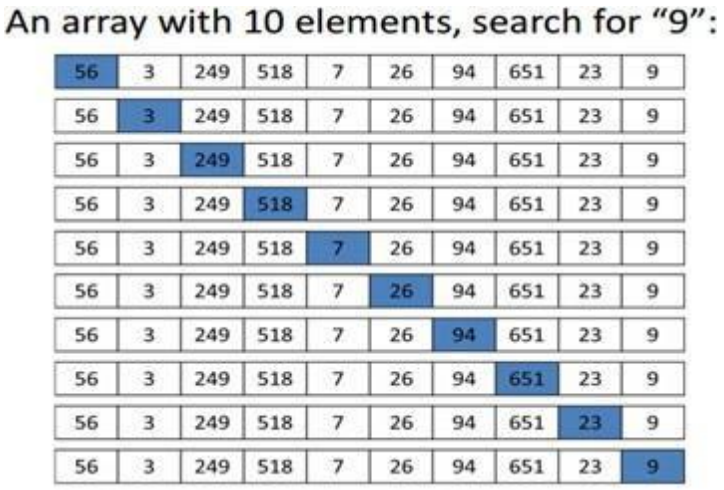

Figure 12 Illustration of Linear search

Advantages

- It is simple and conventional method of searching data. The linear or sequential name implies that the items are stored in a systematic manner.
- The elements in the list can be in any order. i.e. The linear search can be applied on sorted or unsorted linear data structure.

Disadvantage

- This method is insufficient when large number of elements is present in list.
- It consumes more time and reduces the retrieval rate of the system.

BINARY SEARCH

It is a fast search algorithm as the run-time complexity is O(log n). Divide and conquer is the basic principle behind this search algorithm. The data collection must be in the sorted form for better functioning of this algorithm. It searches for a particular data item by equating it to the middle most data item of the collection. If match is found, then the location of data item is returned. If the middle data item is bigger than the search element, then the data item is searched to the left of the middle data item. Else it searched to the right of the middle data item. This process is repeated until the size of the sub array reduces to zero. The objective of this search is to use the information that the array is sorted and reduce the time complexity to O(Logn), shown in Fig. 13.

- Worst case/Average case : Θ(logn)Θ(logn)
- Best Case : Θ(1)Θ(1) ; when key is middle element
- No. of comparisons : Θ(logn)Θ(logn) in worst/average case & 11 in best case

Algorithm

1. Assign n-1 to max and 0 to min.
2. Compute mid as the mean of max and min.
3. If array[mid] is equal to the search element, then stop and return mid.
4. If the mid is less than the search element then assign mid+1 to min.
5. If mid is greater than search element then assign mid-1 to max.
6. Go back to step 2.

Illustration

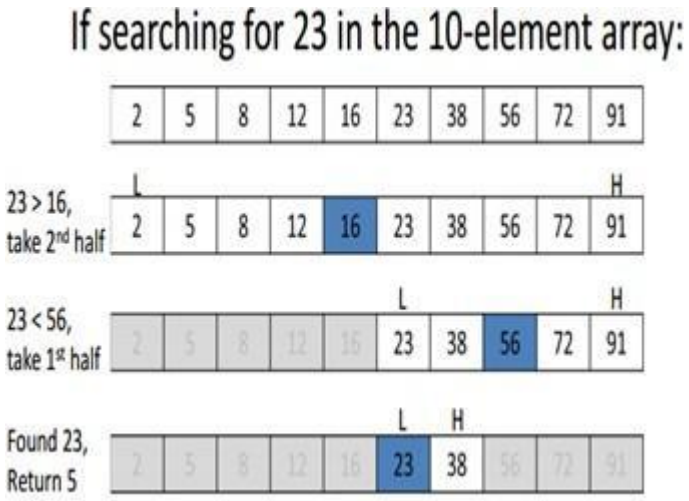

Figure 13 Illustration of Binary search

Advantages

- The binary search is fast as compared to linear search.
- Time complexity is high in both average and worst case.
- Most suitable for sorted arrays.

Disadvantages

- The main disadvantage of binary search is that it works only with an array of sorted order.

HASH SEARCH

Hashing is a technique to convert a range of key values into a range of indexes of an array. We're going to use modulo operator to get a range of key values. Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data. Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from. The time complexity is O(n), shown in Fig. 14.

Algorithm

1. Get the key k to be searched
2. Set counter j = 0
3. Compute hash function h[k] = k % SIZE
4. If the key space at hashtable[h[k]] is occupied
   (4.1) Compare the element at hashtable[h[k]] with the key k.
   (4.2) If they are equal
   (4.2.1) The key is found at the bucket h[k]
   (4.2.2) Stop
   Else
   (4.3) The element might be placed at the next location given by the quadratic function
   (4.4) Increment j
   (4.5) Set h[k] = (k + (j * j)) % SIZE, so that we can probe the bucket at a new slot, h[k].
   (4.6) Repeat Step 4 till j is greater than SIZE of hash table
5. The key was not found in the hash table
6. Stop.

Illustration

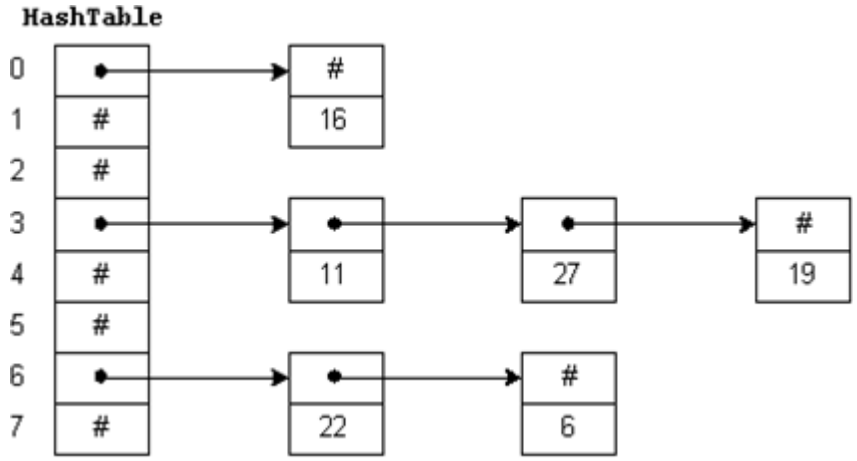

Figure 14 Illustration of Hash search

Advantages

- Hashing provides a more reliable and flexible method of data retrieval than any other data structure.
- Hash tables are efficient when large number of data items can be predicted in advance, so that the bucket array can be allocated once with the optimum size and never resized.

Disadvantages

- Hash tables are not effective when the number of entries is very small
- Listing all n entries in some specific order generally requires a separate sorting step

INTERPOLATION SEARCH

Interpolation search algorithm is improvement over Binary search. The binary search checks the element at middle index. But interpolation search may search at different locations based on value of the search key. The elements must be in sorted order in order to implement interpolation search, shown in Fig 15. The time complexity of interpolation search

- Best case: O(log(log(n)))
- Worst case: O(n)

Algorithm

1. Initialize the values of start to 0 and end to n-1.
2. Calculate the value of $x=start+\frac{(end-start)}{(K[end]-K[start])} * (p - K(start))$, where K is an array and p is the search key.
3. If K[x]==p, then stop and return.
4. If K[x]!=p, then
   If p>K[x] then make start=x+1
   If p<K[x] then make end=x-1.
5. Repeat step 2 till the search element is found.

Advantages

- It requires less time when compared to binary search
- Number of steps required to search an element will be comparatively less than binary search.

Disadvantages

- The calculation of x is complicated and requires more time.
- The elements must be in sorted order.

Illustration

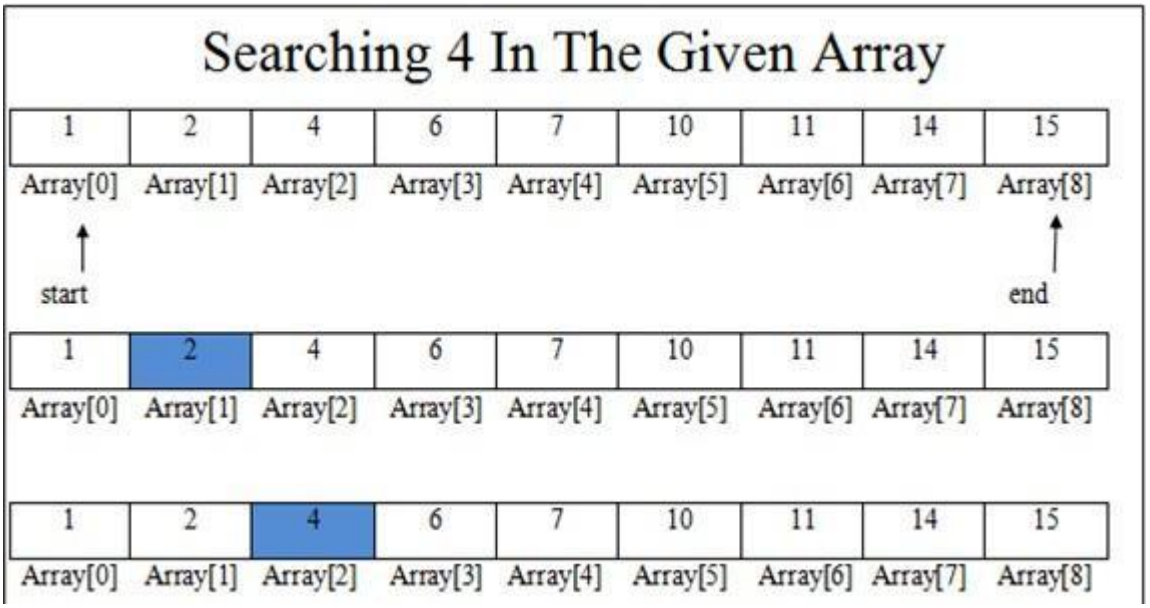

Figure 15 Illustration of Interpolation Search

EXPONENTIAL SEARCH

Exponential search is also called Galloping search, doubling search. Its implementation is similar to implementation of binary search. As the name refers the search starts from first element and continues by doubling the upper limit of the range till the element at that position is larger than the search key. Then it either searches the key using binary search or starts another galloping. The time complexity of exponential search is O(log(i)), where i is the value of index, shown in Fig 15.

Algorithm

1. Initialize the index value to 1.
2. Check the index value with the search key.
3. If the index value is greater than the value of search key.
4. Else, double the value of index till the value of element at the index position is greater than the value of search key.
5. Apply the binary search technique on the array of elements to find the search key.

Illustration

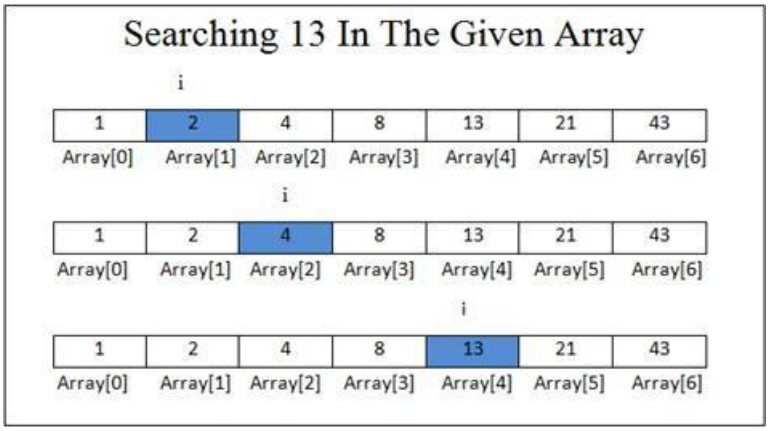

Figure 16 Illustration of Exponential Search

Advantages

- Exponential search can be used for infinite set of elements.
- It can be applied on large set of data items.
- The range\size of the array can be calculated if it is unknown.

TERNARY SEARCH

Ternary search is a searching algorithm where the search key is searched by dividing the given array into three parts. The implementation is similar to binary search. The time complexity is O (log (n)). It works on the principle of divide and conquer technique, shown in Fig 17.

Algorithm

1. Initialize the values of start to 0 and end to n-1, where n is the size of array.
2. Calculate mid1=start+(end-1)/3, mid2=end-(end-1)/3
3. Check if the search key is equal to array[mid1] or array[mid2].
4. If the search key is found to be equal,then stop the process.
5. If key<array[mid1], assign end to mid1-1 or if key>array[mid1], assign start to mid1+1 and repeat step 2.
6. Repeat the step 5 for mid2

Illustration

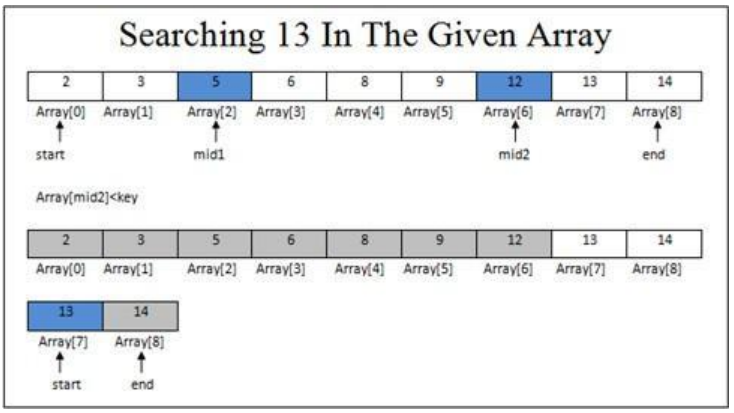

Figure 17 Illustration of Ternary Search

Advantages

- It has more space efficiency.
- Comparatively less comparisons are done in best cases than in binary search.

Disadvantages

More comparisons are done in worst cases when compared to binary search.

6. ANALYSIS OF SEARCHING ALGORITHMS

The comparison of various searching algorithms is depicted in Table 2:

Table 2 Comparison of searching algorithms

Name	Average	Worst	Space	Other Notes
Linear Search	O(N)	O(N)	O(N)	Suitable for small data
Binary Search	O(logN)	O(logN)	O(N)	Suitable formidsized data
Hash Search	O(1)	O(1)	O(N)	Suitable for large data
Interpolation Search	O(log(logN))	O(N)		
Exponential search	O(N)	O(N)	O(N)	Suitable for large data
Ternary Search	O(logN)	O(N)		Suitable for large data sets

7. CONCLUSION

The paper discusses about various sorting and searching techniques. It shows the methodology for various sorting and searching techniques. The analysis shows the advantages and disadvantages of various sorting and searching algorithms along with examples. We analysed the various sorting techniques based on time complexity and space complexity. On analysis, we found that quick sort is productive for large data items and insertion sort is efficient for small list. Also, we found that binary search is suitable for mid-sized data items and is applicable in arrays and in linked list, whereas hash search is best for large data items. Also we found that Exponential search can be used for infinite set of elements.

REFERENCES

[1] Ajay Kumar, Bharat Kumar, Chirag Dawar and Dinesh Bajaj, Comparison Among Different Sorting Techniques, International Journal for Research In Applied Science And Engineering Technology (IJRASET), 2014

[2] Rekhadwivedi and Dr. Dinesh C. Jain, A Comparative Study on Different Types of Sorting Algorithms (On the Basis of C and Java), International Journal of Computer Science & Engineering Technology (IJCSET), 2014.

[3] Ramesh Chand Pandey, Study and Comparison of various Sorting Algorithms, 2008.

[4] V.P.Kulalvaimozhi, M.Muthulakshmi, R.Mariselvi, G.Santhana Devi, C.Rajalakshmi and C. Durai, Performance Analysis of Sorting Algorithm, International Journal of Computer Science and Mobile Computing, 2015.

[5] Pankaj Sareen, Comparison of Sorting Algorithms (On the Basis of Average Case), International Journal of Advanced Research in Computer Science and Software Engineering, 2013.

[6] Kamlesh Kumar Pandey and Narendra Pradhan, A Comparison and Selection on Basic Type of Searching Algorithm in Data Structure, International Journal of Computer Science and Mobile Computing, 2014.

[7] Thomas Niemann, “Sorting and Searching Algorithms.

[8] Brad Miller and David Ranum, Problem Solving with Algorithms and Data Structures, 2013.

[9] Debadrita Roy and Arnab Kundu, A Comparative Analysis of Three Different Types of Searching Algorithms in Data Structure, International Journal of Advanced Research in Computer and Communication Engineering, 2014.

[10] Amy CsizmarDalal, Searching and Sorting Algorithms, 2004.

[11] Khalid Suleiman Al-Kharabsheh, Ibrahim Mahmoud AlTurani, Abdallah Mahmoud Ibrahim AlTurani&NabeelImhammedZanoon, Review on Sorting Algorithms A Comparative Study, International Journal of Computer Science and Security (IJCSS), 2013.

[12] Vimal P.Parmar, Comparing Linear Search and Binary Search Algorithms to Search an Element from a Linear List Implemented through Static Array, Dynamic Array And Linked List, International Journal of Computer Applications, 2015.

[13] Yuvraj Singh Chandrawat, Abhijeet Vajpayee and Aayush Pathak, Analysis and Comparative Study of Searching Techniques, International Journal of Engineering Sciences & Research Technology, 2015.

[14] JehadHammad, A Comparative Study between Various Sorting Algorithms, International Journal of Computer Science and Network Security, 2015.

[15] FahriyeGemciFurat, A Comparative Study of Selection Sort and Insertion Sort Algorithms, International Research Journal of Engineering and Technology, 2016.

[16] Clifford A. Shaffer, A Practical Introduction to Data Structures and Algorithm Analysis, 2009.

[17] http://www.tutorialspoint.com/data_structures_algorithms/data_structures_algorithms_tutorial.pdf

[18] SmitaPaira, Sourabh Chandra, SkSafikulAlam and Subhendu Sekhar Patra, Bi Linear Search a New Session of Searching, International Journal of Advanced Research in Computer Science and Software Engineering, 2014.

[19] “Analysis of Binary Search algorithm And Selection Sort algorithm”, http://www.cartagena99.com/recursos/alumnos/apuntes/Binarysearch_sorting.pdf

[20] Muhammad Usman, Zaman Bajwa and MudassarAfza, “Performance Analysis of Searching Algorithms in C#”, International Journal for research in Applied Science & Engineering Technology, 2015.

[21] Nitin Arora, GarimaBhasin and Neha Sharma,”Two way Linear Search Algorithm”, International Journal of Computer Applications, 2014.

[22] Manpreet Singh Bajwa, Arun Prakash Agarwal and SumatiManchanda, “Ternary Search Algorithm: Improvement of Binary Search”, International Conference on Computing for Sustainable Global Development, 2015.

[23] http://introcs.cs.princeton.edu/java/42sort/

[24] http://www.cprogramming.com/discussionarticles/sorting_and_searching.html

[25] Amy CsizmarDalal, “Searching and Sorting Algorithm”, 2004.

[26] YashvantKanetkar, “Data Structures Through C”, BPB publications, 1999 (first edition)

[27] E Balagurusamy,” Object Oriented Programming with C++”, Tata McGraw Hill Education Private Limited, 2010(fourth edition).

[28] Asokan M, Visualization of Sorting Algorithms Using Flash, Volume 5, Issue 1, January - April 2014, pp. 01-15, International Journal of Graphics and Multimedia.

[29] A Soft Computing Approach For Image Searching Using Visual Reranking, International Journal of Computer Engineering and Technology, International Journal of Computer Engineering and Technology, Volume 4, Issue 2, March – April (2013).

[30] Geeta and AnubhootiPapola, An Efficient Sorting Algorithm: Increcomparision Sort, Volume 5, Issue 9, September (2014), pp. 10-16, International Journal of Advanced Research in Engineering and Technology

[31] http://archive.mu.ac.in/myweb_test/syllFybscit/C++.pdf

[32] http://python-textbok.readthedocs.io/en/1.0/Sorting_and_Searching_Algorithms.html