International Journal of Advance Research in Science and Engineering
Vol. No.4, Special Issue (01), September 2015
www.ijarse.com
IJARSE
ISSN 2319 - 8354

STUDY OFAPPLICATIONS OF DIVIDE AND CONQUER APPROACH IN ALGORITHM AND COMPLEXITY

Piyush Kumar Mudgal

[1]*PG Scholar, MBM Govt. Engineering Collage J.N.V.U , Jodhpur, Rajasthan, (India)*

ABSTRACT

In this paper I am going to present the main applications of divide and conquer approach to design algorithms to solve some specific problems and the impact on the time complexity of the algorithm. Basic of this approach is to break the big problem in to small sub problems and conquer means overcome by effort these sub problem by find the solutions for all the small sub problem after finding all sub solutions it recombine them in order to find the solution to the original problem. Using such approach most of the time it reduces the time complexity of the algorithm for a given problem..

Keywords: BINARY SEARCH, DAC, QUICK SORT, RECURRENCE RALATION, TC.

I. INTRODUCTİON

Algorithm design and analysis plays a vital role in study of computer. In context to this paper a algorithm is a set of finite steps or instruction to solve a specific problem. For a specific problem there may be more than one algorithm this expresses that a specific problem can be solve in different ways but as per the context of computer the efficiency is a important factor so a efficient algorithm means "a algorithm which takes less time and space on computer while executing which is analyzed after designing". In theory of study of algorithm for designing some algorithm some standard approaches are defined. Such approaches are mainly-

a. Divide and conquer approach.

b. Greedy approach.

c. Dynamic approach.

In this paper I am focusing on the first approach which is divide and conquer. This approach is used to find solution for problems which need an exact solution in shorter time basically the optimization problems are not solved using this approach. Divide and conquer approach is a three step approach to design a algorithm for a given problem.

a. Divide.

b. Conquer.

c. Combine.

In first step divide and conquer approach make algorithm to divides the big problem into small sub Problems it may repeatedly do this division till finding the smallest sub problem which can be solved(conquered) easily.

After finding the smallest sub problem in the second step it make algorithm to solve (conquer) that sub problem recursively and return solution recursively. In the last step it make algorithm to it combines the solution of sub problems or solved sub problems in the same manner it divided to get the solution to the given big problem.

1.1 Control Abstraction of Divide and Conquer Approach

Following control abstraction shows the divide and conquer approach.

```
DAC(p,q)
{
    if( small(p, q))
    {
    return (solution(p, q));
    }
    else
    {
      m=divide(p, q);
      DAC(p, m);
      DAC(m+1, q);
   combine(DAC(p, m) DAC(m+1, q));
    }
```

1.2 Applications of Divide and Conquer Approach

Algorithm is a vast aria to study and analysis so there are many algorithms which can be seen as application of divide and conquer approach. All these algorithms cannot be in described in a single paper so here in this paper I am going to discus some fundamental and basic algorithms using the divide and conquer approach. Which are.

a. Finding the power of an element
b. Searching an element in a sorted array
c. Sorting an unordered array
d. Matrix multiplication

II. FINDING THE POWER OF AN ELEMENT

The problem given is

Input: An integer element a>1 and another integer n>1.

Output: Finding a^n (n power of a)

2.1 Algorithm without Using Divide and Conquer Approach

```
Power(a,n)
 {
    int i ,j=1;
    for(i=1;i<=n;i++)
     {
```

```
            j=j*a;
            return j;
          }
    }
```

In above algorithm for loop runs n time so the time complexity of the algorithm without using DAC is O(n).

2.2 Algorithm using Divide and Conquer Approach

While we use the divide and conquer approach th problem a^n is divided in to $a^{n/2}$ and $a^{n/2}$ where we found the value of $a^{n/2}$ using further dividing and conquer once we get $a^{n/2}$ there is no need to find other sub problem $a^{n/2}$. a^n can be found by squaring $a^{n/2.}$ Mathematically $a^n = a^{n/2} * a^{n/2} = (a^{n/2})^2$.The problem of size n is divided into two n/2 sized problems algorithm need to spent its computing resources to find just $a^{n/2}$. As shown in algorithm below-

```
Power(a,n)
{
  int  mid, f=;
  if(n==1)
    return a;
 else
      {
         mid=n/2;
        f=Power(a,n/2);
        f=f*f;
        return f;
       }
   }
```

This algorithm shows the recurrence relation as

$$T(n)=\begin{cases} 1 & \text{if } n=1 \\ T(n/2)+c & \text{if } n>1 \end{cases}$$

$T(n)= T(n/2^2)+c+c$

.

.

……k times running

$T(n)= T(n/2^k)+kc$

Let $2^k=n$

$K= \log_2 n$

$T(n)= 1+ c.\log_2 n$

i.e. Time complexity $T(n)=\log_2 n$

from above analysis it is clear that using divide and conquer approach reduces the time complexity.

III. SEARCHING AN ELEMENT IN A SORTED ARRAY

The problem is given as Input: a sorted array with n element Output : an element from the array which key value is 'x'

3.1 Algorithm without using Divide and Conquer Approach

As it is known for finding a element in a sorted array of size n we have to traverse the array at least once and compare. Whether the element with key 'x' is first or last or in between. So maximum computing time is n and minimum is 1 so the time complexity is Ω(1) in best case and O(n) in worst case. This is also known as linear search.

3.2 Algorithm using Divide and Conquer Approach

While we use divide and conquer approach the array is divided into two part as the array is sorted we compare mid element to the key x if it is x then return the index of mid if the mid element is less then x we will traverse the right sided array with n/2 elements if grater will go with left half of array in both case only half of the array is traversed. This is also called as binary search. As shown in following algorithm.

```
Binary search(a, i, j,x)        /* i and j are index of array
 {
   int mid ;
   if (i==j)
     {  if(a[i]==x)
        return i;
      else
        return -1;
       }
   Else
   { mid=(i+j)/2;
     if (a[mid]==x )
        return mid;
     else
       {  if(a[mid]>x)
          Binary search(a,i,mid-1,x);
         Else
           Binary search(a,mid+1,j,x);
        }
       }
      }
```

The above algorithm gives the recurrence relation as

$$T(n)=\begin{cases} 1 & \text{if } n<=1 \\ T(n/2)+c & \text{if } n>1 \end{cases}$$

Which gives the time complexity $T(n)= \log_2^n$

IV. SORTING AN UNSORTED ARRAY

One of the fundamental issues in computer science is ordering a list of items. Although there is a huge number of sorting algorithms. Such insertion sorting selection sorting and many others but all sorting algorithm based on comparisons having time complexity $O(n^2)$ except quick and merge sort. Quick sort and merge sort are the most popular applications of the divide and conquer approach I am going to discus about the quick sort in this paper.

4.1 Review of Quick sorting

Quick Sort is a sorting algorithm that is efficient in regards to both space and time complexity. It sorts in place for an optimal $\theta(n)$ space complexity and runs generally in $\theta(n\log_2 n)$ time, though $\theta(n^2)$ in worst case.

4.2 Algorithm

Quicksort is compost of two methods:

1. 'QUICKSORT(A,p,r)'and
2.'PARTITION(A, p, r)'.

```
 QUICKSORT(A, p, r)
{
  if p < r
  q=PARTITION(A, p, r);
  QUICKSORT(A, p, q-1 );
  QUICKSORT(A, q+1, r);
}
PARTATION(A, p, r)
{
  x = A[r];
  i = p-1;
  for j = p to r-1
   {
    if A[j] <= x
    i = i+1;
   exchange A[i] with A[j];
}
```

exchange A[i+1] with A[r];

return i+1; }

4.3 Analysis

According to the algorithm there may be two conditions as follows.

1. The pivot is the largest or smallest element in the given sub-array

2. The pivot is the middle element in the given array.

It depends on the nature of the elements of that array. Either array is already sorted in increasing or decreasing order or it is randomly unsorted. Over these condition the array is divided into sub arrays (Large problem to small sub problems *D&C).This division may be in two ways according to first condition T(n)=T(n-1)+1 (n is the size of array). And according to second condition T(n)=T(n/2)+T(n/2).The time taken by partition algorithm is 'n'. So recurrence relation for quick sort is-

$$T(n)=\begin{cases} 1 \quad \text{if } n<=1; \\ T(n-1)+n. \\ \quad \text{Or} \quad \text{if } n>1 \\ T(n/2)+T(n/2)+n \end{cases}$$

$T(n)=2T(n/2)+n;$

Appling substitution method

$T(n)=2^2\,T(n/2^2)+n/2^1+n/2^0$

$=2^3T(n/2^3)+n/2^2+n/2^1+n/2^0$

.

.

. K times continuing the substitution

$= 2^kT(n/2^k)+n+n+\ldots\ldots+n \quad = 2^kT(n/2^k)+nk$

$= 2^kT(n/2^k)+ kn$

Let $2^k=n$

$K= \log_2 n.$

$= n.1+n.k$ i.e. $n+n\log_2 n$ i.e. $\Omega(n\log_2 n)$

This shows the best case time complexity. Where every time when problem divided into sub problem the pivot element is middle element. Or we can say every divided into two half arrays .

Now for

$T(n)=T(n-1)+n$

Appling substitution method

$T(n)=T(n-2)+n-1+n$

.

.

.K times

$T(n)=T(n-k)+n-(k-1)+\ldots.+n$

Let $k=n-1$

$T(n)= T(1)+(2+3+4+\ldots.+n)$

= 1+2+3+……+n

$=\sum_{i=1}^{n} i$

=n(n+1)/2

$=n^2/2+n/2$ i.e. $O(n^2)$

This shows the worst case time complexity where every time the pivot element is the smallest or largest element or we can say array always divided into two parts of size n-1 and one. Basically the worst case condition is occur whenever the array is already sorted in either ascending or descending order. For average case time complexity we assume at every alternative step we got division as two same size array and for remaining we get division as n-1 and 1.

Let

T(n)=2T(n/2) +n is named lucky

T(n)=T(n-1)+n is named unlucky

So for an average case

We got

T(n)= lucky+unlucky+ lucky+unlucky+…….+lucky .

T(n)=2T(n/2)+n

= 2T(n/2-1)+n/2+n

= 2^2T((n/2-1)/2)+(n/2-1)+n/2+n

.

.

…….. Will give $n\log_2 n$ i.e. $\Theta(n\log_2 n)$

V. MATRIX MULTIPLICATION

Matrix multiplication is a well known application of divide and conquer approach for understanding this problem.

Input: Two matrix of order nxn

Output : A matrix of order nxn (multiplication of two matrix)

5.1 Algorithm without using Divide and Conquer

Let A_{nxn} and B_{nxn} then

$C_{nxn}= A_{nxn}*B_{nxn}$

C having n^2 element and each element require n multiplication so all n^2 element require requires $n*(n^2)$ operation. Let time taken by one operation is unit then total time require for all multiplication is n^3 that implies the time complexity of matrix multiplication is $O(n^3)$.

```
MM(A[n][n],B[n][n])
 {
   int  i, j,k,C[n][n];
for(i=0;i<=n;i++)
```

```
{
  for(j=0;j<=n;j++)
    {  C[i][j]=0
       }
  }
 for(i=0;i<=n;i++)
  {
    for(j=0;j<=n;j++)
      {
        for(k=0;k<=n;k++)
          {
            C[i][j]=C[i][j]+A[i][k]+b[k][j]
             }
         }
    }
}
```

Due to three level nested loop the time complexity is $O(n^3)$.

5.2 Algorithm using Divide and Conquer

To understand this algorithm we need to take an example.

M_M_algo (Two matrix to be multiplied)

Step1.If given matrices size are <= 2x2 problem is small so stop. If not then

Step2. Assume given two matrices are square matrices for example.

$$A=\left\{\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right\}\begin{array}{cc} A_{11} & A_{12} \\ & \\ A_{21} & A_{22} \\ & \end{array}\quad 4x4$$

$$B=\left\{\begin{array}{cc|cc} a & b & c & d \\ e & f & g & h \\ \hline i & j & k & l \\ m & n & o & p \end{array}\right\}\begin{array}{cc} B_{11} & B_{12} \\ & \\ B_{21} & B_{22} \\ & \end{array}\quad 4x4$$

$$C=\left\{\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right\}\quad 4x4$$

Here

$C_{11}=A_{11}*B_{11}+A_{12}*B_{21}$

$C_{12}=A_{11}*B_{12}+A_{12}*B_{22}$

$C_{21}=A_{21}*B_{11}+A_{22}*B_{21}$

$C_{22}=A_{21}*B_{12}+A_{22}*B_{22}$

The above shows that a n size problem is broken in to eight n/2 size problem and each n/2 size problem require 4 times of $(n/2)^2$ of time. Which gives the recurrence relation as:

$$T(n)=\begin{cases} c & \text{if } n<=2x2 \\ 8\,T(n/2)+4(n/2)^2 & \text{if } n>2x2 \end{cases}$$

i.e. the time complexity $T(n)=O(n^3)$.

In the matrix multiplication the time complexity using DAC and with out using DAC is same. But Strassen found that using DAC the number of multiplications can be reduce 7 from 8 so the no. of sub problems is reduced and the improved recurrence relation is

$$T(n)=\begin{cases} c & \text{if } n<=2x2 \\ 7\,T(n/2)+4(n/2)^2 & \text{if } n> 2x2 \end{cases}$$

Using this recurrence relation the time complexity is reduced $O(n^{2.81})$ from $O(n^3)$. Strassen's matrix multiplication is also an application of DAC

VI. CONCLUSION

This paper is a study divide and conquers approach to design a algorithm and its application. There are many application of divide and conquer approach some fundamental applications of them are described here. The entire described algorithm are analyzed as when using DAC and without using DAC. The time complexity of the application is reduced when the divide and conquer approach is used to design the algorithm for example the searching sorting and power of an element. This is not true all the time some time the time complexity to any algorithm remains same whether using DAC or not. For example the matrix multiplication. But most of the time it reduces the complexity. In this paper a well known and very popular application of divide and conquer approach Quick sort algorithm is described as its designing and the analysis as well. This paper will help to understand the scenario of divide and conquer approach to design algorithm t the fundamental level.

REFERENCES

[1] Zhijing G.Mou and Paul Hudak, " An algebraic model for divide and conquer and its parallelism", The Journal of Supercomputing, 2, 257-278, Kluwer Academic Publishers, Boston. Manufactured in The Netherlands, 1988.

[2] Douglas R. Smith, "Applications of a strategy for designing divide and conquer Algorithm", Science of computer Programming,1985.

[3] Aho A., Hopcroft J., and Ullman J., The Design and Analysis of Computer Algorithms, Addison Wesley, 1974.

[4] Cormen T., Leiserson C., Rivest R., and Stein C., Introduction to Algorithms, McGraw Hill,2001.

[5] Ellis H., Sartaj Sahani and Sanguthevar R., Fundamental of computer Algorithms, University Press, 1998.

[6] Christian S., Ian Welch and Peter K., "Application of divide and conquer algorithm paradigm to improve the detection speed of high interaction client honeypots", Victoria University of Wellington.

[7] Rudranarayan M. Mukherjee, Kishor D. Bhalerao and Kurt S. Anderson, "A divide-and-conquer direct differentiation approach for multibody system sensitivity analysis", March 2007.

[8] Radu Rugina and Martin Rinard, "Automatic Parallelization of Divide and Conquer Algorithms".