Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 04, April-2021

Decomposed Algorithm for Reducing Time Complexity in Binary Search

Kunal [1], Tushar [2], Gargi Chakraborty [3]
[1]Department of Computer Science, Calcutta Institute of Engineering and Management
[2]Department of Computer Science, Amity University Chhattisgarh
[3]Department of Computer Science, Calcutta Institute of Engineering and Management

***Abstract*: Searching is one of the principal tasks in computer science. Binary Search is one of the most common and efficient algorithms used. Binary Search targets the middle element and checks for the target key in the list. The worst-case Time Complexity of Binary Search is O(log(n)) where n less length of the search list. In this paper, we have proposed a fast and efficient approach to binary Search by decomposing the main search list into multiple search lists. The Time Complexity for the proposed algorithm is O(log(k)) where k < n and n is the length of the sorted list and k is the length of the sub-list.**

Keywords:- Binary search, algorithm efficiency, sorting and searching, time complexity, decomposed algorithm.

I. INTRODUCTION:-

Binary search is one of the well-known searching techniques used to search the key value from the sorted set of data. The algorithms for Binary search can be found in many classic Textbooks such as Algorithms in C [1] and Introduction to Algorithms, Third Edition [2]. The algorithm of classical binary search constantly targets and checks the mid element of the sorted list. The Binary search can be implemented both iteratively and recursively.

The Proposed algorithm works very similarly to the Binary Search. It searches the list using an element as its basis and comparing whether or not present in the list in a given iteration. However, unlike the Binary search the presented algorithm instead of searching the whole list it searches in the decomposed list which is always smaller than the original list.

The time complexity in the worst-case scenario in Linear search, Binary search, and the decomposed binary search are O(n), O(log(n)), and O(log(K)) respectively where K< n and n is the length of the sorted list on which the searches are implemented and K is the length of the decomposed searching list.

(A) POPULAR OF SEARCHING APPROACH

There are multiple techniques for searching data. Each searching techniques have their pros and cons. The most commonly used searching algorithms used are:- Linear Search, Binary Search, Binary Search Tree, and more. We will discuss some of these popular searching algorithms in this section.

1. **Linear Search**:-

Linear Search is a widely used searching algorithm in computer science. In Linear search, each element is checked to start from zeroeth index to the end of the list. The Time Complexity of Linear Search is O(n).

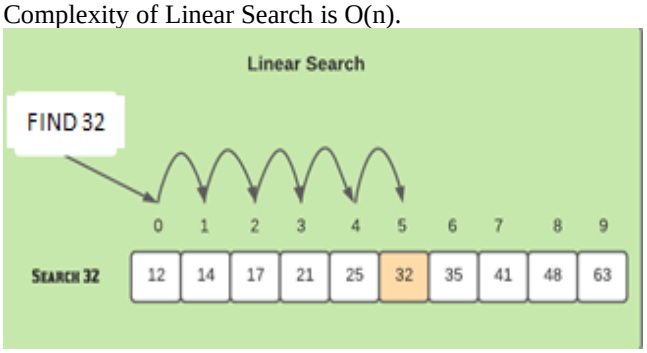

Fig. 1

2. **Binary Search**: -

In classical Binary Search, the complete search array is repeatedly divided into halves. The key is searched in the left half if the middle element is smaller than the key else key is searched in the right half. The process of division of left and right half continues till the key is found as per the condition of the middle element and search key.

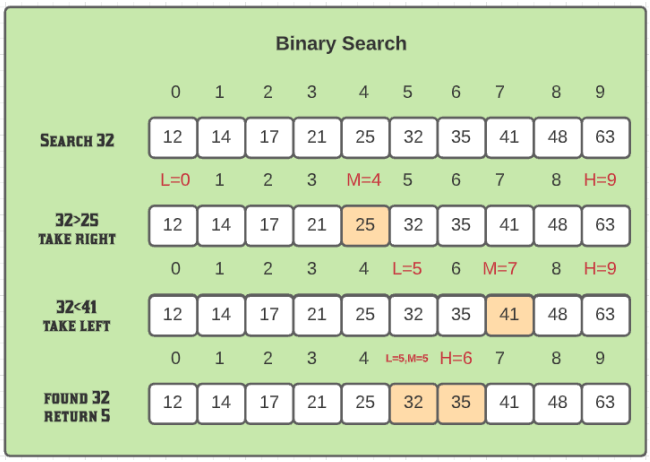

Fig. 2

II. LITERATURE REVIEW

1. Donald Knuth proposed Uniform Binary Search. The proposed algorithm stores the index of the middle element and changes in the middle element between the next two iterations. The main disadvantage of this algorithm is that the proposed methodology is only quick in only in those cases where it is unfit to calculate the middle point. [1]
2. Bently et al. proposed Exponential Search in 1976. They proposed the new searching algorithm by

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 04, April-2021

targeting the first segment with an index which is the power of two and greater than the target value and then shifts to binary Search. The main disadvantage of this proposed algorithm is that it is only fast than binary search when the target value is near the starting index of the array. [2]

3. Chazelle et al. proposed Fractional Cascading in 1986. The authors proposed an efficient searching technique. The proposed algorithm splits the sorted array into multiple sorted arrays and performs the searching operation in each sorted array. The main disadvantage of this proposed algorithm is that it needs some extra space which increases the space complexity and so increases the expense of searching.[3]
4. Praveen Kumar proposed Quadratic Search in 2013. The proposed algorithm is the advancement of binary search. The algorithm targets $1/4^{th}$, middle, and $3/4^{th}$ elements of the sorted array and searches for the target element, if the target element is there in the target index then it will return the index of the target element or else continue to search by reducing search space as per the condition. The biggest disadvantage of this algorithm is that a lot of condition is to be checked to reduce the array to find the target element and also have to maintain more pointers.[4]
5. Nitin Arora et al. proposed Two way linear Search in 2014. The Authors proposed a searching algorithm in which they used two pointers, first at the beginning and the other at the end of the search array. The first pointer is incremented while the second pointer is decremented in each iteration to find the target element in the search array. The major disadvantage of this proposed algorithm is maintaining multiple pointers in linear search.[5]
6. Alwin Francis et al. proposed "Modulo Ten Search – An Alternate to Linear Search" in 2011. The authors proposed a searching algorithm for finding the target element in the search array by getting the modulus Ten of the number and the list will be divided into ten lists based on modulo ten of the number.[6]
7. Karthick Sowndarajan proposed "Odd-Even Based Binary Search" in 2016. The authors proposed a searching algorithm by sorting the list into the odd part and even part and then implement the Binary Search technique based on the type of search element. If the search element is odd then the binary search is only executed in the odd part of the array and the same for the even key element, binary search will be executed in the even part of the array. The proposed algorithm works efficiently only if the array is provided in sorted on an odd-even basis.[7]
8. Ankit R. Chadha et al. proposed the "Modified Binary Search Algorithm" in 2014. The proposed algorithm checks the presence of the input element with the middle element of the search array in each iteration. This proposed algorithm optimizes the time complexity at the worst case of the classical binary search algorithm by comparing the input element with the first and last of the data set. [8]
9. Similarly, there are much research works on searching techniques for different purposes e.g Fibonacci Search[9], Quantum binary Search[10], ASH Search[11]. Each algorithm has its pros and cons.

III. PROPOSED APPROACH

In the proposed **Decomposed binary search** approach first, we derive the binary equivalent of the length of the main list. Each of the 1's in the binary representation is converted to the power of two corresponding to their positions where the index starts from 0 to K-1 and only the values greater than or equal to one is stored in the list. The reverse of the new list created is the **decomposed list**. The main idea behind the creation of the decomposed list is to reduce the search time by reducing the search space. So the Decomposed List consists of the distribution of indices from the main list where the key could be found out. So each element in the Decomposed List will be represented as the index of the main list whose value can be equal to or less than the key. If the value of the element in the Main List referenced by the Decomposed List is equal to Key then the key is found or else if the value is less than the Key then the element referenced from the Decomposed list in the Main list becomes the High Value and the value of One index less than the referenced index in the decomposed list becomes the Low value. And if the key is less than the first value referenced from the first Index of the Decomposed list as the index of the main list then it becomes the High value and Zero becomes the Low value. Now, we have the High and the Low value which we will pass to the binary search function which in turn will reduce the time complexity by only searching the sublist of the main list. So by comparing the Decomposed list value as an index of the main list from the key we can say directly say where is the probability of the element found which is way faster than searching the whole list.

IV. PROPOSED ALGORITHM

DecomposedBinarySearch(List[], N, B, B_N, Search)
Purpose : To get the subset of the given array for which the possibility of occurrence of the element is highest
Input : List[0…N-1] – the list of sorted elements.
: N – the number of elements in the list.
: B – Binary Representation of the length of the list.
: B_N – No. of ones in the binary representation.
: Search – the **key** to be searched.
: RevDec[0…K-1] – Reverse of Decomposed List
Output : Decomp_list[0…K-1] – the list of the decomposed set elements in the power of two of the length of the main list.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 04, April-2021

: High – the Upper Bound of the List for the Binary Search Function.
: Low – the Lower Bound of the List for the Binary Search Function.
: K – the number of decomposed elements in the Decomp_list.

Step 1 : [Check whether the total number of one's in the binary representation of the length of the array is greater than one]
If B_N <= 1 then
Print "Decomposed Binary Search not possible."
Goto Step 10
Endif

Step 2 : [Initialize the variables]
Count ← -1
Length ← length of B
Count_decomp_list ← 0

Step 3 : Repeat Step 4 for i←0 to Length-1

Step 4 : [Creating a list for each '1' present in the binary representation with its equivalent base 10 value]
If B % 10 = 1 then
Count ← Count +1
Decomp_list[i] ← 2^Count
B ← B / 10
Count_decomp_list ← Count_decomp_list + 1
Else
Count ← Count + 1
B ← B / 10
[End of Step 4]

Step 5 : Count ← 0

Step 6 : Repeat Step 7 for i ← 0 to Length -1 for reversed Decomposed List

Step 7 : Count ← Count + RevDec[i]
RevDec[i] ← Count

Step 8 : Repeat Step 9 for i←0 to Count_decomp_list

Step 9 : [Slice the Main array to the range where the element is likely to be found]
If **List[RevDec[i]]** = search then
High ← RevDec[i]
Low ← RevDec[i]
Endif
Goto Step 9
Else if search < **List[RevDec[0]]** then
High ← RevDec[0]-1
Low ← 0
Else if search > **List[RevDec[i]]** and search < **List[RevDec[i+1]]** then
High ← RevDec[i+1]-1
Low ← RevDec[i]+1
Endif
[End of Step 9 loop]

Step 10 : Exit

Once the **DecomposedBinarySearch** algorithm is executed we will get the High and the Low value for the Main list which is then passed to the classical Binary search Algorithm reducing the space and time complexity of the search.

V. IMPLEMENTATION

Consider a list of 13 elements in sorted order arranged in ascending order.

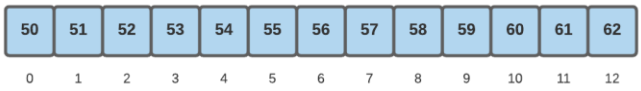

Fig. 4

The binary representation of the length of the list is 1101. Here the position of maxterm is at 3 so we convert it to the power of 2 which is 2^3 = 8 which means the first subarray will contain 8 elements whose index will start from 0 to 8-1 = 7. The next position of one is at 2nd position the power of two will be 2^2 = 4 which means the second subarray will contain 4 elements with minterm 7+1=8 and maxterm 7+4 = 11. The last one is at the minterm which is at position 0 so we have 2^0 = 1 term in the last sub-array having minterm as 11+1=12 and max term as 11+1=12.

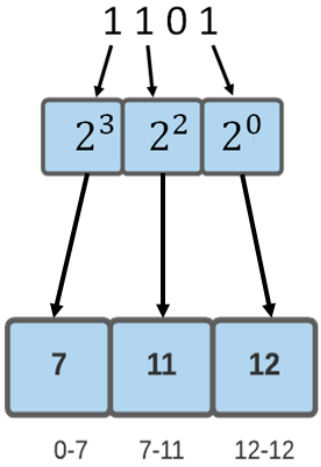

Fig. 3

Now, let us say we have to find Key=56 from the main list. Our algorithm will first compare the element at index 7 in the main List which we can see is **Main_list[**Decomp_list[0] = 7]**=57>Key hence the High = 7 and Low =0 is returned which is then passed to the binary_search function as an argument. Where Decomp_list[0] is the first element of the decomposed list formed.[**Note: In the algorithm, we have used RevDec list to denote the Decom_list to ease the calculation]

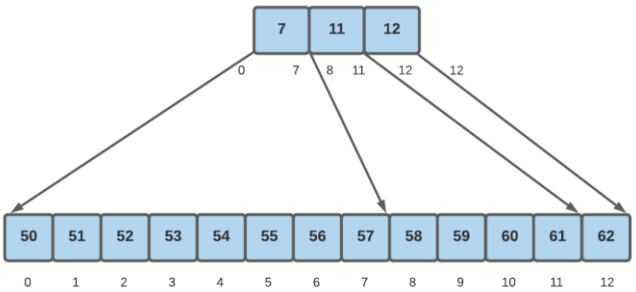

Fig. 5

VI. EXPERIMENTAL ANALYSIS

For experimental analysis, the proposed decomposed binary search algorithm is codded in Python 3.9.4 with configuration i5 preprocessor (2.50GHz Intel processor with 8MB cache memory) and 8GB RAM(DDR4 2933Mhz). We

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 04, April-2021

had tested our algorithm with datasets with more than 1000 pages to find different search keys and compared the steps encountered in the classical binary search to our proposed decomposed binary search.

Key = 1851		
Steps	**Binary Search**	**Decomposed Search**
initial	low=0,mid=511,high=1022	low=513,mid=640,high=767
1	low=512,mid=767,high=1022	low=513,mid=576,high=639
2	low=512,mid=639,high=766	low=577,mid=608,high=639
3	low=512,mid=575,high=638	low=609,mid=624,high=639
4	low=576,mid=607,high=638	low=625,mid=632,high=1022
5	low=608,mid=623,high=638	low=633,mid=636,high=639
6	low=624,mid=631,high=638	low=637,mid=638,high=639
7	low=632,mid=635,high=638	
8	low=636,mid=637,high=638	
9	low=638,mid=638,high=638	
	638	638

Fig. 6

Here we took a random dataset of 1023 pages and applied classical binary search and our proposed decomposed binary search to find 1851 in this dataset. We found that Binary Search took 9 steps and our binary search took only 6 steps to find the index for the same given key after the decomposed algorithm is implemented on the main list.

Key = 2236		
Steps	**Binary Search**	**Decomposed Search**
initial	low=0,mid=511,high=1022	low=769,mid=832,high=895
1	low=512,mid=767,high=1022	low=769,mid=800,high=831
2	low=768,mid=895,high=1022	low=769,mid=784,high=799
3	low=768,mid=831,high=894	low=769,mid=776,high=783
4	low=768,mid=799,high=830	low=769,mid=772,high=775
5	low=768,mid=783,high=798	low=769,mid=770,high=771
6	low=768,mid=775,high=782	low=769,mid=769,high=769
7	low=768,mid=771,high=774	
8	low=768,mid=769,high=770	
9	low=770,mid=770,high=770	
	Element Not Found	Element Not Found

Fig. 7

Here again, we took the same dataset but this time we applied both the approach with different key i.e 2236, and again we found that our proposed algorithm is efficient over the classical binary search approach. Classical Binary search takes 9 steps to identify that the search key is not present in the given dataset where our proposed approach identifies the same in 6 steps.

Key = 3008		
Steps	**Binary Search**	**Decomposed Search**
initial	low=0,mid=511,high=1022	low=1022,mid=1022,high=1022
1	low=512,mid=767,high=1022	
2	low=768,mid=895,high=1022	
3	low=896,mid=959,high=1022	
4	low=960,mid=991,high=1022	
5	low=992,mid=1007,high=1022	
6	low=1008,mid=1015,high=1022	
7	low=1016,mid=1019,high=1022	
8	low=1020,mid=1021,high=1022	
9	low=1022,mid=1022,high=1022	
	1022	1022

Fig. 8

Now, again we took another key 3008 in the same dataset and found that the classical binary search approach took 9 steps to find the index of the search key where our proposed approach took only 1 step to find the index of the search key when the decomposed algorithm is implemented.

VII. CONCLUSION

After applying multiple datasets and different keys in the classical binary search and our proposed searching algorithm, we found that the algorithm we proposed to find the search element in the datasets is more efficient and faster. The Time complexity of the classical binary search is O(log n) where the Time complexity of our algorithm is O(log(k)) where k < n and n is the length of the sorted list and k is the length of the sub-list.

VII. REFERENCES

[1] Robert Sedgewick. 2002. Algorithms in C (3rd. ed.). Addison-Wesley Longman Publishing Co., Inc., USA.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

[3] D.E. Knuth, “The Art of Computer Programming”, Vol. 3: Sorting and Searching, Addison Wesley, 1973

[4] Jon Louis Bentley, Andrew Chi-Chih Yao, An almost optimal algorithm for unbounded searching, Information Processing Letters, Volume 5, Issue 3,1976, Pages 82-87, ISSN 0020-0190,https://doi.org/10.1016/0020-0190(76)90071-5.

[5] B. Chazelle. L. J. Guibas, “Fractional Cascading: A data structure technique”, Algorithm, Vol 1, Issue 1-4, pp. 113-162, November 1986

[6] Parveen Kumar, " Quadratic Search: A New and Fast searching Algorithm (An extension of classical Binary search strategy)".International Journal of Computer Applications 65, no.14 (2013): 43-46.

[7] Arora, Nitin & Bhasin, Garima & Sharma, Neha. (2014). Two-way Linear Search Algorithm. International Journal of Computer Applications. 107. 6-8. 10.5120/19137-9622.

[8] A. Francis and R. Ramachandran, "Modulo Ten Search- An Alternative to Linear Search," 2011 International Conference on Process Automation, Control and Computing, Coimbatore, India, 2011, pp. 1-4, DOI: 10.1109/PACC.2011.5979034.

[9] Karthick Sowndarajan (2016). Odd-Even-based Binary Search. International Journal of Computer Engineering & Technology (IJCET). 7. 40-55.

[10] Ankit R Chadha & Rishikesh Misal & Tanaya Mokashi, (2014). Modified Binary Search Algorithm. International Journal of Applied Information Systems. 7. 10.5120/ijais14-451131.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 04, April-2021

[11] David E. Ferguson, “Fibonaccian searching”, Communications of ACM, Vol. 3, Issue 12, NY, USA, DOI: 10.1145/367487.367496, 1960.
[12] Vladimir Korepin, Ying Xu, “Binary Quantum Search”, International Journal of Modern Physics B, Vol. 21, ISSN 5187-5205, DOI: 10.1117/12.717282, May 2007.
[13] Ashar Mehmood. ASH Search: Binary Search Optimization. International Journal of Computer Applications 178(15):10-17, DOI: 10.5120/ijca2019918788, May 2019.