|  |  |
| --- | --- |
| Fizz buzz:  for num in number:  If num % 15 == 0:  print(“fizz buzz”)  elif num % 3 == 0:  print(“fizz”)  else: print(num) | Program should return n natural numbers in ascending order replacing any number divisible by three with the word "fizz", and any number divisible by five with the word "buzz", and any number divisible by both 3 and 5 with the word "fizzbuzz". |
| Binary search Tree:  Class Node(Obj):  def \_\_init\_\_(self,val):  self.l\_child = None  self.r\_chile = None  self.val = val  class BinarySearchTree(Obj):  def insert(self, root, node):  if root is None:  return node  if root.val < node.val:  root.r\_child = self.insert(  root.r\_child, node)  else:  root.l\_child = self.insert(  root.l\_child, node)  return root  def in\_order\_place(self,root):  if not root:  return None  else:  self.in\_order\_place(root.l\_child)  print root.val  self.in\_order\_place(root.r\_child)  def pre\_order\_place(self,root):  if not root:  return None  else:  self.pre\_order\_place(root.l\_child)  print root.val  self.pre\_order\_place(root.r\_child)  def post\_order\_place(self,root):  if not root:  return None  else:  self.post\_order\_place(root.l\_child)  print root.val  self.post\_order\_place(root.r\_child) | A Binary Search Tree (BST) is a special type of binary tree in which the left child of a node has a value less than the node’s value and the right child has a value greater than the node’s value. This property is called the BST property and it makes it possible to efficiently search, insert, and delete elements in the tree.  The left and right subtree each must also be a binary search tree.  There must be no duplicate nodes (BST may have duplicate values with different handling approaches)  Handling approach for Duplicate values in the Binary Search tree:  You cannot allow the duplicated values at all.  We must follow a consistent process throughout i.e either store duplicate value at the left or store the duplicate value at the right of the root but be consistent with your approach.  We can keep the counter with the node and if we found the duplicate value, then we can increment the counter  Below are the various operations that can be performed on a BST:  Insert a node into a BST: A new key is always inserted at the leaf. Start searching a key from the root till a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node. |
| Dijkstra’s Algirithms:  Finding the shortest path between the nodes in a graph |  |
| Dynamic Programming:  Used for optimization:  Breaking down complicated problems into sub problem in a recursive manner in a bottom up approach. |  |
| Edit distance: |  |
| Weighted Job Scheduling Algorithms: |  |
| Longest Common Subsequence |  |
| Fibonacci Number |  |
| Longest Common Substring |  |
| Fibonacci number |  |

DYNAMIC ALGOS

|  |  |
| --- | --- |
| Kruskal’s Algorithms  ccc |  |
| Greedy Algorithms  Huffman Coding:  Activity Selection Problem:  Change making Problem: |  |
| Prim’s Algorithm  Spanning Tree: |  |
| Bellman FOrd Algorithm  Single source shortest path: |  |
| Floyd Warshall Algorithm  All pair shortest Algorithm: |  |
| Multi-Threaded Algorithm  Squared matrix multiplication:  Vector matrix multiplication:  Merge sort: |  |
| KMP Algorithm  Knuth Morris Pratt: |  |
| Online Algorithm  Paging caching: |  |
|  |  |

Sorting

|  |  |
| --- | --- |
| Bubble sort  Merge sort |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |