1. Write a Program to insert and delete an element into a linear array.

data =[int(x) for x in input("Enter Array: ").split()]  
print("Array is :",data)  
pos=int(input("Enter Position for insert element : "))  
pos=pos-1  
num=int(input("Enter insert Number : "))  
data.insert(pos,num)  
print("After inserted element in Array is :",data)  
dltnum=int(input("Enter delete number : "))  
data.remove(dltnum)  
print("The Final Array is :",data)

1. Write a program to sort a linear array using the bubble sort algorithm.

def bubbleSort(array):  
 # loop to access each array element  
 for i in range(len(array)):  
  
 # loop to compare array elements  
 for j in range(0, len(array) - i - 1):  
  
 # compare two adjacent elements  
 if array[j] > array[j + 1]:  
 # swapping elements  
 temp = array[j]  
 array[j] = array[j + 1]  
 array[j + 1] = temp  
  
  
# Get input from user  
data = [int(x) for x in input("Enter Array: ").split()]  
  
# Calling function  
bubbleSort(data)  
  
# Print Output  
print('Sorted Array in Ascending Order:')  
print(data)

1. Write a program to find an element using a linear search algorithm.

# Linear Search in Python  
def linearSearch(array, element):  
 n = len(array)  
 find = False  
 pos = []  
 # Going through array sequencially  
 for i in range(0, n):  
 if (array[i] == element):  
 pos.append(i + 1)  
 find = True  
 if (find == True):  
 return pos  
 else:  
 return -1  
  
  
data = [int(x) for x in input("Enter Array: ").split()]  
print("Array is :", data)  
element = int(input("Enter your searching element : "))  
result = linearSearch(data, element)  
if (result == -1):  
 print("Element not found")  
else:  
 print("Element found at position(s): ", result)

1. Write a program to find an element using the binary search algorithm.

# Binary Search in python  
def binarySearch(array, x, low, high):  
 # Repeat until the pointers low and high meet each other  
 while low <= high:  
 mid = (low + high) // 2  
 if array[mid] == x:  
 return mid  
 elif array[mid] < x:  
 low = mid + 1  
 else:  
 high = mid - 1  
 return -1  
  
  
data = [int(x) for x in input("Enter Array: ").split()]  
data.sort()  
print("Array is :", data)  
element = int(input("Enter your searching element : "))  
result = binarySearch(data, element, 0, len(data) - 1)  
if result == -1:  
 print("Element not found")  
else:  
 print("Element is present at position ", result + 1)

1. Write a program to sort a linear array using the merge sort algorithm.

# Python program for implementation of MergeSort  
def mergeSort(arr):  
 if len(arr) > 1:  
 # Finding the mid of the array  
 mid = len(arr)//2  
 # Dividing the array elements  
 L = arr[:mid]  
 # into 2 halves  
 R = arr[mid:]  
 # Sorting the first half  
 mergeSort(L)  
 # Sorting the second half  
 mergeSort(R)  
 i = j = k = 0  
 # Copy data to temp arrays L[] and R[]  
 while i < len(L) and j < len(R):  
 if L[i] < R[j]:  
 arr[k] = L[i]  
 i += 1  
 else:  
 arr[k] = R[j]  
 j += 1  
 k += 1  
 # Checking if any element was left  
 while i < len(L):  
 arr[k] = L[i]  
 i += 1  
 k += 1  
  
 while j < len(R):  
 arr[k] = R[j]  
 j += 1  
 k += 1  
  
arr = [int(x) for x in input("Enter Array: ").split()]  
print("Array is :",arr)  
mergeSort(arr)  
print("Sorted array is: ",arr)

1. Write a program to sort a linear array using the Selection Sort algorithm.

# Python program for implementation of Selection  
import sys  
  
data =[int(x) for x in input("Enter Array: ").split()]  
print("Array is :",data)  
  
# Traverse through all array elements  
for i in range(len(data)):  
  
 # Find the minimum element in remaining  
 # unsorted array  
 min\_idx = i  
 for j in range(i + 1, len(data)):  
 if data[min\_idx] > data[j]:  
 min\_idx = j  
  
 # Swap the found minimum element with  
 data[i], data[min\_idx] = data[min\_idx], data[i]  
  
print("Sorted Array is :",data)

1. Write a program to find a given pattern from text using the first pattern matching algorithm.

#https://www.geeksforgeeks.org/naive-algorithm-for-pattern-searching/  
def matching(pat, txt):  
 M = len(pat)  
 N = len(txt)  
 for i in range(N - M + 1): # N -M+1 eto gula window possible  
 j = 0  
 # For current index i, check  
 # for pattern match \*/  
 while (j < M):  
 if (txt[i + j] != pat[j]):  
 break  
 j += 1  
  
 if (j == M):  
 print("Pattern found at Position ", i+1)  
txt = input('Enter Text : ')  
print('Text is : ',txt)  
pat = input('Enter Pattern : ')  
matching(pat, txt)

1. Write a program to solve 𝑛 queen's problem using backtracking.

# https://www.codesdope.com/course/algorithms-backtracking/  
  
def is\_attack(i, j, board, N):  
 # checking for column j  
 for k in range(1, i):  
 if(board[k][j] == 1):  
 return True  
  
 # checking upper right diagonal  
 k = i-1  
 l = j+1  
 while (k>=1 and l<=N):  
 if (board[k][l] == 1):  
 return True  
 k=k+1  
 l=l+1  
  
 # checking upper left diagonal  
 k = i-1  
 l = j-1  
 while (k>=1 and l>=1):  
 if (board[k][l] == 1):  
 return True  
 k=k-1  
 l=l-1  
  
 return False  
  
def n\_queen(row, n, N, board):  
 if (n==0):  
 return True  
  
 for j in range(1, N+1):  
 if not(is\_attack(row, j, board, N)):  
 board[row][j] = 1  
  
 if (n\_queen(row+1, n-1, N, board)):  
 return True  
  
 board[row][j] = 0 #backtracking  
 return False  
  
  
board = [[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0]]  
  
n\_queen(1, 4, 4, board)  
  
 #printing the matix  
for i in range(1, 5):  
 print(board[i][1:])

1. Write a program to find the shortest path from a graph using Kruskal’s Algorithm.

class Graph:  
 def \_\_init\_\_(self, vertices):  
 self.V = vertices  
 self.graph = []  
  
 def add\_edge(self, u, v, w):  
 self.graph.append([u, v, w])  
  
 # Search function  
 def find(self, parent, i):  
 if parent[i] == i:  
 return i  
 return self.find(parent, parent[i])  
  
 def apply\_union(self, parent, rank, x, y):  
 xroot = self.find(parent, x)  
 yroot = self.find(parent, y)  
 if rank[xroot] < rank[yroot]:  
 parent[xroot] = yroot  
 elif rank[xroot] > rank[yroot]:  
 parent[yroot] = xroot  
 else:  
 parent[yroot] = xroot  
 rank[xroot] += 1  
  
 # Applying Kruskal algorithm  
 def kruskal\_algo(self):  
 result = []  
 i, e = 0, 0  
 self.graph = sorted(self.graph, key=lambda item: item[2])  
 parent = []  
 rank = []  
 for node in range(self.V):  
 parent.append(node)  
 rank.append(0)  
 while e < self.V - 1:  
 u, v, w = self.graph[i]  
 i = i + 1  
 x = self.find(parent, u)  
 y = self.find(parent, v)  
 if x != y:  
 e = e + 1  
 result.append([u, v, w])  
 self.apply\_union(parent, rank, x, y)  
 total = 0  
 for u, v, weight in result:  
 print("%d to %d = %d" % (u, v, weight))  
 total = total+weight  
 print("The Total Cost : ",total)  
  
  
vertex = int(input("Enter Total Vertex : "))  
edges = int(input("Enter Total Edges : "))  
g = Graph(vertex)  
for i in range(edges):  
 data = [int(x) for x in input("Enter Vi , Vj and Weight : ").split()]  
 g.add\_edge(*data[0], data[1], data[2])  
g.kruskal\_algo*

# 5 7  
# 0 1 4  
# 0 2 8  
# 1 2 1  
# 1 3 3  
# 2 3 7  
# 2 4 3  
# 3 4 8  
# Output = 11

1. Write a program using greedy method to solve this problem when no of job 𝑛=5, profits (𝑃1,𝑃2,𝑃3,𝑃4,𝑃5) = (3,25,1,6,30) and deadlines (𝑑1,𝑑2,𝑑3,𝑑4,𝑑5) = (1,3,2,1,2).

class Job:  
 def \_\_init\_\_(self, taskId, deadline, profit):  
 self.taskId = taskId  
 self.deadline = deadline  
 self.profit = profit  
  
  
# Function to schedule jobs to maximize profit  
def scheduleJobs(jobs, T):  
 profit = 0  
 # list to store used and unused slots info  
 slot = [-1] \* T  
 # arrange the jobs in decreasing order of their profits  
 jobs.sort(key=lambda x: x.profit, reverse=True)  
  
 # consider each job in decreasing order of their profits  
 for job in jobs:  
 # search for the next free slot and map the task to that slot  
 for j in reversed(range(job.deadline)):  
 if j < T and slot[j] == -1:  
 slot[j] = job.taskId  
 profit += job.profit  
 break  
  
 print('The scheduled jobs are', list(filter(lambda x: x != -1, slot)))  
 print('The total profit earned is', profit)  
  
  
  
jobs = [  
 Job(1, 1, 3), Job(2, 3, 25), Job(3, 2, 1), Job(4, 1, 6),Job(5, 2, 30)  
]  
T = len(jobs)  
scheduleJobs(jobs, T)

1. Write a program to solve the following 0/1 Knapsack using dynamic programming approach profits 𝑃= (15,25,13,23), weight 𝑊=(2,6,12,9), Knapsack 𝐶=20, and the number of items n=4.

# Program for 0-1 Knapsack problem  
def knapSack(W, wt, val, n):  
 dp = [0 for i in range(W+1)] # Making the dp array  
  
 for i in range(1, n+1): # taking first i elements  
 for w in range(W, 0, -1): # starting from back,so that we also have data of  
 # previous computation when taking i-1 items  
 if wt[i-1] <= w:  
 # finding the maximum value  
 dp[w] = max(dp[w], dp[w-wt[i-1]]+val[i-1])  
  
 return dp[W]  
  
  
# Driver code  
P = [15, 25, 13, 23]  
wt = [2, 6, 12, 9]  
C = 20  
n = len(P)  
print("Maximum Profit for 0/1 Knapsack Using Dynamic Programing is : ", knapSack(C,wt,P,n))

1. Write a program to solve the Tower of Hanoi problem for the 𝑁 disk.

#https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi/  
# Recursive Python function to solve tower of hanoi  
def TowerOfHanoi(n, from\_rod, to\_rod, aux\_rod):  
 if n == 0:  
 return  
 TowerOfHanoi(n-1, from\_rod, aux\_rod, to\_rod)  
 print("Move disk", n, "from rod", from\_rod, "to rod", to\_rod)  
 TowerOfHanoi(n-1, aux\_rod, to\_rod, from\_rod)  
  
  
# Driver code  
N = int(input("Enter the disk's number for Tower of Hanoi: "))  
# N = 3  
# A, C, B are the name of rods  
TowerOfHanoi(N, 'A', 'C', 'B')  
  
# Contributed By Harshit Agrawal

1. Write a program to implement a queue data structure along with its typical operations.

queue = [int(x) for x in input("Enter The Elements : ").split()]  
print("Queue Elements is : ",queue)  
# Removing elements from the queue  
print("\nDequeue Elements From Queue are : ",queue.pop(0),end=" ")  
print(queue.pop(0))  
  
print("After removing Queue elements is : ",queue)