AKSHAYA INSTITUTE OF TECHNOLOGY

Lingapura, Tumkur-Koratagere Road, Tumkur-572106.

Module 2 Presentation

Design & Analysis of Algorithms

AKSHAYA INSTITUTE OF TECHNOLOGY

Lingapura, Obalapura Post, Koratagere Road, Tumakuru - 572106

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VISION

To empower the students to be technically competent, innovative and self-motivated with human values and contribute significantly towards betterment of society and to respond swiftly to the challenges of the changing world.

MISSION

M1: To achieve academic excellence by imparting in-depth and competitive knowledge to the students through effective teaching pedagogies and hands on experience on cutting edge technologies.

M2: To collaborate with industry and academia for achieving quality technical education and knowledge transfer through active participation of all the stake holders.

M3: To prepare students to be life-long learners and to upgrade their skills through Centre of Excellence in the thrust areas of Computer Science and Engineering.

EST 2009 AKSHAYA TUMKUR

Program Specific Outcomes (PSOs)

After Successful Completion of Computer Science and Engineering Program Students will be able to

* Apply fundamental knowledge for professional software development as well as to acquire new skills.
* Implement disciplinary knowledge in problem solving, analyzing and decision-making abilities through different domains like database management, networking, algorithms, and programming as well as research and development.
* Make use of modern computer tools for creating innovative career paths, to become an entrepreneur or desire for higher studies.

Program Educational Objectives (PEOs)

PEO1: Graduates expose strong skills and abilities to work in industries and research organizations.

PEO3: Graduates engage in team work to function as responsible professional with good ethical behavior and leadership skills.

PEO3: Graduates engage in life-long learning and innovations in multi disciplinary areas.

SYLLABUS

UNIT – I

INTRODUCTION: Algorithm, Performance Analysis-Space complexity, Time complexity, Asymptotic Notations- Big oh notation, Omega notation, Theta notation and Little oh notation. Divide and conquer: General method, applications-Binary search, Quick sort, Merge sort, Stassen's matrix multiplication.

UNIT – II

Disjoint Sets: Disjoint set operations, union and find algorithms Backtracking: General method, applications, n-queen's problem, sum of subsets problem, graph coloring

UNIT - III

Dynamic Programming: General method, applications-Optimal binary search trees, 0/1 knapsack problem, All pairs shortest path problem, Traveling sales person problem, Reliability design.

UNIT – IV

Greedy method: General method, applications-Job sequencing with deadlines, knapsack problem, Minimum cost spanning trees, Single source shortest path problem.

UNIT - V

Branch and Bound: General method, applications - Travelling sales person problem, 0/1 knapsack problem - LC Branch and Bound solution, FIFO Branch and Bound solution. NP-Hard and NP-Complete problems: Basic concepts, non deterministic algorithms, NP - Hard and NP-Complete classes, Cook's theorem.

TEXT BOOKS

1. Fundamentals of Computer Algorithms, Ellis Horowitz, Satraj Sahni and Rajasekharan,3rd Edition University Press.

REFERENCES

- Design and Analysis of Algorithms, Aho, Ullman and, Pearson education.
- Introduction to Algorithms, second edition, T.H. Coremen, C.E Leiserson, R.L.Rivest and C. Stien, PHI Pvt . Ltd./Pearson Education.
- Algorithm Design; Foundations, Analysis and Internet Examples, M.T. Goodrich and R. Tamassia, John Wiley and sons.

Euclid’s algorithm for GCD

- An algorithm is an effective method for finding out the solution for a given problem. It is a sequence of instruction
That conveys the method to address a problem

- **Algorithm :** Step by step procedure to solve a computational problem is called Algorithm.

or

- An Algorithm is a step-by-step plan for a computational procedure that possibly begins with an input and yields an output value in a finite number of steps in order to solve a particular problem.

Pseudo Code of the Algorithm

Pseudo Code of the Algorithm-

```
Step 1: Let a, b be the two numbers
Step 2: a mod b = R
Step 3: Let a = b and b = R
Step 4: Repeat Steps 2 and 3 until a mod
b is greater than 0
Step 5: GCD = b
Step 6: Finish
```

PROPERTIES OF ALGORITHM

TO EVALUATE AN ALGORITHM WE HAVE TO SATISFY THE FOLLOWING CRITERIA:

1.INPUT: The Algorithm should be given zero or more input.

2.OUTPUT: At least one quantity is produced. For each input the algorithm produced value from specific task.

3.DEFINITENESS: Each instruction is clear and unambiguous.

4.FINITENESS: If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.

5.EFFECTIVENESS: Every instruction must very basic so that it can be carried out, in principle, by a person using only pencil & paper.

ALGORITHM (CONTD…)

- A well-defined computational procedure that takes some value, or set of values, as *input* and produces some value, or set of values, as *output*.

- Written in a pseudo code which can be implemented in the language of programmer's choice.

PSEUDO CODE: A notation resembling a simplified programming language, used in program design.

How To Write an Algorithm

```
Step-1:start
Step-2:Read a,b,c
Step-3:if a>b
        if a>c
        print a is largest
        else
        if b>c
        print b is largest
        else
        print c is largest
Step-4 : stop
```

```
Step-1: start
Step-2: Read a,b,c
Step-3:if a>b then go to step 4
       otherwise go to step 5
Step-4:if a>c then
   print a is largest  otherwise
  print c is largest
Step-5: if b>c then
  print b is largest   otherwise
  print c is largest
step-6: stop
```

Differences

Algorithm	Program
1.At design phase	1.At Implementation phase
2.Natural language	2.written in any programming language
3.Person should have Domain knowledge	3.Programmer
4.Analyze	4.Testing

ALGORITHM SPECIFICATION

Algorithm can be described (Represent) in four ways.

1.Natural language like English:

When this way is chooses, care should be taken, we should ensure that each & every statement is definite. (no ambiguity)

2. Graphic representation called flowchart:

This method will work well when the algorithm is small& simple.

3. Pseudo-code Method:

In this method, we should typically describe algorithms as program, which resembles language like Pascal & Algol(Algorithmic Language).

4.Programming Language:

we have to use programming language to write algorithms like C, C++,JAVA etc.

PSEUDO-CODE CONVENTIONS

1. Comments begin with // and continue until the end of line.

2. Blocks are indicated with matching braces { and }.

3. An identifier begins with a letter. The data types of variables are not explicitly declared.

```
node= record
    {
    data type 1 data 1;
    data type n data n;
    node *link;
    }
```

4. There are two Boolean values **TRUE** and **FALSE**.

Logical Operators

AND, OR, NOT

Relational Operators

<, <=,>,>=, =, !=

5. Assignment of values to variables is done using the assignment statement.

```
<Variable>:= <expression>;
```

6. Compound data types can be formed with records. Here is an example,
Node. Record

```
{
 data type – 1   data-1;
.
.
.
 data type – n  data – n;
  node * link;
}
```

Here link is a pointer to the record type node. Individual data items of a record can be accessed with → and period.

Contd…

7. The following looping statements are employed.

For, while and repeat-until While Loop:

```
While < condition > do
 {
      <statement-1>
          ..
          ..
    <statement-n>
 }
```

For Loop:

```
       For variable: = value-1 to value-2 step step do

{
       <statement-1>
                 .
                 .
                 .
<statement-n>
}
```

repeat-until:

```
repeat
        <statement-1>
                .
                .
                .
<statement-n>
until<condition>
```

8. A conditional statement has the following forms.

```
→ If <condition> then <statement>
→ If <condition> then <statement-1>
  Else <statement-1>
```

Case statement:

```
Case
{
        : <condition-1> : <statement-1>
                                       .
                                       .
                                       .
        : <condition-n> : <statement-n>
        : else : <statement-n+1>
}
```

9. Input and output are done using the instructions read & write. No format is used to specify the size of input or output quantities

Contd…

10. There is only one type of procedure: Algorithm, the heading takes the form,

Algorithm Name (Parameter lists)

consider an example, the following algorithm fields & returns the maximum of n given numbers:

```
algorithm Max(A,n)
// A is an array of size n
{
    Result := A[1];
    for i:= 2 to n do
        if A[i] > Result then
            Result :=A[i];
    return Result;
}
```

Issue in the study of algorithm

1. **How to create an algorithm.**
2. **How to validate an algorithm.**
3. **How to analyses an algorithm**
4. **How to test a program.**

1 .How to create an algorithm: To create an algorithm we have following design technique

a) Divide & Conquer
b) Greedy method
c) Dynamic Programming
d) Branch & Bound
e) Backtracking

2.How to validate an algorithm: Once an algorithm is created it is necessary to show that it computes the correct output for all possible legal input , this process is called algorithm validation.

3.How to analyses an algorithm: Analysis of an algorithm or performance analysis refers to task of determining how much computing Time & storage algorithms required.

a) Computing time-Time complexity: Frequency or Step count method
b) Storage space- To calculate space complexity we have to use number of input used in algorithms.

4.How to test the program: Program is nothing but an expression for the algorithm using any programming language. To test a program we need following

a) Debugging: It is processes of executing programs on sample data sets to determine whether faulty results occur & if so correct them.
b) Profiling or performance measurement is the process of executing a correct program on data set and measuring the time & space it takes to compute the result.

ANALYSIS OF ALGORITHM

PRIORI	POSTERIORI
1.Done priori to run algorithm on a specific system	1.Analysis after running it on system.
2.Hardware independent	2.Dependent on hardware
3.Approximate analysis	3.Actual statistics of an algorithm
4.Dependent on no of time statements are executed	4.They do not do posteriori analysis

Problem: **Suppose there are 60 students in the class. How will you calculate the number of absentees in the class?**

Pseudo Approach

```
1.Initialize a variable called as Count to zero, absent to
  zero, total to 60
2.FOR EACH Student PRESENT DO the following:
  Increase the Count by One
3.Then Subtract Count from total and store the result
  in absent
4.Display the number of absent students
```

Problem: **Suppose there are 60 students in the class. How will you calculate the number of absentees in the class?**

Algorithmic Approach:

```
1.Count <- 0, absent <- 0, total <- 60
2.REPEAT till all students counted
   Count <- Count + 1
3.absent <- total - Count
4.Print "Number absent is:" , absent
```

Need of Algorithm

1. To understand the basic idea of the problem.
2. To find an approach to solve the problem.
3. To improve the efficiency of existing techniques.
4. To understand the basic principles of designing the algorithms. To compare the performance of the algorithm with respect to other techniques.
6. It is the best method of description without describing the implementation detail.
7. The Algorithm gives a clear description of requirements and goal of the problem to the designer.
8. A good design can produce a good solution.
9. To understand the flow of the problem.

PERFORMANCE ANALYSIS

Performance Analysis: An algorithm is said to be efficient and fast if it take less time to execute and consumes less memory space at run time is called Performance Analysis.

1. **SPACE COMPLEXITY:**

The space complexity of an algorithm is the amount of Memory Space required by an algorithm during course of execution is called space complexity .There are three types of space

a) Instruction space :executable program
b) Data space: Required to store all the constant and variable data space.
c) Environment: It is required to store environment information needed to resume the suspended space.

2. **TIME COMPLEXITY:**

The time complexity of an algorithm is the total amount of time required by an algorithm to complete its execution.

Space complexity

Now there are two types of space complexity

a) Constant space complexity

b) Linear(variable)space complexity

1.Constant space complexity: A fixed amount of space for all the input values.

Example : int square(int a)

```
                {
                        return a*a;
                }
```

Here algorithm requires fixed amount of space for all the input values.

2.Linear space complexity: The space needed for algorithm is based on size.

- Size of the variable 'n' = 1 word
- Array of a values = n word
- Loop variable = 1 word
- Sum variable = 1 word

Example:

```
int sum(int A[],int n)
{                          n
int sum=0,i;               1
for (i=0;i<n;i++)          1
Sum=sum+A[i];              1
Return sum;
}
```

Ans : 1+n+1+1 = n+3 words

1. Space Complexity

The space complexity of an algorithm is the amount of memory it needs to run to complete.

Space needed by an algorithm is given by

S(P)=C(fixed part)+Sp(Variable part)

fixed part: independent of instance characteristics.

eg: space for simple variables, constants etc

Variable part: Space for variables whose size is dependent on particular problem instance.

Examples:

```
1.Algorithm sum(a,,b,c)
{
a=10;             a-1
b=20;              b-1
c=a+b;            c-1
}
s(p)=c+sp
        3+0=3
     o(n)=3
```

```
2. algorithm sum(a,n)
{
total-=0;        - 1
Fori=1 to n do -1,1
Total=total+a[i]--n
Return total
```

Algorithm-1

```
Algorithm abc(a,b,c)
{
return a+b*c+(a+b-c)/(a+b) +4.0;

}
```

a→1
b→1
c→1

>= 3 units

Algorithm-3

```
Algorithm RSum(a,n)
{
if(n≤0) then return 0.0;
else return Rsum(a,n-1)+a[n];
}
```

Algorithm-2

```
• Algorithm sum(a,n)
• {
• s=0.0;
• for i=1 to n do
•  s= s+a[i];
•  return s;
• }
```

i,n,s→1 unit each
a→n units

>= n+1 units

DAA

Rsum(a,n)→1(a[n])+1(n)+1(return)=3units
Rsum(a,n-1)→1(a[n-1])+1(n)+1(return)
.
.
.
Rsum(a,n-n)→1(a[n-n])+1(n)+1(return)

Total→>=3(n+1) units

Algorithm-1 Algorithm-2 Algorithm-3:recursive procedure

2. Time Complexity:

The time complexity of an algorithm is the amount of computer time it needs to run to complete.

T(P)= compile time+ execution time

T(P)= tp (execution time)

DAA

Step count:

- For algorithm heading→0
- For Braces→0
- For expressions→1
- for any looping statements→no.of times the loop is repeating.

1.Constant time complexity : If a program required fixed amount of time for all input values is called Constant time complexity .

Example : int sum(int a , int b)

```
{
return a+b;
}
```

2.Linear time complexity: If the input values are increased then the time complexity will changes.

- comments = 0 step
- Assignment statement= 1 step
- condition statement= 1 step
- loop condition for n times = n+1 steps
- body of the loop = n steps

Example :
```
int sum(int A[],int n)
{
 int sum=0,i;
 for (i=0;i<n;i++)
 sum=sum+A[i];
     return sum;
```

cost	repetation	total
1	1	1
1+1+1	1+(n+1)+n	2n+2
2	n	2n
1	1	1
		4n+4

Algorithm-1

```
1. Algorithm abc(a,b,c)                       →0
2. {                                          →0
3. return a+b*c+(a+b-c)/(a+b) +4.0;           →1
4. }                                          →0
                                              --------
                                              1 unit
                                              --------
```

Algorithm-3

```
Algorithm RSum(a,n)
{
if(n≤0) then return 0.0;
else return Rsum(a,n-1)+a[n];
}
```

DAA

Algorithm-2

```
1. Algorithm sum(a,n)     →0
2. {                      →0
3. s=0.0;                 →1
4. for i=1 to n do        →n+1
5.  s= s+a[i];            →n
6.  return s;             →1
7. }                      →0
                          --------
                          2n+3
```

$$T(n) = 2 \quad \text{if } n=0$$
$$= 2 + T(n-1) \quad \text{if } n>0$$

$$T(n) = 2 + T(n-1)$$
$$= 2 + (2+T(n-2)) = 2*2+T(n-2)$$
$$=2*2+(2+T(n-3))=2*3+T(n-3)$$
.
.
$$=2*n+T(n-n)= 2n+T(0)$$
$$T(n) \quad =2n+2$$

TIME COMPLEXITY

The time T(p) taken by a program P is the sum of the compile time and the run time(execution time)

Statement	S/e	Frequency	Total
1. Algorithm Sum(a,n)	0	-	0
2. {	0	-	0
3. S=0.0;	1	1	1
4. for i=1 to n do	1	n+1	n+1
5. s=s+a[I];	1	n	n
6. return s;	1	1	1
7. }	0	-	0
Total			2n+3

KINDS OF ANALYSIS

1.Worst-case: (usually)

- T(n) = maximum time of algorithm on any input of size n.

2.Average-case: (sometimes)

- T(n) = expected time of algorithm over all inputs of size n.
- Need assumption of statistical distribution of inputs.

3.Best-case:

- T(n) = minimum time of algorithm on any input of size n.

COMPLEXITY:

Complexity refers to the rate at which the storage time grows as a function of the problem size

Analysis of an Algorithm

- The goal of analysis of an algorithm is to compare algorithm in running time and also Memory management.
- Running time of an algorithm depends on how long it takes a computer to run the lines of code of the algorithm.

Running time of an algorithm depends on

1.Speed of computer

2.Programming language

3.Compiler and translator

Examples: binary search, linear search

ASYMPTOTIC ANALYSIS:

- Expressing the complexity in term of its relationship to know function. This type analysis is called asymptotic analysis.
- The main idea of Asymptotic analysis is to have a measure of efficiency of an algorithm , that doesn't depends on

1. Machine constants.
2. Doesn't require algorithm to be implemented.
3. Time taken by program to be prepare.

ASYMPTOTIC NOTATION

ASYMPTOTIC NOTATION: The mathematical way of representing the Time complexity.

The notation we use to describe the asymptotic running time of an algorithm are defined in terms of functions whose domains are the set of natural numbers.

Definition : It is the way to describe the behavior of functions in the limit or without bounds.

Asymptotic growth: The rate at which the function grows…

“growth rate” is the complexity of the function or the amount of resource it takes up to compute.

Growth rate $\longrightarrow$ **Time +memory**

Classification of growth

1.Growing with the same rate.

2. Growing with the slower rate.

3.Growing with the faster rate.

They are 3 asymptotic notations are mostly used to represent time complexity of algorithm.

1.Big oh (O)notation

2.Big omega (Ω) notation

3.Theta(Θ) notation

4.Little oh notation

5.Little omega(Ω) notation

1.Big oh (O)notation

1.Big oh (O)notation : Asymptotic “less than”(slower rate).This notation mainly represent upper bound of algorithm run time.

Big oh (O)notation is useful to calculate maximum amount of time of execution.

By using Big-oh notation we have to calculate worst case time complexity.

Formula : $f(n) <= c\ g(n)$ $n >= n_o$, $c > 0$, $n_o >= 1$

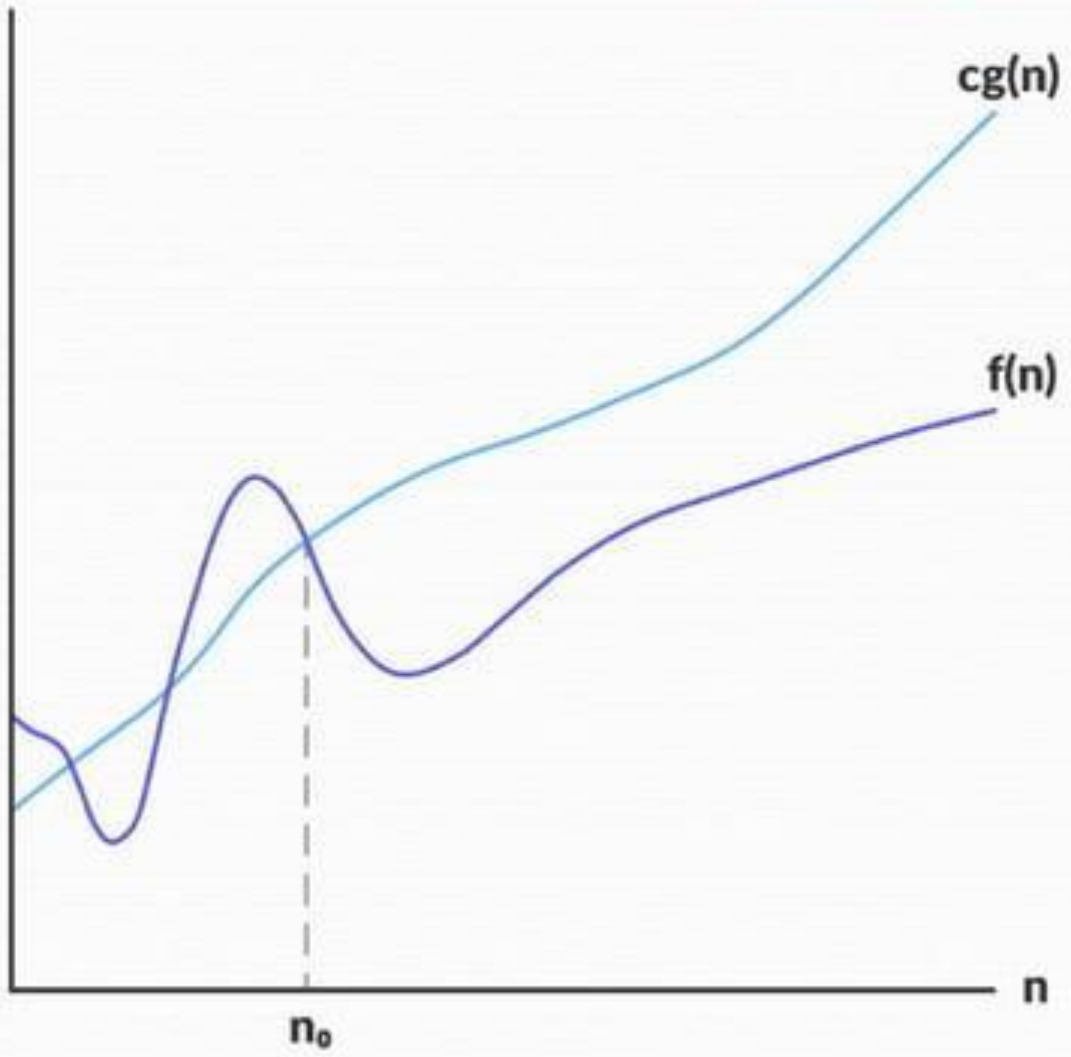
Definition: Let f(n) ,g(n) be two non negative (positive) function now the f(n)=O(g(n)) if there exist two positive constant c,no such that $f(n) <= c.g(n)$ for all value of $n > 0$ & $c > 0$

1.Big O-notation

- For a given function $g(n)$, we denote by $O(g(n))$ the set of functions

$$O(g(n)) = \left\{ \begin{array}{c} f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ s.t.} \\ 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

- We use O-notation to give an asymptotic upper bound of a function, to within a constant factor.
- $f(n) = O(g(n))$ means that there existes some constant c s.t. $f(n)$ is always $\le cg(n)$ for large enough n.



$f(n) = O(g(n))$

Examples

Example : $f(n)=2n+3$ & $g(n)= n$

Formula : $f(n)<=c\ g(n)$ $n>=n_0$, $c>0$, $n_0>=1$

$f(n)=2n+3$ & $g(n)=n$

Now $3n+2<=c.n$

$3n+2<=4.n$

Put the value of n =1

$5<=4$ false

N=2 $8<=8$ true now no>2 For all value of n>2 & c=4

now $f(n)<= c.g(n)$

$3n+2<=4n$ for all value of $n>2$

Above condition is satisfied this notation takes maximum amount of time to execute .so that it is called worst case complexity.

2.Ω-Omega notation

Ω-Omega notation : Asymptotic "greater than"(faster rate).
It represent Lower bound of algorithm run time.

By using Big Omega notation we can calculate minimum amount of time. We can say that it is best case time complexity.

Formula : $f(n) >= c\, g(n)$ $n >= n_0$, $c > 0$, $n_0 >= 1$

where c is constant, n is function

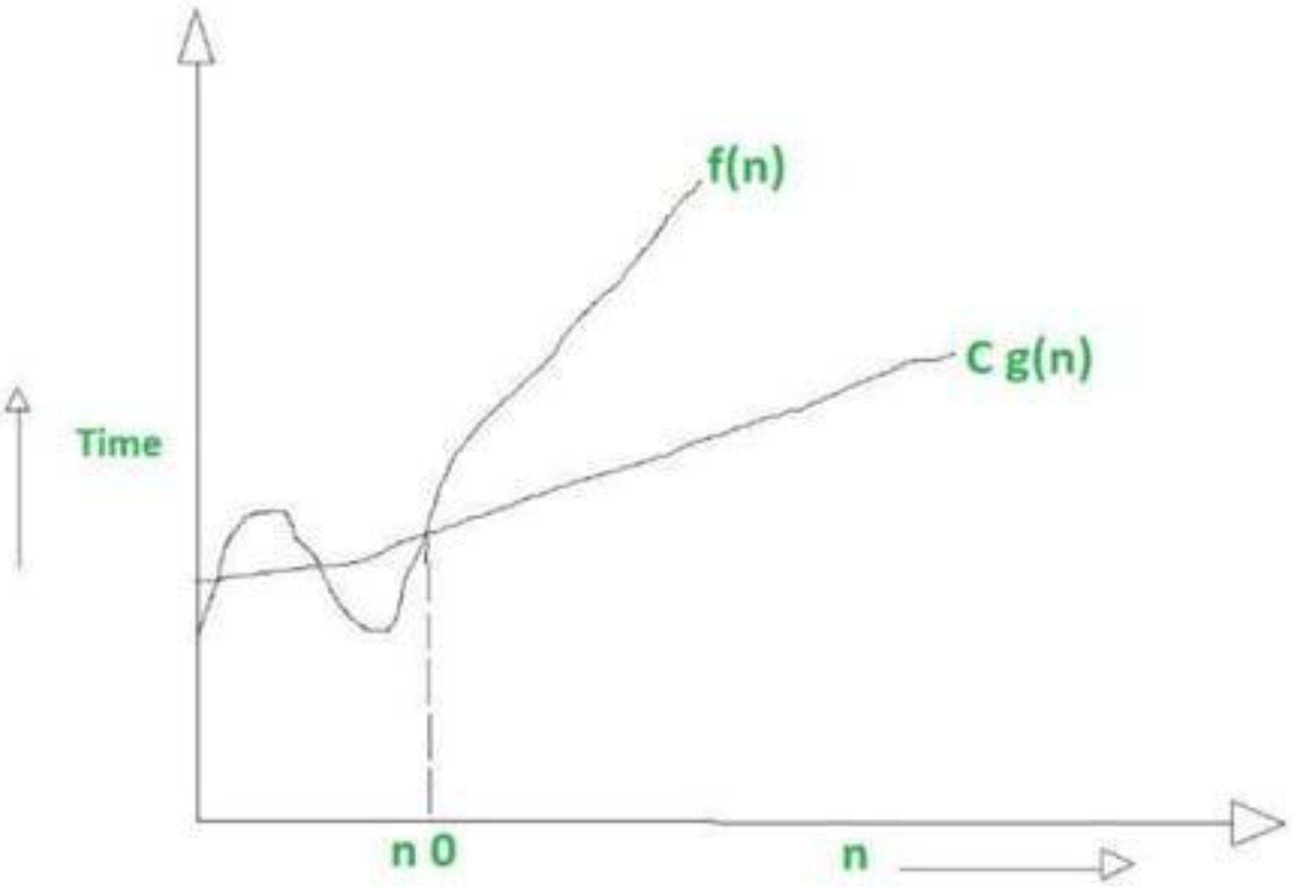
- Lower bound
- Best case

Ω-Omega notation

- For a given function $g(n)$, we denote by $\Omega(g(n))$ the set of functions

$$\Omega(g(n)) = \left\{ \begin{array}{c} f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ s.t.} \\ 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

- We use *Ω*-notation to give an asymptotic lower bound on a function, to within a constant factor.
- $f(n) = \Omega(g(n))$ means that there exists some constant c s.t. $f(n)$ is always $\ge cg(n)$ for large enough n.


f(n)
C g(n)
Time
n 0
n

Examples

Example : $f(n)=3n+2$

Formula : $f(n)>=c\ g(n)$ $n>=n_0$, $c>0$, $n_0>=1$

$f(n)=3n+2$

$3n+2>=1*n$, $c=1$ put the value of $n=1$

$n=1$ $5>=1$ true no>=1 for all value of n

It means that $f(n)= \Omega\ g(n)$.

3.Θ -Theta notation

Theta (Θ) notation : Asymptotic "Equality"(same rate).

It represent average bond of algorithm running time.

By using theta notation we can calculate average amount of time.

So it called average case time complexity of algorithm.

Formula : $c_1\ g(n) <= f(n) <= c_2\ g(n)$

where c is constant, n is function

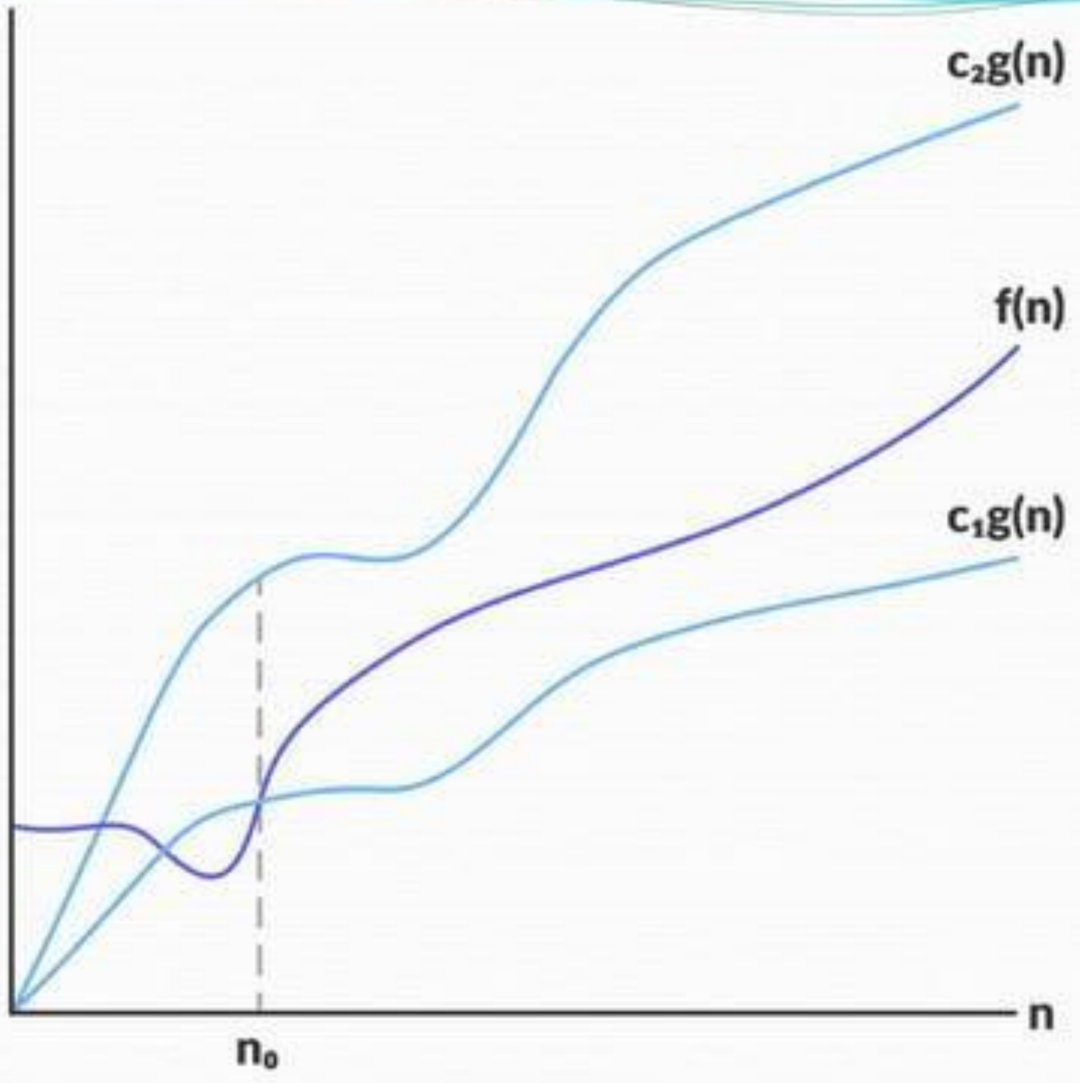
- Average bound

Θ -Theta notation

- For a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions

$$\Theta(g(n)) = \left\{ \begin{array}{c} f(n): \text{there exist positive constants } c_1, c_2, \text{and } n_0 \text{ s.t.} \\ 0 \le \mathrm{c}_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \end{array} \right\}$$

- A function $f(n)$ belongs to the set $\Theta(g(n))$ if there exist positive constants c_1 and c_2 such that it can be "sand-wiched" between $c_1 g(n)$ and $c_2 g(n)$ or sufficienly large n.
- $f(n) = \Theta(g(n))$ means that there exists some constant c_1 and c_2 s.t. $c_1 g(n) \le f(n) \le c_2 g(n)$ for large enough n.



$f(n) = \Theta(g(n))$

Examples

Example : $f(n)=3n+2$

Formula : $c_1\ g(n)<=f(n)<=c_2\ g(n)$

$f(n)=2n+3$

$1*n<=3n+2<=4*n$ now put the value of

$n=1$ we get $1<=5<=4$ false

$n=2$ we get $2<=8<=8$ true

$n=3$ we get $3<=11<=12$ true

Now all value of $n>=2$ it is true above condition is satisfied.

4.Little oh notation

- Little o notation is used to describe an upper bound that cannot be tight. In other words, loose upper bound of f(n).

Slower growth rate

f(n) grows slower than g(n)

- Let f(n) and g(n) are the functions that map positive real numbers. We can say that the function f(n) is o(g(n)) if for any real positive constant c, there exists an integer constant no $\leq$ 1 such that f(n) > 0.

- Using mathematical relation, we can say that f(n) = o(g(n)) means,

  if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

- **Example on little o asymptotic notation:**

1.If $f(n) = n^2$ and $g(n) = n^3$ then check whether

$f(n) = o(g(n))$ or not.

Sol:

$$\lim_{n\to\infty}\frac{n^2}{n^3}$$

$$= \lim_{n\to\infty}\frac{1}{n}$$

$$= \frac{1}{\infty}$$

$$= 0$$

The result is 0, and it satisfies the equation mentioned above. So we can say that f(n) = o(g(n)).

5.Little omega(ω) notation

- Another asymptotic notation is little omega notation. it is denoted by (ω).
- Little omega (ω) notation is used to describe a loose lower bound of f(n).
- Faster growth rate
- F(n) grows faster than g(n)
- If

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \overset{\infty}{\overset{\downarrow}{0}} \qquad \infty$$

Formally stated as f(n)=ωg(n)

Example of asymptotic notation

Problem:-Find upper bond ,lower bond & tight bond range for functions: f(n)= 2n+5

Solution:-Let us given that f(n)= 2n+5 , now g(n)= n

lower bond=2n, upper bond =3n, tight bond=2n

For Big –oh notation(O):- according to definition

f(n)<=cg(n) for Big oh we use upper bond so

f(n)=2n+5, g(n)=n and c=3 according to definition

2n+5<=3n

Put n=1 7<=3 false Put n=2 9<=6 false Put n=3 14<=9 false Put n=4 13<=12 false Put n=5 15<=15 true

now for all value of n>=5 above condition is satisfied. C=3 n>=5

2. Big - omega notation :- f(n)>=c.g(n) we know that this Notation is lower bond notation so c=2

Let f(n)=2n+5 & g(n)=2.n

Now 2n+5>=c.g(n);

    2n+5>=2n put n=1

We get 7>=2 true for all value of n>=1,c=2 condition is satisfied.

3. Theta notation :- according to definition

  c1.g(n)<=f(n)<=c2.g

ANALYSIS OF INSERTION-SORT(CONTD.)

- The worst case: The array is reverse sorted

(t_j =j for j=2,3, ...,n).

$$\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$$

$$T(n) = c_1 n + c_2(n-1) + c_5(n(n+1)/2 - 1)$$
$$+ c_6(n(n-1)/2) + c_7(n(n-1)/2) + c_8(n-1)$$

$$= (c_5/2 + c_6/2 + c_7/2)n^2 + (c_1 + c_2 + c_4 + c_5/2 - c_6/2 - c_7/2 + c_8)n$$

$$T(n) = an^2 + bn + c$$

RANDOMIZED ALGORITHMS

- A **randomized algorithm** is an **algorithm** that employs a degree of randomness as part of its logic.

- The **algorithm** typically uses uniformly **random** bits as an auxiliary input to guide its behavior, in the hope of achieving good performance in the "average case" over all possible choices of **random** bits.

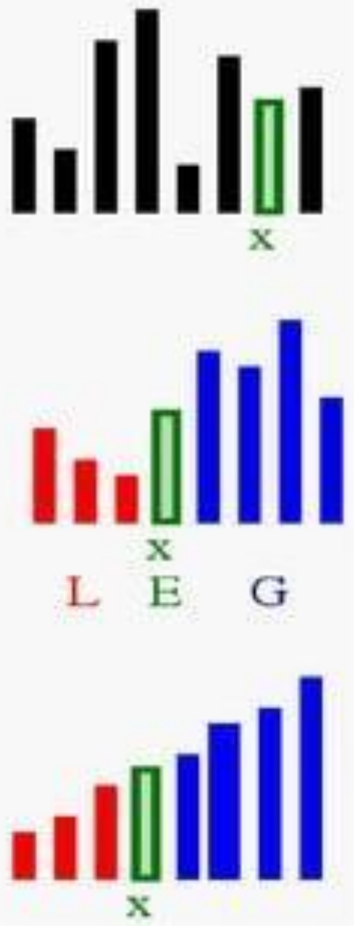
- An algorithm that uses random numbers to decide what to do next anywhere in its logic is called Randomized Algorithm..
- Example: Quick sort

QUICK SORT

Select: pick an arbitrary element x in S to be the pivot.

Partition: rearrange elements so that elements with value less than x go to List L to the left of x and elements with value greater than x go to the List R to the right of x.

Recursion: recursively sort the lists L and R.



DIVIDE AND CONQUER

- Given a function to compute on ‘n’ inputs the divide-and-conquer strategy suggests splitting the inputs into ‘k’ distinct subsets, $1<k<=n$, yielding ‘k’ sub problems.

- These sub problems must be solved, and then a method must be found to combine sub solutions into a solution of the whole.

- If the sub problems are still relatively large, then the divide-and-conquer strategy can possibly be reapplied.

- If the problem p and the size is n , sub problems are n1, n2 ….nk, respectively, then the computing time of D And C is described by the recurrence relation.
- T(n)= { g(n)   n small
- T(n1)+T(n2)+……………+T(nk)+f(n);
- otherwise.

"Where T(n) is the time for D And C on any I/p of size n.

- g(n) is the time of compute the answer directly for small I/p s. f(n) is the time for dividing P & combining the solution to sub problems.

DIVIDE AND CONQUER :GENERAL METHOD

```
Algorithm D And C(P)
{
    if small(P) then return S(P);
    else
    {
        divide P into smaller instances
        P1, P2… Pk, k>=1;
        Apply D And C to each of these sub problems;
        return combine (D And C(P1), D And C(P2),…….,D And C(Pk));
    }
}
```

EXAMPLE

Consider the case in which a=2 and b=2. Let T(1)=2 & f(n)=n. We have,

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

2[2T(n/2/2)+n/2]+n

[4T(n/4)+n]+n

4T(n/4)+2n

4[2T(n/4/2)+n/4]+2n

4[2T(n/8)+n/4]+2n

8T(n/8)+n+2n

8T(n/8)+3n

$2^3T(n/2^3)+3n$

By using substitution method

Let $n=2^k$

$K=logn_2$

$K=3$

$2^kT(n/n)+3n$

$nT(1)+3N$

$2n+kn$

$2n+nlogn$

Time complexity is O(nlogn)

APPLICATIONS

1. **Binary Search** is a searching algorithm. In each step, the algorithm compares the input element x with the value of the middle element in array. If the values match, return the index of middle. Otherwise, if x is less than the middle element, then the algorithm recurs for left side of middle element, else recurs for right side of middle element.

2.Quick sort is a sorting algorithm. The algorithm picks a pivot element, rearranges the array elements in such a way that all elements smaller than the picked pivot element move to left side of pivot, and all greater elements move to right side. Finally, the algorithm recursively sorts the sub arrays on left and right of pivot element.

3.Merge Sort is also a sorting algorithm. The algorithm divides the array in two halves, recursively sorts them and finally merges the two sorted halves.

BINARY SEARCH

```
Algorithm Bin search(a,n,x)
// Given an array a[1:n] of elements in non-decreasing
//order, n>=0,determine whether x is present and
// if so, return j such that x=a[j]; else return 0.
{
low:=1; high:=n;
while (low<=high) do
{
 mid:=[(low+high)/2];
if (x<a[mid]) then high;
else if(x>a[mid]) then
low=mid+1;
else return mid;
}
return 0;  }
```

EXAMPLE

1) Let us select the 14 entries.

–15,6,0,7,9,23,54,82,101,112,125,131,142,151.

Place them in a[1:14] and simulate the steps Binsearch goes through as it searches for different values of x.

Only the variables low, high & mid need to be traced as we simulate the algorithm.

We try the following values for x: 151, -14 and 9.

for 2 successful searches & 1 unsuccessful search.

Table. Shows the traces of Bin search on these 3 steps.

X=151	low	high	mid
	1	14	7
	8	14	11
	12	14	13
	14	14	14
			Found
x=-14	low	high	mid
	1	14	7
	1	6	3
	1	2	1
	2	2	2
	2	1	Not found
x=9	low	high	mid
	1	14	7
	1	6	3
	4	6	5
			Found

MERGE SORT

- Another application of Divide and conquer is merge sort.
- Given a sequence of n elements a[1],…,a[n] the general idea is to imagine then split into 2 sets a[1],…..,a[n/2] and a[[n/2]+1],….a[n].
- Each set is individually sorted, and the resulting sorted sequences are merged to produce a single sorted sequence of n elements.
- Thus, we have another ideal example of the divide-and-conquer strategy in which the splitting is into 2 equal-sized sets & the combining operation is the merging of 2 sorted sets into one.

ALGORITHM FOR MERGE SORT

```
Algorithm MergeSort(low,high)
//a[low:high] is a global array to be sorted
//Small(P) is true if there is only one element
//to sort. In this case the list is already sorted.
{
if (low<high) then    //if there are more than one element
{
//Divide P into subproblems
//find where to split the set
mid = [(low+high)/2];
//solve the subproblems.
mergesort (low,mid);
mergesort(mid+1,high);  //combine the solutions .
merge(low,mid,high);
}
}
```

Algorithm: Merging 2 sorted subarrays using auxiliary storage.

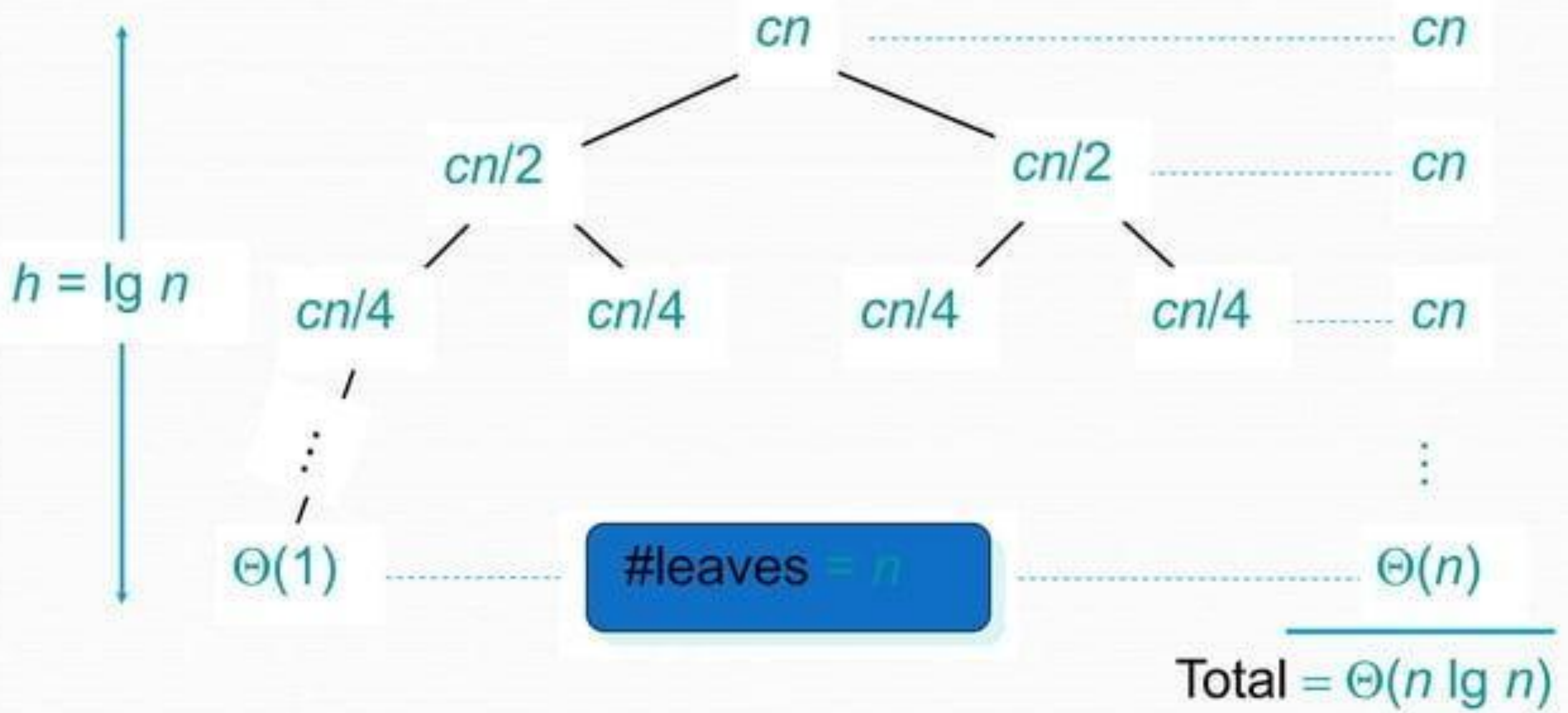
```
1. Algorithm merge(low,mid,high)
2. /*a[low:high] is a global array containing  two sorted subsets in a[low:mid]  and in a[mid+1:high].The
   goal is to merge these 2 sets into  a single set residing in a[low:high].b[] is an auxiliary global array.
   */
3. {
4. h=low; I=low; j=mid+1;
5. while ((h<=mid) and (j<=high)) do {
6. if (a[h]<=a[j]) then {
7. b[I]=a[h];
8. h = h+1; }
9. else {
10.b[I]= a[j];
11.j=j+1; }
12.I=I+1; }
13.if (h>mid) then
14.for k=j to high do {
15.b[I]=a[k];
16.I=I+1;
17.}
18.else
19.for k=h to mid do
20.{
21.b[I]=a[k];
22.I=I+1; }
23.for k=low to high do a[k] = b[k]; }
```

EXAMPLE

- Consider the array of 10 elements a[1:10] =(310, 285, 179, 652, 351, 423, 861, 254, 450, 520)
  Algorithm Mergesort begins by splitting a[] into 2 sub arrays each of size five (a[1:5] and a[6:10]).
- The elements in a[1:5] are then split into 2 sub arrays of size 3 (a[1:3]) and 2(a[4:5])
- Then the items in a [1:3] are split into sub arrays of size 2 a[1:2] & one(a[3:3])
- The 2 values in a[1:2] are split to find time into one-element sub arrays and now the merging begins.

Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



QUICK SORT

- In Quick sort, the division into 2 sub arrays is made so that the sorted sub arrays do not need to be merged later.
- This is accomplished by rearranging the elements in a[1:n] such that a[I]<=a[j] for all I between 1 & n and all j between (m+1) & n for some m, 1<=m<=n.
- Thus the elements in a[1:m] & a[m+1:n] can be independently sorted.
- No merge is needed. This rearranging is referred to as partitioning.

Algorithm: Partition the array a[m:p-1] about a[m]

```
Algorithm Partition(a,m,p)
/*within a[m],a[m+1],......,a[p-1] the elements  are rearranged in such a manner
that if  initially t=a[m],then after completion  a[q]=t for some q between m and
p-1,a[k]<=t for m<=k<q, and a[k]>=t for q<k<p. q is returned Set a[p]=infinite. */
{
v=a[m];I=m;j=p;
repeat
{
repeat
I=I+1;
until(a[I]>=v);
repeat
j=j-1;
until(a[j]<=v);
if (I<j) then interchange(a,i.j);
}until(I>=j);
a[m]=a[j]; a[j]=v;
retun j;
}
Algorithm Interchange(a,I,j)  //Exchange a[I] with a[j]
{
p=a[I];
a[I]=a[j];
a[j]=p;
}
```

- **Algorithm:** Sorting by Partitioning
- Algorithm Quicksort(p,q)
- //Sort the elements a[p],….a[q] which resides
- //is the global array a[1:n] into ascending
  //order; a[n+1] is considered to be defined
- // and must be >= all the elements in a[1:n]
- {
- if(p<q) then // If there are more than one element
- {
- // divide p into 2 subproblems
- j=partition(a,p,q+1);
- //"j" is the position of the partitioning element.
- //solve the subproblems.
- quicksort(p,j-1);
- quicksort(j+1,q);
- //There is no need for combining solution.
- }
- }

WEBSITES

1. www.mit.edu
2. www.soe.stanford.edu
3. www.grad.gatech.edu
4. www.gsas.harward.edu
5. www.eng.ufl.edu
6. www.iitk.ac.in
7. www.iitd.ernet.in
8. www.ieee.org
9. www.ntpel.com
10. WWW.JNTUWORLD.COM
11. www.firstrankers.com
12. www. studentgalaxi.blogspot.com

SUGGESTED BOOKS

TEXT BOOKS

1. Fundamentals of Computer Algorithms, Ellis Horowitz,Satraj Sahni and Rajasekharam,Galgotia publications pvt. Ltd.

2. Algorithm Design: Foundations, Analysis and Internet examples, M.T.Goodrich and R.Tomassia,John wiley and sons.

REFERENCES

1. Introduction to Algorithms, secondedition,T.H.Cormen,C.E.Leiserson, R.L.Rivest,and C.Stein,PHI Pvt. Ltd./ Pearson Education

2. Introduction to Design and Analysis of Algorithms A strategic approach,
R.C.T.Lee, S.S.Tseng, R.C.Chang and T.Tsai, Mc Graw Hill.

3. Data structures and Algorithm Analysis in C++, Allen Weiss, Second edition, Pearson education.

Thank You