Paw Search-A Searching Approach for Unsorted Data Combining with Binary Search and Merge Sort Algorithm

Paw Search Algorithm

Md. Harun Or Rashid[1], Ahmed Imtiaz[2]
Department of Computer Science and Engineering, Rangamati Science and Technology University, Jhagrabil, Rangamati, 4500, Chattogram, Bangladesh[1, 2]

***Abstract*—Searching is one of the oldest core mechanism of nature. Nature is changing gradually along with searching approaches too. Data Mining is one of the most important industrials topic now-a-days. Under this area all social networks, governmental or non-governmental institutions and ecommerce industries produce a huge number of unsorted data and they are to utilize it. For utilizing this huge number of unsorted data there needs some specific features based unsorted data structure tools like searching algorithm. At present there are several sorted data based searching algorithms like Binary Search, Linear Search, Jump Search and Interpolation Search and so on. In this paper of Paw Search Algorithm, it is fully focused to develop a new approach of searching that can work on unsorted data merging several searching techniques and sorting techniques. This algorithm starts its operation by breaking down the given unsorted array into several blocks by making the square root of the length of the given array. Then these blocks will be searched within its specific formula till the target data is found or not, and in the inner side of each block there will be performed Merge Sort and Binary Search approach gradually. Time and Space Complexity of this Paw Search algorithm is comparatively optimal.**

***Keywords*—Paw; search; unsorted; data; blocks; square root**

I. Introduction

In this present world, technology is the heart of all activities, operations and so on, and the large amount of unsorted data sets generated from different sites of the world as well as different institutions are the largest and best resources of the present technology. Managing this large amount of data with proper data structure techniques is the best tool for leading this IT world now-a-days. In this paper, we will go through a new technique of searching of data from the given unsorted array of data. There are several techniques raised now-a-days, but here we will go through a new dimension of searching and merging several built in techniques along with some new approaches to generate an optimal output.

In present world there are tons of unsorted data produced within a minimal time randomly. There are several searching algorithms like Linear Search [1, 2, 3], Binary Search [1, 2], Jump Search [1, 4], Hybrid Search [1, 5] and Interpolation Search [1, 6] now-a-days which work only on sorted data. But till now there are less approaches that work on randomly generated unsorted data. Several optimal data structures tools are badly required to operate this very large number of unsorted data which are producing day by day. Data Mining is one of the most important industrials topic now-a-days. Under this area of data mining for all social networks, governmental or non-governmental institutions and ecommerce industries produce tons of unsorted data and they are to utilize it. For utilizing this tons of unsorted data there is a need of some specific feature based unsorted data structures tools like searching algorithm [6] based on unsorted data array. So, a data structure tool [7] that will work on directly unsorted data is the prime concern for developing another searching approach.

Data scientists are trying to develop several data structures tools to utilize the tons of unsorted data randomly around the globe continuously. With a view to helping the data scientists here I am trying to develop data structures tools for finding out of any data from any given array of unsorted data. We know that sorting of data consumes a large number of time; so, from this concept of time utilization there needs some specific approach that can perform searching operation on unsorted data which minimize the uses of time. As the technology and technology related models/industries appreciate the approaches that minimize time consuming, so this is the demand of time to have high performer approach consuming less time without sorting the large data set at a single time.

Through this whole paper we will go through the approaches to develop a specific feature based searching algorithm entitled paw search algorithm that will be capable to perform searching operations on unsorted data, and here we may also go through the help of some existing searching approaches and sorting approaches at the inner phase of searching operation to ensure the high performance of searching.

The main principle of this Paw Search Algorithm is to work on (i) unsorted data segmenting the given array of data into several (ii) blocks.

Initially it starts working with x blocks of unsorted data by making the square root of the length n of the given array of data i.e., x=ceiling⌈ √ n⌉ where n is length of the given array of unsorted data.

This algorithm will never check all of the blocks of unsorted data linearly, rather than it will go the blocks of unsorted data all but as like as binary approach but not fully follow the binary approaches. And for the inner block operation we will also call here the merge sort approach for the better performance of this paw search algorithm.

II. Literature Review

In this section we will go through the several existing searching algorithms, and most of them here work only on sorted data:

A. Classification of Searching Techniques

There are several searching techniques present now-a-days. Depending on external and internal issue there are two types of searching techniques as (i) external search and (ii) internal search, and based on sequential and interval issue there are two types of searching techniques as (i) sequential search and (ii) interval search.

B. Present Searching Algorithms

There are several searching algorithms based on sorted data. Some of them are listed below-

1) Linear search algorithm: Linear search algorithm [3] could be an easy search algorithmic program. It's a sequent search that performed on sequences of numbers that are ascending or down or unordered. And it checks every component of the whole list to look a specific information from the list. If the comparison is equal, then the search is stopped and declared productive. For a listing with n things, the most effective case is once the worth of item to be searched is adequate to the primary component of the list, during this case only one comparison is required. Worst case is once the worth isn't within the list or happens one time at the top of the list, during this case n comparisons are required.

2) Binary search algorithm: It is a quick search formula because the run-time quality is O (log n). Divide and conquer Principle is used here as its' search formula. This formula performs higher for sorted knowledge assortment. In binary search [8], we tend to 1st compare the key with the item within the middle position of the info assortment. If there's a match, we are able to come forthwith. If the secret's but middle key, then the item should lie the lower 1/2 the info collection; if it's bigger, then the item should lie the higher 1/2 the info assortment.

3) Hybrid search algorithm: Hybrid Search algorithmic [3, 9] rule combines properties of each linear search and binary search and provides a far better and economical algorithmic rule. This algorithmic rule may be accustomed search in associate degree unsorted array whereas taking less time as compared to the linear search algorithmic rule. As mentioned this algorithmic rule is combines 2 looking algorithms, viz. Linear Search and Binary Search. Like Hybrid Search algorithmic rule, the array is split into 2 sections so searched in every of the sections. The algorithmic rule starts with examination the key component to be searched with the 2 extreme components of the array, the primary and therefore the last, further because the middle component. If a match is found, the index worth comes back. However, if it's not, the array is split into two sections, from the center index. Currently the search is meted out within the section on the left in a very similar method. The acute components and therefore the middle component of the left division are compared with the key worth for a match, that if found, returns the index worth. If not, the left section is once more divided into two components and this method goes on until a match is found within the left section. If no match is found within the left division, then the algorithmic rule moves on to the proper division, and therefore the same procedure is meted out to search out a match for the key worth. Now, if no worth is found that matches the key worth even when ransacking through all sections, then it's more divided and therefore the method repeats iteratively till it reaches the atomic state. If the worth isn't gift within the array, as a result of that the algorithmic rule returns -1.

4) Interpolation search algorithm: Interpolation search [2, 10] rule is improvement over Binary search. The binary search checks the part at middle index. However, interpolation search could search at completely different locations supported price of the search key. The weather should be in sorted order so as to implement interpolation search. As mentioned the Interpolation Search is Associate in Nursing improvement over Binary explore for instances, wherever the values in a very sorted array are uniformly distributed. Binary Search continuously goes to the center part to ascertain. On the opposite hand, interpolation search could head to completely different locations in line with the worth of the key being searched. For example, if the worth of the secret's nearer to the last part, interpolation search is probably going to start out search toward the tip facet.

5) Jump search algorithm: Jump search algorithmic [11, 12] rule, additionally known as block search algorithmic rule. Solely sorted list of array or table will use the Jump search algorithmic rule. In jump search algorithmic rule, it's not in any respect necessary to scan each component within the list as we have a tendency to liquidate linear search algorithmic rule. We have a tendency to simply check the m component and if it's but the key component, then we have a tendency to move to the m + m component, wherever all the components between the m and m + m component square measure skipped. This method is sustained till m component becomes adequate to or larger than key component known as boundary price. The worth of m is given by $m = \sqrt{n}$, wherever n is that the total range of components in associate array. Once the m components attain the boundary price, a linear search is finished to seek out the key price and its position within the array. And also the numbers of comparisons square measure adequate to (n/m + m -1). It should be noted that in Jump search algorithmic rule, a linear search is finished in reverse manner that's from boundary price to previous price of m.

Though there is a large number of searching approaches [13] on different aspects like strings [14], numeric values and

so on there is still a concern of optimizing [15, 16] these approaches.

Now-a-days industry requires specific feature based searching tools like audio, video and/or image based searching [17, 18], and as the industry is changing day by day with the help of upgraded technology, searching approaches are also gradually being changed as needed [19, 20].

And, still there needs of most powerful, high performer, fast searching unsorted data based searching approaches; keeping this conscious in mind, this paw search approach for unsorted data is demand of time now.

III. METHODOLOGY

In this Methodology section, we will go through several sections like Planning, Design, Paw Search Algorithm etc. for the development of the proposed approach of search precisely and clearly:

A. Planning

Define To develop this algorithm I have planned several data structure approaches, arrays, sub-arrays or blocking, sorting approaches and so on.

- First plan is to manage several unsorted data sets that may be generated from different environment like weather data, space data and son.
- Second plan is to find out the length of the array with filling this array with that unsorted raw data.
- Third plan is to divide the unsorted array into several sub-arrays which are termed as data blocks in the later chapters of this paper.
- Fourth plan is to find out an optimal way to have operations on these blocks by traversing them.
- Fifth plan is to operate a searching approach on the blocks for finding out the optimal outputs.
- Sixth plan is to calculate the time and space complexity of this algorithm.
- Seventh plan is to compare these time and space complexity with different present searching algorithms properly.

The designation process of this algorithm is briefly described in part B of this section.

B. Design

To design this algorithm we are to go through a list of unsorted data set firstly as the main principle of this Paw Search - A Searching Approach for Unsorted Data Combining with Binary Search and Merge Sort Algorithm is to work on (i) unsorted data segmenting the given array of data into several (ii) blocks.

Initially it starts working with x blocks of unsorted data by making the square root of the length n of the given array of data i.e., x=ceiling⌈ √n ⌉ where n is length of the given array of unsorted data. but when the length of this array isn't a perfect square root number then the block number becomes a fraction number, but the block number can't be a number as a fraction number in real, so we are to operate here the ceiling operator to get the integer number of blocks. But in this situation there needs some dummy data as like zero to make the block size perfect i.e., same length of each block.

For example let assume an unsorted array arr1[ ] of data of the length of 16 which is a perfect square root number that is shown in Table I.

TABLE I. UNSORTED ARRAY ARR[ ] OF 16 LENGTH

5	0	6	3	7	1	9	2	4	17	10	8	11	16	13	15

Here n=16; Since 16 is a perfect square root number

So the block numbers, x = int⌈ √ 16⌉ = 4

TABLE II. X BLOCKS FROM ARR1[ ]

5	0	6	3	7	1	9	2	4	17	10	8	11	16	13	15
Block 1				**Block 2**				**Block 3**				**Block 4**			

Here the Block1, Block2, Block3 and Block4 are the four blocks of the segmented arr1 [ ] shown in Table II

Now let assume another unsorted array arr2 [ ] of data of the length of 8 which is not a perfect square root number that is shown in Table III

TABLE III. UNSORTED ARRAY ARR2[ ] OF 8 LENGTH

7	5	10	3	21	1	6	9

Here n=8; Since 8 is not a perfect square root number

So the block numbers, x = int⌈ √ 8 ⌉

= int⌈2.828427125⌉

= 3 [By applying ceiling operation]

TABLE IV. X BLOCKS FROM ARR2[ ]

7	5	10	3	21	1	6	9	0
Block 1			Block 2			Block 3		

Here the Block1, Block2 and Block3 are the three blocks of the segmented arr2[ ] shown in Table IV. In Block4 there is putted an extra zero as a dummy data for remaining the blocks size same.

However, this paw search algorithm will never visit all of the blocks of unsorted data linearly, rather than it will go through the blocks of unsorted data all but as like as binary approach. But it won't fully follow the binary approach.

The designing resources and working procedures list of this algorithm is listed here-

- Unsorted Data Set
- Square Root Generating Function
- Ceiling Operator

- Generating Blocks
- Block Visiting Loop
- Inner Block Searching Approach
- Exit

C. Paw Search Algorithm

Assume that there is an array with the length of n of the nodes value of any given graph or other randomly generated unsorted data, now let's demonstrate our desired Paw Search Algorithm for finding out the target data from this given array of unsorted data. Here, we will go through the procedural steps of this Paw Search Algorithm. The procedures of this Paw Search Algorithm are shown below-

PAW (SEARCH ALGORITHM)

```
Divide the given array into x blocks where x = ceiling [ √ n]
Loop for Block
//For Block1:
Block[ ] =  x1[ ] = [ ]
        mergesort (Block[ ], L, U])
If(Block[last_element] ⩾ Target)
{
        If(x[last_element] == target)
        {Print "TARGET FOUND"}
          Exit
        Else
        {
        BinarySearch ( Block(x)[ ], 1,U)
                If (x[mid] == target)
                        {Print "TARGET FOUND"}
                            Exit
                Else
                        {Jump to the next Block}
        }
Else
        {Jump Block[ ] = x[last]}
Update Loop
Loop Exit
If (target == not found)
        {Print "Unsuccessful"}
Exit
```

```
    //Merge Sort Function
mergesort (Block[ ], l, U)
If U > l
Find the middle point to divide the array into two halves:
middle m = l+ (U-l)/2
    Call mergeSort for first half:
    Call mergeSort(arr, l, m)
    Call mergeSort for second half:
    Call mergeSort(arr, m+1, U)
    Merge the two halves sorted in step 2 and 3:
    Call merge(arr, l, m, U)
    //Binary Search Function
Binary Search (Block[ ], 1,U)
Input the Block[ ] array  of x elements I sorted form
LB = 0,UB = n; mid = int((LB+UB))/2)
Repeat step 4 and 5 while(LB <= UB and (A[mid] != item)
If (item < A[mid]) UB = mid-1
Else
LB = mid+1
mid = int((LB+UB)/2)
If (A[mid] == item)
Print "Item is found"
Else
Print "Item is not found"
Exit
```

The above mentioned procedures are the proposals of the Paw Search Algorithm, which also includes the Binary Search and Merge Sort Algorithm for completing its operation more efficiently. The further explanation of this algorithm is discussed later sections with proper examples.

D. Explanation and Implementation

Let's understand the block visiting procedures now, a graphical view is illustrated in Fig. 1 to show the working flow of the x blocks generated from the length of the array by making square root on it, and the length of each block is also x i.e., the block size and the block numbers are same.

Here x is the block number and l, k, m, p, y are also the sub number of x and they are the right mid, right-right mid, ……. , left mid, left-right mid, …….. , and gradually so on.

For implementing this algorithm let assume an array A[] with the length of n as shown in the following Table V.

TABLE V. GIVEN ARRAY WITH N ELEMENTS

ARRAY ELEMENTS										----					
INDEX NUMBER	0	1	2	3	4	5	6	7	8	----	n-4	n-3	n-2	n-1	n

So the initial step of this algorithm is to calculate the square root value x of the length of the given array A[].

$$x = \text{sqrt}(n) = \text{ceiling}(\sqrt{n})$$

It generates x number of blocks as Block(1), Block(2), Block(3), Block(4), ……..… , Block(x-3), Block(x-2), Block(x-1), Block(x) which are shown in Fig. 2.

The operational steps of these x number of blocks are also shown in Fig. 2. This algorithm follows the Left to Right approach. According to this Fig. 2, the first Block[1] is to go under the algorithmic operation firstly, secondly the last Block[x], then Block [mid], then Block [Right-mid], then Block [Right-mid], ……..,…..… , ……. , Block [Left-mid], Block [Left-Right-mid], …… , Block [Left-Left-mid], …… , …… , …….. etc.

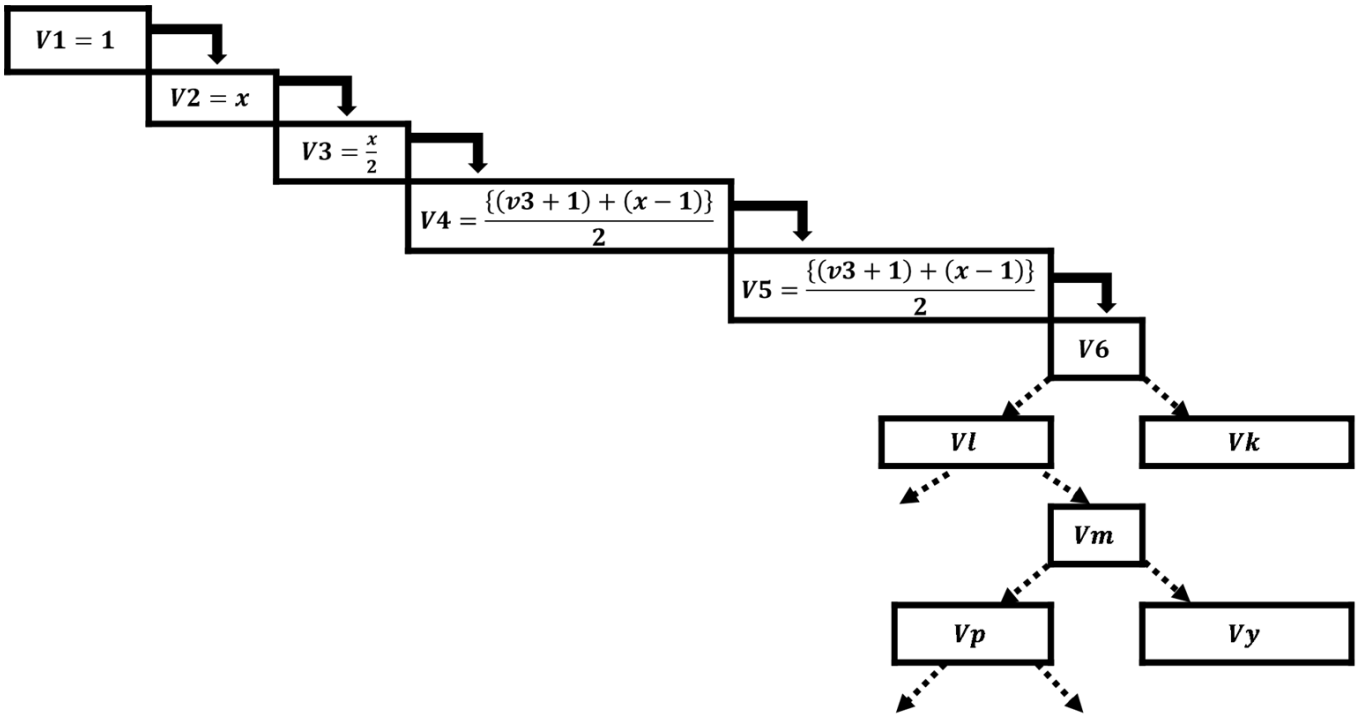

Fig. 1. Traversing procedure tree of the x blocks

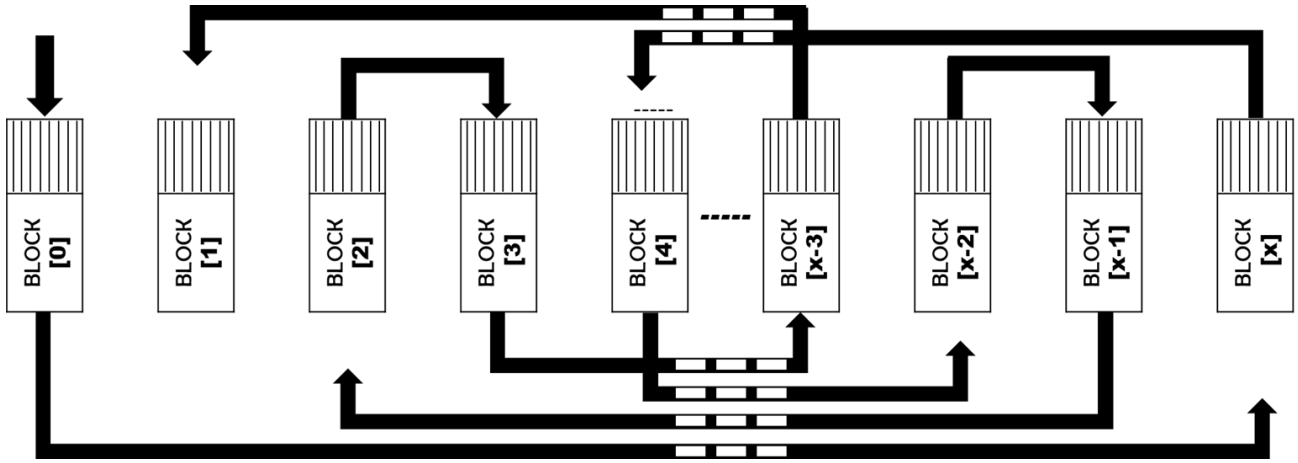

Fig. 2. Graphical view with paw search algorithm of the given array (table 5) with elements

For better understanding the implementation of this Paw Search Algorithm we will go through an example with proper explanation. Lets' assume another array B[] with the length of n, where n=9 as shown in Table VI. And let the SEARCH ITEM = Target = 5.

TABLE VI. ARRAY B[] WITH 9 ELEMENTS

4	6	1	9	3	5	8	7	2

Firstly, Lets' find out the number of Blocks:

$$x = \text{ceiling}\lceil \sqrt{n} \rceil = \text{ceiling}\lceil \sqrt{9} \rceil = 3$$

So, the blocks are shown in Table VII follows:

TABLE VII. BLOCKS OF ARRAY B[] WITH 9 ELEMENTS

4	6	1	9	3	5	8	7	2
Block 1			Block 2			Block 3		

For Block1:

```
//Sort the block using merge sort approach
mergesort (Block1[ ], 0, 2]) //input
{
Block(x)[]={3, 5, 9} //sorted
}
If((Block1(last)==6)⩾(Target==5))
{
If(x(last)==target)
{
Print "TARGET FOUND"
}
Else
{
//Binary Search
beg = lower_bound = 0
end = upper_bound = 2
mid =(beg + end)/2=(0 + 2)/2= 1
Block1[mid]= x[1]= 4
if (x[1]== target)
{
Print "TARGET FOUND"
Exit
}
Else
{
Jump to the next Block
}
}
Else
{
Jump x(last)
}
x++
Exit
```

For Block2:

```
//Sort the block using merge sort approach
mergesort (Block2[ ], 3, 5]) //input
{
Block(x)[]={3, 5, 9} //sorted
}
If((Block1(last)==9)⩾(Target==5))
{
If(x(last)==target)
{
Print "TARGET FOUND"
}
Else
{
//Binary Search
beg = lower_bound = 3
end = upper_bound = upper_bound+2=5
mid =(beg + end)/2=(3 + 2)/2= 4
Block1[mid]= x[4]= 5
if (x[4]== target)
{
Print "TARGET FOUND"
Exit
}
Else
{
Jump to the next Block
}
}
```

```
    Else
    {
    Jump x(last)
    }
    x++
Exit
```

So, here the target is found in Block2.

IV. Performance Analysis

Some fundamental key terms related to performance measurement of this proposed searching approach of paw search algorithm will be discussed through this section briefly. Basically, here we will cover the time and space complexity of this proposed searching approach and also cover a brief comparison of different existing searching approaches with this proposed searching approach:

A. Space Complexity

The Now lets' go through the time complexity phase of this Paw Search Algorithm. For calculating Time complexity of this algorithm we are to go through the divide and conquer approach of recursive method through traversing the x blocks generated by squaring root the length n of the given array of data.

Let the block1 of the length of x elements generated by squaring root the length n of the given array of data i.e., Block1(x) = {E(1),E(2),E(3),E(4),….,E(x-3),E(x-2),E(x-1),E(x)}.

First of all we are to calculate the space complexity of merge sort approach for sorting this sub array i.e., Block1 of x length. And we already know that the space complexity of this merge sort approach is O(x) that means that it needs of space for sorting this sub array data is as equal as the length of the sub array, here which is x. As the size of each and every block is same and at a time only one block will be sorted, so here the space complexity is O(x).

Now, let's calculate the space complexity of this paw search algorithm to find out the target value for this sub array x i.e., Block1

```
For Block1:
    //Space Complexity Calculation
    If(x(last)>=target)
    {
    If(x(last)==target)
    {
    Print "TARGET FOUND"
    }
    Else
    {
    Binary Search Algorithm
    if (x[i]== target)
    {
    Print "TARGET FOUND"
    i++
    Exit
    }
    Else
    {
    Jump to the next Block
    }
    }
    }
    Else
    {
    Jump x(last)
    }
Exit
```

So, there needs space as same as the length of the array x for performing this operation successfully. We can also see the graphical view (push and pop operation of stack method) of the recursive method of this Block1 x as below in Fig. 3.

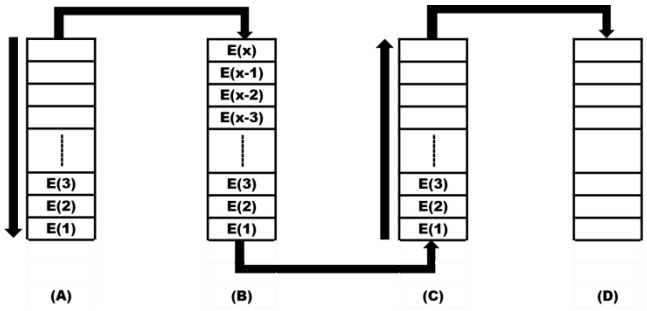

Fig. 3. Space complexity calculation

Where Fig. 3(A) shows the PUSHING of Block1's data into the STACK, Fig. 3(B) shows Block1 fully PUSHED into the STACK, Fig. 3(C) shows the POPPING of Block1's data from the STACK and Fig. 3(D) shows the Block1 which is fully popped from the STACK.

So, this Block1 needs same space as the length of this Block1 i.e., x. Similarly, all rest of the blocks need same space as their block size. So, here the space complexity is log (x).

Now, for finding out the target element from each block we will operate here the Binary Search approach. And we already know that the space complexity of the Binary Search approach is log (x).

So, the space complexity under the categories of worst case, average case and best case of this Paw Search Algorithm as below:

Paw_(Space_complexity) S(n)=O{(Space Complexity of the Block of given array+Space Complexity of Merge Sort Algorithm + Space Complexity of Binary Search Algorithm)}

⇒Paw_(Space_complexity) S(n)=O(log n + n + log n)

= O(n+2 log n)

=O(2 log n)

∴ Paw_(Space_complexity) S(n)=O(log n) [∵ 2 is the constant]

Hence the space complexity of this paw search algorithm is log n

B. Time Complexity

Now lets' go through the time complexity phase of this Paw Search Algorithm. For calculating Time complexity of this algorithm we are to go through the divide and conquer approach of recursive method through traversing the x blocks generated by squaring root the length n of the given array of data.

Let the Block1 of the length of x elements generated by squaring root the length n of the given array of data i.e., Block1(x) = {E(1), E(2), E(3), E(4), …. , E(x-3), E(x-2), E(x-1), E(x)}.

Firstly, we are to calculate an extra time for generating this x blocks by making square root of the given data array of the length of n.

Secondly, we are to consider the time complexity of merge sort approach for sorting this sub array i.e., Block1 of x length. And we already know that the time complexity of this merge sort approach is x log (x).

Now, let's calculate the space complexity of this paw search algorithm to find out the target value for this sub array x i.e., Block1

```
//For Block1:
If(x(last)>=target)
    If(x(last)==target)
    Print "TARGET FOUND"
    Exit
Else
    BinarySearch( Block[ ], L, U)
        if (x[i] == target)
            Print "TARGET FOUND"
            Exit
        Else
            Jump: Update Block
    Else
            Jump: Update Block
    Exit Block
Exit Loop
```

Thirdly, we are to find out the time complexity for the Binary Search approach for this Block1 i.e., x. And we already know that the time complexity of the Binary Search approach is log (x).

So, the time complexity under the best case category is √ n

And the time complexity under the category of worst case and average case of this Paw Search Algorithm as bellow:

Paw_(Time_Complexity) T(n)=O{(Time for making Square Root of the given arry+Time Complexity of Merge Sort Algorithm + Time Complexity of Binary Search Algorithm)*Number of Blocks}

⇒Paw_(Time_Complexity) T(n)=O{(1+n log n + log n)* √ n}

= O{ √ n(1+(n+1) log n)}

= O{ √ n (n log n)}

= O{n* √ n (log n)}

= O[√ (n^3)(log n)]

∴ Paw_(Time_Complexity) T(n) = O(√ (n^3) log n)

Hence the time complexity under the categories of worst case and average case of this paw search algorithm is √ (n^3) log n

C. Difference between Binary Search and Paw Search

The Paw Search Algorithm and the Binary Search Algorithm aren't same. There are several distinct difference between this two approaches. A difference chart between these two algorithms is shown in Table VIII follows:

TABLE VIII. DIFFERENCE BETWEEN PAW SEARCH AND BINARY SEARCH ALGORITHM

Paw Search Algorithm	Binary Search Algorithm
It begins its operation with the unsorted array of data.	It begins its operation with the sorted array of data.
It divides the given array of unsorted array of data of n length into x blocks by squaring root the length i.e., $\boldsymbol{x = sqrt(n)}$	It doesn't divide the array into blocks.
The input data is either unsorted or unsorted doesn't fact here.	The input data must be sorted here.
Time Complexity here in Worst Case is $\boldsymbol{\sqrt{n^3}\ log\ n}$	Time Complexity here in Worst Case is $\boldsymbol{log\ n}$
Time Complexity here in Average Case is $\boldsymbol{\sqrt{n^3}\ log\ n}$	Time Complexity here in Average Case is $\boldsymbol{log\ n}$
Time Complexity here in Best Case is $\boldsymbol{\sqrt{n}}$	Time Complexity here in Best Case is $\boldsymbol{log\ n}$
Space Complexity here in Worst Case is $\boldsymbol{log\ n}$	Space Complexity here in Worst Case is $\boldsymbol{log\ n}$
Space Complexity here in Average Case is $\boldsymbol{log\ n}$	Space Complexity here in Average Case is $\boldsymbol{log\ n}$

Time Complexity here in Best Case is $\boldsymbol{log\ n}$	Time Complexity here in Best Case is $\boldsymbol{log\ n}$
It is a combined searching system.	It is a unique searching system.

So, the Paw and Binary Searching technique isn't similar at all rather than it is quite different and comparatively more efficient than Binary Searching technique. It also should be mentioned that the Paw Search Algorithm solves the limitation of taking fully sorted array as an input of Binary Search Algorithm.

D. Difference between Jump Search and Paw Search

The Paw Search Algorithm and the Jump Search Algorithm aren't same. There are several distinct difference between this two approaches. A difference chart between these two algorithms is shown in Table IX follows:

TABLE IX. DIFFERENCE BETWEEN PAW SEARCH AND JUMP SEARCH ALGORITHM

Paw Search Algorithm	Jump Search Algorithm
It begins its operation with the unsorted array of data.	It begins its operation with the sorted array of data.
It divides the given array of unsorted array of data of n length into x blocks by squaring root the length i.e., $\boldsymbol{x = sqrt(n)}$	It also divides the given array of sorted array of data of n length into x blocks by squaring root the length i.e., $\boldsymbol{x = sqrt(n)}$
It doesn't follow the linear approach for traversing its blocks.	It follows the linear approach for traversing its blocks.
It is faster.	It is comparatively slower.
It doesn't travel the blocks sequentially.	It travels the blocks sequentially.
Under the block operation it operates here binary search approach as an inner approach.	Under the block operation it operates here linear search approach as an inner approach.
The input data is either unsorted or unsorted doesn't fact here.	The input data must be sorted here.
It is a combined searching system.	It is a unique searching system.

So, the Paw and Jump Searching technique isn't similar at all rather than it is quite different and comparatively more efficient than Jump Searching technique. It also should be mentioned that the Paw Search Algorithm solves the limitation of taking fully sorted array as an input of Jump Search Algorithm.

E. Time Complexity Comparisons with others Algorithms

A comparison list of time complexity of different search algorithms like linear search, binary search, hybrid search, interpolation search and paw search in different cases like worst case, average case and best case is shown in Table X as follows:

TABLE X. TIME COMPLEXITY COMPARISONS

	Best Case	Average Case	Worst Case
Linear Search	**O(1)**	**O($\boldsymbol{n}$)**	**O($\boldsymbol{n}$)**
Binary Search	**O(1)**	**O($\boldsymbol{log\ n}$)**	**O($\boldsymbol{log\ n}$)**
Hybrid Search	**O(1)**	**O($\boldsymbol{log\ 2n}$)**	**O($\boldsymbol{n}$)**
Interpolation Search	**O(1)**	**O($\boldsymbol{log(log\ N)}$)**	**O($\boldsymbol{n}$)**

F. Space Complexity Comparisons with Others Algorithms

A comparison list of space complexity of different search algorithms like linear search, binary search, hybrid search, interpolation search and paw search in different cases like worst case, average case and best case is shown in Table XI as follows:

TABLE XI. SPACE COMPLEXITY COMPARISONS

	Best Case	Average Case	Worst Case
Linear Search	**O(1)**	**O($\boldsymbol{n}$)**	**O($\boldsymbol{n}$)**
Binary Search	**O(1)**	**O($\boldsymbol{log\ n}$)**	**O($\boldsymbol{log\ n}$)**
Hybrid Search	**O(1)**	**O($\boldsymbol{log\ 2n}$)**	**O($\boldsymbol{n}$)**
Interpolation Search	**O(1)**	**O($\boldsymbol{log(log\ N)}$)**	**O($\boldsymbol{n}$)**
Paw Search	**O($\boldsymbol{log\ n}$)**	**O($\boldsymbol{log\ n}$)**	**O($\boldsymbol{log\ n}$)**

V. CONCLUSION WITH FUTURE WORK

By developing this long discussion of this research paper, I come to know that research is the fundamental weapon of this globalizing world i.e., IT world, and the large number of unsorted data is the heart of each and every research now-a-days. And managing this large number of unsorted data properly with proper searching technique is the core point of this paw search algorithm. The prime attraction of this research work is to develop a specific as well as more optimal formula of searching purposes from the unsorted list or array of data with the help of other searching and sorting techniques like merge sort and binary search. This Paw Search Algorithm shows the optimal way to generate a proper searching output taking an unsorted data list or array of data along with optimal time and space complexity, several comparisons of different searching approaches with this paw search algorithm are shown in Table VIII, IX, X and XI consecutively.

However, research is a continuous process. It will be upgraded with the demand of time day by day. There are also available a lot of future works here, some of them are listed below:

- Developing a more optimal logic/formula to optimize this algorithm
- Developing a Machine Learning Model to predict the desired block containing the desired data with Machine Learning Approach

ACKNOWLEDGMENT

It is a great pleasure for me to present this thesis paper titled as "Paw Search - A Searching Approach for Unsorted Data Combining with Binary Search and Merge Sort Algorithm".

I express heartiest thanks to friends and my well-wisher for their continuous inspiration and support, which led me to complete this research work.

Finally, I express my appreciation to my parents and other family members for their unconditional support as without their support and inspiration, it would be impossible for me to complete this research successfully.

REFERENCES

[1] Sultana, N., Paira, S., Chandra, S., & Alam, S. S. (2017, February). A brief study and analysis of different searching algorithms. In 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT) (pp. 1-4). IEEE.

[2] Roopa, K., & Reshma, J. (2018). A comparative study of sorting and searching algorithms. International Research Journal of Engineering and Technology (IRJET).

[3] Das, P., & Khilar, P. M. (2013). A randomized searching algorithm and its performance analysis with binary search and linear search algorithms. International Journal of Computer Science & Applications (TIJCSA), 1(11).

[4] Pathak, A. (2015). Analysis and Comparative Study of Searching Techniques. International Journal of Engineering Sciences & Research Technology, 4(3), 235-237.

[5] Subbarayudu, B., Gayatri, L. L., Nidhi, P. S., Ramesh, P., Reddy, R. G., & Reddy, C. K. K. (2017). Comparative analysis on sorting and searching algorithms. International Journal of Civil Engineering and Technology (IJCIET), 8(8), 955-978.

[6] Rahim, R., Nurarif, S., Ramadhan, M., Aisyah, S., & Purba, W. (2017, December). Comparison searching process of linear, binary and interpolation algorithm. In Journal of Physics: Conference Series (Vol. 930, No. 1, p. 012007). IOP Publishing.

[7] Data Stuctures, Seymour Lipschutz and G A Vijayalakshmi Pai, SCHAUMS'S OUTLINES, 2013-2014.

[8] Jacob, A. E., Ashodariya, N., & Dhongade, A. (2017, August). Hybrid search algorithm: Combined linear and binary search algorithm. In 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS) (pp. 1543-1547). IEEE.

[9] Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Third Edition, 2017-2018

[10] Harman, M., & McMinn, P. (2009). A theoretical and empirical study of search-based testing: Local, global, and hybrid search. IEEE Transactions on Software Engineering, 36(2), 226-247.

[11] Shneiderman, B. (1978). Jump searching: A fast sequential search technique. Communications of the ACM, 21(10), 831-834.

[12] Mahboob, T., Akhtar, F., Asif, M., Siddique, N., & Sikandar, B. (2015). Survey and Analysis of Searching Algorithms. International Journal of Computer Science Issues (IJCSI), 12(3), 169.

[13] Boyer, R. S., & Moore, J. S. (1977). A fast string searching algorithm. Communications of the ACM, 20(10), 762-772.

[14] Bentley, J. L., & Sedgewick, R. (1997, January). Fast algorithms for sorting and searching strings. In Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms (pp. 360-369).

[15] Mehlhorn, K., Sanders, P., & Sanders, P. (2008). Algorithms and data structures: The basic toolbox (Vol. 55, p. 56). Berlin: Springer.

[16] Tuparov, G., Tuparova, D., & Jordanov, V. (2014). Teaching sorting and searching algorithms through simulation-based learning objects in an introductory programming course. Procedia-Social and Behavioral Sciences, 116, 2962-2966.

[17] Wang, A. (2003, October). An industrial strength audio search algorithm. In Ismir (Vol. 2003, pp. 7-13).

[18] Zabinsky, Z. B. (2009). Random search algorithms. Department of Industrial and Systems Engineering, University of Washington, USA.

[19] Shareef, H., Ibrahim, A. A., & Mutlag, A. H. (2015). Lightning search algorithm. Applied Soft Computing, 36, 315-333.

[20] Bentley, J. L., & Yao, A. C. C. (1976). An almost optimal algorithm for unbounded searching. Information processing letters, 5(SLAC-PUB-1679).