To check whether an element X is present in an array or not using Divide and Conquer Algorithm

Indian Institute of Information Technology, Allahabad

Aditya Aggarwal
iit2019210@iiita.ac.in

Divy Agrawal
iit2019211@iiita.ac.in

Aman Rubey
iit2019212@iiita.ac.in

***Abstract*—In this paper we have devised an algorithm to find an element X in an array of given elements solely using the concept of divide and conquer. Also in the later part of the paper we have discussed the space and time complexity of the devised algorithm.**

***Index Terms*—Algorithm, Divide and Conquer, Searching, complexity**

I. Introduction

Divide and conquer algorithm is an efficient algorithm that decompose a given problem into small sub-problems recursively until it is relatively easier to solve those small sub-problems. Then the solution to the original problem is computed by combining the solutions to the sub-problems. Implementation of this algorithm could be seen in sorting algorithms such as merge-sort and quick-sort.

II. Algorithm Design

We have taken elements (Integer) as input from user and stored them in a vector. We will be using this vector to check whether an element X is present in this vector.

An important concept used in this algorithm is that if vector contains a singleton element only and if this element is equal to the required element X the answer would be true else it would be false.

- Here the main problem is to check whether an element is present in a vector or not.
- This main problem could be divided into two sub-problems by dividing this initial vector (parent vector) into two vectors (child vectors) of approximately same size and checking whether the required element is present in these two vectors formed(child vectors).
- We will divide these new sub-vectors formed until the new vectors comprise of singleton element only.
- As discussed earlier in this section, answer for these vectors comprising of single elements will be either true or false depending on whether this single element is equal to required element or not.
- Suppose, we know that whether the required element X is present in two vectors or not. Then the answer to the problem-vector comprising of the elements of both the vectors simply depends upon the answer for the two child vectors.
  - If element is present in either or in both of the child vectors then the element will be present in the parent vector comprising of these two child vectors.
  - If element is not present in both of the child vectors then the element will not be present in the parent vector comprising of these two child vectors.
- Similarly, we will be checking whether the required element is present in all of the sub-vectors (child vectors) previously created or not. And, this then would be used to compute whether element is present in the parent vector or not.

These 6 steps of the algorithm would become more clear by the following example:

- Consider a vector V.

  $$V \equiv \{34, 4, 56, 999\}$$

  We want to check whether the element X=999 is present or not in this vector using divide and conquer.
- We will now divide this vector V into two sub-vectors V_1 and V_2 which is equivalent to dividing problem-1 into two sub-problems problem-1a and problem-1b.

  $$V_1 \equiv \{34, 4\}$$
  $$V_2 \equiv \{56, 999\}$$

  Now we will check whether X is present in these sub-problems formed or not.
- We will now divide vector V_1 into two sub-vectors V_{11} and V_{12} which is equivalent to dividing problem-1a further into two sub-problems problem-1a1 and problem-1a2.

  $$V_{11} \equiv \{34\}$$
  $$V_{12} \equiv \{4\}$$

  Similarly, we will now divide vector V_2 into two sub-vectors V_{21} and V_{22} which is equivalent to dividing problem-1b further into two sub-problems problem-1b1 and problem-1b2.

  $$V_{11} \equiv \{56\}$$
  $$V_{12} \equiv \{999\}$$

  Now we will check whether X is present in these sub-problems formed or not.
- As discussed earlier, solutions to the sub-problems problem-1a1, problem-1a2, problem-1b1, problem-1b2 will be either true or false depending upon whether these

singleton element is equal to required element or not.
Therefore, solution to these sub-problems will be:

1) Problem-1a1 is false as 34 is not equal to 999.
2) Problem-1a2 is false as 4 is not equal to 999.
3) Problem-1b1 is false as 56 is not equal to 999.
4) Problem-1b2 is true as 999 is equal to 999.

This now will be used to compute solution for the sub-problems problem-1a and problem-1b.

- Solution to the sub-problem problem-1a will be false as solution to the sub-problems problem-1a1 and problem-1a2 is false. This means that 999 is not present in the vector V_1 which is true.
  Solution to the sub-problem problem-1b will be true as solution to the sub-problem problem-1b2 is true despite of solution to problem-1b1 is false. This means that 999 is present in the vector V_2 which is true.
  Therefore, solution to these sub-problems will be:
  1) Problem-1a is false.
  2) Problem-1b is true.

  This now will be used to compute solution for the main-problem, problem-1.
- Solution to the main-problem problem-1 will be true as solution to the sub-problem problem-1b is true despite of solution to problem-1a is false. This means that 999 is present in the vector V which is true.
  Thus, the required element 999 is present in the vector V (main-problem).

These above 6 steps of the example clearly show how we have used divide and conquer algorithm i.e., we firstly divided the main-problem into small sub-problems until they we were able to solve them directly. Then we computed the answer for these sub-problems and used these answers to compute answer for the main-problem.

III. Pseudo Code

Algorithm using Divide and Conquer

Global Variables:
vector$\langle int \rangle v$;
$int\ n$;
$int\ x$;

function $main()$
$Get\ n$;
$Get\ n\ elements\ and\ store\ them\ in\ a\ vector$;
$solution \leftarrow solve(0, n-1)$;
$print\ solution$;
$return\ 0$;

function $solve(low, high)$
$if\ low == high\ then$
$if\ v[low] == x\ then$
$return\ true$;
$else$
$return\ false$
$mid \leftarrow low + (high - low)/2$
$return\ (solve(low,\ mid)\ OR\ solve(mid+1,\ high))$

IV. Complexity Analysis

In the next two sub-sections we analyse the time and space complexity of algorithm we devised for the given problem.

A. Time-Complexity

Let N denote the number of elements in the vector in which we have to check whether the required element is present or not.

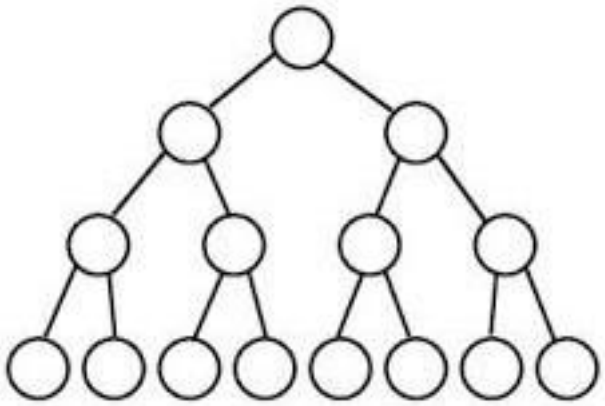

Fig. 1. Problem Dividing in Divide and Conquer

Taking the reference of the above tree, where root node denotes the main-problem and rest nodes denote the sub-problems formed from the main-problem. Suppose if the time-complexity of the problem consisting of N elements is T(N) then the time-complexity of sub-problems of this problem will be T(N/2) approximately and let time-complexity for executing rest statements be C_1.
Then,

$$T(N) = 2T(N/2) + C_1 \tag{1}$$

which is equivalent to,

$$T(N) = 2T(N/2) + \Theta(1) \tag{2}$$

Therefore, upon applying master's theorem we get,

$$T(N) = O(N) \tag{3}$$

B. Auxiliary Space-Complexity

Let N denote the number of elements in the vector in which we have to check whether the required element is present or not.

The function solve(low,high) is being called approximately 2N-1 times that is O(N) times and each time when it is called 3 variables of size 4 bytes is created. Therefore, Auxiliary Space-Complexity will be,

$$T(N) = O(N) \tag{4}$$

V. Theortical complexity analysis

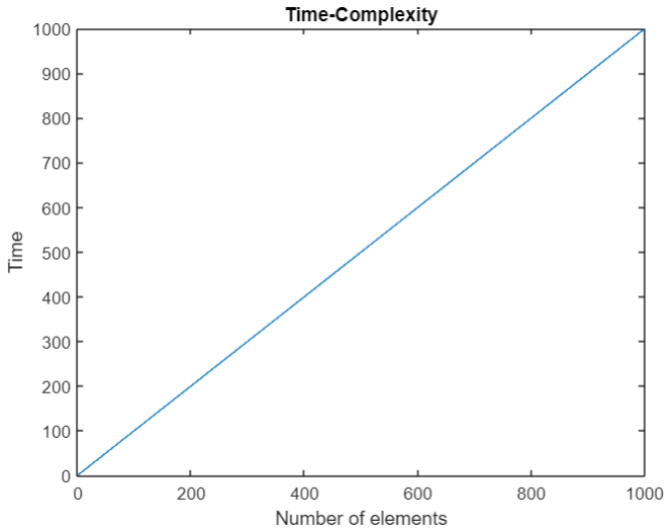

Fig. 2. Time–Complexity Graph (O(N))

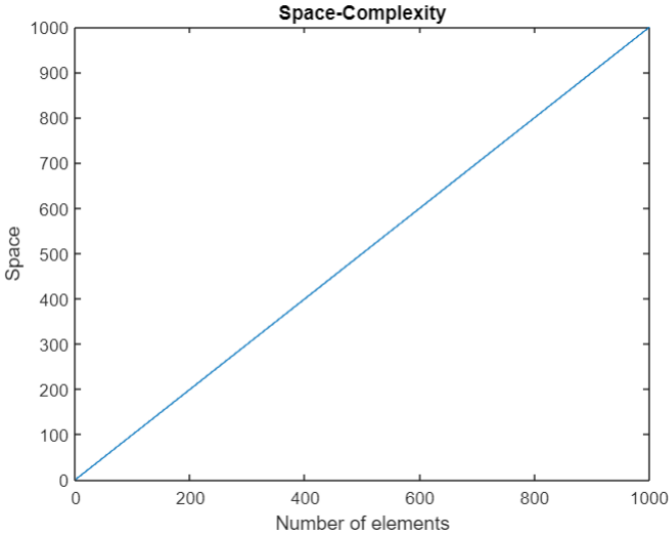

Fig. 3. Auxiliary Space–Complexity Graph (O(N))

VI. Advantages Against Linear uni-processor Search

1) In the computer systems containing a single processor the execution of processes tends to be slow as a single process hogs the CPU till it's cycle is complete or execution of the process stops. So, OS generally implements multi-threading by itself at the software level without kernel knowing about it.

So in a Linear Uni-processor Search, the process of checking whether the element is present or not will be taking several CPU cycles before the we finally get the answer. In this the whole process we be running on a single main thread because of which it will become slow as the number of elements to search from increases.

But in Divide and Conquer Algorithm the main-task is divided into sub-tasks which are completely independent of each other. So, these independent sub-tasks could be executed using threads which are basically light-weight processes needing relatively less context switching time then that required for context switching of a process. These, solution to the sub-problems executed using threads could be used to compute the solution to other sub-problems depending upon these on a separate thread.

Because of this, when the number of inputs becomes very large Divide and Conquer Algorithm executes faster than Linear Uni-processor Search despite of the fact that both the algorithms have same time complexity. When the number of inputs are low the time of execution is same for both.

2) Another advantage of Divide and Conquer Algorithm is the use of memory caches efficiently as when the sub problems becomes very simple, they can be solved within the cache, without having to access the slower main memory(Cache is second fastest accessible memory after Registers), which saves a lot of time and makes the algorithm more efficient. And in some cases, it can even produce more precise outcomes in computations with rounded arithmetic than iterative methods would.

VII. Conclusion

Therefore, using the concept of divide and conquer we have devised an algorithm to check whether an element is present in a vector of elements or not. The time complexity of the above devised algorithm is O(N) where as auxiliary space complexity is O(N) where N is number of elements given by the user.

References

[1] Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest
[2] Algorithm Design by J Kleinberg and E Tardos
[3] Abraham Silberschatz Peter B. Galvin and Greg Gagne, Operating System Concepts, Wiley 8th Edition, 2008
[4] Divide and Conquer
[5] Complexity Analysis
[6] Divide and Conquer Paradigm