International Journal of Research
Available at https://pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 20
September 2018

A New Approach for Selecting Pivot Element in Quick Sort to Reduce Execution Time

K Jhansi & Sumayya Afreen
[1]Student, Dept. of CSE, Stanley College of Engineering & Technology for Women
[2]Assistant Professor, Dept. of CSE, Stanley College of Engineering & Technology for Women

Abstract:

The Pivot element is the element which is selected first in the array in the quick sort, and based on which array is partitioned. Any element can be selected as pivot element but first or last element of the array is usually selected as Pivot element. In this paper, an approach to select pivot element is discussed with examples which reduces the execution time. This approach can be applied to almost sorted arrays.

Keywords*: Pivot element*

1. Introduction

An algorithm is a detailed series of instructions for carrying out an operation or solving a problem. They lay a foundation for creating a code for the given problem. We also use algorithms in our day to day life in solving our real life problems knowingly or unknowingly. Technically, computers use algorithms to list the detailed instructions for carrying out an operation [1].

In general, we model an algorithm according to the given problem. Like we have algorithms on sorting, searching etc. These sorting and searching algorithms provide steps in getting the work done of sorting or searching.

In a non-technical approach, we use algorithms in everyday tasks, such as a recipe to bake a cake or a do-it-yourself handbook.

Technically, computers use algorithms to list the detailed instructions for carrying out an operation. For example, to compute an employee's paycheck, the computer uses an algorithm. To accomplish this task, appropriate data must be entered into the system. In terms of efficiency, various algorithms are able to accomplish operations or problem solving easily and quickly.[9]

2. Algorithm Strategies

we have different approaches in creating an algorithm and how the problem can be tackled. Like one may like to break the given huge problem into small manageable parts, where one can work on those small parts and later join those parts back and give the solution to the problem. Some algorithm strategies are divide and conquer, dynamic programming, greedy approach, brute force, branch and bound etc [2] . By knowing the strengths and weakness of different algorithms we can pick the best suitable one for the task at hand [3].

2.1. Brute Force

Brute force is a type of algorithm that tries a large number of patterns to solve a problem. In some cases, they are extremely simple and rely on raw computing power to achieve results.

A common example of a brute force algorithm is a security threat that attempts to guess a password using known common passwords. Such an algorithm might also try dictionary words or even every combination of ASCII strings of a certain length. [13] Brute forcing Solves a problem in the most simple, direct, or obvious way but May do more work than necessary[12].

Brute forcing is just like an exhaustive search.

2.2. Branch and Bound

A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions by means of state space search: the set of candidate solutions is thought of as forming a rooted tree with the full set at the root. The algorithm explores *branches* of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated *bounds* on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far by the algorithm. The algorithm depends on efficient estimation of the lower and upper bounds of regions/branches of the

International Journal of Research
Available at https://pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 20
September 2018

search space. If no bounds are available, the algorithm degenerates to an exhaustive search.[14].

2.3. Dynamic Programming

The idea is very simple, If you have solved a problem with the given input, then save the result for future reference, so as to avoid solving the same problem again. If the given problem can be broken up in to smaller sub-problems and these smaller subproblems are in turn divided in to still-smaller ones, and in this process, if you observe some over-lapping subproblems, then its a big hint for Dynamic Programming. Also, the optimal solutions to the subproblems contribute to the optimal solution of the given problem[15].

2.4. Greedy Algorithm

A greedy algorithm, as the name suggests, always makes the choice that seems to be the best at that moment**.** This means that it makes a locally-optimal choice in the hope that this choice will lead to a globally-optimal solution.in this approach we will be greedy about some particular thing that may be the cost of executing the algorithm etc.

How do you decide which choice is optimal? Assume that you have an objective function that needs to be optimized (either maximized or minimized) at a given point. A Greedy algorithm makes greedy choices at each step to ensure that the objective function is optimized. The Greedy algorithm has only one shot to compute the optimal solution so that it never goes back and reverses the decision.[16].

2.5. Divide and conquer approach

Divide problem into several smaller subproblems.
Normally, the subproblems are similar to the original problem.
Conquer the subproblems by solving them recursively.
Base case: solve small enough problems by brute force.

Combine the solutions to get a solution to the subproblems.

And finally a solution to the orginal problem
Divide and Conquer algorithms are normally recursive[17].

3. Applications of Algorithms in Day-to-Day Life

We can have many examples like below where we unknowingly use or apply the algorithmic strategy to our day-to-day life .

Let's say that you have a friend arriving at the airport, and your friend needs to get from the airport to your house. Here are four different algorithms that you might give your friend for getting to your home:

3.1. The Taxi Algorithm:

1. Go to the taxi stand.
2. Get in a taxi.
3. Give the driver my address.

3.2. The Call-me Algorithm:

1. When your plane arrives, call my cell phone.
2. Meet me outside baggage claim.

3.3. The Rent-a-car Algorithm:

1. Take the shuttle to the rental car place.
2. Follow the directions to get to my house.

3.4. The Bus algorithm:

1. Outside baggage claim, catch bus number 70.
2. Transfer to bus 14 on Main Street.
3. Get off on Elm street.
4. Walk two blocks north to my house.

All four of these algorithms accomplish exactly the same goal, but each algorithm does it in completely different way. Each algorithm also has a different cost and a different travel time. Taking a taxi, for example, is probably the fastest way, but also the most expensive. Taking the bus is definitely less expensive, but a whole lot slower. You choose the algorithm based on the circumstances.[8].

4. Analysis of Algorithms

The basic notations for run time analyses are worst-case run time, average case run time, expected run time, amortized run time, and the analysis of competitiveness and they area represented by the symbols $(O,\Omega,\Omega\infty,\Theta,o,\omega)$

We say af function f:

- is constant, if $f(n) = \Theta(1)$
- grows logarithmically, if $f(n) = O(\log n)$
- grows polylogarithmically, if $f(n) = O(\log^k (n))$ for a $k \in N$.
- grows linearly, if $f(n) = O(n)$

• grows quadraticly, if f(n) = O(n 2) [10]

There are many different kinds of algorithms used everywhere.we use algorithms in signal processing, in allocation and deallocation of memory,in cryptography, in encryption,in networks, in graph theory, in bioinformatics, in geoscience ,in astronomy and many more .[11]

5. Sorting

Sorting is arranging the given elements into ascending order or descending order. We also have many sorting algorithms like merge sort, quick sort etc. Both merge sort and quick sort come under same strategy of algorithms. They both come under the category of divide and conquer.
Merge sort divide the array at the middle and work on the other 2 halves, it goes on dividing until it reaches minimum number of elements and then after it starts combining back. Where as in quick sort it selects a pivot element, depending upon the pivot element it divides the array in such a way that the elements left to it are lesser than it and the elements right to it are greater than it. Again, in the divided parts it again selects the pivot and the procedure goes on until we get the sorted array.

5.1. Quick Sort

The key process in quicksort is partition(), Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x, all this should be done in linear time [4]. Quick sort (sometimes called partition exchange sort) is an efficient algorithm, serving as a systematic method for placing the elements of an array in order, Developed by Tony Hoare in 1959 and published in 1961 it is still a commonly used algorithm for sorting [5].
Among sorting algorithms with O(N log N) average computing time, median-of-3 quicksort is considered to be a good choice in most contexts,it sorts in place, and is usually faster than other in-place algorithms such as heapsort [6].

5.2. Bubble sort

Is based on the idea of repeatedly comparing pairs of adjacent elements and then swapping their positions if they exist in the wrong order.[18]

6. Problem with Quicksort

During the selection of the pivot element, the element may sometimes be the largest or the smallest of all the elements or may also be the middle element. If the pivot element is the extreme element then the time complexity of the quick sort algorithm increases to O(n^2).

6.1. Approach to Select Pivot Element

The problem in quicksort is only due to selecting a random element as the pivot element. Here is an algorithm for selecting the pivot element for almost sorted list.

```
Algorithm pivot (high, low)
{
        h :=0;
        for k := 1 to n do
        {
                if(k=0) then
                        {
                                if(a[k]>low and
                        a[k]<high) then
                                {
                                        b[h]
                                :=a[k];
                                        h++;
                                }
                        }
                else
                {
                        if(a[k]>low and a[k]<high
                and a[k]>a[k-1])
                        {
                                b[h] :=a[k];
                                h++;
                        }
                }
        }
        Mid :=h/2;
        pivot_element :=b[mid];
}
```

Algorithm Explanation
For proceeding with this algorithm, we need to have the highest and the least element of the given list. These highest and least elements are passed as arguments to this algorithm.Here a[] is the array of elements which has to be sorted and b[] is the temporary array into which we insert the elements only if those are greater than the least element and lesser than the highest element of a[] and should be greater than the element in b[]. At lastthe pivot element is at the middle of array b[].

International Journal of Research
Available at https://pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 05 Issue 20
September 2018

Example
Consider array a[]={19,15,4,1,3,16}
Here highest and least elements are 19 and 1 respectively.
Step 1:b[] will be empty

a[]

19	15	4	1	3	16

b[]

Step2:
19>1 and 19<19 // condition failed
15>1 and 15<19 // so insert into b[]

15					

4>1 and 4<19 and 4>15 // condition failed
1>1 and 1<19 and 1>15 // condition failed

3>1 and 3<19 and 3>15 // condition failed
16>1 and16<19 and 16>15 //so insert into b[]

15	16				

Step 3:
Now the middle element of b[]=3/1
So b[1] ie..., 15 will the pivot element using which the quick sort algorithm can be proceeded.

Limitations of the algorithm
Algorithm can only work for almost sorted list. Sometimes the algorithm does not exactly selects the middle element as thepivot.

Example
consider array a[]={3,6,4,2,5}
here the highest and the least elements are 6 and 2 respectively.
Step 1:
a[]

3	6	5	2	4

b[]

Step 2:
3>2 and 3>6 // insert into b[]
6>2 and 6<6 and 6>3 //condition failed
5>2 and 5<6 and 5>3 //insert into b[]
2>2 and 2<6 and 2>5 // condition failed
4>2 and 4<6 and 4>5 // condition failed

3	5			

Step 3:
Middle element of b[]=3/2
So b[1]=3 will be the pivot element which is not middle one.

7. References

1. https://www.techopedia.com/definition/3739/algori thm
2. Algorithm Strategies, Department of Computer Science University of Maryland, College Park
3. https://www.geeksforgeeks.org/quick-sort/
4. https://en.wikipedia.org/wiki/Quicksort
5. David R. Musser: Introspective Sorting and Selection Algorithms, Computer Science Department Rensselaer Polytechnic Institute, Troy, NY 12180 musser@cs.rpi.edu
6. https://www.geeksforgeeks.org/heap-sort/
7. https://www.geeksforgeeks.org/bubble-sort/
8. https://computer.howstuffworks.com/what-is-a-computer-algorithm.html
9. https://www.techopedia.com/definition/3739/algorithm
10. http://www.mi.fuberlin.de/wiki/pub/ABI/DiscretMathWS10/runtime.pdf
11. https://en.wikipedia.org/wiki/List_of_algorithms#Astronomy
12. https://www.eecs.umich.edu/courses/eecs281/f04/lecnotes/25-greedybrute.pdf
13. https://simplicable.com/new/brute-force
14. https://en.wikipedia.org/wiki/Branch_and_bound
15. https://www.codechef.com/wiki/tutorial-dynamic-programming#Introduction
16. https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/tutorial/
17. https://www.radford.edu/~nokie/classes/360/divcon.html
18. https://www.hackerearth.com/practice/algorithms/sorting/bubble-sort/tutorial/