Comparative study of Divide and Conquer and Brute Force search algorithm

Upasana Ghosh

Roll no: UE143110
U.I.E.T, Panjab University
Email: ghoshupasana05@gmail.com

AIM

This paper presents comparative study of divide and conquers algorithm and brute force search algorithm to find maximum element and minimum element in an array. It is observed that the time required to execute and run divide and conquer algorithm is more than the time required to execute and run brute-force search algorithm. This result is contrary to the believe that divide and conquer algorithm always give the best result than that of brute force algorithm.

APPROACH

Maximum-minimum search algorithms are extensively used in computer science, especially in data analysis. They are further used in decision theory, game theory, statistics and philosophy for minimizing the possible loss for a worst case (maximum loss) scenario and maximizing the profit. It has also been extended to more complex games and to general decision-making in the presence of uncertainty.

To find maximum and minimum elements in an unsorted array, different array sizes are taken and they are randomly filled with integers values generated by random function. Maximum and minimum element of an array is searched using divide and conquer and brute force search method.

In divide and conquer algorithm, array is recursively broken down into two sub-arrays (divide), until array contain at most two elements. If array contain two elements then those elements are compared and the larger and smaller elements are returned. If array contain only one element then that element is treated as both largest and smallest and is returned to the calling function. These largest and the smallest elements are further compared to other sub-problems and then largest and the smallest element among those are returned to the original problem.

Another approach to find maximum and minimum elements that is presented in this paper is brute-force search. Brute-force search is a very general problem-solving technique that consists of systematically checking whether each candidate satisfies the problem's statement. In this, two variable called maximum and minimum are initialized with the first element of the array. These maximum and minimum variables are then compared with other elements of the array. The value of the variables maximum and minimum will be changed only when an array element is greater than that of maximum variable and an array element is smaller than that of minimum variable else that element will be ignored. Each array element is compared until the array is exhausted and the value of maximum and minimum is returned.

The time taken to search is calculated for both methods and a graph is plotted.

EXPERIMENT

In this experiment, an array of size of 500 is taken and is populated randomly with integer values. Divide and conquer and brute-force methods are invoked one by one to find maximum and minimum element in an array. The time taken is calculated. The array size is then increased by 300 and again time taken by both these methods is calculated. For every array size, the array elements are filled 5,00,000 times, searching operation is called 5,00,000 times and time is calculated 5,00,000 times and average time is calculated. This analysis is done with the following bounds:
Initial array size: 500
Step size: 300
Final array size: 10000

Platform used for compilation and execution:
Operating system: Ubuntu 15.10
RAM: 4GB
IDE: Dev-C++ version 4.9.9.2
Compiler: GCC 3.4.2
Graph plotted: Octave-3.6.4.

I. ALGORITHM

Algorithm used for searching minimum and maximum by brute force method:
Declaration:

a: array consisting of integer values.
n: total array element

max: maximum element
min: minimum element
i: index of array a

```
Algorithm brute_force(a, n, max, min)
{
    max=min=a[1];
    for(i=1 to n)
     {
        if(a[i]>max)
        max=a[i];
            else if(a[i]<min)
            min=a[i];
      }
}
```

Algorithm used for searching minimum and maximum by divide and conquer method:

Initialization:
a: array consisting of integer values.
low: The lowest index of array .
high: The highest index of array
max: maximum element
min: minimum element

```
Algorithm divide_conquer(a, low, high,
 max, min)
{    if(high==low)
        then max=min=a[low];
        else if(low==high-1)
        {
                if(a[low]<a[high])
                {
                        min=a[low];
                        max=a[high];
                }
                else
                {
                        min=a[high];
                        max=a[low];
                }
        }
        else
        { divide_conquer (a, low, mid,
         max, min);
        divide_conquer (a, mid+1,high,
        max1, min1);
                if(max1>max)
                max=max1;
                if(min1<min)
                min=min1;
        }
}
```

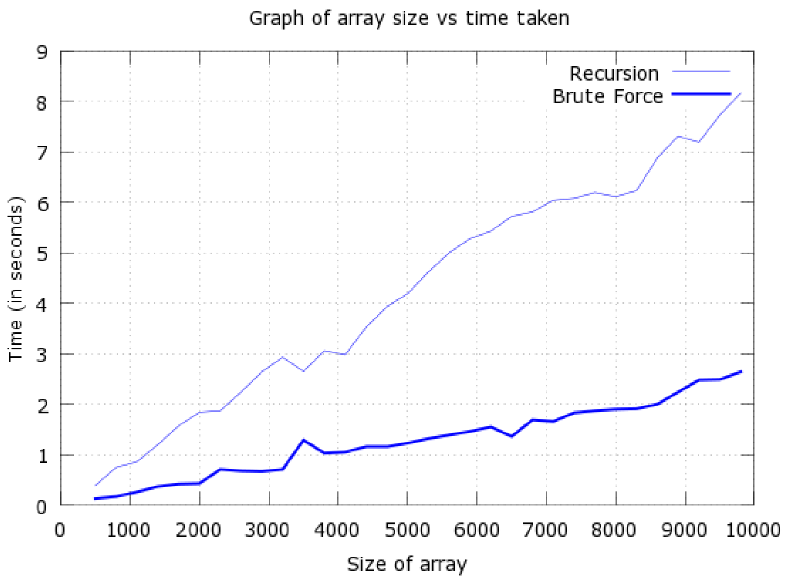

Fig. 1.

RESULT

Table 1 and Graph 1 depict the results of searching maximum and minimum through both methods. Table 1 shows the array size and the corresponding time taken (average) in seconds for searching the maximum and minimum elements of array. The result is depicted graphically in Graph 1. Table 2 depicts the total number of recursion calls required with different array sizes in divide and conquer strategy where 5,00,000 times divide and conquer is called for a particular array size.

DISSCUSSION

The graph of brute-force search is nearly linear but has some peaks and valleys. Theoretically, the graph should be linear, but it is not reflected in the graph due to some peaks (Array size: 2300, 3500, 9200) and valleys (Array size: 2000, 7100). Background system processes like batch processes or system interrupts are the reasons for this change. As Ubuntu 15.10 operating systems supports multi-tasking and multi-threading, so program gets interrupted while processing.

It is evident from the graph that time required for searching maximum and minimum element in array by this method increases with increasing array sizes. It is because in brute-force approach, the entire array is traversed and each element is compared with maximum and minimum value to get the maximum and minimum element of the array. Each element is compared with both maximum and minimum elements even if the element is not larger than the maximum value there are chances that it may be smaller than the minimum value. But, if the array element is greater than current maximum value, the value of the maximum variable will change with

TABLE I
AVERAGE TIME REQUIRED FOR DIVIDE AND CONQUER AND BRUTE FORCE V/S ARRAY SIZE

Array size	Time(D&C)	Time(B-F)
500	0.39	0.13
800	0.75	0.17
1100	0.86	0.26
1400	1.2	0.37
1700	1.57	0.42
2000	1.84	0.43
2300	1.87	0.71
2600	2.24	0.68
2900	2.64	0.67
3200	2.8	0.71
3500	2.92	1
3800	3.06	1.03
4100	3.42	1.05
4400	3.52	1.17
4700	3.93	1.17
5000	4.19	1.23
5300	4.62	1.32
5600	5	1.39
5900	5.28	1.46
6200	5.43	1.55
6500	5.72	1.6
6800	5.81	1.69
7100	6.04	1.66
7400	6.08	1.83
7700	6.19	1.87
8000	6.11	1.9
8300	6.23	1.91
8600	6.88	2
8900	7.31	2.24
9200	7.19	2.48
9500	7.73	2.49
9800	8.16	2.64

TABLE II
NUMBER OF RECURSION CALLS IN DIVIDE AND CONQUER STRATEGY

Array Size	No. of recursions
500	1.28E+08
800	3.84E+08
1100	6.78E+08
1400	1.12E+09
1700	1.63E+09
2000	2.15E+09
2300	2.78E+09
2600	3.57E+09
2900	4.51E+09
3200	5.53E+09
3500	6.56E+09
3800	7.58E+09
4100	8.61E+09
4400	9.78E+09
4700	1.11E+10
5000	1.26E+10
5300	1.42E+10
5600	1.60E+10
5900	1.79E+10
6200	2.00E+10
6500	2.20E+10
6800	2.41E+10
7100	2.61E+10
7400	2.82E+10
7700	3.02E+10
8000	3.23E+10
8300	3.44E+10
8600	3.66E+10
8900	3.90E+10
9200	4.16E+10
9500	4.43E+10
9800	4.71E+10

that greater value. As the element of array with maximum value cannot act as the minimum value of that array at the same, (assuming size of array is greater than one) only one comparison is required. Same can be stated for the array element which is lesser than the current minimum value. Hence, comparison of array element with both maximum and minimum is necessary except when the value of variable maximum and minimum changes with the array element. As the array size increases, number of comparisons also increases and time taken to run the program also increases. Hence, graph increases (approximately) linearly with time.

The graph of divide and conquer search is also increases with increase in the array size. As it follows divide and conquer strategy, hence it divide the array into two equal sub-parts recursively untill the array size is at most two. Maximum and minimum value is then calculated by comparing elements in the array (only if there is more than one element in array otherwise the single element is assigned to both maximum and minimum variables). So, with the increases in the array size, the recursions also increases and the time to solve each recursion also increases.

On comparing both graphs, it is observed that slope of divide and conquer algorithm is approximately 0.5 times more than brute force search algorithm. Time complexity of brute-force search is O(n) and the time complexity of divide and conquer is also O(n) where *n* is the number of element. In divide and conquer method, problem is divided into two sub-parts and these sub-parts are further divided into two more sub-parts recursively and finally the result of all these sub-parts are then merged together to get the final result of the problem. The array is divided into two parts and both the parts are required to be solved and merged with the complexity:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where, b is the number of parts (in which the array is divided into nearly equal parts), a is the number of parts required to solve and f(n) is the effort for merging sub-parts of the problem. So, the time complexity is:

$$T(n) = 2T\left(\frac{n}{2}\right) + 2$$

On solving the recurrence relation, time complexity is (3 n/2- 2). In brute-force algorithm, time complexity is (2n-2). Slope of brute-force search is 0.5 times more than that of divide and conquer.

But results are contradicting the fact that divide and conquer algorithm always give the best result than that of brute force algorithm. Here, Brute-force search which is an iterative process takes lesser time to run and execute than Divide and Conquer method which is a recursive process. Recursion means a function which calls itself repeatedly untill certain condition is satisfied. It uses system stack to accomplish this task. As stack uses LIFO approach (Last In First Out) and when a function is called, the controlled is moved to where function is defined which is stored in memory with some address, which is also stored in stack. In Brute-force search algorithm, only one variable is required to increment the loop apart from maximum and minimum variables but in Divide and conquer algorithm, stack is required to store and retrieve the value of address for recursion. Return addresses of recursions are saved each time and even after that merging of array is required before giving solution to other sub-problems.

All these tasks take up a lot of CPU time and memory, making recursion slower. As both the sub-parts of array is solved and both are merged, the benefit of recursion is not obtained here. It might be beneficial if after dividing the array into sub-parts, only few parts are required to solve, but in this problem, both parts are needed to solve to get the desired result. The main advantage of recursion over iteration is that it adds clarity and reduces programmer time to write code and debug the program but its usually slower due to the overhead of maintaining stack.

To get the benefit of divide and conquer, array can be sorted initially and then maximum and minimum can be obtained in constant time complexity only. Merge sort can be used as it divide the algorithm into two nearly equal parts recursively and then merge them to get a sorted array. After sorting in ascending order, the first element is always the smallest and the last element of the array is always the largest. Therefore, time complexity of finding maximum and minimum elements after sorting array element is O(1), but the time required for sorting the elements through merge sort is O(nlogn), which is log(n) times higher than finding the elements without sorting. With smaller array element, time difference will be very small but with the increase in array size, the time difference will increase logarithmic times which are not desirable when only maximum and minimum values are required.

To decrease the processing time of divide and conquer search, parallel computing can be a solution. Four threads of CPU can be assigned at one quarter of the array and each would return the largest number and the smallest number they find. After that all those number from the four threads would be compared to get the final result. If the data is in large amount, say terabytes or petabytes, data could be divided into multiple computers which would further be subdivided into threads. But if the data is not very large, then this approach would be inefficient as processing the data in local machine will be faster than transferring it across network and then merge those results by again transferring those results into local machine. Even on a single machine, process won't actually get 4 time speed from four threads as they fight for memory bandwidth. But process would get some speedup from this. From both the methods, time complexity for searching maximum and minimum values is in O(n).

But there exist another algorithm which can search the desired result in O(n) time. But, this requires some hardware that doesn't exist (yet), namely a quantum computer. The approach is to use Grover's algorithm. This uses a superposition state that could resolve to be any of the items. The system runs this superposition state repeatedly through a quantum operation which selectively adjusts the mix such that the correct answer becomes the likely resolution. The operation acts on all n states at once. That sounds similar to parallel computing, but it differs radically: instead of several pieces of hardware working at once, this would use a single piece of hardware, working across multiple possible universes.

But as quantum devices that exist today can process only few data items so using it is inefficient and expensive.

II. Conclusion

In divide and conquer, the number of comparisons are lesser then that for brute force method but due to large number of recursive calls it takes a lot of time to process the data, so brute force approach is more efficient than divide and conquer strategy.

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stin *Introduction to Algorithm*, 3rd ed. The MIT Press, Cambridge, Massachusetts, London, England

[2] Yossi Shiloach and Uzi Vishkin, *Finding the maximum, merging and sorting in a parallel computing model*, 1st ed. Publisher: Springer Berlin Heidelberg

[3] Brute-force search in *Wikipedia*. Retrieved February 26, 2009, from https://en.wikipedia.org/wiki/Brute-force_search

[4] Divide and conquer algorithms in *Wikipedia*. Retrieved February 26, 2009, from https://en.wikipedia.org/wiki/Divide_and_conquer_algorithmsR

[5] Recursions in algorithms in *Wikipedia*. Retrieved February 26, 2009, from http://www.cs.odu.edu/~cs381/cs381content/recursive_alg/rec_alg.html

[6] Merge sort algorithms in *Wikipedia*. Retrieved February 26, 2009, from https://en.wikipedia.org/wiki/Grovers_algorithm

[7] Quantum Computing in *uwaterloo*. Retrieved February 26, 2009, from https://uwaterloo.ca/institute-for-quantum-computing/quantum-computing-101