Learn DAA: From B K Sharma

Subject Title:
Design and Analysis of Algorithm
Subject Code: TCS- 503

Total Number of Units: 5

Maximum Marks: 100

Unit I: Contains 7 Chapters:

- Introduction
- Growth of Functions
- Master Theorem
- Heap Sort
- Quick Sort
- Sorting in Linear Time
- Order Statistics

Unit II: Contains 6 Chapters:

- Red Black Tree
- Augmenting Data Structures
- B-Trees
- Binomial Heaps
- Fibonacci Heaps
- D.S. for Disjoint Sets

Learn DAA: From B K Sharma

Subject Title:

Design and Analysis of Algorithm

Subject Code: TCS- 503

Unit III: Contains 4 Chapters:

- Dynamic Programming
- Greedy Method
- Amortized Analysis
- Backtracking

Unit IV: Contains 6 Chapters:

- BFS & DFS
- MST
- Single-Source Shortest Paths
- All-Pairs Shortest Paths
- Maximum Flow
- TSP

Learn DAA: From B K Sharma

Subject Title:
Design and Analysis of Algorithm
Subject Code: TCS- 503

Unit V: Contains 4 Chapters:

- Randomized Algorithms
- String Matching
- NP-Completeness
- Approximation Algorithms

Total: 27 Chapters

Good Luck!!!

Learn DAA: From B K Sharma

TCS-503: Design and Analysis of Algorithms

Introduction

Unit I: Syllabus

- Introduction:
  - Algorithms
  - Analysis of Algorithms
  - Growth of Functions
  - Master's Theorem
  - Designing of Algorithms

Unit I: Syllabus

- Sorting and Order Statistics
  - Heap Sort
  - Quick Sort
  - Sorting in Linear Time
    - Counting Sort
    - Bucket Sort
    - Radix Sort
  - Medians and Order Statistics

Algorithm

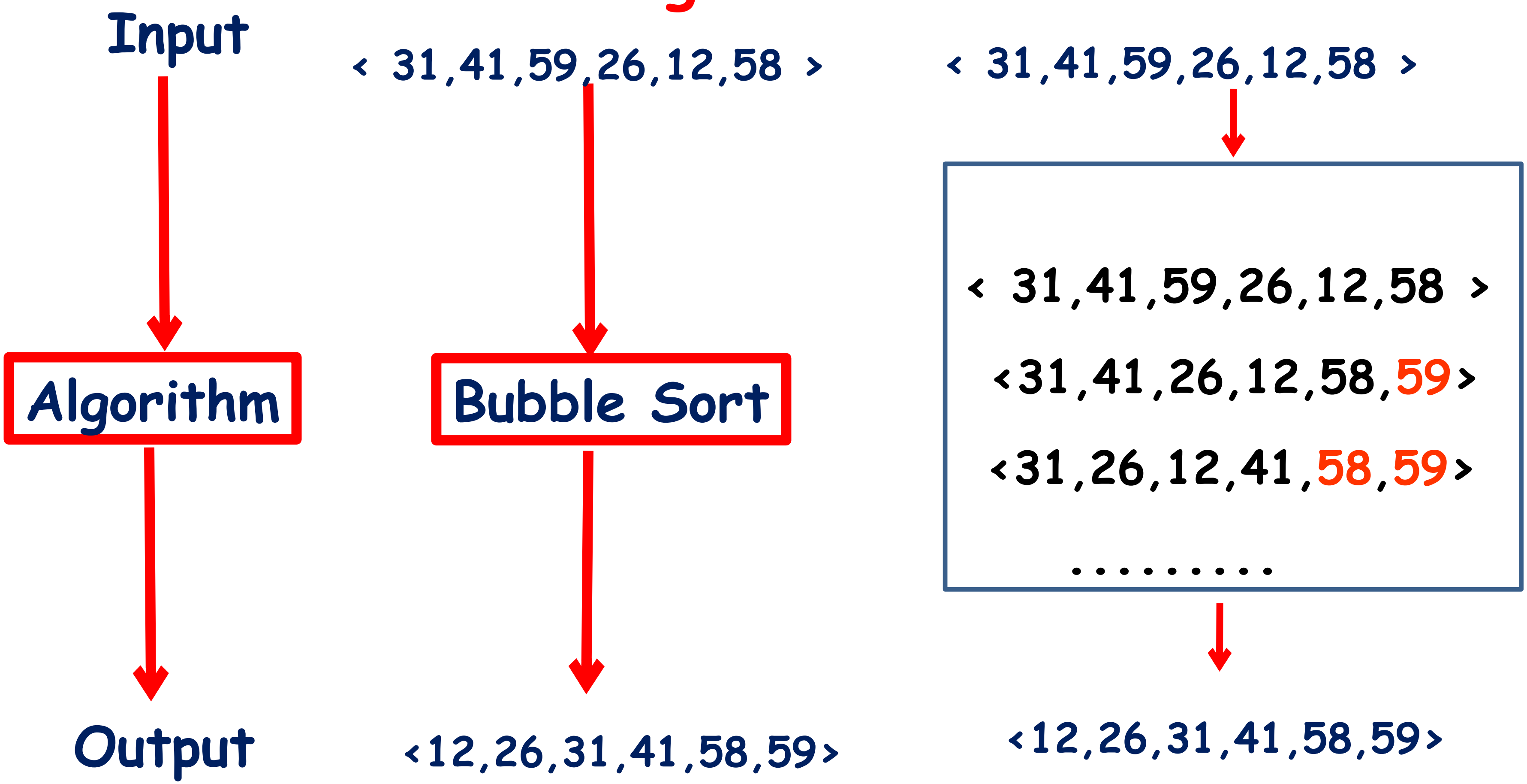
Input → **Algorithm** → Output

< 31,41,59,26,12,58 > → **Bubble Sort** → <12,26,31,41,58,59>

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

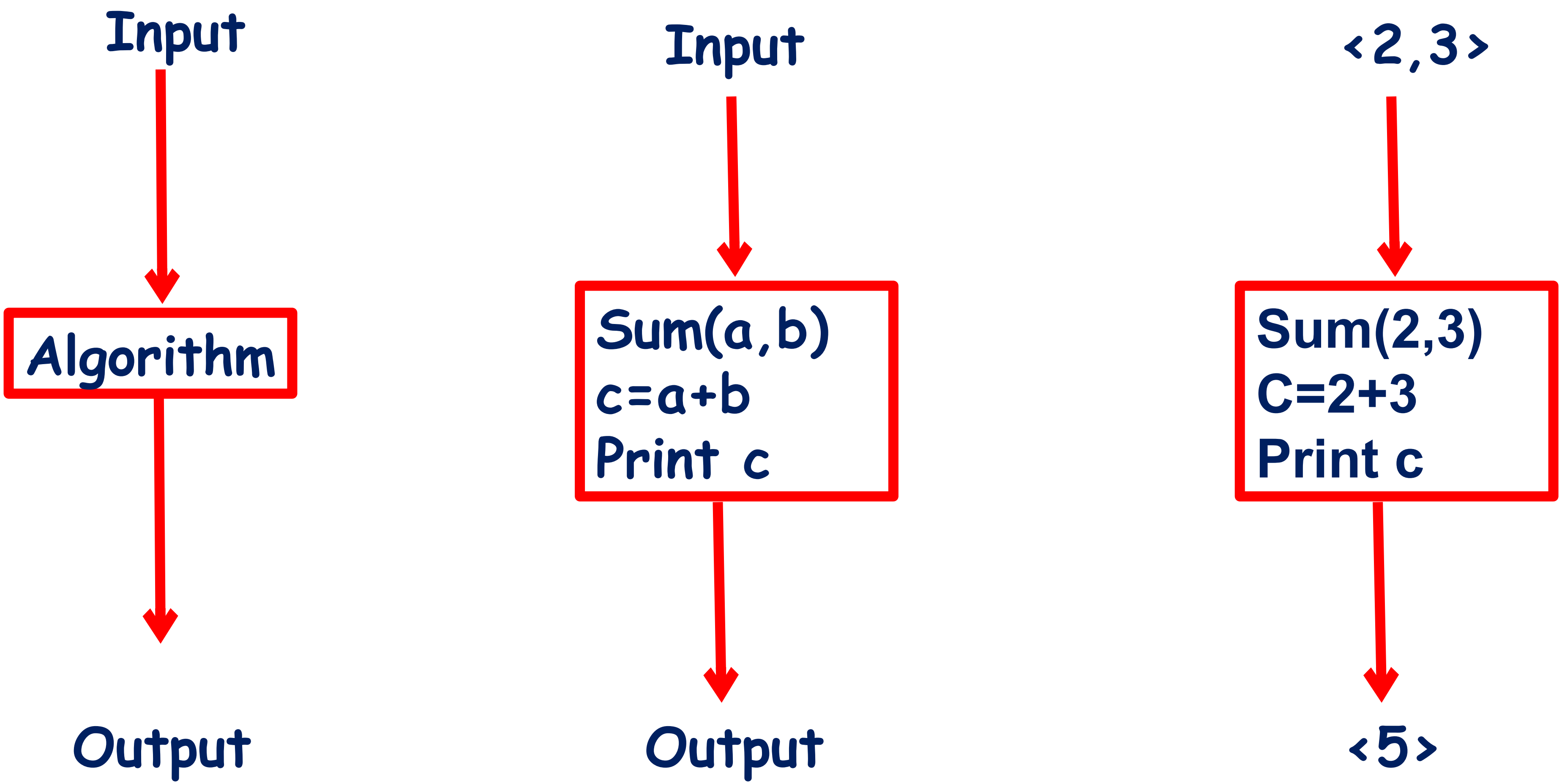
Learn DAA: From B K Sharma

Algorithm



Learn DAA: From B K Sharma

Algorithm



Design of Algorithm

Design algorithms which minimize the cost.

Analysis of Algorithm

predict the cost of an algorithm in terms of resources

Cost:

Memory Space Occupied by Algorithm

Running Time taken by Algorithm

Resource:

Memory

CPU Time

DAA: Design and Analysis of Algorithm

ADA: Algorithm Design and Analysis

Features of Algorithm

1. Finiteness

Terminates after a finite number of steps

2. Definiteness

Rigorously and unambiguously specified

3. Input

Zero or more valid inputs are clearly specified

4. Output

At least one output can be proved to produce the correct output given a valid input

5. Effectiveness

Steps are sufficiently simple and basic

Algorithm Vs Pseudo Code Vs Flowchart

An algorithm is a formal structure for solving a problem that may be expressed in Pseudo Code or Flowchart.

We say that write an algorithm in Pseudo Code to implement Bubble Sort.

We say that write an algorithm using Flow Chart to implement Bubble Sort.

Algorithm Vs Pseudo Code Vs Flowchart

Algorithm Types: Pseudo Code and Flowchart.

Pseudo Code: describes how you would implement an algorithm without getting into syntactical details of programming languages.

: Written in Natural Language.

: Preferred notation for describing algorithms

Learn DAA: From B K Sharma

Algorithm Vs Pseudo Code Vs Flowchart

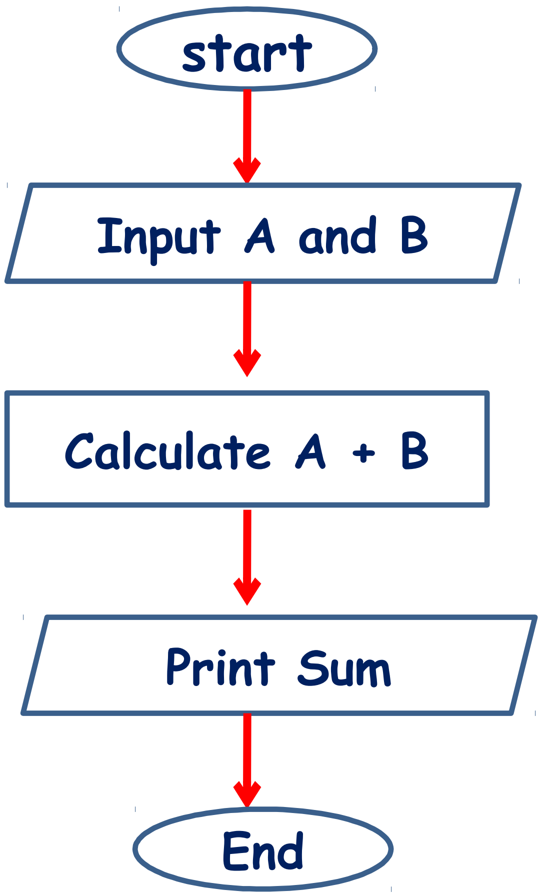
Pseudo Code to implement sum algorithm

Pseudo Code:

1. Begin
2. Input A and B
3. Calculate A + B
4. Print result of SUM
5. End

Flow Chart to implement sum algorithm

Flow Chart:



Learn DAA: From B K Sharma

Different Algorithm Design Techniques

Divide And Conquer

Reduce the problem to smaller problems (by a factor of at least 2), solves recursively and then combine the solutions

Greedy Approach

"Take what you can get now" strategy.

Work in phases.

In each phase the currently best decision is made

Dynamic programming

Bottom-Up Technique in which the smallest sub-instances are *explicitly* solved first and the results of these used to construct solutions to progressively larger sub-instances.

Backtracking

Generate-and-Test methods:

Based on exhaustive search in multiple choice problems

DAA: B K Sharma

Different Algorithm Design Techniques

Divide And Conquer

Examples:

- Binary Search
- Merge sort
- Quick sort

Greedy Approach

Examples:

- Fractional Knapsack Problem
- Activity Selection Problems
- Shortest Path Problems
- Finding Minimum Spanning Tree

Dynamic programming

Examples:

- Binary(0/1) Knapsack Problem
- Assembly Line Scheduling
- Matrix Chain Multiplication

Backtracking

Examples:

- n-queens Problems
- Sum of Subset Problems

Why to Analyze Algorithm?

Factors affecting the running time:

Computer   Translator   Algorithm used   Input to the algorithm

Example: sorting ten million (10^7) numbers

Insertion sort:

Computer A: 10^9 instruc/s

World's craftiest programmer

Machine language

$$T(n) = c_1 n^2$$

$$T(n) = 2n^2$$

$$t = \frac{2 \cdot (10^7)^2 \text{instruc}}{10^9 \text{instruc/s}} = 2 \times 10^5 \text{s} \approx 55.56\text{h}$$

Merge sort:

Computer B: 10^7 instruc/s

Average programmer

High-level language

$$T(n) = c_2 n \lg n$$

$$T(n) = 50 n \lg n$$

$$t = \frac{50 \cdot 10^7 \lg 10^7 \text{instruc}}{10^7 \text{instruc/s}} \approx 19.38\text{m}$$

How do we compare/analyze algorithms?

1. Compare execution times?

    Not good: times are specific to a particular computer !!

2. Count the number of statements executed?

    Not good: number of statements vary with the programming language as well as the style of the individual programmer.

3. Express running time as a function of the input size *n* (i.e., *f(n)*).

    Ideal Solution: Compare different functions corresponding to running times.

    Such an analysis is independent of machine time, programming style, etc.

How do we compare/analyze algorithms?

Growth of Functions

↓

Asymptotic Notations

Learn DAA: From B K Sharma

Subject Title:

Design and Analysis of Algorithm

Subject Code: TCS- 503

Total Number of Units: 5

Maximum Marks: 100

Unit I: Contains 7 Chapters:

- Introduction
- Growth of Functions
- Master Theorem
- Heap Sort
- Quick Sort
- Sorting in Linear Time
- Order Statistics

Unit II: Contains 6 Chapters:

- Red Black Tree
- Augmenting Data Structures
- B-Trees
- Binomial Heaps
- Fibonacci Heaps
- D.S. for Disjoint Sets

Different Algorithm Analysis Techniques

Best-Case Analysis

Minimum number of steps for any possible input.

Provides a lower bound on running time

Not a useful measure.

Worst-Case Analysis

Maximum number of steps the algorithm takes for any possible input.

Provides an upper bound on running time

Most tractable measure.

Average-Case Analysis

Average of the running times of all *possible* inputs.

Demands a definition of probability of each input, which is usually difficult to provide and to analyze.

Assumes that the input is random.

Provides a prediction about the running time.

Search for 12

Search for 98

1	2	3	4	5	6	7	8	9	10	11
12	15	21	23	27	45	76	77	90	95	98

One step

Search for 200

What is more important to analyze?

Best case? Worst case? Average case?

The worst-case running time gives a guaranteed upper bound for any input.

For many algorithms, the worst case occurs often.

e.g. When searching, the worst case often occurs when the item being searched for is not present, and searches for empty items may be frequent.

The average case is interesting and important because it gives a closer estimation of the realistic running time.

However, its consideration usually requires more efforts (algebraic transformations, etc.)

The worst case is the most interesting.

The average case is interesting, but often is as "bad" as the worst case and may be estimated by the worst case.

The best case is the least interesting.

Random Access Machine Model

Algorithms can be analyzed in a machine-independent way using the Random Access Machine (RAM) model.

This model assumes a single processor.

Executes operations sequentially.

In the RAM model, instructions are executed one after the other, with no concurrent operations.

Each memory access takes one time step.

Each simple operation(+,-,*,/,if) takes 1 time step.
All ops cost 1 unit.

Loops and subroutines are *not* considered simple operations.

Simplifying assumption:

Eliminates dependence on the speed of our computer, otherwise impossible to verify and to compare.

Learn DAA: From B K Sharma

Algorithm Analysis: Example

```
int sum(int n)
 {
    int psum;
1   psum=0;                 1
2   for(int i=1;i<=n;i++)   2n+2
3       sum+=i*i*i;         4n
4   return sum;             1
 }
```

Lines 1 and 4 count for one unit each

Line 2: Explanation : Suppose n=5

1: for initialization

1<=5: Yes; 2<=5: Yes; 3<=5: Yes

4<=5: Yes; 5<=5: Yes; 6<=5: No

Total No. of Comparison =6 =5+1 =n+1

Total No. of increments =5 =n

Total cost of line 2 =1 + n+1 +n

=2n +2

Line 3: Explanation

Line 3 will be executed n times,

each time 4 units of time.

So, Total Cost of line 3==4*n units

Total cost: 6n + 4 ⇒ linear with respect to n

DAA: B K Sharma

Algorithm Analysis: Example

Alg.: MIN (a[1], …, a[n])

$m \leftarrow a[1];$	1
for $i \leftarrow 2$ to n	$n-1 + 1 = n$
if $a[i] < m$	$n-1$
then $m \leftarrow a[i];$	$n-2$

T(n) =1 [first step] + (n) [for loop] + (n-1) [if condition] + (n-2) [the assignment in then] = 3 n - 2

11	2	9	13	60	1
1	2	3	4	5	6

Algorithm Analysis: Example

LinearSearch(A, key)	cost	times
$i \leftarrow 1$	c_1	1
while $i \leq n$ and $A[i]$!= *key do*	c_2	x
i++	c_3	$x-1$
if $i \leq n$	c_4	1
then return *true*	c_5	1
else return *false*	c_6	1

Best case: $x=1$

1	2	3	4	5	6	Key
11	2	9	13	60	1	11

$$T(n) = c_1.1 + c_2.1 + c_3.0 + c_4.1 + c_5.1$$

Worst case: $x=n+1$

1	2	3	4	5	6	Key
11	2	9	13	60	1	100

$$T(n) = c_1 + c_2(n+1) + c_3 n + c_4 + c_6$$

Algorithm Analysis: Example

LinearSearch(A, key)	cost	times
1 $i \leftarrow 1$	1	1
2 while $i \le n$ and $A[i]$!= *key* do	1	x
3. i++	1	x-1
4. if $i \le n$	1	1
5. then return *true*	1	1
6. else return *false*	1	1

Assign a cost of 1 to all statement executions.

Best case:

$$T(n) = c_1.1 + c_2.1 + c_3.0 + c_4.1 + c_5.1$$
$$= 1 + 1 + 0 + 1 + 1 = 4$$

Worst case:

$$T(n) = c_1 + c_2(n+1) + c_3 n + c_4 + c_6$$
$$= 1 + (n+1) + n + 1 + 1 = 2n+4$$

Algorithm Analysis: Example

LinearSearch (*A*, *key*)	cost	times
1 $i \leftarrow 1$	1	1
2 while $i \leq n$ and $A[i]$!= *key* do	1	x
3. i++	1	x-1
4. if $i \leq n$	1	1
5. then return *true*	1	1
6. else return *false*	1	1

Average Case:

Suppose Array contains n=2 elements and key=50

50	10
1	2

50

50	10
1	2

10

Line 2 will be executed n/2=2/2=1 time

Then consider key=10.

Line 3 will be executed n/2=2/2=1 time

If we assume that we search for a random item in the list, on an average, Statements 2 and 3 will be executed $n/2$ times.

Hence, average-case complexity is

$$1 + n/2 + n/2 + 1 + 1 = n + 3$$

Learn DAA: From B K Sharma

Binary Search Technique: Example

Locates a target value in a *sorted* array / list by successively eliminating half of the array from consideration.

Example: Searching the array below for the value 42:

Which Indexes does the algorithm examine to find value 42?

8

12

10

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

min mid max

Binary Search Technique: Example

For an array of size N, it eliminates ½ elements until 1 element remains.

N, N/2, N/4, N/8, ...,8, 4, 2, 1

How many divisions does it take?

Think of it from the other direction:

How many times do we have to multiply 1 by 2 to reach N?

1, 2, 4, 8, ..., N/4, N/2, N

Call this number of multiplications "x".

$2^x = N$ $x = \log_2 N$

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103
	min								mid								max

Binary Search Technique: Example

Search for 45

1	2	3	4	5	6	7	8	9	10	11
12	15	21	23	27	45	76	77	90	95	98

mid

Found on first step, 45 equals the middle item

Search for 27

1	2	3	4	5	6	7	8	9	10	11
12	15	21	23	27	45	76	77	90	95	98

mid

Search for 27

1	2	3	4	5	6	7	8	9	10	11
12	15	21	23	27	45	76	77	90	95	98

mid

Binary Search Technique: Example

Search for 27

1	2	3	4	5	6	7	8	9	10	11
12	15	21	23	27	45	76	77	90	95	98

mid

Found !

Search for 91

1	2	3	4	5	6	7	8	9	10	11
12	15	21	23	27	45	76	77	90	95	98

mid

Search for 91

1	2	3	4	5	6	7	8	9	10	11
12	15	21	23	27	45	76	77	90	95	98

mid

$91 < 95$

Search Completed, No value found

Binary Search Technique : Algorithm

```
int BinarySearch(int A[], int N, int Num)
 {
    int lb = 0;
    int ub = N - 1;
    int mid = (lb + ub) / 2;
    while ((A[mid] != Num) && (lb <= ub))
        {
           if (A[mid] > Num)
                ub = mid - 1;
           else
                lb = mid + 1;

           mid = (lb + ub) / 2;
        }

     if (A[mid] == Num)
           return mid;
     else
           return -1;
 }
```

0	1	2	3	4	5	6	7	8	9	10
12	15	21	23	27	45	76	77	90	95	98

mid

For an array of size N, it eliminates ½ elements until 1 element remains.

N, N/2, N/4, N/8, ...,8, 4, 2, 1

How many divisions does it take?

Think of it from the other direction:

How many times do we have to multiply by 2 to reach N?

1, 2, 4, 8, ..., N/4, N/2, N

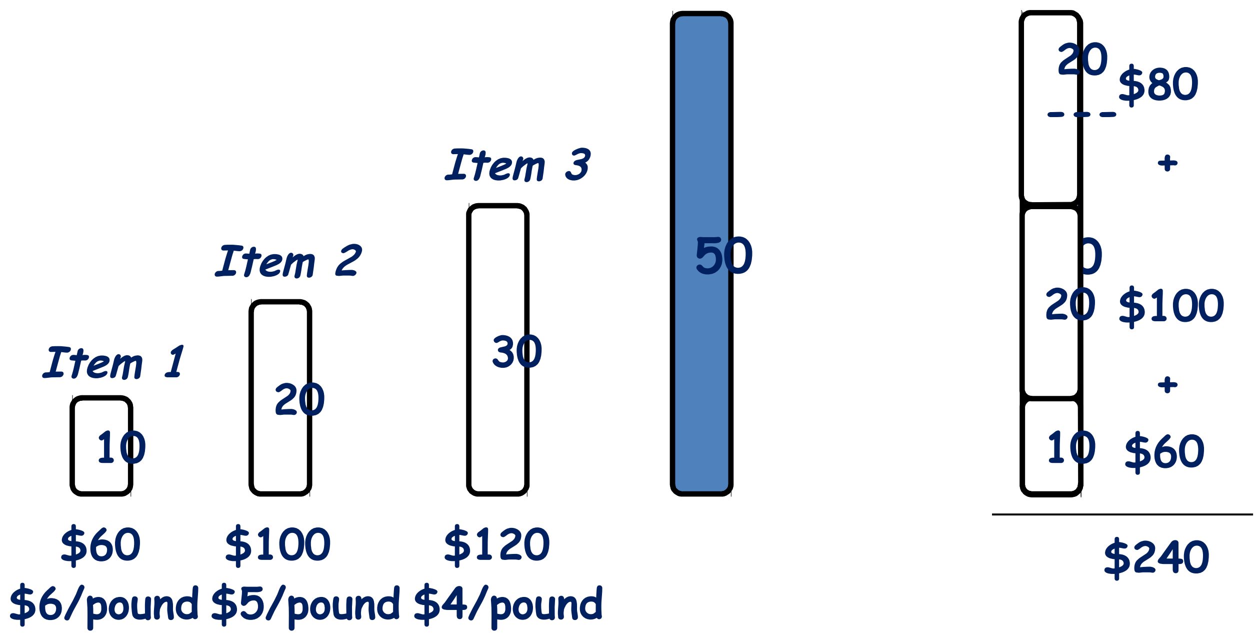
Call this number of multiplications "x".

$2^x = N$ $x = \log_2 N$

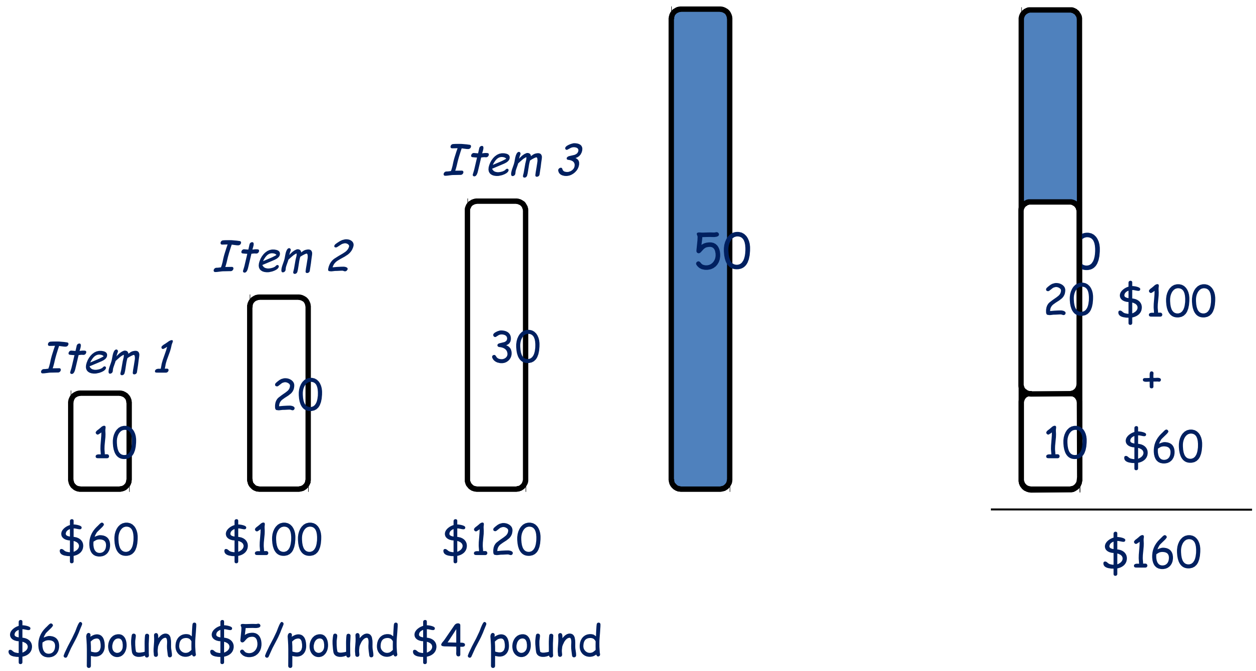
Example: Fractional Knapsack Problem

i	1	2	3
v_i	60	100	120
w_i	10	20	30
v_i/w_i	6	5	4

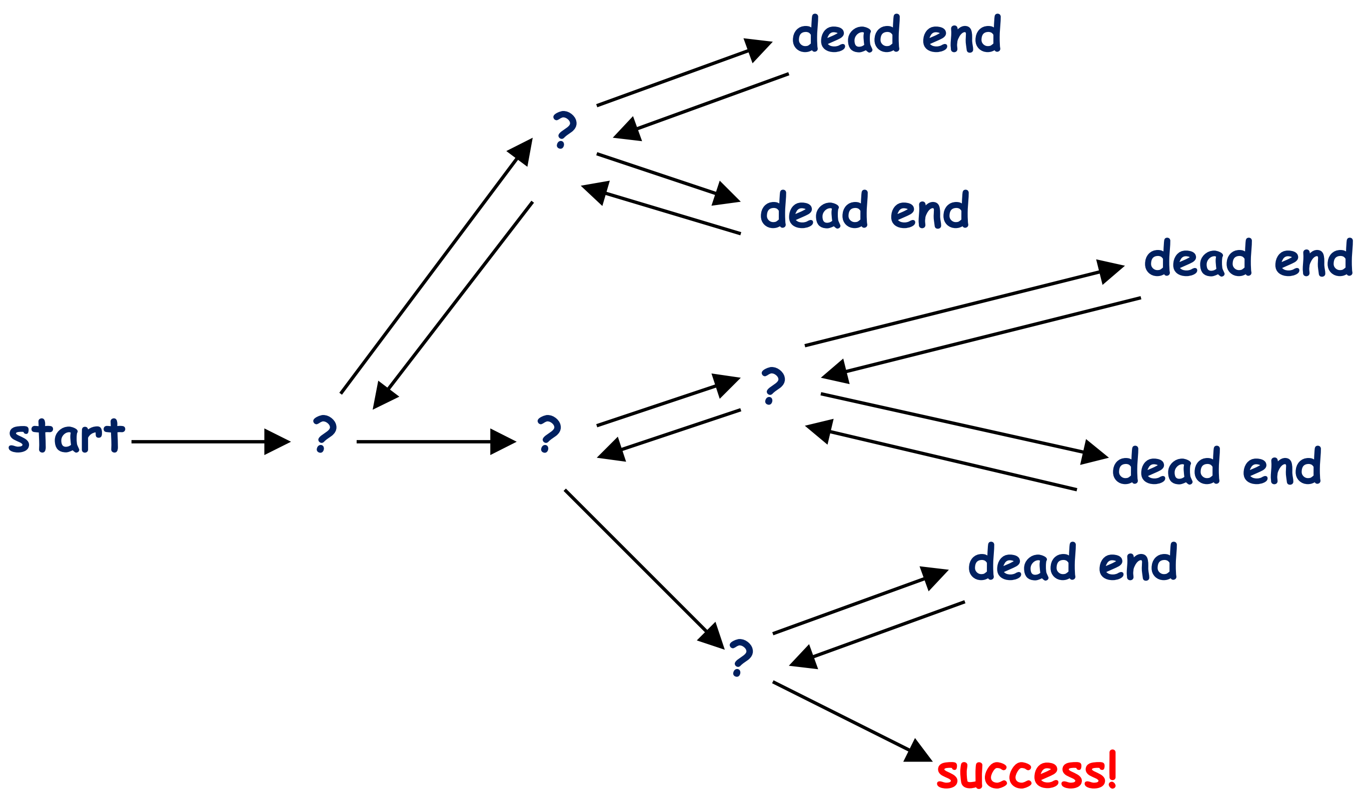
$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots \geq \frac{v_n}{w_n}$$



Example: 0/1 Knapsack Problem



Backtracking

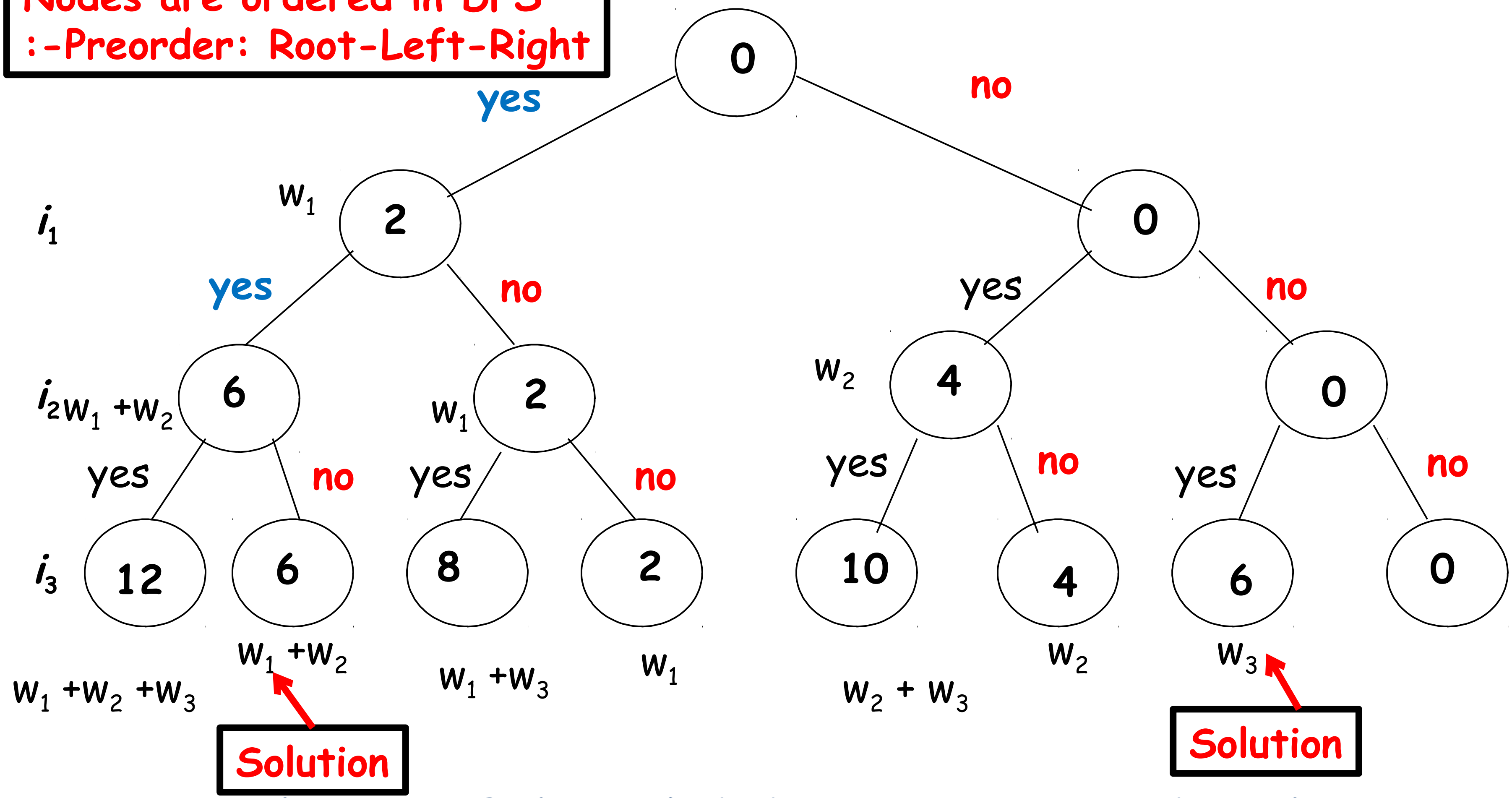


DAA: B K Sharma

Sum of subset Problem: State SpaceTree for 3 items

$w_1 = 2, \quad w_2 = 4, \quad w_3 = 6$ and $S = 6$

Nodes are ordered in DFS :-Preorder: Root-Left-Right



The sum of the included integers is stored at the node.

Backtracking to solve n=4 queens problem