International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2020): 7.803

A Brief Survey and Examination of Various Searching Techniques

Cherukuri Nischay Sai

Bachelor's Student, Computer Science Engineering (CSE) Malla Reddy College of Engineering, Maisamaguda, Hyderabad

Abstract: ***Searching is a traversal technique in the data structure, and it is used to find a particular element in a given list of items. This research paper presents the primary searching algorithm types and analyses their working process, time complexity, space complexity, advantages, disadvantages, and examples. Our study found that hash search is efficient for more significant data items, and exponential search is used for an infinite set of elements. Whereas binary search is suitable for mid - sized data items and applicable for arrays and linked lists, Dependent upon the analysis, a comparative study is being made so that the user can choose the type of searching technique based on the requirement.***

Keywords: Searching, Linear search, Binary search, Interpolation search, Jump search, Exponential search, Hash Search

1. Introduction

In computer science, searching algorithms are used to find an element and fetch the element from any data structure. Searching is the process of finding an item with some described properties from a collection of items. The item may be stored in arrays, text in files, records in the database. In this present generation, computers store a lot of information. To retrieve this information adroitly, we need very efficient searching algorithms to search and fetch the data in less time and be efficient. This study will discuss linear search, binary search, Interpolation search, Jump search, Exponential search, Hash search algorithms based on the efficiency and time and space complexity. Searching techniques are broadly classified into two categories.

Internal searching: A searching method is entitled internal search, in which searching is involved with the data, which are entirely stored in the internal memory of a computer. These internal searching methods are again categorized into two types: with key comparisons and without any comparisons. Searching with key comparisons are classified into two sub - groups: linear search, where items are stored in an array, linked list, etc., and non - linear search, where items are stored in non - linear data structures like trees, graphs, sets, etc.

External searching: external searching methods deals with a large amount of data that is stored in secondary memory like magnetic tapes, disks, tapes, etc. Data directed from secondary memory to the main memory has speed as one of the constraints. So, data has to be arranged and retrieved to improve the speed of the process.

2. Different Types of Searching Algorithms

2.1 Linear Search

Linear search is the most straightforward search algorithm and is frequently called sequential search. This search applies to a small size of the array or linked list data structure. When a search is to be performed, we look through the array sequentially and scan the complete array and see whether the element is present in the given list or not. Linear search is mostly used to search in a list in which elements are not sorted.

Time complexity:

- **Best case:** O (1); The element being searched is found at the first position.
- **Worst case:** O (n); The element being searched is present at the last position or not present in the array at all.

Space complexity:
O (1); we don't require any extra space to store anything. We need to compare the given value with the elements in the given list one after the other.

Sequential search uses (array implementation) uses N+ 1 comparison for an unsuccessful search (always) and about N/2 comparisons for a successful search (on an average). If we assume each record is equally likely to be sought for a successful search, then the average number of comparisons is N+1/2.

Algorithm:
Linear Search (Array D, Value N, Value P)
Step 1: Set i to 0
Step 2: if i > N (size of the list) then go to step 7
Step 3: if D [i] = V then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print element P is found at index i and goto step 8
Step 7: Print element not found
Step 8: Exit

Implementation:

```
intlinearSearch (int D [], intN, int P) {
for (i=0; i<N; i++) {
if (i>N)
return - 1; // element not found
else if (D [i] == P) {
return i; //element is found at index i
}}
```

Example: An array with 10 elements, find "43":

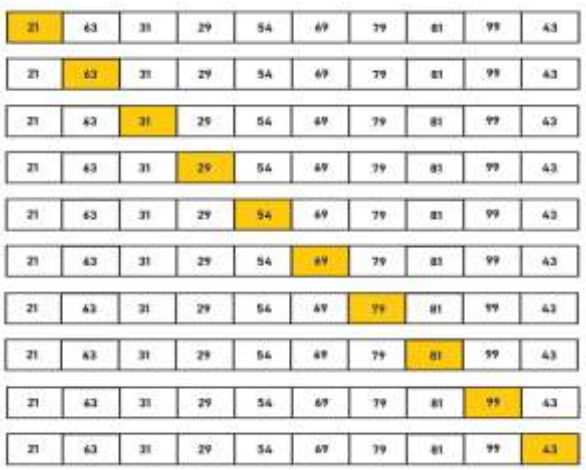


Figure 1: Example for Linear search

Advantages:

- Linear search does not require the list to be in sorted order, and we can add or remove elements; other searching algorithms may have to reorder the elements after adding or removing the elements. In this case, linear search will be more efficient.
- Linear search can operate both ordered and unordered arrays and linked lists.
- The linear search uses memory effectively.

Disadvantages:

- Not suitable for enormous data set.
- The linear search technique is less efficient and not faster than other searching algorithms.

2.2 Binary Search

Suppose you log on to Instagram. When you do, Instagram has to verify that you have an account on the site. So, it needs to search the name in its database. Suppose your username starts with L. Instagram could begin from the as and search for your name – but it makes more sense to begin somewhere in the middle. This is a search problem. To solve this type of problem, we can use binary search.

Suppose you're searching for a person in the phone book. Their name starts with N. You could start at the beginning and keep flipping pages until you get to the Ns. But you're more likely to start at a page in the middle because you know the Ns will be near the centre of the phone book. Binary search works in the same way. The algorithm that implements such an approach is introduced as binary search.

The run - time complexity of the binary search is O (log n) so, it is one of the fast search algorithms. Binary search is based on the principle of divide and conquer, in which it is dividing the items in half. A reasonable way to separate the elements into parts is to keep the list in sorted order. The elements can be sorted in ascending order if the elements are numbers and dictionary order if the elements are strings. For an unsorted list, first, we need to sort the list with some sorting technique, and we need to apply the binary search.

The binary Search Algorithm searches an element by juxtaposing it with the middle element of the list.
Case 1: If the element needed to be searched in the middle element, then its index is returned.
Case 2: If the element that needs to be searched is smaller than the middle element, then its search is continued in the left sub - array of the middle element.
Case 3: If the element that needs to be searched is greater than the middle element, then its search is continued in the right sub - array of the middle element.

This process keeps on repeating on the sub - array until the search element is found or the size of the sub - array is reduced to zero.

Time Complexities:
Recurrence for binary search is T (n) = T (n/2) + 1. This is because we are always considering only half of the input list and throwing out the remaining half. Using the Divide and Conquer theorem, we get T (n) = O (logn).

- **Best case:** O (1); when the search element is equal to the middle element in the array
- **Worst case / Average case:** O (logn);

Space complexity:
Space complexity in Binary search depends on the way how the algorithm has been implemented. Binary search can be implemented in the Iterative and Recursive methods.

- **Iterative method:** In the Iterative method, the iterations are controlled through looping conditions. The space complexity of binary search in the iterative method is O (1).
- **Recursive method:** In this method, there is no loop, and the new values are passed to the next recursion of the loop. Here, the low and high values are used as the boundary condition. The space complexity of binary search in the recursive method is O (log n).

Algorithm:
Step 1: Assign low to 0 and high to n - 1 where n is the size of the array.
Step 2: Find the middle element in the sorted list mid = (low) + (high - low) /2.
Step 3: Compare the search element with the middle element in the list.
Step 4: If both are equal, then display then return the index of the middle element.
Step 5: If both aren't matched, then check whether the search element is smaller or larger than the middle element.
Step 6: If the middle element is smaller than the search element, set low to mid +1 and repeat steps 2, 3, 4, and 5 for the right sublist of the middle element.
Step 7: If the middle element is larger than the search element, set high to mid - 1 repeat steps 2, 3, 4, and 5 for the left sublist of the middle element.
Step 8: Repeat this process until we find the search element within the list or until the sublist contains just one element.

```
// iterative Binary Search Algorithm
intBinarySearchIterative (int a [], int n, int data) {
int low = 0;
int high = n - 1;
while (low <= high) {
mid = low + (high - low) /2;
```

```
if (a [mid] == data)
return mid;
else if (a [mid] < data)
low = mid + 1;
else if (a [mid] > data)
high = mid - 1;
 }
return - 1;
}
//Recursive Binary Search Algorithm
intBinarySearchRecursive (int a [], int low, int high, int data)
int mid = low + (high - low) /2;
if (low>high)
return - 1;
if (a [mid] == data)
return mid;
else if (a [mid] < data)
returnBinarySearchRecursive (a, mid+1, high, data);
else return BinarySearchRecursive (a, low, mid - 1, data);
return - 1;
}
```

Example:

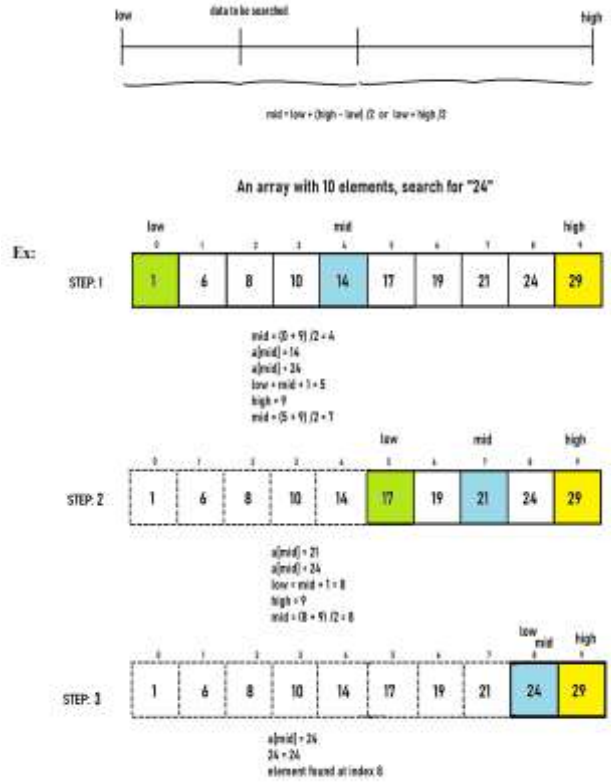


Figure 2: Example for Binary search

Advantages:

- If the list is large, Binary search works significantly faster than linear search.
- It removes half of the list from further searching by using the result of each comparison.

Disadvantages:

- The algorithm requires the elements in the list to be in sorted order.

2.3 Interpolation search

Interpolation search is the improvement over the binary search, where the values in the list must be in the sorted order, and these elements must be uniformly distributed. Binary search always chooses the middle element and compares it with the element to search. On the other hand, an Interpolation search may go to different locations based on the value of the search element. If the value of the search element is closer to the last element, interpolation search is likely to start search toward the end side.

The position to be searched can be computed by the following formula
Position = lo + [(x - arr [lo]) * (hi - lo) / (arr [hi] - arr [Lo])]
Where,
lo = lowest index in the list
hi = highest index in the list
x = element to be searched
arr [lo]=value stored at the lowest index
arr [hi]=value stored at the highest index

Time complexities:

- **Best case:** O (1); when the given array is uniformly distributed and calculates the index of the search key in one step.
- **Average case:** O (log (logn)); It occurs when the array is sorted and not uniformly distributed.
- **Worst case:** O (n); where the values of the elements increase exponentially

Space complexity: O (1); The algorithm doesn't require any other data structure other than temporary variables.

Algorithm:
Step 1:Assign lo= 0 and hi= n-1 where n is the size of the list
Step 2: In a loop, calculate the value of "pos" using the probe position formula
pos = lo + [(x - arr [lo]) * (hi - lo) / (arr [hi] - arr [Lo])]
Step 3: If arr [pos] is equal to search element, return the index of that item.
Step 4: If the arr [pos] is less than the search element then calculate the probe position of the right - sub list. Otherwise, calculate the same for the left - sub list.
Step 5: Repeat this process until a match is found or the sub - list is reduced to zero.

Implementation:

```
intinterpolationsearch (intarr [], int n, int x) {
int lo = 0, hi = n - 1, pos;
while (lo <= hi) {
pos = lo + [(x - arr [lo]) * (hi - lo) / (arr [hi] - arr [Lo]) ];
if (arr [pos] == x) {
returnpos;
}
else if (arr [pos] < x) {
lo = pos + 1;
}
else
hi = pos – 1;
}
return - 1;
}
```

Example: Find an element 4 from the given list

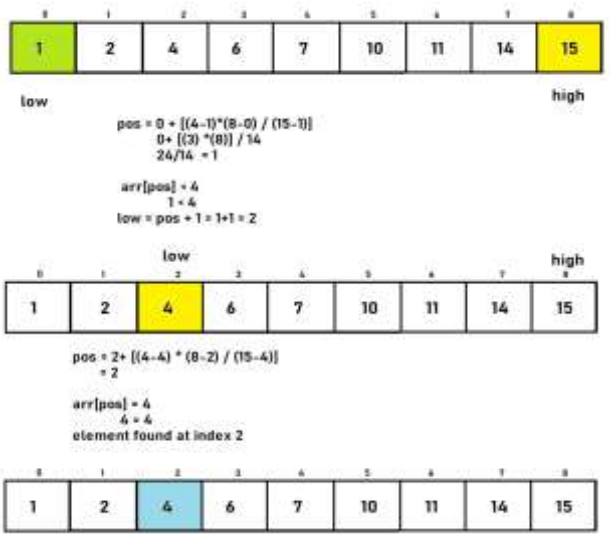


Figure 3: Example for Interpolation search

Advantages:

- When the elements are in sorted order and are uniformly distributed, then executing time will be comparatively lower than the binary and linear search.

Disadvantages:

- Works only one sorted element
- If the elements in the list are increased exponentially, the execution time of the interpolation search is more.

2.4 Jump search

Jump search is the relatively new searching algorithm for searching in which elements must be in sorted order. The basic idea of the jump search is to reduce the number of comparisons required by skipping some elements instead of scanning every element in the array (Linear search). In jump search to find an element, the array is divided into "m" blocks, and the size of the block is based on the size of the array. If the size of the array is N, then the block size will be$\sqrt{N}$, and tries to find the element in one block. If the element is not present, then shift to the next block. When the algorithm finds the correct block in which it contains the element to be searched, then it uses a linear search algorithm and finds the exact index in the particular block. Jump search lies in between the linear search and binary search.

Time complexities:

- **Best case:** O (1); The element to be searched in the first element in the array.
- **Average case:** O ($\sqrt{N}$); If N is the size of the list and 'm' is the block size, then we do N/m jumps and linear search requires m - 1 comparisons and making the total time of expression N/m + m - 1, the most optimal value of m minimizing time Expression is $\sqrt{N}$. so, the time complexity of jump search is O ($\sqrt{N}$).
- **Worst - Case:** O ($\sqrt{N}$); The worst - case occurs when we do N/m jumps, and the last value we checked is greater than the searching element, and m - 1 comparisons are performed for linear search. The worst - case time complexity is O ($\sqrt{N}$).

Space complexity: O (1); The algorithm doesn't require any other data structure other than temporary

Algorithm:

Step 1: set i = 0 and m = $\sqrt{N}$
Step 2: compare Arr [m] with data. If Arr [m] != data and Arr [m] < key, then jump to the next block Also, do the following:
2 - 1: set i = m
2 - 2: increment m by $\sqrt{\square}$
Step 3: Repeat the step 2 till m < N
Step 4: check if m > N – 1, then set m = N
Step 5: if Arr [m] > key, then move to the beginning of the current block and perform a linear search
5 - 1: set x = i
5 - 2: compare Arr [x] with data, if Arr [x] = key, then print x as the valid location else set x++
5 - 3: repeat steps 4 - 1 and 4 - 2 until x < m
Step 6: exit

Implementation:

```
IntjumpSearch (intArr [], int N, int key) {
int i = 0, x;
int m = sqrt (N); // initializing block size = √□
while (arr [m] <= key && m < N) {
i = m;
m = m + sqrt (N);
if (m > N - 1)
m = N
}
for (x=i; x < m; x++) {
if (arr [x] == key) {
return x;
}
return - 1;
}
```

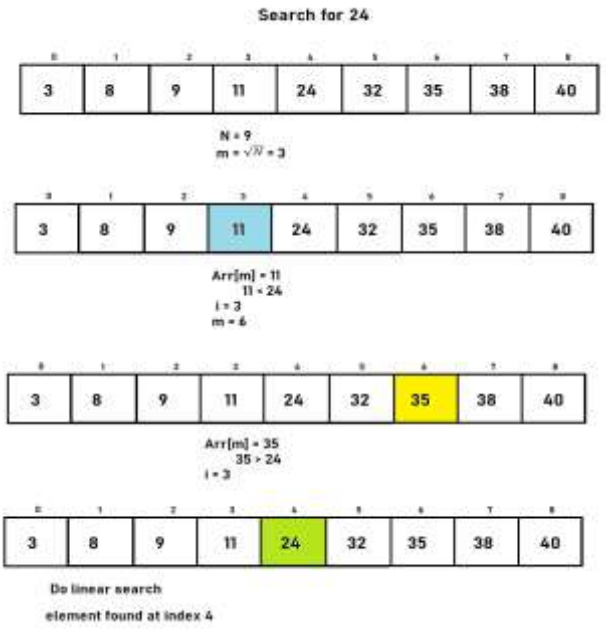


Figure 4: Example for Jump search

Advantages:
- Jump search is faster than the Linear search.
- In jump search, backtracking is done for the only one who is lesser than the binary search.

Disadvantages:
- It requires the array to be in sorted order.
- Jump search is slower than the binary search algorithm.

2.5 Exponential search

Exponential search is also called a Galloping search, doubling search. Exponential search is used for searching sorted, unbounded arrays. The idea of exponential search is to find a range where the target value resides and perform the binary search within that range. The algorithm looks for the first exponent, *l*, where the value 2^l is greater than the search key. This value, 2^l becomes the upper bound for the binary search, and 2^l - 1, being the lower bound for the binary search. For lists with infinite size, exponential search finds the solution much faster than binary search.

Time complexity:
- **Best case:** O (1); the search element is the first element in the list.
- **Average case / worst case:** O (log i); where i is the location where search key is present.

Space complexity: O (1); because it does not require any extra space other than temporary variables.

Algorithm:
Let us consider sorted array Arr [] containing N elements, and we want to find an element key.
Step 1: Check if the first element is equal target element, i. e. Arr [0] == key. If it is true, return the index of the first element.
Step 2: Initialize i value to 1.
Step 3: While i<N and Arr [i] <= key do
Increment i in powers of 2 i. e. i=i*2.
Step 4: Apply binary search on the range i/2 to min (i, N - 1).

Implementation:
```
intexponentialSearch (intArr [], int N, int key)
{
if (Arr [0] == key) //if key is present at first location
return 0;
// Find range for binary search by
int i = 1;
while (i < N &&Arr [i] <= key)
 i = i*2;
 // Call binary search for the found range.
returnbinarySearch (arr, i/2,
min (i, n - 1), x);
}
```

Example: Find an element 14 from the given list

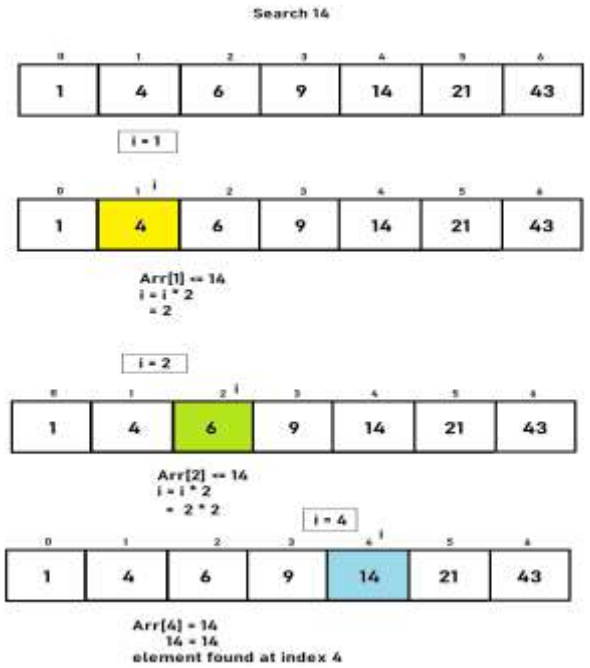


Figure 5: Example for Exponential search

Advantages:
- Exponential search is used for an infinite set of elements.
- It can be applied to a large set of data items.

2.6 Hash Search

All the searching techniques that we have previously discussed (Linear, Binary, Jump, Exponential, Interpolation) where search time is dependent on the number of elements and number of comparisons performed. Whereas, Hashing is a technique that does not depend on the size of the data. In the hash table, the data is stored in an associative manner. In hashing, each data value has its unique index value. Accessing the data becomes efficient and fast if we know the index of the desired data. To determine the index for a given value, we can use the hash function. The hash function is a function which when given a key, generates an address in the table.

Characteristics of a good hash function:
- It should be easy and quick to compute.
- It must decrease the collisions.
- It should distribute the key values evenly in the hash table.

If H is a hash function and K is key, H (K) is called the hash of the key. So a hash function H (K) transforms a key into a hash table index position. If the key is a non - integer value, then we convert the integer value. Some of the techniques for hash function are given below:

a) Mid square: Mid square is a hashing technique in which the key is multiplied by itself (key is squared), and address is obtained by selecting an appropriate number of digits from the middle of the squared number.

b) Folding method: In this method, key K is partitioned into several parts K1, K2, K3…Kn and each part have an equal number of digits, and these parts are added together

in the hash function H (K) =K1 + K2 + K3…. . +Kn. There are two methods in the folding method.

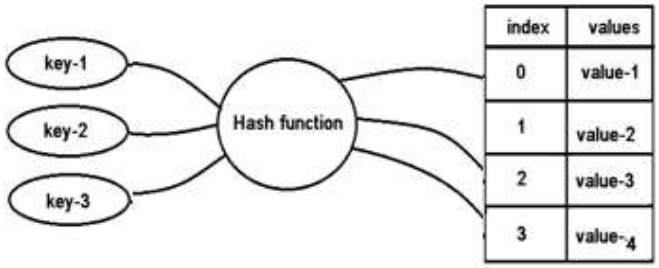
- **Fold shifting method:** The key value is divided into parts and is added together.
- **Fold boundary method:** After the partition, the boundary parts are reversed before the addition.

c) Division method: It is a simple method to calculate the hash function. The hash function can be represented as H (K) where H (K) = K mod m or H (K) = K mod m+1. Here, H (K) is the hash value obtained by dividing the key - value K by the size of the hash table "m" using the remainder. If the list size is a prime number, then it produces fewer collisions.

In the hash function, sometimes there is a possibility that two keys re=sult in the same value in this situation. The newly inserted key maps to an already occupied slot in the hash table are called a collision. They must be handled by collision handling techniques such as chaining or open addressing.

Time complexities of Hash search:

- **Best case / Average case:** O (1); if the hash function that we are using does not store every element into the same key
- **Worst case:** O (n); if too many elements are stored into the same key.



Algorithm:
Step 1: obtain the key (K) to be searched
Step 2: Set J value to zero
Step 3: Compute hash function H (K) = K % size
Step 4: check If the keyspace at the hash table H (K) is occupied
(4.1) Compare the element at hash table H (K) with the given key k.
(4.2) check If they are equal
(4.2.1) the key is found at the bucket H (K)
(4.2.2) else STOP
(4.3) Element may 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 we can probe the bucket at a new slot, H (K).
(4.6) repeat Step 4 till J is greater than the size of the hash table
Step 5: The key is not found in the hash table
Step 6: STOP

Example: Consider 3 students and are assigned three digital student_id and our hash table is 10 from 0 to 9. We need to find an index number using hash function.

Student_id = 112, 114, 127
H (112) = (112 mod 10) = 2
H (114) = (114 mod 10) = 4
H (127) = (127 mod 10) = 7

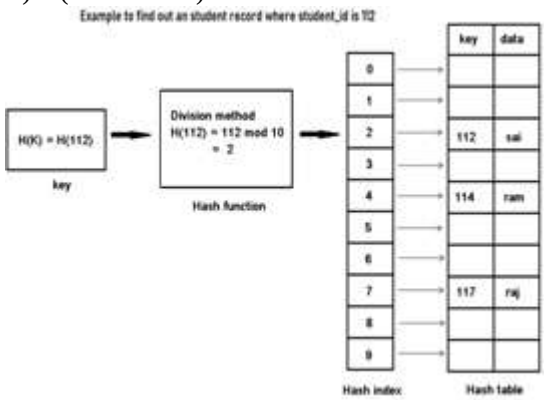


Figure 6: Example for Hash search

Advantages:

- Compared to other searching algorithms, searching is fast and more efficient in hash search.
- Hash search is more flexible and reliable than other search algorithms.

Disadvantages:

- Hash search utilizes a large amount of memory.
- Hash search is not efficient in the small hash table.
- In hash search, the hash function results in the same index for different keys and leads to collisions.

Analysis of Searching Algorithms:

The comparison of various search algorithms is shown below. In exponential search the average and worst case complexity is analyzed. In Exponential search the average case and worst case is O (log i) where i indicates the where search key is present.

Table 1: Analysis of Various Searching Algorithms

Name	Best	Average	Worst	Space
Linear Search	O (1)	O (n)	O (n)	O (1)
Binary Search	O (1)	O (log n)	O (log n)	O (1)
Interpolation Search	O (1)	O (log (log n))	O (log (log n))	O (1)
Jump Search	O (1)	O ($\sqrt{n}$)	O ($\sqrt{n}$)	O (1)
Exponential Search	O (1)	O (log i)	O (log i)	O (1)
Hash Search	O (1)	O (1)	O (n)	O (1)

3. Conclusion

This paper explores several methods of searching. It demonstrates a variety of approaches for various searching techniques. On the basis of time and space complexity, we compared the searching algorithms. The analysis includes examples and the Pseudocode, as well as the benefits and drawbacks of various searching methods. Based on our findings, hash search is best suited for bigger data sets, whereas binary search is best suited for medium - sized data sets and is applicable to arrays and linked lists. In addition, we discovered that exponential search is employed for an unlimited collection of items.

References

[1] Thomas Niemann, “Sorting and Searching Algorithms”.
[2] NarasimhaKarumanchi, ”Data Structures and Algorithms”.
[3] Robert Sedgewick, “Algorithms in C++ Parts 1 - 4: Fundamentals, Data Structure, Sorting, Searching”.
[4] Kurt Mehlhorn, “Data Structures and Algorithms 1: Sorting and Searching Kurt Mehlhorn”.
[5] Cormen T. H., Leiserson C. E., Rivest R. L. and Stein C. (2003) Introduction to Algorithms MIT Press, Cambridge, MA, 2nd edition.
[6] D. Knuth, in The art of programming sorting and searching, 1988.
[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, Introduction to Algorithms is a book on computer programming.
[8] AdityaBhargava, “Grokking Algorithms: An Illustrated Guide for Programmers and Other Curious People”.
[9] Mark Allen Weiss, Data Structures and Algorithm Analysis in C++.