International Journal of Engineering, Management & Technology (IJEMT)
www.ijemt.com, Volume 1 Issue II, April 2022, PP 49-55, ISSN (Online): 2583 – 4517

Conceptual & Mathematical Analysis on Operations of Array: Searching

Pankaj Kumar Gupta[*1], Prof. R. K. Agrawal[2]
1.Assistant Professor & Head, BCA Department, Durga Prasad Baljeet Singh (PG) College, Anoopshahr District Bulandshahr
2.Professor & Head, Department of Mathematics, Durga Prasad Baljeet Singh (PG) College, Anoopshahr District Bulandshahr

Abstract: *This Paper describes full conceptual & mathematical analysis about the operation (Searching of an item) of Array Data Structures. Discussions about Algorithms and Procedures of same above defined operations also made in this paper. Complexities of algorithms are also described in this paper.*

Keywords: *Array, Linear Search, Binary Search, Homogenous, Contiguous, Sorted, Unsorted, Complexity, Time Complexity, Space Complexity, Worst Case, Best Case, Average Case.*

1. MEANING OF ARRAY

Computer Application we define Array as: it is a Linear Data Structure. It is also termed as a collection of homogenous data elements. All the elements of an array share common name and stored on contiguous locations. It is also categorized into derived data types.

Examples

1. A collection of aadhar numbers of 50 employees (in integer datatype).
2. A collection of marks of 500 students in 3-3 subjects for a class (in float datatype).
3. A collection of names with surname of 500 employees (in character datatype).

Categories of Arrays

Arrays are categorized into 3 categories that are defined as follows:

1. Single Dimensional Arrays.
2. Two Dimensional Arrays.
3. Multidimensional Arrays.

Operations of Array Data Structures

Various operations are performed on Array Data Structure, that are as follows:
Here Discussion is made on Single Dimensional Array.

1. Traversal
2. Insertion
3. Deletion
4. Searching
5. Sorting
6. Merging

But We will discuss about Search operation in this paper.

* **Corresponding Author:** Pankaj Kumar Gupta
Published online on www.ijemt.com : April 25, 2022

2. SEARCHING IN ARRAY

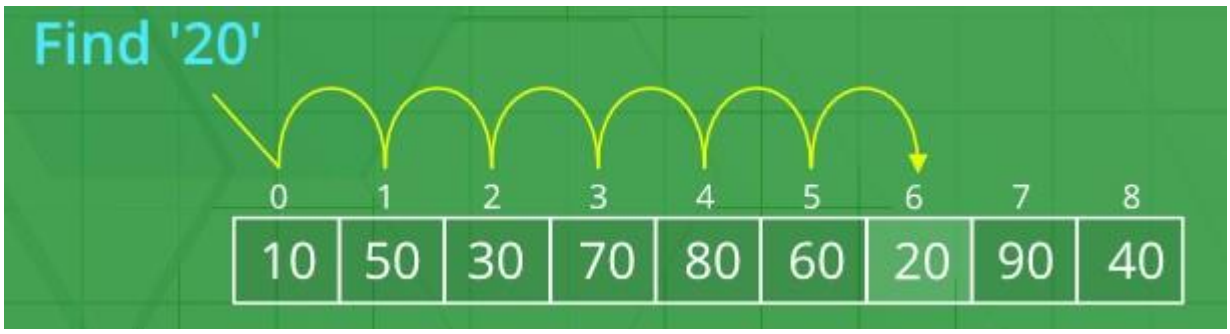
Searching is one of the important operations of data structures. In the Array searching can be said to as **the operation of finding a particular element or a group of elements in the array**. We can say in other words to find the location of given item in the given list. If item is present in the list, we say search is successful and if not present we say search is unsuccessful.

There are several searching algorithms. Some of them are as follows:

- Linear Search
- Binary Search

Linear Search

Linear Search is a search technique in which item is compared with each element of array until it is matched or list is ended. if item is matched with any element of array we say search is successful and show the location of matched element Otherwise we say search is unsuccessful. it also known as Sequential search.



Item may be present in following two ways in Array:

1. Only once in the Array.
2. Two or more places in the Array.

If item is present only at single place in the array we use following Algorithm:

Algorithm LINEARSEARCH(A,N,ITEM,LOC)

```
//Here LINEARSEARCH is an algorithm which is used to find the
//location LOC of the ITEM in the Array A with N elements using
//Linear Search Technique.
Note: Item is considered to be present only once in the Array
Step-1 Start
Step-2 Set LOC=-1.
Step-3 Set K:=0.
Step-4 Repeat steps 5 & 6 while K<N
Step-5 if ITEM=A[K].
            Then set LOC:=K and go to step 7.
Step-6 Set K:=K+1.
Step-7 if LOC != -1
            Then Print "Search is Successful and Location is ", (LOC+1).
       Otherwise
            print "Search is unsuccessful".
```

Step-8 Exit.

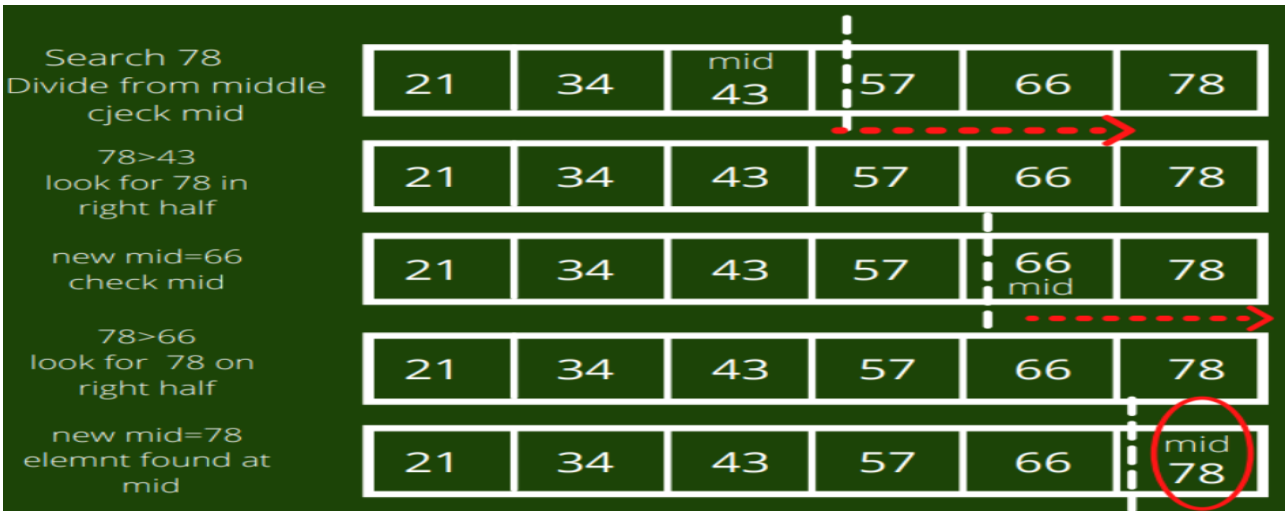
If item is present on two or more places in the array we use following Algorithm:

Algorithm LINEARSEARCH(A,N,ITEM,LOC)

//Here LINEARSEARCH is an algorithm which is used to find the
//location LOC of the ITEM in the Array A with N elements using
//Linear Search Technique.
Note: Item may be present more than once in the Array
Step-1 Start
Step-2 Set LOC=-1.
Step-3 Set K:=0.
Step-4 Repeat steps 5 & 6 while K<N
Step-5 if ITEM=A[K].
Then set LOC:=K and print "Search is Successful and Location is ", (LOC+1).
//For printing all the locations of ITEM
Step-6 Set K:=K+1.
Step-7 if LOC == -1
Then print "Search is unsuccessful".
Step-8 Exit.

Binary Search

If we have an array that is sorted, we can use a much more efficient algorithm called a Binary Search. In binary search each time we divide array into two equal halves and compare middle element with search element. If middle element is equal to search element, then we got that element and return that index otherwise if middle element is less than search element we look right part of array and if middle element is greater than search element we look left part of array.



Item may be present in following two ways in Array:

1. Only once in the Array.
2. Two or more places in the Array.

If item is present only at single place in the array we use following Algorithm:

Algorithm BINARYSEARCH(A,N,ITEM,LOC,LB,UB)

```
//Here BINARYSEARCH is an algorithm which is used to find the
//location LOC of the ITEM in the Array A with N elements using
//Binary Search Technique. LB is the Lower bound of A and UB is the
//Upper bound of A. Consider that starting Index of A is 0.
Step-1 Start
Step-2 Set BEG:=LB, END:=UB & Set LOC=-1.
Step-3 Repeat steps 4 & 5 while BEG<=END
Step-4 Set MID:=(INT) (BEG+END)/2.
Step-5 if (ITEM = A[MID])
            Then set LOC:=MID and go to step 6.
       Else if ITEM<A[MID]
            Then set END:=MID-1.
       Else
            Set BEG:=MID+1.
Step-6 if LOC = -1
            Then print "Search is unsuccessful".
       Otherwise
            Print "Search is Successful and Location is ", (LOC+1).
Step-7 Exit.
```

If item is present on two or more places in the array we use following Algorithm:

Algorithm BINARYSEARCH(A,N,ITEM,LOC,LB,UB)

```
//Here BINARYSEARCH is an algorithm which is used to find the
//location LOC of the ITEM in the Array A with N elements using
//Binary Search Technique. LB is the Lower bound of A and UB is the
//Upper bound of A. Consider that starting Index of A is 0.
Note: Item may be present more than once in the Array
Step-1 Start
Step-2 Set BEG:=LB, END:=UB & Set LOC=-1.
Step-3 Repeat steps 4 & 5 while BEG<=END
Step-4 Set MID:=(INT) (BEG+END)/2.
Step-5 if (ITEM = A[MID]) Then
            a. set LOC:=MID.
            b. Set MID = MID-1.
            c. Repeat while ITEM = A[MID]
                  i.  Set LOC = MID+1.
                  ii. Set MID = MID-1.
            d. Set MID = MID+1
            e. Repeat while ITEM = A[MID]
                  i.  Set LOC = MID+1.
                  ii. Set MID = MID+1.
                  iii. Print the LOC.  //For printing all the locations of ITEM
            f. go to step 6.
       Else if ITEM<A[MID]
            Then set END:=MID-1.
       Else
            Set BEG:=MID+1.
Step-6 if LOC = -1
```

Then print "Search is unsuccessful".

Step-7 Exit.

Note: But we suggest for Searching the items that have the frequency more than one in a sorted List (Array), we must use the Linear Search Technique instead of Binary Search Technique.

Complexity

The complexity of a procedure or an algorithm is used to compute the amount of time and spaces required by an algorithm for an input of size (n). The complexity of an algorithm can be categorized into two categories: The **time complexity** and the **space complexity**.

Time Complexity

The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the size of the input to the problem. The time complexity of an algorithm is commonly expressed using big O notations, which suppresses multiplicative constants and lower order terms. When expressed this way, the time complexity is said to be described asymptotically, i.e. as the input size goes to infinity. For example: if the time required by an algorithm on all inputs of size n is at most $5n^3+3n$, the time complexity is $O(n^3)$. Time complexity can be measured in three cases:

1. Best case
2. Average case
3. Worst case

Space Complexity

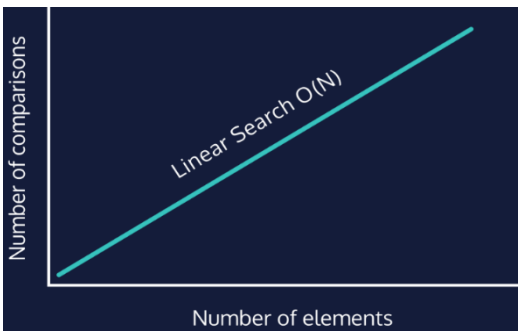
Space complexity can be defined as the process of computing how much memory space is required for the successful execution of the algorithm. The memory space is generally considered to be as the primary memory.

Complexity Measure for Linear Search:

The complexity of linear search for an array of ***n elements*** is measured according two types i.e. Time Complexity & Space Complexity in three Cases i.e. best case, average case, and worst case.

Time Complexity

Case	Time Complexity
Best Case	O(1)
Average Case	O(n)
Worst Case	O(n)



- **Best Case Complexity –** if item to be searched is at first position in the list, it will be the best case in Linear search. So the best-case time complexity of linear search is **O(1).**

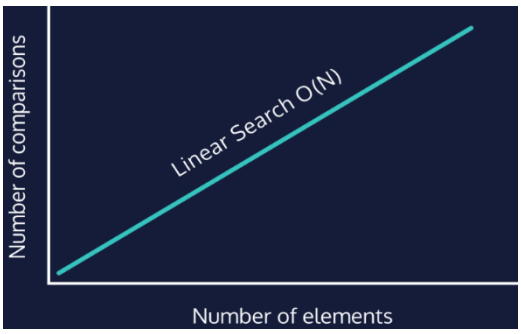
- **Average Case Complexity -** if item to be searched is at any position except first and last position in the list, it will be the average case in Linear search. So the average case time complexity of linear search is **O(n).**
- **Worst Case Complexity -** if item to be searched is at Last position in the list, it will be the Worst case in Linear search. If item to be searched is not present in the list, it will again be the Worst Case. So in the worst-case time complexity of linear search is **O(n).**

  *Since every element in the array is compared only once, the time complexity of linear search is **O(n)**.*

Space Complexity

Space Complexity	O(1)

*The space complexity of linear search is **O(1)**.*

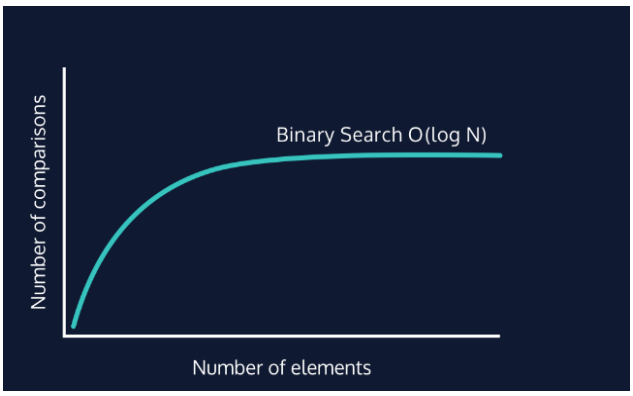


Complexity Measure for Binary Search

The complexity of binary search for an array of ***n elements*** is measured according two types i.e. Time Complexity & Space Complexity in three Cases i.e. best case, average case, and worst case.

Time Complexity

Case	Time Complexity
Best Case	O(1)
Average Case	O(log(n))
Worst Case	O(log(n))



- **Best Case Complexity –** if item to be searched is at middle position in the list, it will be the best case in Linear search. So the best-case time complexity of linear search is **O(1).**
- **Average Case Complexity -** if item to be searched is at any position except first and last position in the list, it will be the average case in Linear search. So the average case time complexity of linear search is **O(log(n)).**

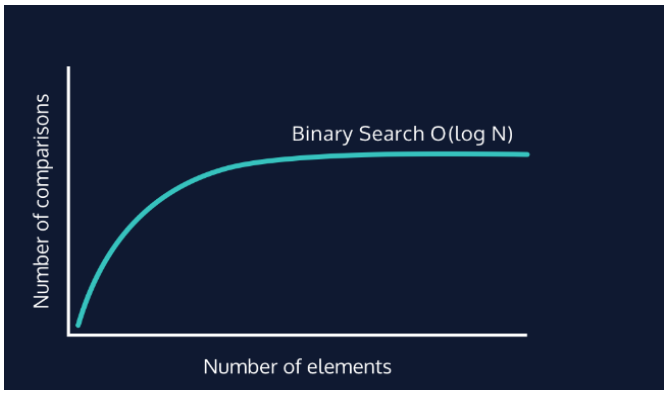
Worst Case Complexity - if item to be searched is at First or Last position in the list, it will be the Worst case in Linear search. If item to be searched is not present in the list, it will again be the Worst Case. So in the worst-case time complexity of linear search is **O(log(n)).**

Since every element in the array is compared only once, the time complexity of linear search is ***O(log(n))****.*

Space Complexity

Space Complexity (Iterative)	O(1)
Space Complexity (Recursive)	O(log(n))

The space complexity of linear search is ***O(log(n))****.*



CONCLUSION

An array can also be termed as a collection of homogeneous values. Anybody can treat an array as a single object by referring to it through a variable. Variable name is succeeded by square brackets (that is [ ] symbols) But you can also treat the components of the array as if they are themselves variables. If elements are unsorted, we should use Linear/Sequential Search Algorithm and if it is sorted, we should use Binary Search Algorithm for finding the location of an item in the list.

References

1. www.biitonline.co.in (Author Pankaj Kumar Gupta, Head, BCA Department, DPBS College, Anupshahr)
2. Computer Science with C++ By SumitaAroraby SumitaArora.
3. Let Us C by YashavantKanetkar
4. Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.
5. Sartajsahni, “Data structures, algorithms and applications in C++”, University press.
6. Seymour Lipschutz. “Theory and problems of data structures”, Tata Mcgraw hill international editions”.
7. The C Programming Language by Brian W. Kernighan / Dennis Ritchie
8. www.en.wikipedia.org/wiki/Array.
9. https://www.codecademy.com
10. www.geeksforgeeks.org
11. www.w3schools.com