

## CHAPTER IV

### DIVIDE - and - CONQUER

Divide and conquer is the best known algorithm design technique. Divide and conquer algorithms work according to following general plan

1. A problem's instance is divided into several

smaller problems of same size

A. M. PRASAD  
Assistant Professor

Department of Computer Science & Engineering  
Dayananda Sagar College of Engineering  
Bangalore - 560 078

2. The smaller instances are solved.

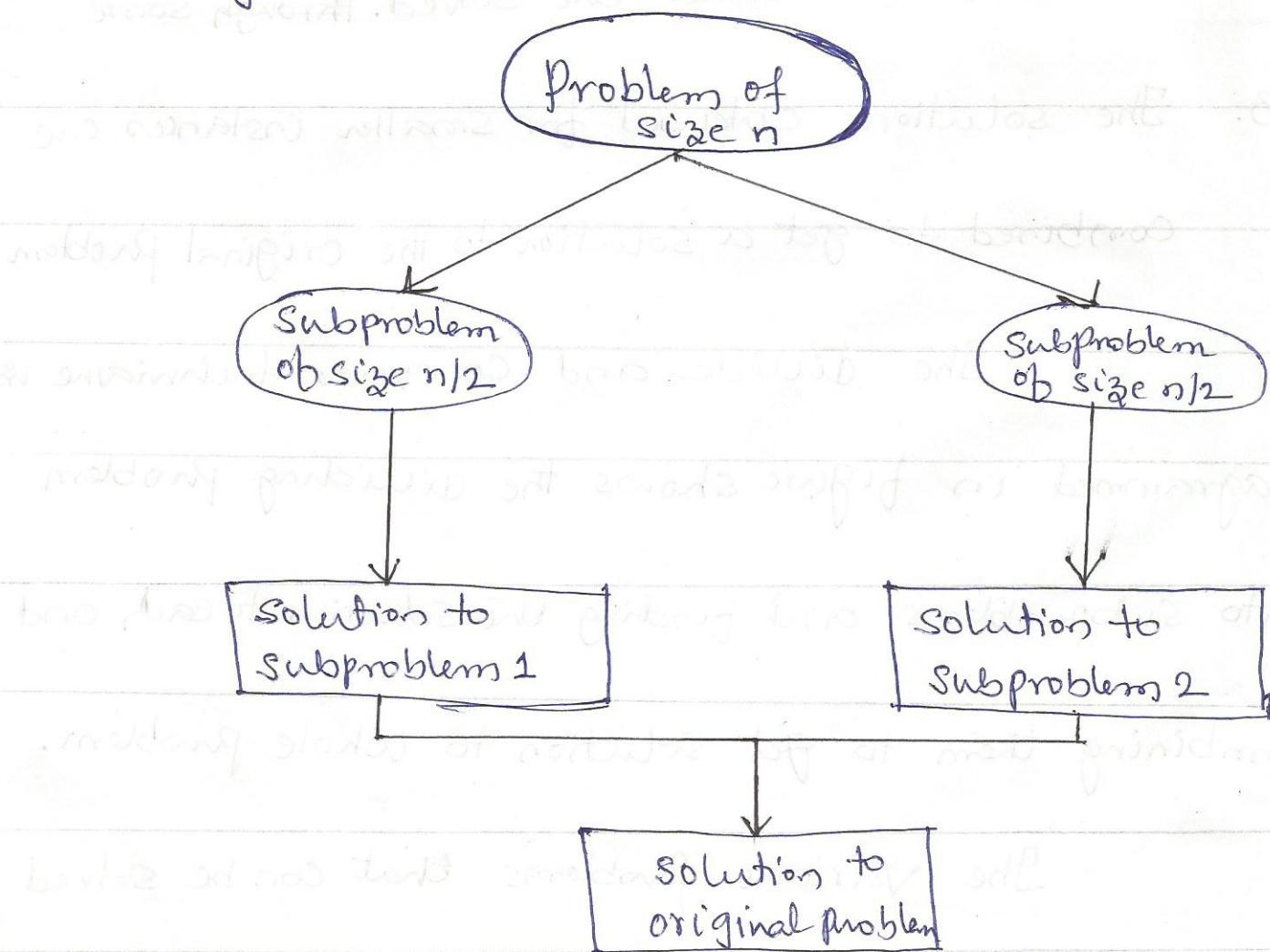
3. The solutions obtained for smaller instances are combined to get a solution to the original problem

The divide and conquer technique is diagrammed in figure shows the dividing problem into subproblems and finding the solution to each and combining them to get solution to whole problem.

The various problems that can be solved using divide and conquer approach are

1. Quick Sort
2. Merge Sort
3. Binary Search
4. Binary tree traversals.
5. Multiplication of numbers
6. Matrix Multiplication using Strassen's algorithm

The divide and conquer technique is shown pictorially as shown below



# QUICK SORT

Quick sort is the divide and conquer

method used to arrange numbers in Ascending order.

A. M. PRASAD

Assistant Professor

Department of Computer Science & Engineering

Dayananda Sagar College of Engineering

Bangalore - 560 078

## Method:

The Quick sort method assumes the

first element of the list of 'n' numbers as key or pivot element.

Then the key element is compared with the elements

one by one from second element until the key is greater. Then

the key element is compared from  $n^{th}$  element one by one

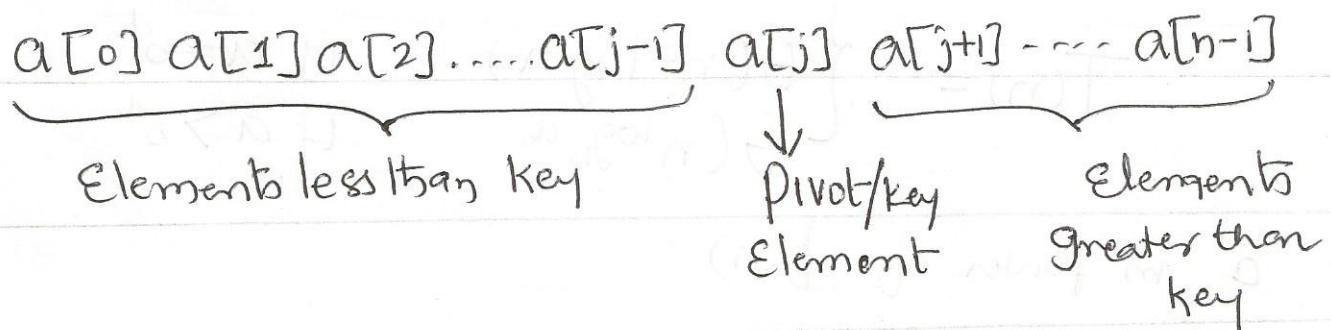
until the key element is less. Then compare the position

of two pointers i and j, if  $i < j$  swap  $i^{th}$  and  $j^{th}$

elements else swap pivot element and  $j^{th}$  element.

After swapping key element is at  $j^{th}$  position, to left are

less and to right are greater than key elements.



Now sort left part of the array  $a[0] \dots a[j-1]$  recursively

using the same method and sort right part of the array

The time efficiency of divide and conquer algorithms can be computed using

### 1. Master theorem

### 2. Back substitution method.

## Master Theorem

An instance of size  $n$  can be divided into

Several instances say  $a$  of size  $n/b$  then general

divide and conquer recurrence relation is

$$T(n) = aT(n/b) + f(n)$$

\* where  $a$  and  $b$  are positive constants

\*  $f(n)$  is time required to divide and combine subproblems

Time Efficiency can be calculated using following relation

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^{d \log n}) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

$d$  is power of  $b$

$a[j+1]$  to  $a[n-1]$  recursively.

## Illustration of Quicksort

Eg:

Key  $\rightarrow$   $\boxed{9}$   $i^o$   $12, 4, 16, 3, 15, 5, 18, 6, 14$   $j^o$   
false less, move  $j$  to 6 and false

$\boxed{9} \quad 12 \quad 4 \quad 16 \quad 3 \quad 15 \quad 5 \quad 18 \quad 6 \quad 14 \quad i^o \quad j^o$   
swap

$\boxed{9} \quad 6 \quad 4 \quad 16^o \quad 3 \quad 15 \quad 5^o \quad 18 \quad 12 \quad 14 \quad i^o \quad j^o$   
swap

$\boxed{9} \quad 6 \quad 4 \quad 5 \quad 3^o \quad 15^o \quad 16 \quad 18 \quad 12 \quad 14$

Now  $i^o$  is more than  $j^o$ , so swap pivot and  $j^o$  item

3, 6, 4, 5,  $\boxed{9} \quad j^o \quad 15, 16, 18, 12, 14$

After swapping key is exactly in middle, to left of key are less and right are greater than key.

Now sort Low to  $j^o-1$  recursively and  $j^o+1$  to high

elements recursively.

A. M. PRASAD

Assistant Professor

Department of Computer Science & Engineering

Dayananda Sagar College of Engineering

Bangalore - 560 078

Since Quicksort method works on partitioning the set of elements, it is also called as PARTITION EXCHANGE Sort. The partitioning of should be done until each subpartition can have maximum two elements.

ie.  $low < high$

Algorithm: Quicksort ( $a$ ,  $low$ ,  $high$ )

//  $a$ : array containing unsorted elements

//  $low$ : position of first element

//  $high$ : position of last element

// O/P: array  $a$  consisting sorted elements.

if ( $low < high$ ) then

begin     $j \leftarrow \text{partition}(a, low, high)$

Quicksort ( $a$ ,  $low$ ,  $j-1$ )

Quicksort ( $a$ ,  $j+1$ ,  $high$ )

endif

Algorithm: partition ( $a$ ,  $low$ ,  $high$ )

begin

Key  $\leftarrow a[low]$

$i \leftarrow low + 1$

$j \leftarrow high$

for

while ( $key > a[i]$ ) do

$i \leftarrow i + 1$

while ( $key < a[j]$ ) do

$j = j - 1$

if ( $i < j$ ) then swap ( $a[i], a[j]$ )

Else swap ( $a[low], a[j]$ ), return  $j$

Endwhile

# TIME EFFICIENCY: The Time complexity of Quick

Sort can be computed using

A. M. PRASAD

Assistant Professor

Department of Computer Science & Engineering  
Dayananda Sagar College of Engineering  
Bangalore - 560 078

① Master theorem

② Substitution method.

The recurrence relation for this

algorithm can be written as

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

↑    ↖  
 Time required to      Time required to  
 sort left part of      sort right part of  
 the array               the array

Time required to  
partition n elements

So

$$T(n) = \begin{cases} 0 & \text{if } n=1 \\ 2T(n/2) + n & \text{otherwise} \end{cases}$$

1. By master theorem: The general relation is

$T(n) = aT(n/b) + f(n)$ , the recurrence

relation of Quicksort is

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

By comparing, we have  $a=2$ ,  $b=2$ ,  $f(n)=n^d=n^1=n^d$ ,  $d=1$

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log_b n) & \text{if } a = b^d \\ \Theta(n \log_b a) & \text{if } a > b^d \end{cases}$$

Here  $a=2$ ,  $b^d = b = 2^1 = 2$ . So second relation  $a = b^d$  holds good and time efficiency is given by

$$\begin{aligned} T(n) &= \Theta(n^d \log_b n) \\ &= \Theta(n \log_2 n) \end{aligned}$$

## 2. By Substitution method

$$\begin{aligned} T(n) &= T(n/2) + T(n/2) + n \\ &= 2T(n/2) + n \end{aligned}$$

Let, number of elements ' $n$ ' =  $2^k$

$$\text{so } T(2^k) = 2T\left(\frac{2^k}{2}\right) + 2^k$$

$$T(2^k) = 2T(2^{k-1}) + 2^k, \text{ Substitute for } T(2^{k-1})$$

$$= 2 \left[ 2T(2^{k-2}) + 2^{k-1} \right] + 2^k$$

$$= 2^2 T(2^{k-2}) + 2 \cdot 2^{k-1} + 2^k$$

$$= 2^2 T(2^{k-2}) + 2^k + 2^k$$

$$= 2^2 T(2^{k-2}) + 2 \cdot 2^k$$

Again,  $T(2^{k-2})$  can be substituted

$$T(2^k) = 2^3 T(2^{k-3}) + 3 \cdot 2^k$$

A. M. PRASAD

Assistant Professor

Department of Computer Science & Engineering

Dayananda Sagar College of Engineering

Bangalore - 560 078

When  $k^{th}$  value is substituted

$$\begin{aligned} T(2^k) &= 2^k T(2^{k-k}) + k \cdot 2^k \\ &= 2^k T(2^0) + k \cdot 2^k \\ &= 2^k T(0) + k \cdot 2^k \end{aligned}$$

Time required to sort 1 element is zero, so

$$T(2^k) = k \cdot 2^k, \text{ but } n = 2^k$$

$$= n \cdot k, \quad k = \log_2 n$$

$$T(n) = \Theta(n \log_2 n)$$

So, time complexity of Quick Sort is

$$T(n) = \Theta(n \log_2 n)$$

## Merge Sort

Merge Sort is divide and conquer technique used to arrange numbers in ascending order. The different steps of merge sort are

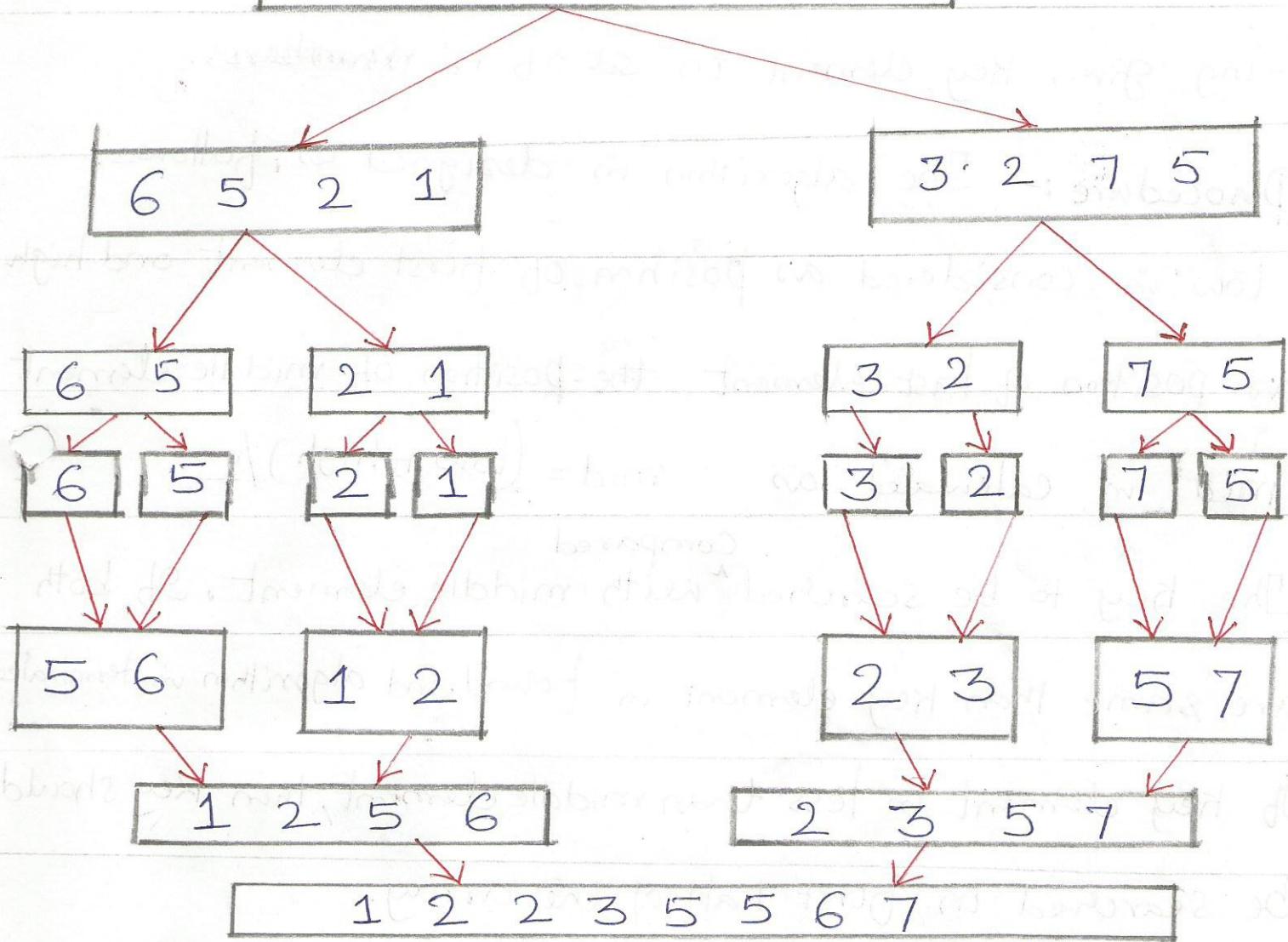
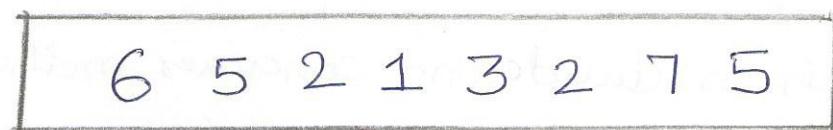
1. Divide array of  $n$  elements to two subarrays of  $n/2$  each; left part and right part
2. Sort left array and right array recursively
3. Merge sorted left part and sorted right part to get single sorted array.

The array is divided into subarrays until only one element is in subarray. we  $\text{low} \leq \text{high}$

Illustration: Consider the following elements

6, 5, 2, 1, 3, 2, 7, 5

The trace of merge sort can be shown in tree.



Algorithm Mergesort( $a$ , low, high)

Begin if ( $low \leq high$ )

$mid = low + high / 2$

    Mergesort( $a$ , low, mid)

    Mergesort( $a$ , mid+1, high)

    merge( $a$ , low, mid, high)

Endif

A. M. PRASAD

Assistant Professor

Department of Computer Science & Engineering

Dayananda Sagar College of Engineering

Bangalore - 560 078

merge( $low$ ,  $mid$ ,  $high$ )

begin  $i = low$ ,  $j = mid + 1$ ,  $k = low$

    while ( $i \leq mid$  and  $j \leq high$ )

        if ( $a[i] < a[j]$ )

$b[k] = a[i]$ ,  $i \leftarrow i + 1$ ,  $k \leftarrow k + 1$

        else

$b[k] = a[j]$ ,  $j \leftarrow j + 1$ ,  $k \leftarrow k + 1$

    endwhile

    while ( $i \leq mid$ )  $b[k] = a[i]$   
 $i \leftarrow i + 1$ ,  $k \leftarrow k + 1$

    while ( $j \leq high$ )  $b[k] = a[j]$   
 $j \leftarrow j + 1$ ,  $k \leftarrow k + 1$

## BINARY SEARCH

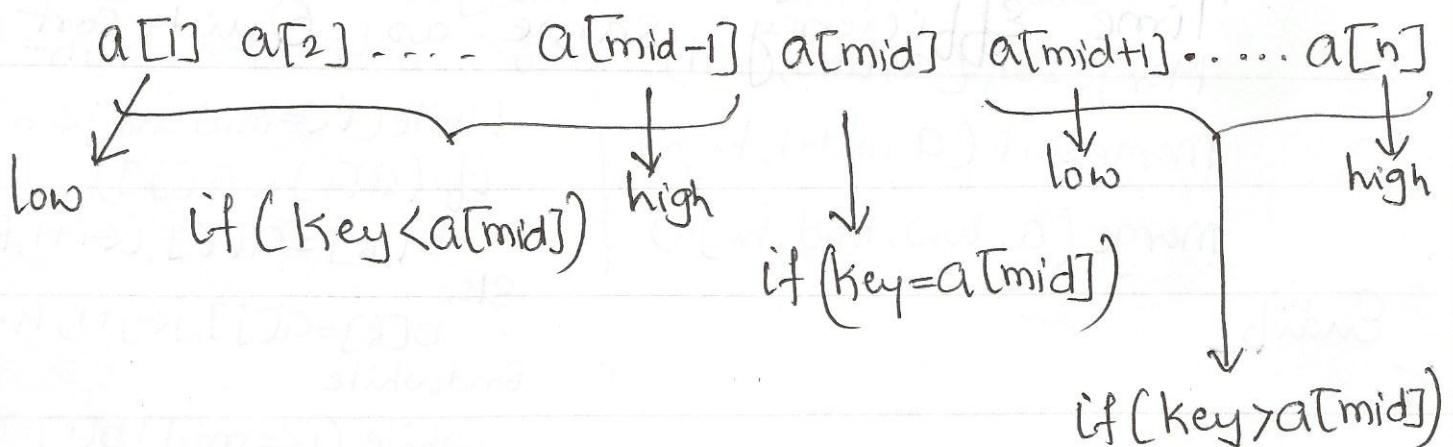
Binary Search is divide and conquer method of searching given key element in set of 'n' numbers.

Procedure :- The algorithm is designed as follows,  
low is considered as position of first element and high  
as position of last element, the position of middle element  
'mid' is calculated as  $\text{mid} = (\text{low} + \text{high}) / 2$

The key to be searched is compared with middle element. If both  
are same then key element is found and algorithm is terminated.

If key element is less than middle element, then key should  
be searched in first half of the array.

If key element is greater than middle element, then key  
should be searched in second half of the array.



## Algorithm:

Algorithm binary Search (a, n, key)

// Key is the element to be searched

// n is the number of elements

// a is the array which contains elements

begin

low  $\leftarrow$  1

high  $\leftarrow$  n

while ( low  $\leq$  High) do

    mid  $\leftarrow$  (low+high)/2

    if ( Key = a[mid] ) return mid

    if ( Key < a[mid] ) then

        high = mid-1

    else

        low = mid+1

    Endif

Endwhile

return -1

End

A. M. PRASAD

Assistant Professor

Department of Computer Science & Engineering

Dayananda Sagar College of Engineering

Bangalore - 560 078

## TIME EFFICIENCY:

The Best case occurs if the item is found in the first comparison. So the time efficiency is  $T(n) = \Omega(1)$  ie if item is in middle

The Worst case occurs if the item is present in first or last position. The recurrence relation is

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

↓      ↳ Time required to compare  
Time required to search      middle element  
either upper or lower part

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

$$\text{Let } n = 2^k$$

$$T(2^k) = T\left(\frac{2^k}{2}\right) + 1$$

$$= T(2^{k-1}) + 1$$

Now substitute for  $T(2^{k-1})$

$$\begin{aligned} T(2^k) &= [T(2^{k-2}) + 1] + 1 \\ &= T(2^{k-2}) + 2 \end{aligned}$$

Now we can substitute for  $T(2^{k-2})$

$$= T(2^{k-3}) + 3$$

by substituting again for function  $T(2^{k-i})$  till  
 $i$  reaches  $k$ , when  $i=k$ .

A. M. PRASAD  
Assistant Professor

Department of Computer Science & Engineering  
Dayananda Sagar College of Engineering  
Bangalore - 560 078

$$T(n) = T(2^{k-k}) + k$$

$$= T(2^0) + k$$

$$= T(1) + k$$

$T(1)$  is time taken to search one element in 1

$$= 1 + k$$

But  $n = 2^k$  so  $k = \log_2 n$

$$= 1 + \log_2 n, \text{ constant cannot be considered}$$

$$T(n) = \log_2 n$$

So

$$T(n) = O(\log_2 n)$$

The average case is when  $n=1, 2, 4, 8, 16$  etc.

When  $n=1, 2^0$  no of comparison is 1

$n=2, 2^1 \rightarrow 11 \rightarrow 11 \rightarrow 2$

$n=4, 2^2 \rightarrow 11 \rightarrow 11 \rightarrow 11 \rightarrow 3$

$n=8, 2^3 \rightarrow 11 \rightarrow 11 \rightarrow 11 \rightarrow 11 \rightarrow 4$

$n=16, 2^4 \rightarrow 11 \rightarrow 11 \rightarrow 11 \rightarrow 11 \rightarrow 5$

$n=2^{k-1} \rightarrow 11 \rightarrow 11 \rightarrow \dots \rightarrow k$

In general  $T(n) = \frac{1}{n} \sum_{i=1}^k i \cdot 2^{i-1}$

$$2^{i-1} = \frac{2^i}{2} = 2^i - \frac{2^i}{2} = 2^i - 2^{i-1}$$

$$= \frac{1}{n} \sum_{i=1}^k i (2^i - 2^{i-1})$$

$$= \frac{1}{n} \left\{ \sum_{i=1}^k i 2^i - \sum_{i=1}^k i 2^{i-1} \right\}$$

$$= \frac{1}{n} \left\{ \sum_{i=1}^k i \cdot 2^i - \sum_{i=1}^k (i-1) 2^i \right\}$$

$$= \frac{1}{n} \left\{ \sum_{i=1}^k i \cdot 2^i - \sum_{i=1}^k i \cdot 2^i - \sum_{i=1}^k 2^i \right\}$$

$$= \frac{1}{n} K \cdot 2^K - \sum_{i=1}^K 2^i$$

$$= \frac{1}{n} k 2^k [1+2^1+2^2+\dots+2^{k-1}]$$

$$= \frac{1}{n} \left\{ k \cdot 2^k - 1 \left[ \frac{2^{k-1}}{2-1} \right] \right\} \text{ using } S = a \left[ \frac{r^n - 1}{r - 1} \right]$$

where  $a=1, r=2, n=k$

$$= \frac{1}{n} \{ kn - 1(n-1) \}$$

$$= \frac{1}{n} \{ kn - n + 1 \}$$

$$= k-1 + \frac{1}{n}$$

it is approximated to  $k-1$

$$T(n) = k-1 \quad \text{But } n = 2^{k-1}$$

Substituting  $k-1$

$$(k-1) = \log_2 n$$

$$T(n) = \Theta(\log_2 n)$$

So average case is

$$T(n) = \Theta(\log_2 n)$$

**A. M. PRASAD**

Assistant Professor

Department of Computer Science & Engineering

Dayananda Sagar College of Engineering

Bangalore - 560 078