Analysis and Performance of Divide and Conquer Methodology

Dr.S.Dhanalakshmi[1], Mr.D.Krishna Kishore[2], Dr.T.Prabakaran[3]

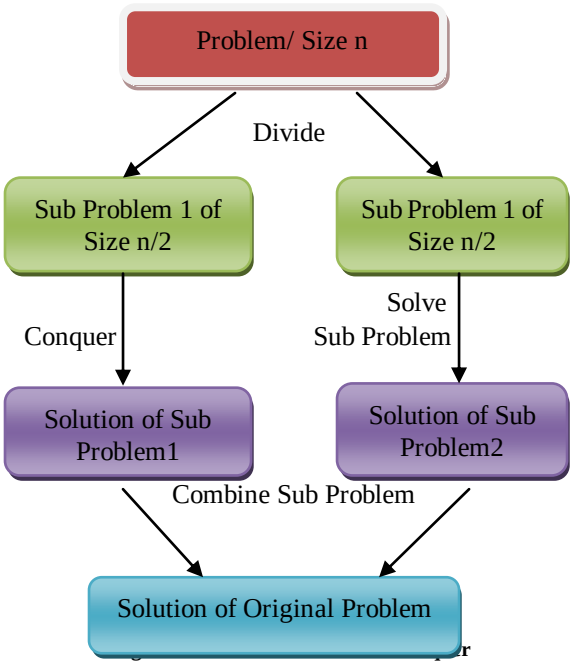
Professor[1], Department of Computer Science and Engineering, Malla Reddy Engineering College (Autonomous), Telugana, India
Associate Professor[2], Department of Computer Science and Engineering, Malla Reddy Engineering College (Autonomous), Telugana, India
Associate Professor[3], Department of Computer Science and Engineering, Malla Reddy Engineering College (Autonomous), Telugana, India

***Abstract*— this paper presents a survey on Divide and Conquer algorithms. Divide and conquer algorithms that follow the many applications and problem solving of quick sort, merge sort, binary search, finding maximum or minimum element, and strassens matrix multiplications algorithms. This discussion is centered overview of performance analysis of divide and conquers methods & the application of binary search, quick sort and merge sort. Divide and conquer methodology determine the control abstraction of solving more number of sub problems in the given original problem, after solving sub problems then combine the solutions, to find the complexity of given problems.**

***Index Terms* — Binary Search, Quick Sort, Complexity, Merge Sort etc.,**

1. INTRODUCTION

Divide and conquer strategy, the n inputs of a given function are splitted into p distinct subsets, resulting of $1< p \leq n$, p sub problems. This methodology determine the breaks the problem into sub problems that are similar to the original problem then recursively solves the sub problems, and finally combines the solutions to the sub problems of original problems. Fig.1. Indicate the recursion calls of divide and conquer algorithms.



This methodology determines the three parts;

1) **Divide** the problems in to two or more sub problems
2) **Conquer** the sub problems by solving them Recursively, solve the sub problems as base cases
3) **Combine** the sub problems in to solutions of original Problems

2. RECURRENCE RELATIONS

Divide and conquer algorithm is described by the recurrence relations.

$$T(n) = \begin{cases} g(n) & n \text{ small} \\ T(n_1) + T(n_2) + \ldots + T(n_K) + f(n) & \text{otherwise} \end{cases}$$

g (n) is the time to compute the answer directly for small inputs. f (n) is the time for dividing P and combining the solutions to sub problems. Recursion of these problems can be solved by substitution method in master theorem. Fig.2. indicate the recursion tree

$$T(n) = \begin{cases} T(1) & n=1 \\ aT(n/b) + f(n) & n>1 \end{cases}$$

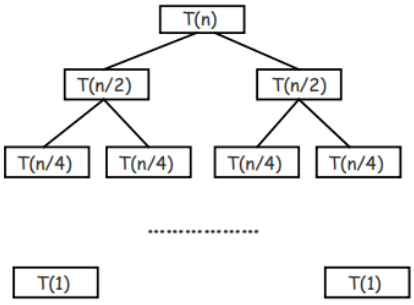


Fig.2. Recursion Tree

Substitution Method in Master Theorem

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

n = n/2 substitute in equ 1.

$T(n/2) = 2T(n/4) + n/2$

$T(n) = 2(2T(n/4) + n/2) + n$

$T(n) = 4T(n/4) + 2n$ **n = n/4**

$T(n/4) = 2T(n/8) + n/4$

$T(n) = 4(2T(n/8) + n/4) + 2n$

$T(n) = 8T(n/8) + 3n$

T (n) = n T (n/2) + n — **n = 2^k**
T (n) = n T (n/n) +? n — **k = log n**
T (n) = n T (1) + k n — **n T(1) is constant**
T (n) = O (n log n)

There are 3 possible cases of Master Theorem in substitution method:

T (n) £ { O (n^d) — a < b^d ; O (n^{dlogn}) — a = bd ; O (n^{logba}) — a > bd

3. RELATED WORK

Several works were done in the divide and conquer methodology. The first one is quick sort, these algorithms to choose the pivot elements and then finding the complexity of partitioning methods, Second one is merge sort, divide the problems into sub problems and conquer the problems & combine the solution of given problems and third one is binary search, for finding the mid value of given elements. These three algorithms can be differing from one another in time complexity of best, worst and average case. Table.1. indicate the complexity analysis of quick sort, merge sort and binary search techniques.

4. QUICK SORT

Quick sort is not a stable search, but it is very fast and requires very less additional space, it's also called partition exchange sort, to picks an element as pivot and partitions the given array around the picked pivot. Pivot element is an middle, the left side element is less value and right side element is greater value. The element is a middle space required by quick sort is very less. Best and Average case complexity is O (n log n) and Worst case complexity is O (n^2). Fig.3. indicate the simple graphical representation of quick sort algorithms

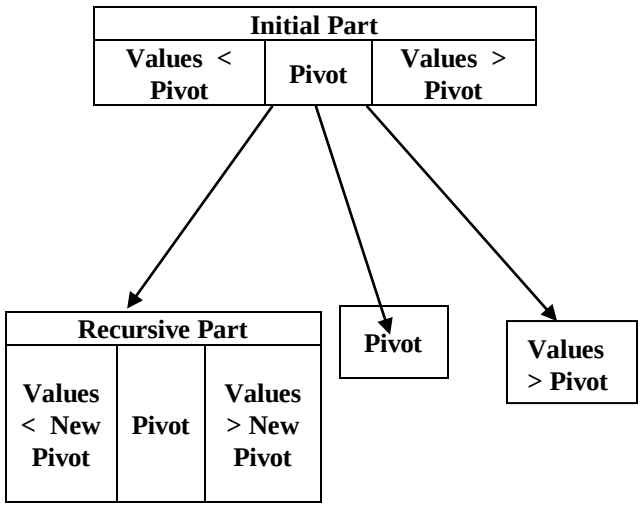


Fig.3. Graphical Representation of Quick Sort

4.1 Algorithm Quick Sort

if (low<high) then
Low=0, high=n
Pi=partition (low, high)
Quick sort (low, pi-1)
Quick sort (pi+1, high)
No need to combine the solutions

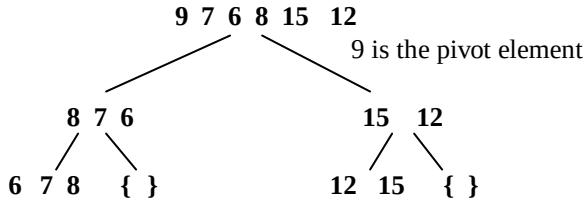
Algorithm Partition

```
int i = left, j = right;
/* partition */
While (i <= j)
While (arr[i] < pivot)
i++;
While (arr[j] > pivot)
j--;
if (i <= j)
tmp = arr[i];
arr[i] = arr[j];
arr[j] = tmp;
i++;
j--;
```

Recursion Method

```
if (left < j)
Quick Sort (arr, left, j);
if (i < right)
Quick Sort (arr, i, right);
```

Quick Sort Partition Methods Example



Example Problem (5 3 1 9 8 2 4 7)

5	3	1	9	8	2	4	7
5	3	1	4	8	2	9	7
5	3	1	4	2	8	9	7
5	3	1	4	2	8	9	7
5	3	1	4	8	2	9	7
2	3	1	4	5	8	9	7
2	1	3	4	5	8	9	7
1	2	3	4	5	8	9	7
1	2	3	4	5	8	7	9
1	2	3	4	5	7	8	9

4.2 Analysis of Quick Sort

4.2.1 Worst Case Complexity

T (n) = T (n-1) + n — **n=n-1**
T (n-1) = T (n-2) + n-1
T (n) = T (n-2) + (n-1) +n
n=n-2
T (n-2) =T (n-3) + n-2
T (n) = T (n-3) + (n-2) + (n-1) + n — **k=3**

T (n) = T (n-k) + (n-k+1) + (n-k+2) + n
k=n-1
T (n) = T (1) + (2) + (3) +… + n
T (n) = 1+2+3+…+n = n.(n+1) / 2
T (n) = O (n^2)

4.2.2 Average Case Complexity

T (n) = O (n log n)

4.2.3 Best Case Complexity

T (n) = a T (n/b) + f (n)

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

T (n/2) = 2 T (n/4) + n/2

T (n) = 4 T (n/4) + 2 n **n=n/4**

T (n) = 8 T (n/8) + 3 n

T (n) = n T (n/n) + k n **k= log n**

T (n) = n T (1) + log n. n **T(1)=1**

T (n) = n is constant + log n. n

T (n) = O (n log n)

5. MERGE SORT

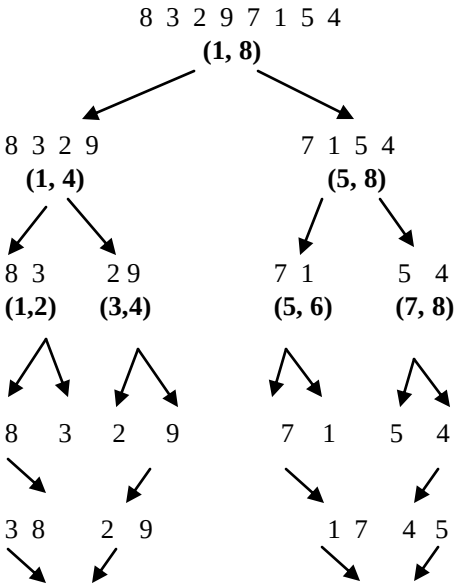
Merge Sort is one of the most popular sorting algorithms and a great way to develop confidence in building recursive algorithms. Then the elements are to be sorted in non decreasing order. Given a sequence of 'n' elements a [1] … a[n] is called keys, and merged to produce a single sorted sequence.

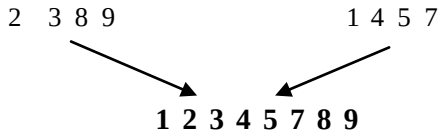
1. Divide Step - A[p], --- ,A [q] and A[q + 1], …, A[r], then each set is sorted individually
2. Conquer Step - Conquer by recursively sorting the two sub arrays A [p…q] and A [q + 1 ... r]
3. Combine Step - merging the two sorted sub arrays A [p…q] and A [q + 1 ... r] into a sorted sequence, then MERGE (*A*, *p*, *q*, *r*).

5.1 Algorithm Merge Sort

Merge Sort (A, p, r)
If p > r
Return
q = (p + r) / 2
merge sort (A, p, q)
merge sort (A, q+1, r)
merge sort (A, p, q, r)

5.2 Tree Calls & Example of Merge Sort





5.3 Complexity Analysis of Merge Sort

Best Case Time Complexity – O (n log n)
Worst Case Time Complexity – O (n log n)
Average Case Time Complexity – O (n log n)
Complexity of best, worst and average case is similar to master theorem in quick sort method.
T (n) = 2 T (n/2) + n so

T (n) = O (n log n)

6. BINARY SEARCH

It is the most popular Search algorithm. It is efficient and also one of the most commonly used techniques that are used to solve problems, it works only on a sorted set of elements. An array *A* of *n* elements with values or records A0, … , An. Begin with an interval covering the whole array, if the value search key is less than the item in the middle of the interval; narrow the interval to lower half. Otherwise narrow it to the upper half, and then repeatedly check until the value is found interval is empty.

1. Low = 0 and high = n.
2. Low > high, the search terminates as unsuccessful.
3. Set m (the position of the middle element) to the floor (the largest previous integer) of (low + high)/2.
4. m < T, set low = m + 1
5. m > T, set high = m - 1
6. Then m = T, the search is done; return m.

6.1 Algorithm Binary Search

```
int binary Search(int low, int high ,int key)
low = 0,  high = n-1
While (low<=high)
mid=(low + high)/2;
if (a[mid]<key)
Low=mid+1;
Else if (a[mid]>key)
High=mid-1;
Else return mid;
Return -1;              //key not found
```

6.2 Example of Binary Search

1	2	3	4	5	6	7	8	9	10
2	5	8	12	16	23	38	56	72	91

Case 1: Searching the key elements 16
Value = a[mid]
Value = 16
Low = 1, high=10 mid= (1+10)/2=5
Mid value and key value to be equal

Searching the key elements 23

Case 2:

Value > a [mid]
Value = 23
Low = 1, high=10 mid= (1+10)/2=5
Key value is found the right side of mid value, so low=mid+1
Low = mid+1=5+1=6
Low = 6, high=10 mid=(10+6)/2=8

Case 3:

Value < a [mid]
Key value is found the left side of mid value, so high=mid-1
Low = 6, high=mid – 1= 8-1=7
Low = 6, high=7, mid = (6+7)/2=6
The mid value and key value to be equal in a [6], then return the key value is a [6] =23.

6.3 Complexity Analysis of Binary Search

Best Case Time Complexity – O (1)
Worst Case Time Complexity – O (log n)
Average Case Time Complexity – O (log n)

Method/ Analysis	Best Case	Average Case	Worst Case
Quick Sort	O (n log n)	O (n log n)	O (n^2)
Merge Sort	O (n log n)	O (n log n)	O (n log n)
Binary Search	O (1)	O (log n)	O (log n)

Table.1. Complexity of Divide and Conquer Methodology

7. CONCLUSION

In this survey, we have presented the overview of performance analysis of divide and conquer methodology, studying the various applications of binary search, quick sort & merge sort algorithms, calculate the performance analysis of time complexity, and example of divide and conquer applications. In this paper can be useful for analysis of algorithms. In proposed method, various new algorithms can be implemented in divide and conquer methodology.

REFERENCES

[1] The Design and Analysis of Algorithms by Nitin Upadhayay
[2] Algorithms Design and Analysis by Udit Agarwal
[3] Introduction to the Design and Analysis of Algorithms by Anany Levitin
[4] Fundamentals of Computer Algorithms by Ellis Horowitz
[5] Rohit Yadav and Kratika Nitin Verma, Analysis of Recursive and Non-recursive Merge Sort Algorithm, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 11, November 2013
[6] Rami Mansi, Enhanced Quick sort Algorithm, The International Arab Journal of Information Technology, Vol. 7, No. 2, April 2010
[7] Reyha Verma1 and Jasbir Singh, A Comparative Analysis of Deterministic Sorting Algorithms based on Runtime and Count of Various Operations, International Journal of Advanced Computer Research, ISSN (Print): 2249-7277 ISSN (Online): 2277-7970 Volume-5 Issue-21 December-2015
[8] Pooja et al., Comparative Analysis & Performance of Different Sorting Algorithm in Data Structure International Journal of Advanced Research in Computer Science and Software Engineering Volume 3 Issue 11, November 2013, pp.500-507
[9] Sushma Nallamalli and M. M. Vamsi Priya, Performance Analysis of Divide and Conquer Sorting Algorithms, International Journal of Advanced Technology in Engineering and Science, Volume 3, Issue 1, January 201,ISSN : 2348 – 7550,pp.170-180.