International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 05 Issue: 04 | Apr42018 www.irjet.net p-ISSN: 2395-0072

Significance of Searching and Sorting in Data Structures

Megharaja D.S [1], Rakshitha H J [2], Shwetha K [3]

[1,2,3] *Lecturer, Department of Computer Science, DVS College of Arts and Science, Shivamogga*

Abstract - *A data structure is a specialized format for organizing and storing data. General data structure types include the array, the file, the record, the table, the tree, and so on. Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways. Searching is the process of finding a particular item in a group of items. Not even a single day pass, when we don't have to search for something in our day to day life, like car keys, mobile charger, books etc., Same is the life of computer, there is so much data stored in it, that whenever a user asks for some data, computer has to search its memory to look for the data and make it available to the user. Sorting refers to rearrangement of data items in a particular order. We sort the items on a list into alphabetical or numerical order. We have well-known Sorting techniques to sort elements either in ascending or descending order. Searching and Sorting are the most basic problems in computer science, as it is used in most of the software applications. The computer has its own techniques to search and sort the elements through its memory, which we look here.*

Key words: **Algorithm, Linear Search, Binary Search, Insertion Sort and Quick Sort**.

1. INTRODUCTION

Search is process of finding a value in a list of values. In other words, Searching is the process of locating given value position in a list of values.

Sorting refers to the operation of arranging data in some given sequence i.e., increasing or decreasing order. Sorting is categorized as internal sorting and external sorting. Internal sorting means we are arranging the elements within the array which is only in computer primary memory. Whereas the external sorting is the sorting of elements from the external file by reading it from secondary memory.

2. SEARCH ALGORITHMS

Algorithm is a sequence of instructions or a set of rules that are followed to complete a task. In the discussion that follows, we use the term search term to indicate the item for which we are searching. We assume the list to search is an array of integers, although these algorithms will work just as any other primitive data type (doubles, characters, etc.). We refer to the array elements as items and the array as a list

2.1 Linear search

Linear Search is one of the basic and simplest search algorithm and it is also called as Sequential search algorithm. It is used for unsorted and unordered small list of elements. In this technique we search for a given key item in the list in linear order i.e., one after the other. The item to be search is often called key item. Linear search algorithm finds given element in a list of elements with O(n) time complexity where n is the total number of elements in the list.

2.1.1 Implementation of Linear Search

Following are the steps of implementation that we will be following:

1. Traverse the array using a **for** loop.
2. In every iteration, compare the **key item** value with the current value of the array.
   - If the value match, return the current index of the array.
   - If the values do not match, move on to the next array element.
3. If no match is found, return -1.

2.1.2 Function Code:

```
int linearSearch( int values[ ], int keyitem, int n)
{
        for(int i = 0, i < n; i++)
        {
                if( values[i] == keyitem)
                {
                        return i;
                }
        }
        return -1;
}
```

2.1.3 Performance of linear search

When comparing search algorithms, we have to see number of comparisons required, since we don't swap any values while searching. Often, when comparing performance, we look at three cases:

Best case: the number of comparisons in this case is 1 i.e., O(1)

Worst case: it takes N comparisons and is equal to the size of the array i.e., O(n)

Average case: On average, the search term will be somewhere in the middle of the array i.e., O(n).

2.2 Binary search

A binary search is a simple searching technique which can be applied if the items to be compared are either in ascending or descending order. The general idea used in binary search is similar to the way we search for the address of a person in a address book. Obviously we don't use linear search. Instead, we open the book from the middle and the name is compared with the element at the middle of the book. If the name is found, the corresponding address is retrieved and the searching has to be stopped. Otherwise, we search either the left part of the book or right part of the book. If the name to be searched is less than the middle element, search towards left otherwise, search towards right. The procedure is repeated till Target items is found or Target item is not found.

Implementation of Binary Search: While designing the program the low is considered as the position of first element it is initialized to 0 and high as the position of the last element it is initialized to n-1, the middle element position can be obtained using

mid= (low+high)/2;

The **Target Element** to be searched is compared with middle element. If they are equal the position of item in the array is returned. If the condition is false, the target element may be present in either left part of the array or in the right part of the array. If **Target** element is less than the middle element then the left part of the array has to be compared from *low to mid-1.*Otherwise, the right part of the array has to be compared from *mid+1 to high.*

Finally when low exceeds high, it indicates that item not found in the array.

2.2.1 Function code :

```
void bsearch(int item. Int a[ ],int n, int *pos)
 {
        Low=0;
        High=n-1;
while(low <= high)
{
        mid = (low+high)/2;
        if( item== a[mid])
        {
                *pos = mid;
                return;
        }
        if(item<a[mid])
                high= mid-1;
        else
                low= mid+1;
}
*pos= -1;  /* Item not Found*/
}
```

2.2.2 Performance of binary search

Best case: An operation which is done to reach an element directly i.e., O(1)

Worst case: In this scenario the search term is not in the array, or search term is the first or last item in the array. i.e., O(log n)

Average case: search term is anywhere in the list i.e., O(log n).

3. Sorting Algorithms

3.1 Insertion sort:

Insertion sort is a very simple method to sort numbers in an ascending or descending order. This method follows the incremental method. It can be compared with the technique how cards are sorted at the time of playing the game.

Here the sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be inserted in this sorted sub-list, has to find its appropriate list and then it has to be inserted there.

Hence the name insertion sorts. This algorithm is not suitable for large data sets.

3.1.1 Implementation of Insertion Sort:

Following are the steps involved in insertion sort:

1. We start by making the second element of the given array, i.e. element at index 1, the key.
2. We compare the key element with the element(s) before it, in this case, element at index 0:
    - If the key element is less than the first element, we insert the key element before the first element.
    - If the key element is greater than the first element, then we insert it after the first element.

3. Then, we make the third element of the array as key and will compare it with elements to it's left and insert it at the right position.
4. And we go on repeating this, until the array is sorted.

3.1.2 Function Code:

```
void insertion(int a[ ],int len)
{
        for(int i=1;i<len;i++)
        {
                j=i;
        while(j>0 && a[j-1] > a[j])
        {
                key=a[j];
                a[j]=a[j-1];
                a[j-1]=key;
                j--;
}
    }
}
```

3.2 Quick sort:

Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays.

Quick sort is a divide and conquer algorithm to gain same advantages as the merge sort, while not using additional storage.

3.2.1 Implementation of Quick Sort:

This sorting technique is well suited for large set of data. First we partition the given array into two sub-arrays such that the elements towards left are less than the key element and elements towards right are greater than key element.

To partition the array into two sub-arrays, two index variables i and j are maintained. The index variable i points to low+1 where low points to the first element and the index variable j points to high which points to the last element. The item a[low] is used as a key element. This key item has to be placed in a position j such that a[k] <= a[j] for low <= k < j and a[j] >= a[1] for j + 1 <= 1 <= high. This can be achieved by comparing the item key with a[i] and a[j]. Keep incrementing the index i whenever key >= a[i]. Immediately when this condition fails, keep decrementing the index j whenever key < a[j]. At this stage if i is less than j, exchange a[i] with a[j] and repeat the process. If i is greater than or equal to j then exchange a[low] with a[j] and return j which gives the position of the partitioned element. Now the element towards left of a[j] are all less than a[j] and elements towards right of it are greater.

3.2.2 Algorithm for Quick Sort :

Input : a,n, item,key;

```
algorithm  partition( a, low, high)
{
        key=a[low];
        i = low+1;
        j=high;
        while(1)
        {
                while(i<high && key >= a[i])
                        i++;
                while(key < a[j])
                        j--;
                if   i<j  then
                {
                Exchange  a[i] and a[j]
                }
                else
                {
                Exchange a[low] and a[j]
                        return j;
                }
         }
}
algorithm quicksort( a,  low,  high)
{
                if  low < high then
                {
                j=partition(a,low,high);
```

```
        quicksort(a,low,j-1);

        quicksort(a,j+1,high);

                }
}
```

4. CONCLUSION & FUTURE SCOPE

In this research paper we have studied about different Searching and sorting algorithms. Every searching and sorting algorithm has advantage and disadvantage. Some sorting algorithms have been compared on the basis of different factors like complexity, number of passes, number of comparison etc. It is also seen that many algorithms are problem oriented so we will try to make it global oriented. Hence we can say that there are many future works which are as follows.

1) Remove disadvantage of various fundamental sorting and advance sorting.

2) Make problem oriented sorting to global oriented.

In the end we would like to say that there is huge scope of the sorting algorithm and searching in the near future, and to find optimum-sorting algorithm, the work on sorting algorithm will go on forever.

Each sorting algorithm has its own advantages and disadvantages. Selection sort O(N2) sorts and quick sort O(NlogN) sorts.

• The advantage of Selection sort is that the number of swaps is O(N), and the disadvantage is that it does not stop early if the list is sorted and it looks at every element in the list in turn. It is suitable in cases where you need to select max element in the list. With time complexity same as Bubble Sort o(n^2), its performance however is better than Bubble Sort.

• The advantage of Quick sort is that it is the fastest sort and is O(NlogN) in both the number of comparisons and the number of swaps and the disadvantage is that the algorithm is a bit tricky to understand.

5. REFERENCES

[1] Savina & SurmeetKaur, "Study of Sorting Algorithm to Optimize Search Results", International Journal of Emerging Trends & Technology in Computer Science, Volume 2, Issue 1.

[2] Md. Khairullah, "Enhancing Worst Sorting Algorithms", International Journal of Advanced Science and Technology, Vol. 56

[3] Er. Rahul Kaushal, "Why Sorting is So Important in Data Structures", International Journal of Scientific Research and Development Vol. 5, Issue 07

[4] W. Sarada, Dr. P. V. Kumar, "A Comparative Study and Analysis of Searching and Sorting algorithms" , International Journal of Advanced Research in Computer Engineering & Technology (IJARCET). Volume 5, Issue 5

[5] Systematic Approach to Data Structures using C- A.M Padma Reddy

[6] https://www.studytonight.com

[7] http://btechsmartclass.com