1. Explain time complexity with example

Time complexity is defined as the amount of time taken by an algorithm to run, as a function of the length of the input. It measures the time taken to execute each statement of code in an algorithm. It is not going to examine the total execution time of an algorithm. Rather, it is going to give information about the variation (increase or decrease) in execution time when the number of operations (increase or decrease) in an algorithm. Yes, as the definition says, the amount of time taken is a function of the length of input only.

There are different types of time complexities used, let's see one by one:

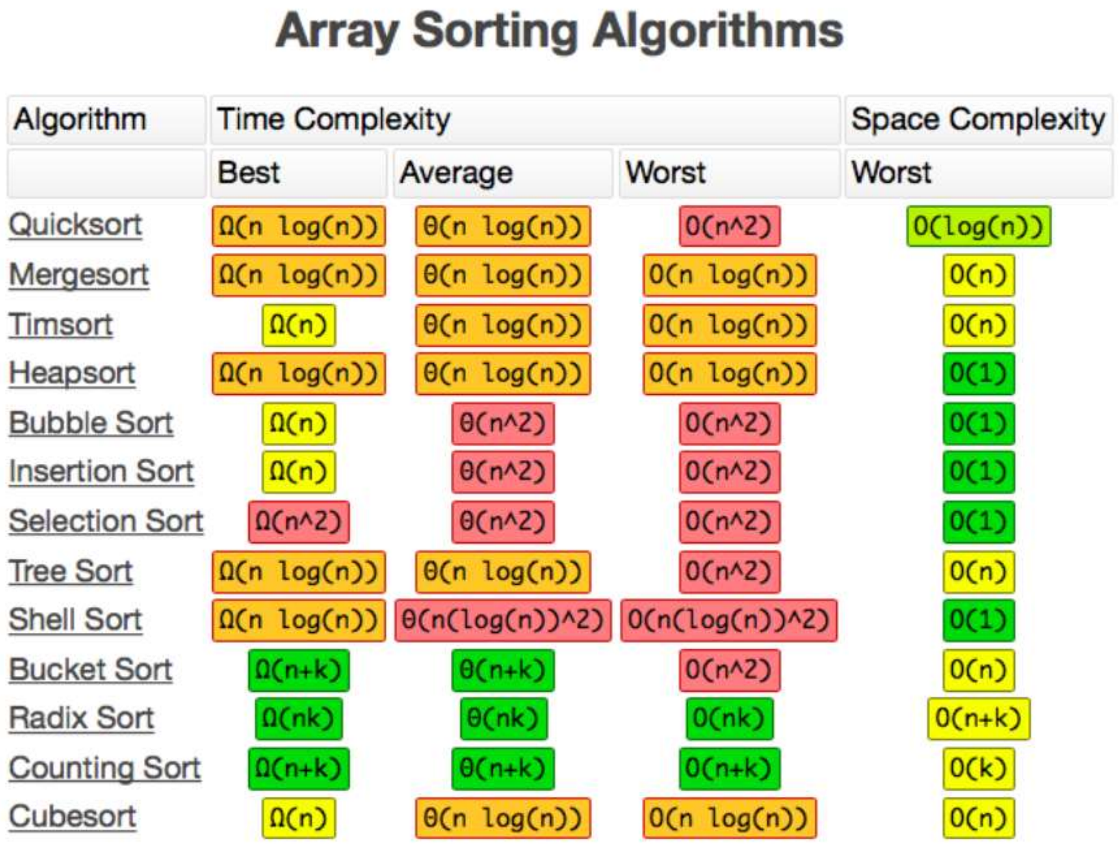
1. Constant time – O (1)

2. Linear time – O (n)

3. Logarithmic time – O (log n)

4. Quadratic time – O (n^2)

5. Cubic time – O (n^3)



Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

2. Explain maximum optimized solution.

An optimal solution is a feasible solution where the objective function reaches its maximum (or minimum) value – for example, the most profit or the least cost. A globally optimal solution is one where there are no other feasible solutions with better objective function values.

An algorithm is designed to achieve optimum solution for a given problem. In greedy algorithm approach, decisions are made from the given solution domain. As being greedy, the closest solution that seems to provide an optimum solution is chosen.

Greedy algorithms try to find a localized optimum solution, which may eventually lead to globally optimized solutions. However, generally greedy algorithms do not provide globally optimized solutions.

The maximized optimized solution is considered to be the best possible solution for a given set of constraints.

A greedy algorithm, as the name suggests, always makes the choice that seems to be the best at that moment. This means that it makes a locally-optimal choice in the hope that this choice will lead to a globally-optimal solution which is known as maximized optimized solution.

3. Explain which data structure used in DFS and how?

Depth First Search (DFS) algorithm traverses a graph in a depth ward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

We use the LIFO queue, i.e. stack, for implementation of the depth-first search algorithm because depth-first search always expands the deepest node in the current frontier of the search tree.

The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the frontier, so then the search "backs up" to the next deepest node that still has unexplored successors.

A LIFO queue means that the most recently generated node is chosen for expansion. This must be the deepest unexpanded node because it is one deeper than its parent — which, in turn, was the deepest unexpanded node when it was selected.

BFS	DFS
BFS stands for ''Breadth First Search"	DFS stsnds for Depth First Search.
BFS traverses the tree level wise.	DFS traverses the tree Depth wise.
BFS is implemented using queue which is FIFP list.	DFS is implemented using Stack which is LIFO list.
It's single step algorithm, wherein the visited vertices are removed from the queue and then displayed at once.	It's two step algorithm. In first stage, the visited vertices are pushed onto the stack and later on when there is no vertex further to visit those are popped out.
BFS requires more memory compare to DFS.	DFS requires less memory compare to BFS.
Applications of BFS: i) To find shortest path. ii) In Spanning tree. iii) In Connectivity.	Applications of DFS i)Useful in Cycle detection. ii)In Connectivity testing. iii)In finding Spanning trees & forest.
Always provides shallowest path solution.	Does'nt guarantee shallowest path solution.
No backtracking is required in BFS.	Backtracking is implemented in DFS.
Can never get trapped into finite loops.	Generally gets trapped into infinite loops.

4. Where NP classes used

P	Easily solvable in polynomial time.
NP	Yes, answers can be checked in polynomial time.
Co-NP	No, answers can be checked in polynomial time.
NP-hard	All NP-hard problems are not in NP and it takes a long time to check them.
NP-complete	A problem that is NP and NP-hard is NP-complete.

The NP in NP class stands for Non-deterministic Polynomial Time. It is the collection of decision problems that can be solved by a non-deterministic machine in polynomial time.

Features:

1. The solutions of the NP class are hard to find since they are being solved by a non-deterministic machine but the solutions are easy to verify.
2. Problems of NP can be verified by a Turing machine in polynomial time.

This class contains many problems that one would like to be able to solve effectively:

Boolean Satisfiability Problem (SAT).

Hamiltonian Path Problem.

Graph coloring.

5. Illustrate applications of Dijkstra algorithm.

Digital Mapping Services in Google Maps: Many times we have tried to find the distance in G-Maps, from one city to another, or from your location to the nearest desired location.

Social Networking Applications: In many applications you might have seen the app suggests the list of friends that a particular user may know.

Telephone Network: As we know, in a telephone network, each line has a bandwidth, 'b'. The bandwidth of the transmission line is the highest frequency that line can support. Generally, if the frequency of the signal is higher in a certain line, the signal is reduced by that line.

IP routing to find Open shortest Path First: Open Shortest Path First (OSPF) is a link-state routing protocol that is used to find the best path between the source and the destination router using its own Shortest Path First.

Flighting Agenda: For example, If a person needs software for making an agenda of flights for customers. The agent has access to a database with all airports and flights.

Designate file server: To designate a file server in a LAN(local area network), Dijkstra's algorithm can be used. Consider that an infinite amount of time is required for transmitting files from one computer to another computer.

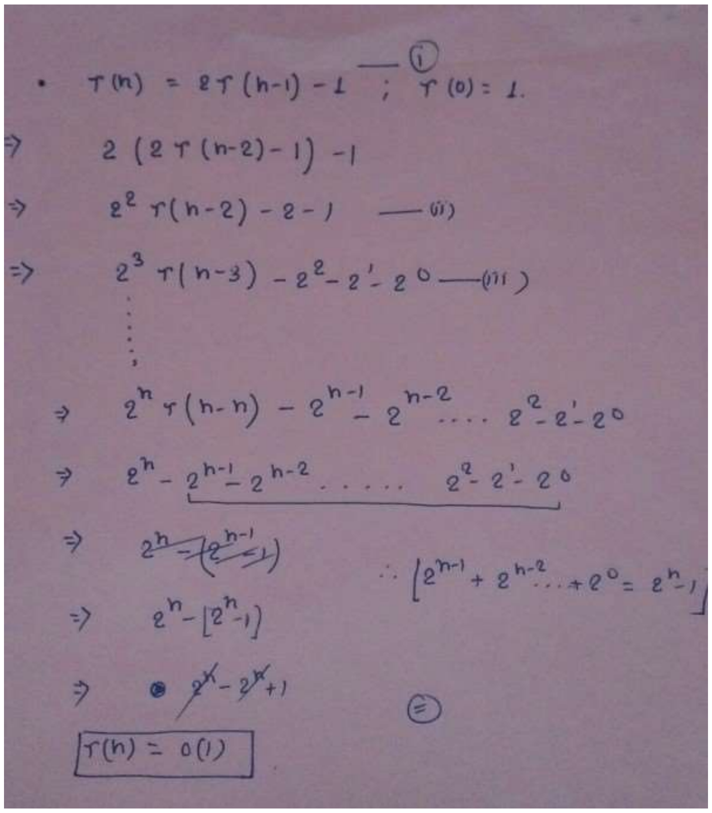
Robotic Path: Nowadays, drones and robots have come into existence, some of which are manual, some automated.

6.

Find the complexity of the recurrence:

T(n) = {2T(n-1) – 1, if n>0,

{1, otherwise



• $T(n) = 2T(n-1) - 1$ — (i) ; $T(0) = 1.$

$\Rightarrow 2(2T(n-2) - 1) - 1$

$\Rightarrow 2^2 T(n-2) - 2 - 1$ — (ii)

$\Rightarrow 2^3 T(n-3) - 2^2 - 2^1 - 2^0$ — (iii)

$\vdots$

$\Rightarrow 2^n T(n-n) - 2^{n-1} - 2^{n-2} \cdots 2^2 - 2^1 - 2^0$

$\Rightarrow 2^n - \underbrace{2^{n-1} - 2^{n-2} \cdots\cdots 2^2 - 2^1 - 2^0}$

$\Rightarrow$ ~~$2^n - (2^{n-1} - 1)$~~

$\therefore \left[2^{n-1} + 2^{n-2} \cdots + 2^0 = 2^n - 1\right]$

$\Rightarrow 2^n - [2^n - 1]$

$\Rightarrow \cancel{2^n} - \cancel{2^n} + 1$

$\boxed{T(n) = O(1)}$

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

$$= 2(2T(n-2)) - 2$$

....

$$= 2(2(2.\ldots.(2T(n-n)))) - (n+1)$$

$$= 2^{n+1}T(0) - (n+1)$$

So, time complexity $O(2^n)$

7.

Consider the following set of integers.
{20,25,57,48,37,12,92,86,33}
If one uses the quick sort algorithm to sort the above set of integers, how many p to completely sort the file?
Note: you may choose middle element as a pivot?

The number of p's that are required to completely sort the given array by the method of the Quick Sort Algorithm is found to be: 5

Explanation:

The Quick sort is based on the concept of a divide-and-conquer algorithm that is also used in merge sort. The difference is that in fast sort, the most important or required work is done when splitting (or dividing) an array into sub-arrays, whereas in merge sort, all the main work is done by merging sub-arrays. That is for quicksort, the join step is irrelevant.

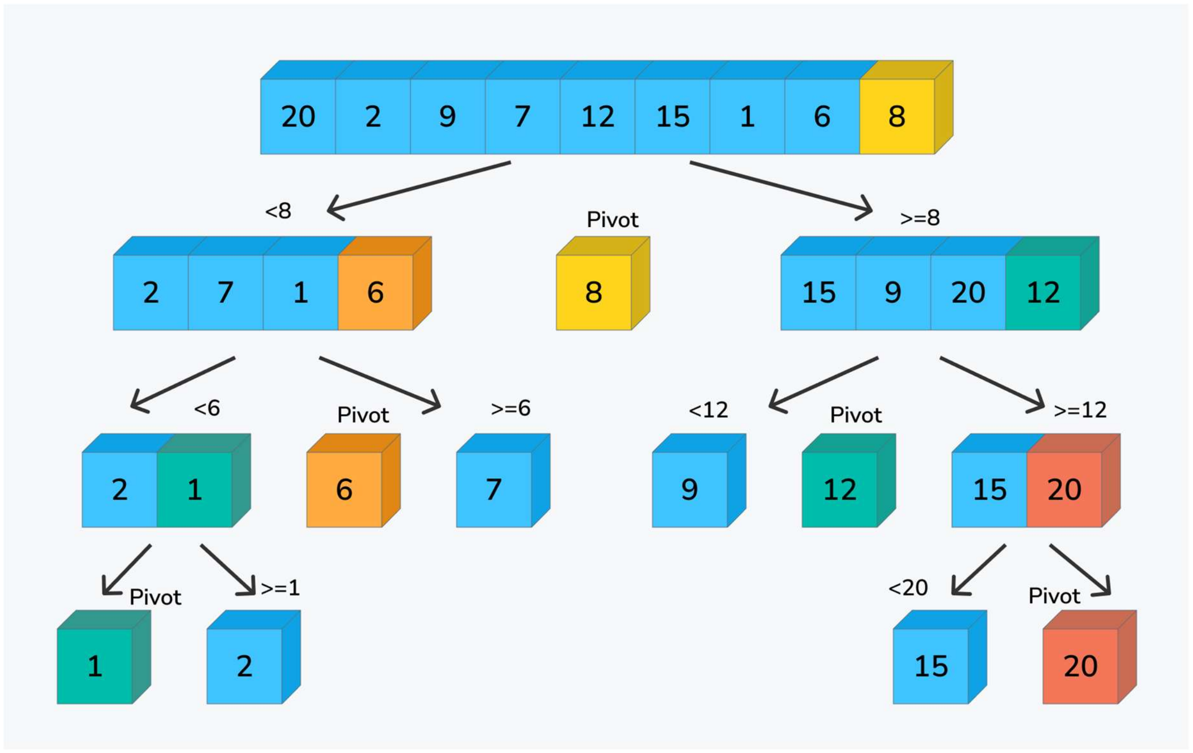
Quick sort is also known as partition exchange sort. This algorithm divides an array of specified numbers into three main parts.

- Elements less than pivot elements
- Pivot element
- Elements larger than the pivot element

Pivot elements can be selected in various ways from the specified numbers.

- Selection of the first element
- Last item selection (example shown)
- Select a random item

- Median selection



8. Illustrate functioning of BFS with suitable example

Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.

Applications of BFS algorithm

The applications of breadth-first-algorithm are given as follows -

- BFS can be used to find the neighboring locations from a given source location.

- In a peer-to-peer network, BFS algorithm can be used as a traversal method to find all the neighboring nodes. Most torrent clients, such as BitTorrent, uTorrent, etc. employ this process to find "seeds" and "peers" in the network.
- BFS can be used in web crawlers to create web page indexes. It is one of the main algorithms that can be used to index web pages. It starts traversing from the source page and follows the links associated with the page. Here, every web page is considered as a node in the graph.
- BFS is used to determine the shortest path and minimum spanning tree.
- BFS is also used in Cheney's technique to duplicate the garbage collection.
- It can be used in ford-Fulkerson method to compute the maximum flow in a flow network.

Algorithm

The steps involved in the BFS algorithm to explore a graph are given as follows -

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set

their STATUS = 2

(waiting state)

[END OF LOOP]

Step 6: EXIT

BFS, Breadth-First Search, is a vertex-based technique for finding the shortest path in the graph. It uses a Queue data structure that follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue. It is slower than DFS.

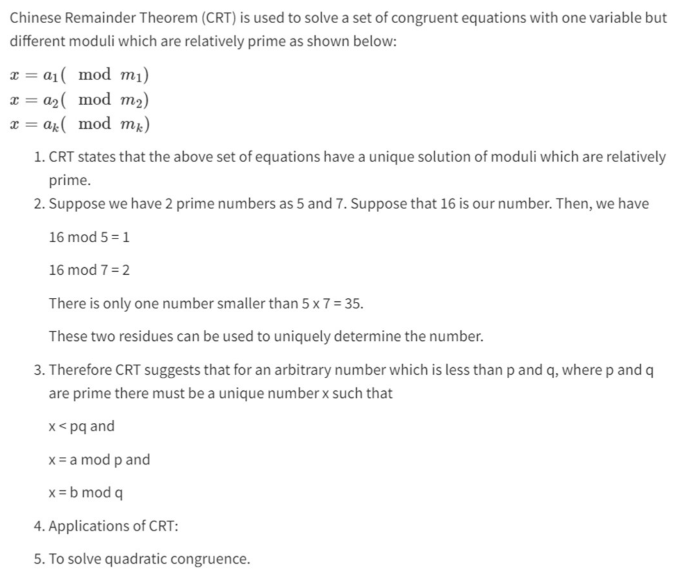
9.

Solve using Chinese Remainder theorem.

X=2(mod 3)

X=3(mod 5)

X=2(mod 7)



Chinese Remainder Theorem (CRT) is used to solve a set of congruent equations with one variable but different moduli which are relatively prime as shown below:

$$x = a_1 (\mod m_1)$$
$$x = a_2 (\mod m_2)$$
$$x = a_k (\mod m_k)$$

1. CRT states that the above set of equations have a unique solution of moduli which are relatively prime.
2. Suppose we have 2 prime numbers as 5 and 7. Suppose that 16 is our number. Then, we have

   16 mod 5 = 1

   16 mod 7 = 2

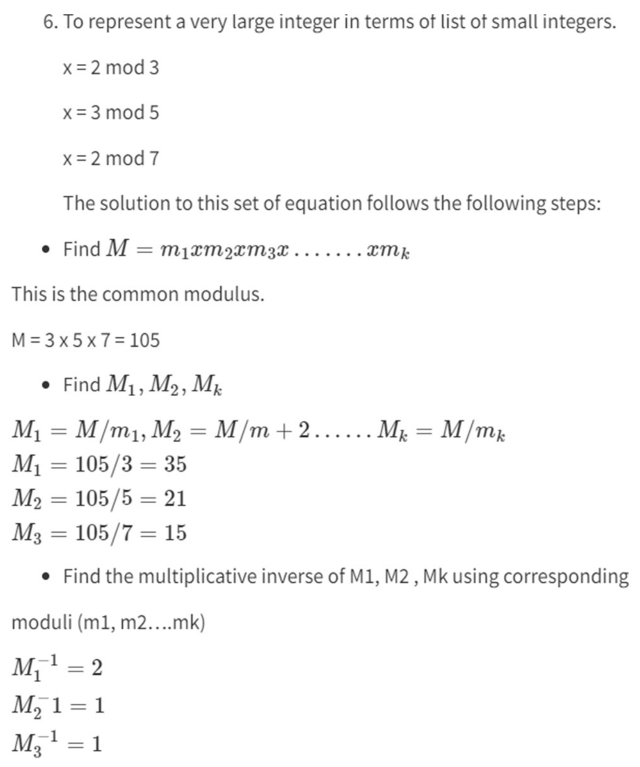
   There is only one number smaller than 5 x 7 = 35.

   These two residues can be used to uniquely determine the number.
3. Therefore CRT suggests that for an arbitrary number which is less than p and q, where p and q are prime there must be a unique number x such that

   x < pq and

   x = a mod p and

   x = b mod q
4. Applications of CRT:
5. To solve quadratic congruence.



6. To represent a very large integer in terms of list of small integers.

   x = 2 mod 3

   x = 3 mod 5

   x = 2 mod 7

   The solution to this set of equation follows the following steps:

- Find $M = m_1 x m_2 x m_3 x \ldots\ldots. x m_k$

This is the common modulus.

M = 3 x 5 x 7 = 105

- Find M_1, M_2, M_k

$$M_1 = M/m_1, M_2 = M/m + 2 \ldots\ldots M_k = M/m_k$$
$$M_1 = 105/3 = 35$$
$$M_2 = 105/5 = 21$$
$$M_3 = 105/7 = 15$$

- Find the multiplicative inverse of M1, M2 , Mk using corresponding

moduli (m1, m2….mk)

$$M_1^{-1} = 2$$
$$M_2^{-}1 = 1$$
$$M_3^{-1} = 1$$

M = 3 x 5 x 7 = 105

- Find M_1, M_2, M_k

$M_1 = M/m_1, M_2 = M/m + 2 \ldots\ldots M_k = M/m_k$
$M_1 = 105/3 = 35$
$M_2 = 105/5 = 21$
$M_3 = 105/7 = 15$

- Find the multiplicative inverse of M1, M2 , Mk using corresponding

moduli (m1, m2….mk)

$M_1^{-1} = 2$
$M_2^{-}1 = 1$
$M_3^{-1} = 1$

- The solution to simultaneous equation is

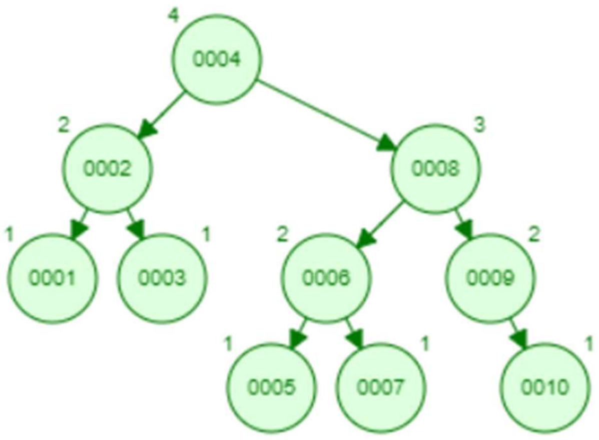
$x = (a_1 x M_1 x M_1^{-1} + a_2 x M_2 x M_2^{-1} + a_3 x M_3 x M_3^{-1}) \mod M$
$x = (2x35x2 + 3x21x1 + 2x15x1) \mod 105$
$x = (140 + 63 + 30) mod 105$
$x = 233 mod 105$
$[233/105 \rightarrow dividend = 2, remainder(\mod) = 23]$

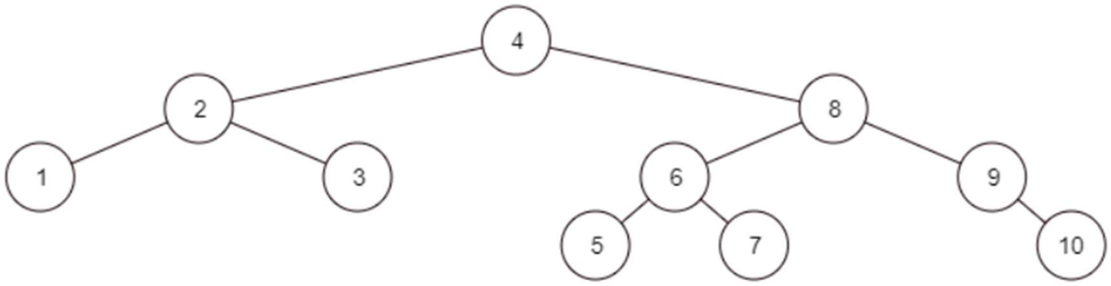
Therefore, x = 23 mod 105

10.

Construct AVL tree: 1 2 3 4 5 6 7 8 9 10

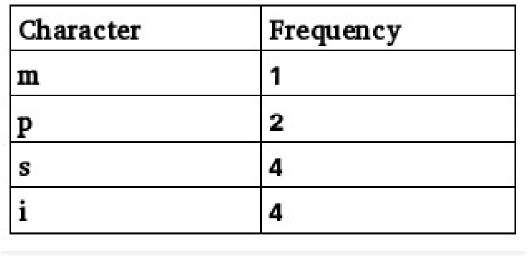
AVL Tree: An AVL tree is a binary search tree in which the heights of the left and right subtrees of the root differ by at most 1 and in which the left and right subtrees are again AVL trees. An AVL Tree is a form of binary tree, however unlike a binary tree, the worst case scenario for a search is O(log n). The AVL data structure achieves this property by placing restrictions on the difference in height between the sub-trees of a given node, and re-balancing the tree if it





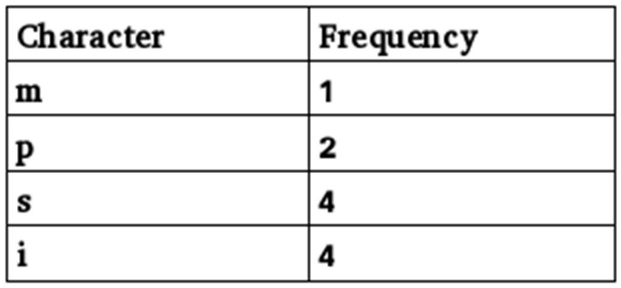
11.

a) How many bits may be required for encoding the message ‘mississippi’? Following is the frequency table of characters in ‘mississippi’ in non-decreasing order of frequency:



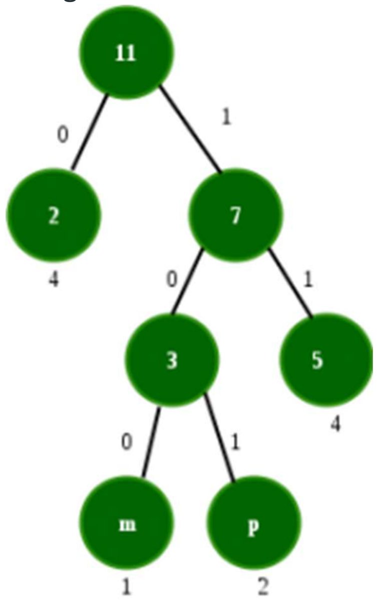
Character	Frequency
m	1
p	2
s	4
i	4

Following is the frequency table of characters in ‘mississippi’ in non-decreasing order of frequency:



Character	Frequency
m	1
p	2
s	4
i	4

The generated Huffman tree is:



Following are the codes:

Character	Frequency	Code	Code Length
m	1	100	3
p	2	101	3
s	4	11	2
i	4	0	1

Total number of bits
= freq(m) * codelength(m) + freq(p) * code_length(p) + freq(s) * code_length(s) + freq(i) * code length(i)
= 1*3 + 2*3 + 4*2 + 4*1 = 21

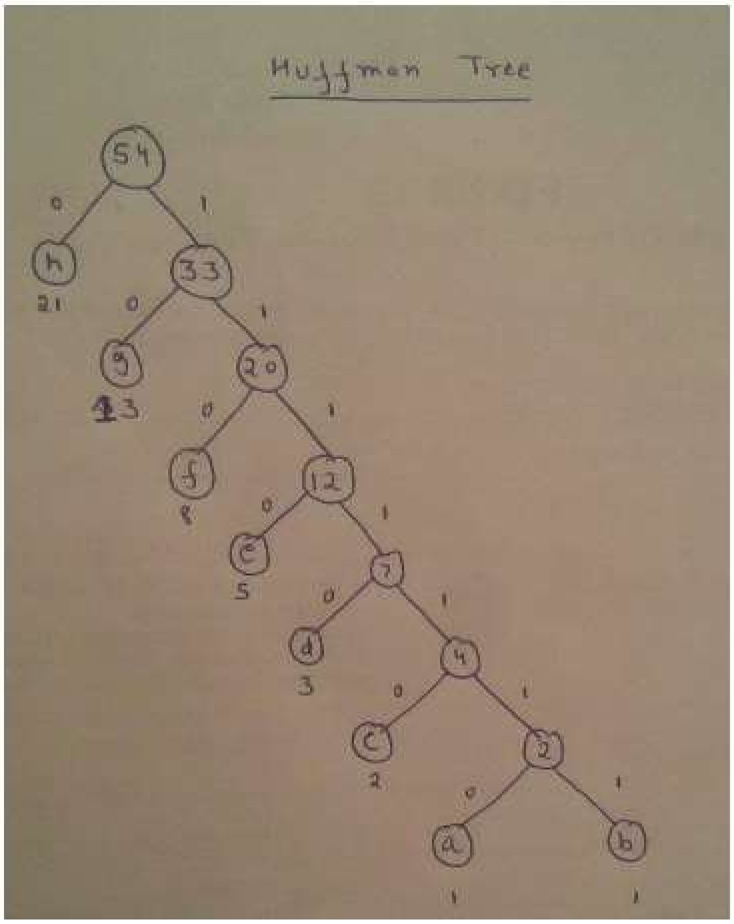
Also, average bits per character can be found as:
Total number of bits required / total number of characters = 21/11 = 1.909

b) The characters a to h have the set of frequencies based on the first 8 Fibonacci numbers as follows: a : 1, b : 1, c : 2, d : 3, e : 5, f : 8, g : 13, h : 21

A Huffman code is used to represent the characters. What is the sequence of characters corresponding to the following code?

110111100111010

Generating Prefix codes using Huffman Coding.
First we apply greedy algorithm on the frequencies of the characters to generate the binary tree as shown in the Figure given below. Assigning 0 to the left edge and 1 to the right edge, prefix codes for the characters are as
below.



a – 1111110
b – 1111111
c – 111110
d – 11110
e – 1110
f – 110
g – 10
h – 0

Given String can be decomposed as

110 11110 0 1110 10

f d h e g

c) What are the steps to build a Huffman Tree from input characters? Explain with the help of an example.

Step 1: For each character of the node, create a leaf node. The leaf node of a character contains the frequency of that character.

Step 2: Set all the nodes in sorted order according to their frequency.

Step 3: There may exist a condition in which two nodes may have the same frequency. In such a case, do the following: Create a new internal node.

1. The frequency of the node will be the sum of the frequency of those two nodes that have the same frequency.
2. Mark the first node as the left child and another node as the right child of the newly created internal node.

Step 4: Repeat step 2 and 3 until all the node forms a single tree. Thus, we get a Huffman tree.

Example

12.

Explain spurious hits in Rabin-Karp string matching algorithm with example. Working modulo q=13, how many spurious hits does the Rabin-Karp matcher encounter in the text T = 2359023141526739921 when looking for the pattern P = 31415?

Given pattern P = 31415, and prime number q = 13

P mod q = 31415 mod 13 = 7

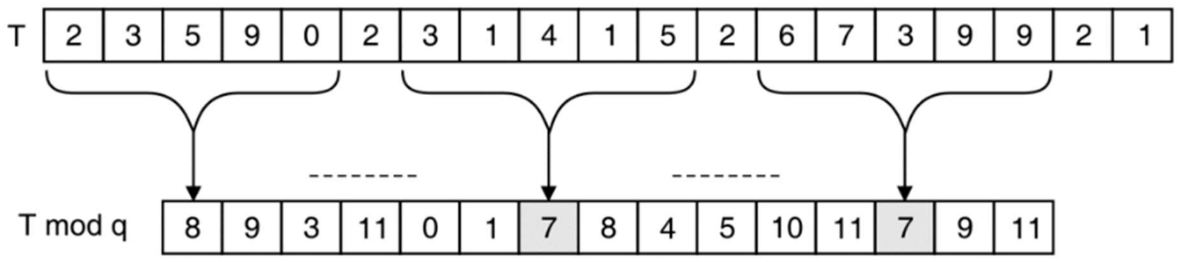
Let us find hash value for given text :

T[1…5] = 23590 mod 13 = 8

T[2…6] = 35902 mod 13 = 9

:

T[(m – 4) … m] = 39921 mod 13 = 11



- The hash value of pattern P is 7. We have two such values in a hash(T). So there may be a spurious hit or actual string. In given text T, one hit is an actual match and one is spurious hit.
- If the hash of pattern and any substring in the text is same, we have two possibilities :

(i) **Hit :** Pattern and text are same

(ii) **Spurious hit :** Hash value is same but pattern and corresponding text is not same

Here, m = length of pattern = 5.

Consider

t_s = 23590. Next value t_{s+1} is derived as,

$t_{s+1} = 10(t_s) - 10^m * T[s+1] + T[s + m + 1])$

$= (10*23590) - 10^5 * 2 + 2$

$= 235900 - 200000 + 2 = 35902$

t_{s+2} = 10(t_{s+1}) – 10^{m}* T[s+2] + T[s + m + 2])

= (10*35902) – 10^5 * 3 + 3

= 359020 – 300000 + 3 = 59023

- In same way, we can compute the next t_{s+i} using incremental approach.
- Rabin Karp algorithm matches hash value, rather than directly comparing actual string value. If hash value of pattern P and the hash value of subsequence in string T are same, the actual value of strings is compared using brute force approach. Like t_{s+1}, we can also derive hash value incrementally as shown below.

Calculation for hash of 14152

If t_s is known, the hash value can directly be derived for t_{s+1} as follow.

Hash for t_{s+1} = (d*(t_s – T[s+1]h) + T[s + m + 1]) mod q

= 10*(31415 – (3*10^4 mod 13)) + 2 mod 13

= 10(31415 – 9) + 2 mod 13

= 314062 mod 13 = 8