

Semester-4

(Design & Analysis of Algorithm)

(According to Purvanchal University Syllabus)

Unit – 1

Introduction

Algorithm–

- An algorithm is the step by step problem solving technique.
- In an algorithm we can use general English for write the code this code is known as pseudo code.

Characteristics of Algorithms

The main characteristics of algorithms are as follows –

- Algorithms must have a unique name
- Algorithms should have explicitly defined set of inputs and outputs
- Algorithms are well-ordered with unambiguous operations
- Algorithms halt in a finite amount of time. Algorithms should not run for infinity, i.e., an algorithm must end at some point

Pseudocode

- Pseudocode gives a high-level description of an algorithm without the ambiguity associated with plain text but also without the need to know the syntax of a particular programming language.

Difference between Algorithm and Pseudocode

- An algorithm is a formal definition with some specific characteristics that describes a process, which could be executed by a Turing-complete computer machine to perform a specific task. Generally, the word "algorithm" can be used to describe any high level task in computer science.
- On the other hand, pseudocode is an informal and (often rudimentary) human readable description of an algorithm leaving many granular details of it. Writing a pseudocode has no restriction of styles and its only

objective is to describe the high level steps of algorithm in a much realistic manner in natural language.

Analysis of Algorithm–

- Algorithm analysis is an important part of computational complexity theory which provide theoretical estimation for the require resource of algorithm to solve a specific computational problem.
- Most algorithm are design to work with inputs of arbitrary length. Analysis of algorithm is determination of the amount of time and space resource require to execute.

Designing Algorithm–

- An algorithm is the series of instruction often refer to as a process, which is to be follow when solving a particular problem.
- Since modern computing uses algorithm much more frequently then at the other point in human history. The field of algorithm design require a strong mathematical background.
- The key to algorithm design to be process by asking to yourself a sequence of questions to guid your thought process.
- There is a various techniques for designing an algorithm
 1. Brute Force.
 2. Greedy algorithm.
 3. Divide and conquer.
 4. Dynamic programming.
 5. Backtracking and branch and bound.

Growth of functions–

The growth of function is directly related to the complexity of algorithms.

... In order to get a handle on its complexity, we first look for a function that gives the number of operations in terms of the size of the problem, usually measured by a positive integer n , to which the algorithm is applied.

Given functions f and g , we wish to show how to quantify the statement: “ g grows as fast as f ”.

The growth of functions is directly related to the complexity of algorithms. We are guided by the following principles.

- We only care about the behavior for “large” problems.

- We may ignore implementation details such as loop counter incrementation.

Growth-rate Functions



- $O(1)$ – **constant** time, the time is independent of n , e.g. array look-up
- $O(\log n)$ – **logarithmic** time, usually the log is base 2, e.g. binary search
- $O(n)$ – **linear** time, e.g. linear search
- $O(n \log n)$ – e.g. efficient sorting algorithms
- $O(n^2)$ – **quadratic** time, e.g. selection sort
- $O(n^k)$ – **polynomial** (where k is some constant)
- $O(2^n)$ – **exponential** time, very slow!
- Order of growth of some common functions
 $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$

Summation–

- Summation (Σ) just means to “add up.” For example, let’s say you had 5 items in a data set: 1,2,5,7,9; you can think of these as x-values. If you were asked to add all of the items up in summation notation, you would see: $\Sigma(x)$ which equals $1 + 2 + 5 + 7 + 9 = 24$.

When using summation notation, X_1 means “the first x-value”, X “the second x-value” and so on. For example, let’s say you had a list of weights: 150lb, 153lb and 202lb. The weights and their corresponding x

X_1 : 100lb

X_2 : 150lb

X_3 : 153lb

X_4 : 202lb

$$\sum_{i=1}^n X_i = X_1 + X_2 + X_3 + \dots + X_n$$

$$\sum_{i=1}^n X_i = X_1 + X_2 + X_3 + \dots + X_n$$

The “ $i=1$ ” at the base of Σ means “start at your first x-value”. This would be (100lb in this example). The “ n ” at the top of Σ means “end at n ”. In statistics, n

is the number of items in the data set. So what this summation is asking you to do is the number of items in the data set. So what this summation is asking you to do

is “add up all of your x-values from the first to the last values from the first to the last.” For this set of data, that would be:

$$100 \text{ lb} + 150 \text{ lb} + 153 \text{ lb} + 202 \text{ lb} = 605 \text{ lb}.$$

Note: if you see a number above Σ , instead of n , it means to add up to a certain point. For example, a “3” above the Σ means to sum up the the third item (X point. For example, a “3” above the Σ means to sum up the the third item (X point. For example, a “3” above the Σ means to sum up the the third item (X_3) in the set.

Recurrence—

A recurrence is an equation or inequality that describes a function in terms of its on smaller inputs. Recurrences are generally used in divide-and-conquer paradigm.

Let us consider $T(n)$ to be the running time on a problem of size

If the problem size is small enough, say $n < c$ where c is a constant, the straightforward solution takes constant time, which is written if the problems with size n .

division of the problem yields a number of sub-problems with size

To solve the problem, the required time is $a.T(n/b)$. If we consider the time

To solve the problem, the required

sub-problems is $C(n)$, the recurrence

relation can be represented as

, the recurrence relation can be

represented as

Recurrence Relation

A **recurrence relation** is an equation that function in terms of itself by using smaller The expression:

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

describes the **running time** for a function recursion.

1. Substitution Method:

The Substitution Method Consists of two main steps:

Use the mathematical induction to find the boundary condition and shows that the guess is correct.
For Example1 Solve the equation by Substitution Method.

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

Guess the Solution.

mathematical induction we prove that our assumption was correct.

Recursion Tree Method

Recursion Tree Method – In this method, a recurrence tree is formed where each node represents the cost.

Master's Theorem – This is another important technique to find the complexity of a recurrence relation.

Sets–

- A set is defined as a collection of distinct objects of the same type or class of objects. The purposes of a set are called elements or members of the set.
An object can be numbers, alphabets, names, etc.

Examples of sets are:

- A set of rivers of India.
- A set of vowels.

We broadly denote a set by the capital letter A, B, C, etc. while the fundamentals of the set by small letter a, b, x, y, etc.

If A is a set, and a is one of the elements of A, then we denote it as $a \in A$. Here the symbol \in means -"Element of."

Sets Representation:

Sets are represented in two forms:-

a) Roster or tabular form: In this form of representation we list all the elements of the set within braces $\{ \}$ and separate them by commas.

Example: If A= set of all odd numbers less than 10 then in the roster form it can be expressed as $A = \{ 1, 3, 5, 7, 9 \}$.

b) Set Builder form: In this form of representation we list the properties fulfilled by all the elements of the set. We note as $\{x: x \text{ satisfies properties } P\}$. and read as 'the set

of those entire x such that each x has properties P .'

Example: If $B = \{2, 4, 8, 16, 32\}$, then the set builder representation will be: $B = \{x: x=2^n, \text{ where } n \in \mathbb{N} \text{ and } 1 \leq n \leq 5\}$

Counting and probability–

- Counting in probability refers to the techniques used to count the set of possible outcomes. The dependence or independence of the events is the main factor in determining how to calculate the total possible outcomes. Independent events don't affect each other. For example, the number of outfits possible from two pants and four shirts is found by multiplying the number of possibilities for each selection together, 2×4 . Dependent variables, or permutations, do affect each other. An example is ranking three favorite items: once the first is picked, there are only two choices for second place, with third place as the last picked. Thus, there are possible rankings.
- The word 'Probability' means the chance of occurring of a particular event. It is generally possible to predict the future of an event quantitatively with a certain probability of being correct. The where the outcome of the trial is uncertain.

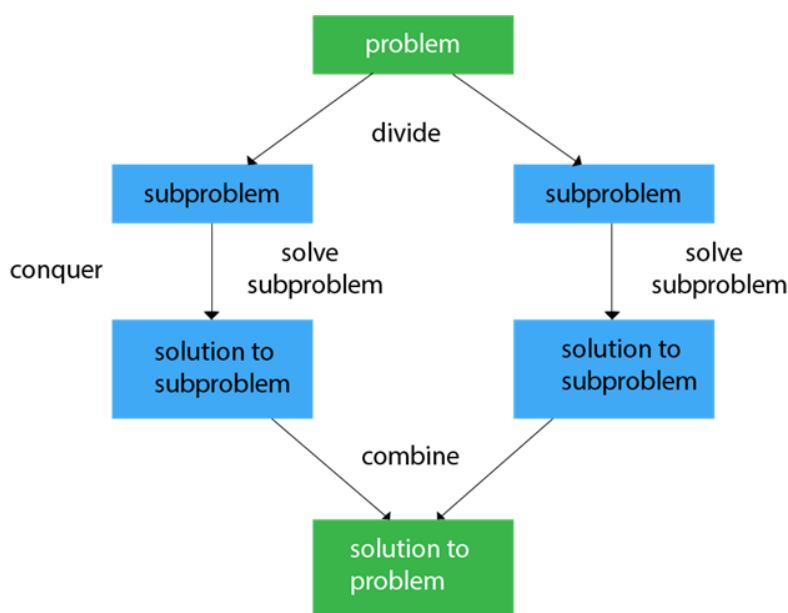
Unit – 2

Divide and conquer

- Divide and Conquer is an algorithmic pattern. In algorithmic methods, the design is to take a dispute on a huge input, break the input into minor pieces, decide the problem on each of the small pieces, and then merge the piecewise solutions into a global solution. This mechanism of solving the problem is called the Divide & Conquer Strategy.

Divide and Conquer algorithm consists of a dispute using the following three steps.

1. **Divide** the original problem into a set of subproblems.
2. **Conquer**: Solve every subproblem individually, recursively.
3. **Combine**: Put together the solutions of the subproblems to get the solution to the whole problem.



Generally, we can follow the **divide-and-conquer** approach in a three-step process.

Searching:

Binary Search

1. In Binary Search technique, we search an element in a sorted array by recursively dividing the interval in half.

2. Firstly, we take the whole array as an interval.

3. If the Pivot Element (the item to be searched) is less than the item in the middle of the interval, We discard the second half of the list and recursively repeat the process for the first half of the list by calculating the new middle and last element.

4. If the Pivot Element (the item to be searched) is greater than the item in the middle of the interval, we discard the first half of the list and work recursively on the second half by calculating the new beginning and middle element.

5. Repeatedly, check until the value is found or interval is empty.

Example

In this example, we are going to search element 63.

Binary Search											
	0	1	2	3	4	5	6	7	8	9	
Search 23	2	5	8	12	16	23	38	56	72	91	
	L=0	1	2	3	M=4	5	6	7	8	9	
23 > 16 take 2 nd half	2	5	8	12	16	23	38	56	72	91	
	0	1	2	3	4	L=5	6	M=7	8	H=9	
23 < 56 take 1 st half	2	5	8	12	16	23	38	56	72	91	
	0	1	2	3	4	L=5, M=5	H=6	7	8	9	
Found 23, Return 5	2	5	8	12	16	23	38	56	72	91	

Sorting:

Counting sort-

- It is a linear time sorting algorithm which works faster by not making a comparison. It assumes that the number to be sorted is in range 1 to k where k is small.
- Basic idea is to determine the "rank" of each number in the final sorted array.

1. Counting-Sort(A, B, k)
2. Let $C[0 \dots k]$ be a new array
3. for $i=0$ to k
4. $C[i] = 0$;
5. for $j=1$ to A.length or n
6. $C[A[j]] = C[A[j]] + 1$;
7. for $i=1$ to k
8. $C[i] = C[i] + C[i-1]$;
9. for $j=n$ or A.length down to 1
10. $B[C[A[j]] - 1] = A[j]$;
11. $C[A[j]] = C[A[j]] - 1$;

Points to be noted:

1. Counting sort is efficient if the range of input data is not significantly greater than the number of objects to be sorted. Consider the situation where the input sequence is between range 1 to 10K and the data is 10, 5, 10K, 5K.
2. It is not a comparison based sorting. Its running time complexity is $O(n)$ with space proportional to the range of data.
3. It is often used as a sub-routine to another sorting algorithm like radix sort.
4. Counting sort uses a partial hashing to count the occurrence of the data object in $O(1)$.
5. Counting sort can be extended to work for negative inputs also.

Radix sort-

- The idea of Radix Sort is to do digit by digit sort starting from least significant digit to most significant digit. Radix sort uses counting sort as a subroutine to sort.

The Radix Sort Algorithm

1) Do following for each digit i where i varies from least significant digit to the most significant digit.

.....a) Sort input array using counting sort (or any stable sort) according to the i 'th digit.

Example:

Original, unsorted list:

170, 45, 75, 90, 802, 24, 2, 66

Sorting by least significant digit (1s place) gives:

[*Notice that we keep 802 before 2, because 802 occurred before 2 in the original list, and similarly for pairs 170 & 90 and 45 & 75.]

170, 90, 802, 2, 24, 45, 75, 66

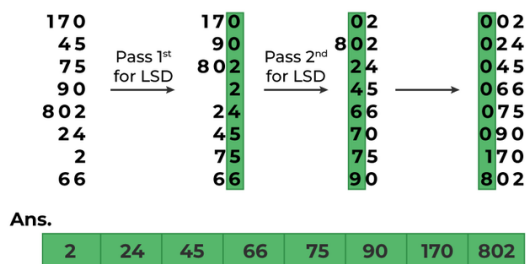
Sorting by next digit (10s place) gives:

[*Notice that 802 again comes before 2 as 802 comes before 2 in the previous list.]

802, 2, 24, 45, 66, 170, 75, 90

Sorting by most significant digit (100s place) gives:

2, 24, 45, 66, 75, 90, 170, 802



Bucket sort

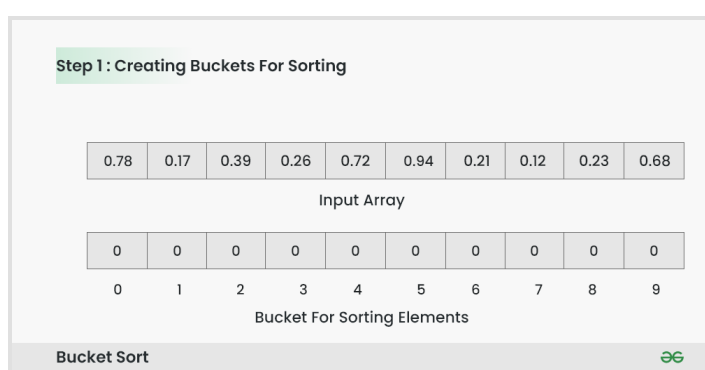
Bucket sort is mainly useful when input is uniformly distributed over a range. For example, consider the following problem.

Sort a large set of floating point numbers which are in range from 0.0 to 1.0 and are uniformly distributed across the range. How do we sort the numbers efficiently?

bucket algorithm.

bucketSort(arr[], n)

- 1) Create n empty buckets (Or lists).
- 2) Do following for every array element arr[i].
 -a) Insert arr[i] into bucket[n*array[i]]
- 3) Sort individual buckets using insertion sort.
- 4) Concatenate all sorted buckets.



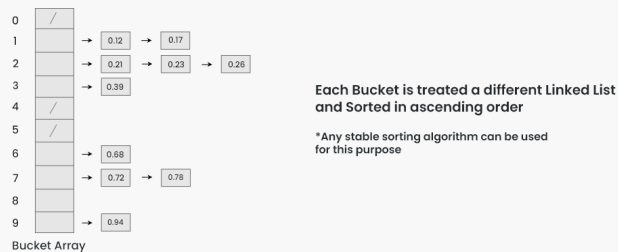
Step 2 : Inserting Array elements into respective buckets



Bucket Sort



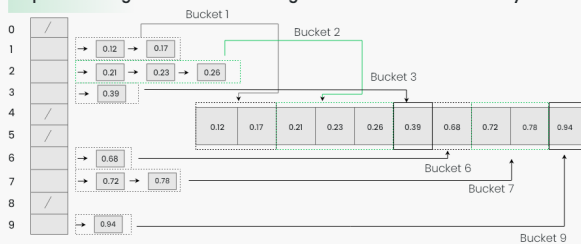
Step 3: Sorting individual Bucket



Bucket Sort



Step 4: Inserting buckets in ascending order into the result array



Bucket Sort



Step 5 : Return the Sorted Array

0.12 0.17 0.21 0.23 0.26 0.39 0.68 0.72 0.78 0.94

Bucket Sort



Selection sort-

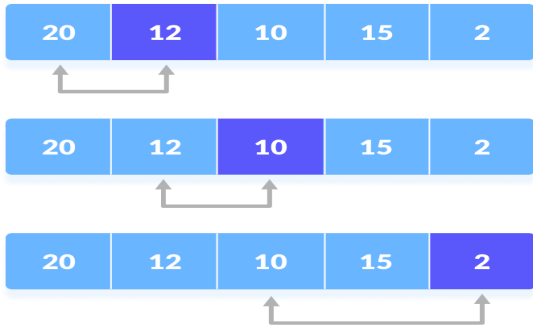
The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

- 1) The subarray which is already sorted.
- 2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.



2)

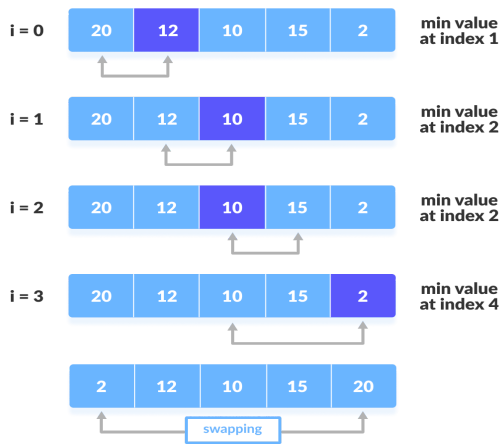


3)



4)

step = 0



5)

step = 3



Heap sort-

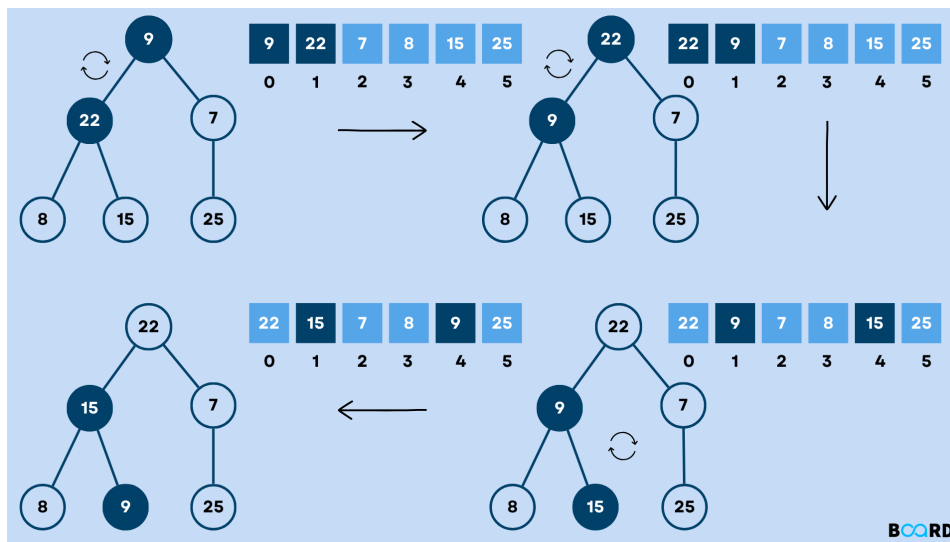
- Heap sort is a comparison based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for remaining element.

Heap Sort Algorithm for sorting in increasing order:

1. Build a max heap from the input data.
2. At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.
3. Repeat above steps while size of heap is greater than 1.

How to build the heap?

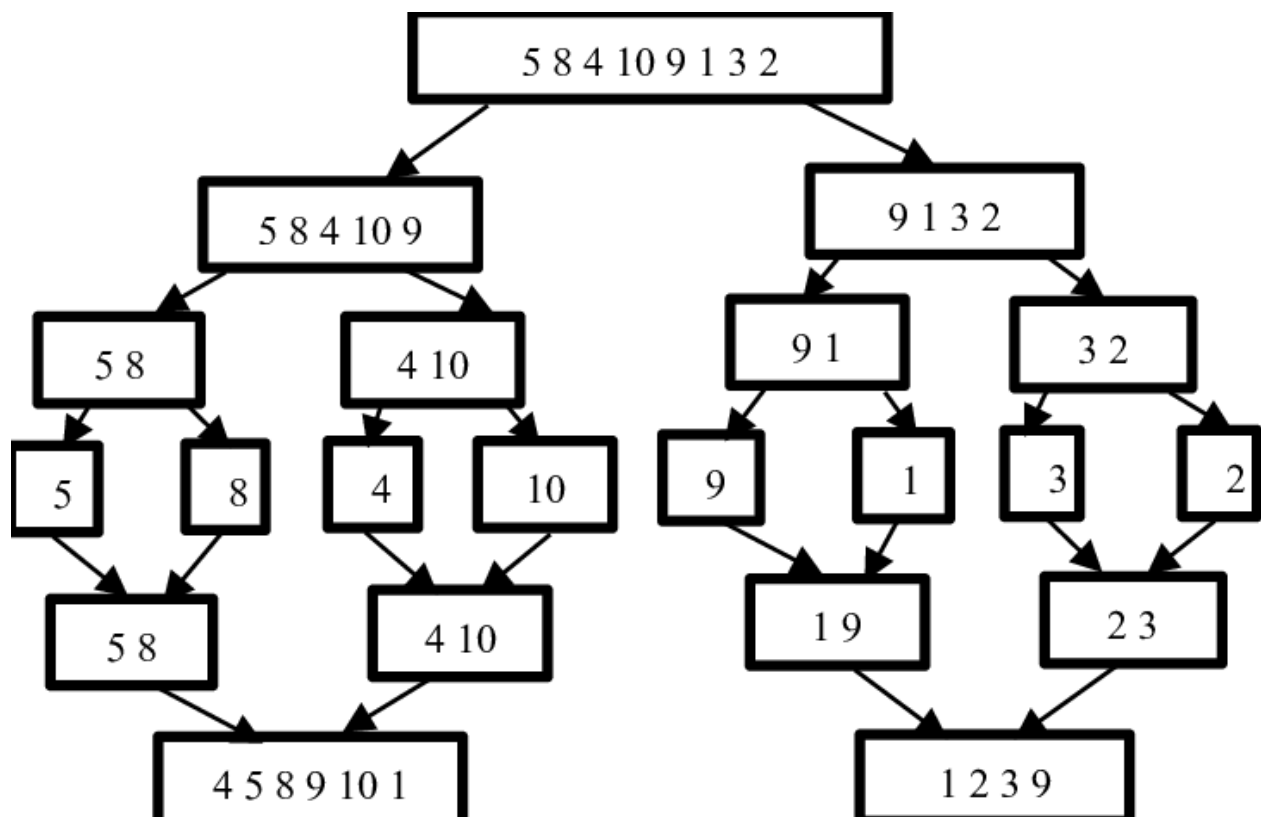
Heapify procedure can be applied to a node only if its children nodes are heapified. So the heapification must be performed in the bottom up order. Lets understand with the help of an example:



Merge sort -

Like QuickSort, Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging two halves.

The following diagram shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}. If we take a closer look at the diagram, we can see that the array is recursively divided in two halves till the size becomes 1. Once the size becomes 1, the merge processes comes into action and starts merging arrays back till the complete array is merged.

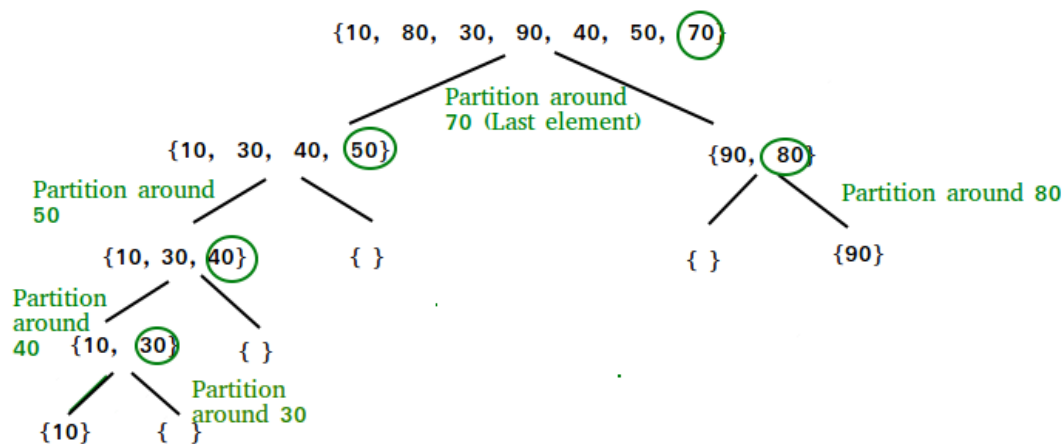


Quick sort-

It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.



Greedy method:

- Among all the algorithmic approaches, the simplest and straightforward approach is the Greedy method. In this approach, the decision is taken on the basis of current available information without worrying about the of the current decision in future.
- This approach is mainly used to solve optimization problems.
- Greedy method is easy to implement and quite efficient in most of the cases. Hence, we can say that Greedy algorithm is an algorithmic paradigm based on heuristic that follows local optimal choice at each step with the hope of finding global optimal solution.

Components of Greedy Algorithm

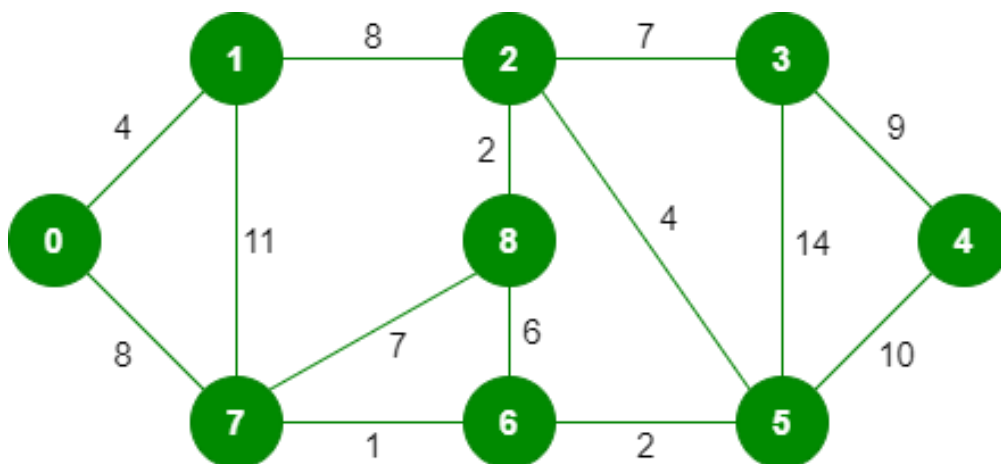
Greedy algorithms have the following five components –

- **A candidate set** – A solution is created from this set.
- **A selection function**-
Used to choose the best candidate to be added to the solution.
- **A feasibility function** – Used to determine whether a candidate can be used to contribute to the solution.
- **An objective function** – Used to assign a value to a solution or a partial solution.
 - **A solution function**-
Used to indicate whether a complete solution has been reached.

Minimum spanning tree-

- A spanning tree is a subset of an undirected Graph that has all the vertices connected by minimum number of edges.
- A Minimum Spanning Tree (MST) is a subset of edges of a connected weighted undirected graph that connects all the vertices together with the minimum possible total edge weight. To derive an MST, Prim's algorithm or Kruskal's algorithm can be used.
- As we have discussed, one graph may have more than one spanning tree. If there are n number of vertices, the spanning tree should have $n - 1$ number of edges. In this context, if each edge of the graph is associated with a weight and there exists more than one spanning tree, we need to find the minimum spanning tree of the graph.

Moreover, if there exist any duplicate weighted edges, the graph may have multiple minimum spanning tree.



Dijkstra's Algorithm for shortest path for a single source-

Dijkstra's algorithm solves the single-source shortest-paths problem on a directed weighted graph $G = (V, E)$, where all the edges are non-negative (i.e., $w(u, v) \geq 0$ for each edge $(u, v) \in E$).

In the following algorithm, we will use one function `Extract-Min()`, which extracts the node with the smallest key.

Algorithm: Dijkstra's-Algorithm (G, w, s)

for each vertex $v \in G.V$

$v.d := \infty$

$v.\pi := \text{NIL}$

```

s.d := 0
S :=  $\Phi$ 
Q := G.V
while Q  $\neq \Phi$ 
  u := Extract-Min (Q)
  S := S  $\cup$  {u}
  for each vertex v  $\in$  G.adj[u]
    if v.d > u.d + w(u, v)
      v.d := u.d + w(u, v)
      v. $\pi$  := u

```

Analysis

The complexity of this algorithm is fully dependent on the implementation of Extract-Min function. If extract min function is implemented using linear search, the complexity of this algorithm is $O(V^2 + E)$.

In this algorithm, if we use min-heap on which Extract-Min() function works to return the node from Q with the smallest key, the complexity of this algorithm can be reduced further.

Example

Let us consider vertex 1 and 9 as the start and destination vertex respectively. Initially, all the vertices except the start vertex are marked by ∞ and the start vertex is marked by 0.

matrix is shown in Fig. 1.

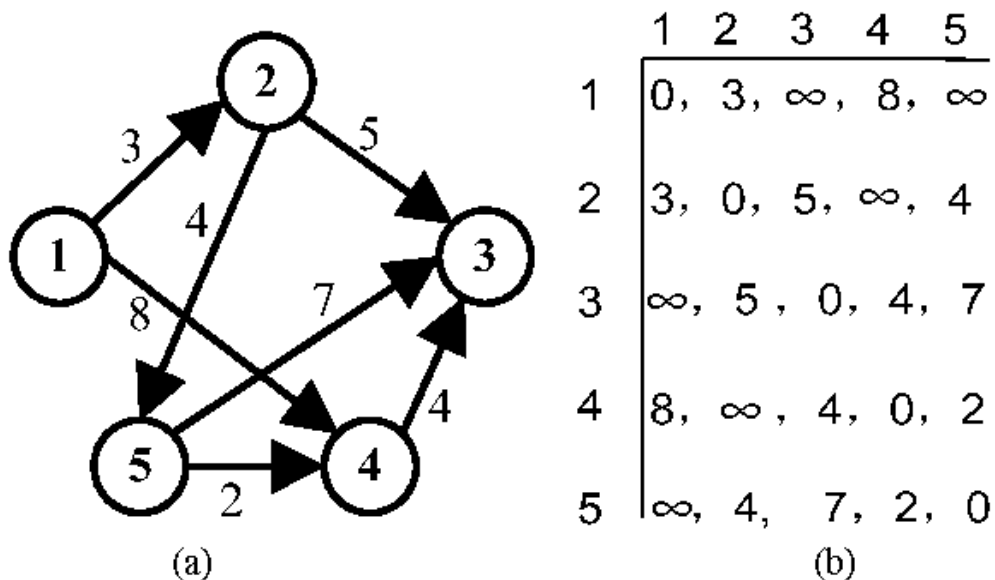


Figure 1. Map representation (a). weighted digraph (b). adjacency matrix

Fractional knapsack problem

Knapsack

- Given a set of items, each with a weight and a value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

Fractional Knapsack

In this case, items can be broken into smaller pieces, hence the thief can select fractions of items.

According to the problem statement,

- There are n items in the store
- Weight of i^{th} item $w_i > 0$
- Profit for i^{th} item $p_i > 0$ and
- Capacity of the Knapsack is W

In this version of Knapsack problem, items can be broken into smaller pieces. So, the thief may take only a fraction x_i of i^{th} item.

$$0 \leq x_i \leq 1$$

The i^{th} item contributes the weight $x_i \cdot w_i$ to the total weight in the knapsack and profit $x_i \cdot p_i$ to the total profit.

Hence, the objective of this algorithm is to

$$\text{maximize } \sum_{i=1}^n (x_i \cdot p_i)$$

subject to constraint,

$$\sum_{i=1}^n (x_i \cdot w_i) \leq W$$

It is clear that an optimal solution must fill the knapsack exactly, otherwise we could add a fraction of one of the remaining items and increase the overall profit.

Thus, an optimal solution can be obtained by

$$\sum_{i=1}^n (x_i \cdot w_i) = W$$

In this context, first we need to sort those items according to the value of $\frac{p_i}{w_i}$, so that $\frac{p_{i+1}}{w_{i+1}} \leq \frac{p_i}{w_i}$. Here, x is an array to store the fraction of items.

Algorithm: Greedy-Fractional-Knapsack ($w[1..n]$, $p[1..n]$, W)

for $i = 1$ to n

do $x[i] = 0$

weight = 0

for $i = 1$ to n

if weight + $w[i] \leq W$ then

$x[i] = 1$

weight = weight + $w[i]$

else

```

x[i] = (W - weight) / w[i]
weight = W
break
return x

```

Example

Let us consider that the capacity of the knapsack $W = 60$ and the list of provided items are shown in the following table –

Fractional Knapsack Problem						
Objects	1	2	3	4	5	6
Profit	5	10	15	7	8	9
Weight	1	3	5	4	1	3

As the provided items are not sorted based on $\frac{p_i}{w_i}$. After sorting, the items are as shown in the following table.

Optimal storage on tapes-

- Given programs stored on a computer tape and length of each program is L_i , find the order in which the programs should be stored in the tape for which the Mean Retrieval Time (MRT given as \bar{L}) is minimized.

Example:

Input : $n = 3$

$L[] = \{ 5, 3, 10 \}$

Output : Order should be $\{ 3, 5, 10 \}$ with $MRT = 29/3$

Let us first break down the problem and understand what needs to be done.

- A magnetic tape provides only sequential access of data. In an audio tape/cassette, unlike a CD, a fifth song from the tape can't be just directly played. The length of the first four songs must be traversed to play the fifth song. So in order to access certain data, head of the tape should be positioned accordingly.
- Now suppose there are 4 songs in a tape of audio lengths 5, 7, 3 and

2 mins respectively. In order to play the fourth song, we need to traverse an audio length of $5 + 7 + 3 = 15$ mins and then position the tape head.

Retrieval time of the data is the time taken to retrieve/access that data in its entirety. Hence retrieval time of the fourth song is $15 + 2 = 17$ mins.

- Now, considering that all programs in a magnetic tape are retrieved equally often and the tape head points to the front of the tape every time, a new term can be defined called the Mean Retrieval Time (MRT).

Unit – 3

Dynamic Programming

0-1 knapsack problem

In 0-1 Knapsack, items cannot be broken which means the thief should take the item as a whole or should leave it. This is reason behind calling it as 0-1 Knapsack.


Hence, in case of 0-1 Knapsack, the value of x_i can be either 0 or 1, where other constraints remain the same.

0-1 Knapsack cannot be solved by Greedy approach. Greedy approach does not ensure an optimal solution. In many instances, Greedy approach may give an optimal solution.

The following examples will establish our statement.

Example

Let us consider that the capacity of the knapsack is $W = 25$ and the items are as shown in the following table



0-1 Knapsack Algorithm

```

for w = 0 to W
    B[0,w] = 0
for i = 0 to n
    B[i,0] = 0
    for w = 0 to W
        if  $w_i \leq w$  // item i can be part of the solution
            if  $b_i + B[i-1, w-w_i] > B[i-1, w]$ 
                 $B[i, w] = b_i + B[i-1, w-w_i]$ 
            else
                 $B[i, w] = B[i-1, w]$ 
        else  $B[i, w] = B[i-1, w]$  //  $w_i > w$ 
  
```

8/14/2014 12

Without considering the profit per unit weight (p_i/w_i), if we apply Greedy approach to solve this problem, first item A will be selected as it will contribute maximum profit among all the elements.

After selecting item A, no more item will be selected. Hence, for this given set of items total profit is 24. Whereas, the optimal solution can be

achieved by selecting items, B and C, where the total profit is $18 + 18 = 36$.

Matrix chain multiplication problem

It is a Method under Dynamic Programming in which previous output is taken as input for next.

Here, Chain means one matrix's column is equal to the second matrix's row [always].

In general:

If $A = [a_{ij}]$ is a $p \times q$ matrix

$B = [b_{ij}]$ is a $q \times r$ matrix

$C = [c_{ij}]$ is a $p \times r$ matrix

Then

Given following matrices $\{A_1, A_2, A_3, \dots, A_n\}$ and we have to perform the matrix multiplication, which can be accomplished by a series of matrix multiplications

$$A_1 \times A_2 \times A_3 \times \dots \times A_n$$

Matrix Multiplication operation is **associative** in nature rather commutative. By this, we mean that we have to follow the above matrix order for multiplication but we are free to **parenthesize** the above multiplication depending upon our need.

In general, for $1 \leq i \leq p$ and $1 \leq j \leq r$

It can be observed that the total entries in matrix 'C' is 'pr' as the matrix is of dimension $p \times r$. Also each entry takes $O(q)$ times to compute, thus the total time to compute all possible entries for the matrix 'C' which is a multiplication of 'A' and 'B' is proportional to the product of the dimension $p \times q \times r$.

It is also noticed that we can save the number of operations by reordering the parenthesis.

Optimal binary search tree-

- A Binary Search Tree (BST) is a tree where the key values are stored in the internal nodes. The external nodes are null nodes.
- An optimal binary search tree is a BST, which has minimal expected cost of locating each node

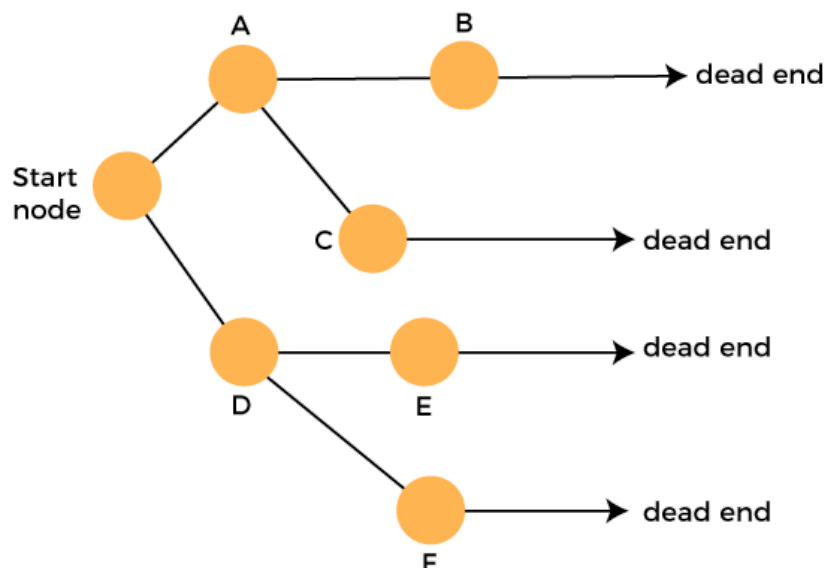
Unit – 4

Back Tracking

- The Backtracking is an algorithmic-method to solve a problem with an additional way. It uses a recursive approach to explain the problems. We can say that the backtracking is needed to find all possible combination to solve an optimization problem.
- **Backtracking** is a systematic way of trying out different sequences of decisions until we find one that "works."

In the following Figure:

- Each non-leaf node in a tree is a parent of one or more other nodes (its children)
- Each node in the tree, other than the root, has exactly one parent



8 Queen problem

- The eight queens problem is the problem of placing eight queens on an 8×8 chessboard such that none of them attack one another (no two are in the same row, column, or diagonal). More generally, the n queens problem places n queens on an $n \times n$ chessboard.

8 Queen problem Algorithm:

```

putQueen(row)
{
for every position col on the same row
if position col is available
place the next queen in position col
if (row<8)
putQueen(row+1);
else success;
the queen from position col
}

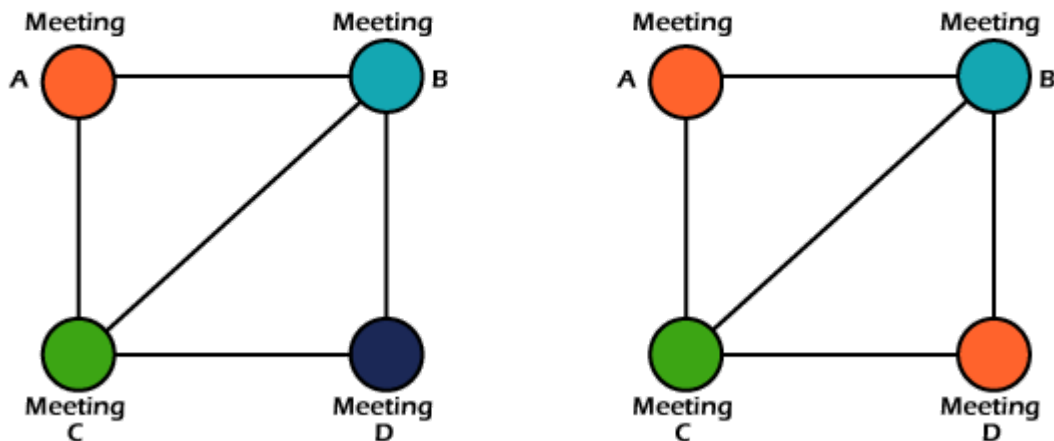
```

Chromatic number-

- The minimum number of colors required for vertex coloring of graph 'G' is called as the chromatic number of G, denoted by $\chi(G)$.
- $\chi(G) = 1$ if and only if 'G' is a null graph. If 'G' is not a null graph, then $\chi(G) \geq 2$

Example:

Colorings



Note – A graph 'G' is said to be n-coverable if there is a vertex coloring that uses at most n colors, i.e., $X(G) \leq n$.

Graph coloring-

- Graph coloring is nothing but a simple way of labelling graph components such as vertices, edges, and regions under some constraints. In a graph, no two adjacent vertices, adjacent edges, or adjacent regions are colored with minimum number of colors. This number is called the chromatic number and the graph is called a properly colored graph.

Applications of Graph Coloring:

Graph coloring is one of the most important concepts in graph theory. It is used in many real-time applications of computer science such as –

- Clustering
- Data mining
- Image capturing
- Image segmentation
- Networking
- Resource allocation
- Processes scheduling

Unit – 5

Branch and Bound

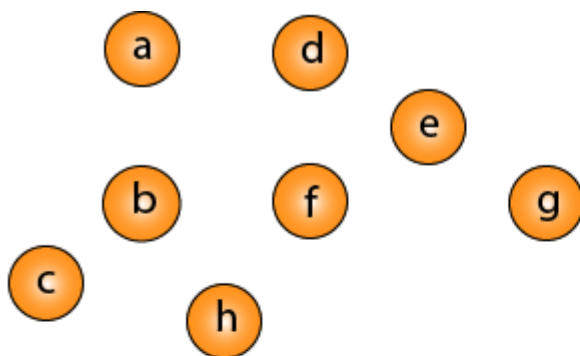
- Branch and bound (BB, B&B, or BnB) is an algorithm design paradigm for discrete and combinatorial optimization problems, as well as mathematical optimization. ... The algorithm explores branches of this tree, which represent subsets of the solution set.

Travelling salesman problem-

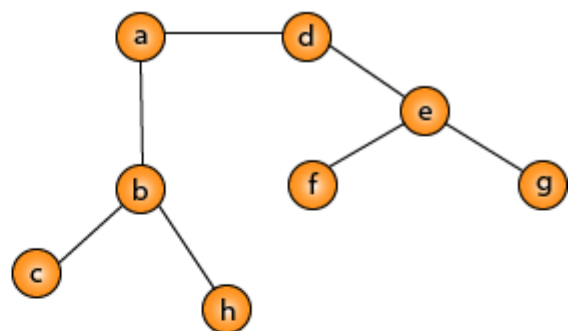
- The traveling salesman problem (TSP) is an algorithmic problem tasked with finding the shortest route between a set of points and locations that that must be visited. ... The 's goal is to keep both the travel costs and the distance traveled as low as possible.

Example

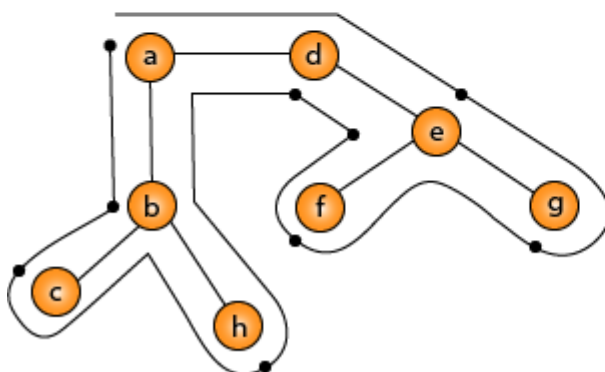
In the following example, we will illustrate the steps to solve the travelling salesman problem.



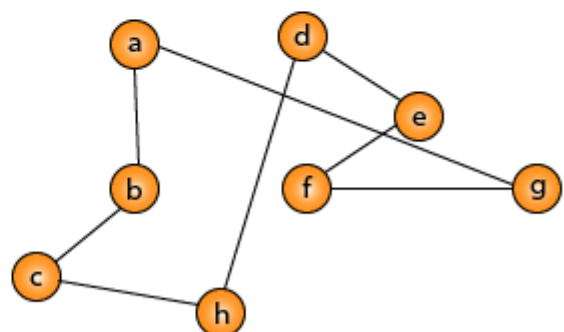
(1) A given set of points



(2) MST T



(3) Full tree walk on T.



(4) A preorder sequence gives a tour H.

