Code:

```
def FibonacciIterative(no, res):
    a, b = 0, 1
    print(a,b, end= " ")
    for i in range(2, no+1):
        c = a + b
        res += 1
        print(c, end=" ")
        a, b = b, c
    return res + 1

def FibonacciRecursive(n):
    global res
    if n <= 1:
        return n
    res += 1
    return FibonacciRecursive(n-1) + FibonacciRecursive(n-2)

def FibonacciDP(n, res):
    lst = [0] * (n + 1)
    lst[0], lst[1] = 0, 1
    for i in range(2, n+1):
        res += 1
        lst[i] = lst[i-1] + lst[i-2]

    print(lst)
    return res

n = int(input("Enter number : "))
res = 0
print("Iterative : ", end="")
res = FibonacciIterative(n, res)
print("\nNumber of Basic operations : ",res, "\n")
res = 0
print("Recursive : ", FibonacciRecursive(n),end="")
print("\nNumber of Basic operations : ",res, "\n")
res = 0
print("Dynamic : ", end = "")
res = FibonacciDP(n, res)
print("Number of Basic operations : ",res)
```

Output :

```
========= RESTART: C:/Users/Admin/Documents/DAA PGM/DAA ASSIGNMENT 1.py =
Enter number : 6
Iterative : 0 1 1 2 3 5 8
Number of Basic operations :  6

Recursive :  8
Number of Basic operations :  12

Dynamic : [0, 1, 1, 2, 3, 5, 8]
Number of Basic operations :  5
```

Code:

```
def knapsack(maxw,val,wt,n,memo):

    if n == 0 or maxw == 0 :
        # memo[n][maxw] = 0
        return 0
    if memo[n][maxw] != 0:
        return memo[n][maxw]
    if wt[n-1]<=maxw:
        memo[n][maxw] = (max(val[n-1]+knapsack(maxw-wt[n-1],val,wt,n-
1,memo),knapsack(maxw,val,wt,n-1,memo)))
        # print(memo)
        return memo[n][maxw]
    else:
        memo[n][maxw] = knapsack(maxw,val,wt,n-1,memo)
        return memo[n][maxw]

if __name__ == "__main__":
    #maxw=5
    #val=[12,10,20,15]
   # wt=[2,1,3,2]
    maxw = int((input("Enter Weight of Sack : ")))
    val = list(map(int, input("Enter Profit : ").split()))
    wt = list(map(int, input("Enter weights : ").split()))

    memo = [[0 for i in range(maxw + 1)] for j in range(len(val) + 1)]
    print("Maximun Profit :",knapsack(maxw,val,wt,len(val),memo))
    print("Memoization : ")
    for i in memo:
        print(i)
```

Output:

```
============ RESTART: C:/Users/Admin/Documents/DAA PGM/DAA LAB 4.py
Enter Weight of Sack : 5
Enter Profit : 12 10 20 15
Enter weights : 2 1 3 2
Maximun Profit : 37
Memoization :
[0, 0, 0, 0, 0, 0]
[0, 0, 12, 12, 12, 12]
[0, 0, 12, 22, 0, 22]
[0, 0, 0, 22, 0, 32]
[0, 0, 0, 0, 0, 37]
```

Code:

```
import random

def quicksort(arr, start, stop):
    if(start < stop):
        pivotindex = partitionrand(arr,start, stop)
        quicksort(arr , start , pivotindex)
        quicksort(arr, pivotindex + 1, stop)

def partitionrand(arr , start, stop):
    randpivot = random.randrange(start, stop)
    arr[start], arr[randpivot] =arr[randpivot], arr[start]
    print("Pivot : ",arr[start], end=" ")
    print("Elements in array : ",arr)
    return partition(arr, start, stop)

def partition(arr,start,stop):
    pivot = start
    i = start - 1
    j = stop + 1
    while True:
        while True:
            i = i + 1
            if arr[i] >= arr[pivot]:
                break
        while True:
            j = j - 1
            if arr[j] <= arr[pivot]:
                break
        if i >= j:
            return j
        arr[i] , arr[j] = arr[j] , arr[i]

if __name__ == "__main__":
    array1 = list(map(int, input("Enter elements in array : ").split()))
    print("Array before sorting : ", array1)
    quicksort(array1, 0, len(array1) - 1)
    print("Array after sorting : ", array1)
```

Output:

```
Enter elements in array : 2 3 4 5 9 8 7 6 1 0
Array before sorting :  [2, 3, 4, 5, 9, 8, 7, 6, 1, 0]
Pivot :  1 Elements in array :  [1, 3, 4, 5, 9, 8, 7, 6, 2, 0]
Pivot :  8 Elements in array :  [0, 8, 4, 5, 9, 3, 7, 6, 2, 1]
Pivot :  4 Elements in array :  [0, 1, 4, 5, 9, 3, 7, 6, 2, 8]
Pivot :  5 Elements in array :  [0, 1, 2, 5, 9, 3, 7, 6, 4, 8]
Pivot :  4 Elements in array :  [0, 1, 2, 4, 3, 9, 7, 6, 5, 8]
Pivot :  5 Elements in array :  [0, 1, 2, 3, 4, 5, 7, 6, 9, 8]
Pivot :  6 Elements in array :  [0, 1, 2, 3, 4, 5, 6, 7, 9, 8]
Pivot :  9 Elements in array :  [0, 1, 2, 3, 4, 5, 6, 9, 7, 8]
Pivot :  8 Elements in array :  [0, 1, 2, 3, 4, 5, 6, 8, 7, 9]
Array after sorting :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Code :

```
from collections import dequefrom
ctypes import sizeof

class Queue:

    def init (self): self.q =
        deque()

    def enq(self, value):
        self.q.appendleft(value)

    def deq(self):
        if len(self.q) > 0: return
            self.q.pop()
        else:
            return None

    def__len__(self): return
        len(self.q)

    def___repr__(self):
        if len(self.q) > 0:
            s = "<enqueue here>\n_____\n"
            s += "\n_____\n".join([str(item) for item in self.q])s += "\n
            _____\n<dequeue here>"
            return s
        else:
            return "<queue is empty>"


class Node(object):

    def init (self, value=None):self.value
        = value self.left   =   None
        self.right = None

    def set_value(self, value):self.value
        = value

    def get_value(self): return
        self.value

    def set_left_child(self, left):self.left =
        left
```

```
    def set_right_child(self, right):self.right =
        right

    def get_left_child(self):return
        self.left

    def get_right_child(self):return
        self.right


    def has_left_child(self): return
        self.left != None

    def has_right_child(self): return
        self.right != None

    def__repr__(self):
        return f"Node({self.get_value()})"

    def__str__(self):
        return f"Node({self.get_value()})"

class Tree:

    def init (self): self.root =
        None

    def set_root(self, value): self.root =
        Node(value)

    def get_root(self): return
        self.root

    def repr (self):level = 0
        q = Queue() visit_order =
        list() node = self.get_root()
        q.enq((node, level))
        while (len(q) > 0):
            node, level = q.deq()if node
            == None:
                visit_order.append(("<empty>", level))continue
            visit_order.append((node, level))if
            node.has_left_child():
                q.enq((node.get_left_child(), level + 1))else:
                q.enq((None, level + 1))
```

```
                if node.has_right_child(): q.enq((node.get_right_child(), level + 1))
                else:
                    q.enq((None, level + 1))

            s = "Tree\n"
            previous_level = -1
            for i in range(len(visit_order)):node, level
                = visit_order[i] if level ==
                previous_level:
                    s += " | " + str(node)else:
                    s += "\n" + str(node)
                    previous_level = level

            return s

def return_frequency(data):
    # Take a string and determine the relevant frequencies of the charactersfrequency = {}
    for char in data:
        if char in frequency:
            frequency[char] += 1
        else:
            frequency[char] = 1
    lst = [(v, k) for k, v in frequency.items()]
    # Build and sort a list of tuples from lowest to highest frequencieslst.sort(reverse=True)
    return lst


# A helper function to the build_tree()def
sort_values(nodes_list, node):
    node_value, char1 = node.valueindex =
    0
    max_index = len(nodes_list)while
    True:
        if index == max_index:
            nodes_list.append(node)return
        current_val, char2 = nodes_list[index].valueif current_val
        <= node_value:
            nodes_list.insert(index, node)return
        index += 1

# Build a Huffman Tree: nodes are stored in list with their values(frequencies) in
descending order.
# Two nodes with the lowest frequencies form a tree node. That node getspushed back into the
list and the process repeats
def build_tree(data):
    lst = return_frequency(data)
    nodes_list = []
```

```
    for node_value   in   lst: node =
        Node(node_value)
        nodes_list.append(node)


    while len(nodes_list) != 1: first_node =
        nodes_list.pop() second_node =
        nodes_list.pop() val1, char1 =
        first_node.value val2, char2 =
        second_node.value
        node = Node((val1 + val2, char1 + char2))
        node.set_left_child(second_node)
        node.set_right_child(first_node)
        sort_values(nodes_list, node)

    root = nodes_list[0]tree =
    Tree() tree.root = root


    while start_index != max_index:
        if data[start_index : end_index] in reversed_dict:
            s += reversed_dict[data[start_index : end_index]]start_index =
            end_index
        end_index += 1
    return tree

# the function traverses over the huffman tree and returns a dictionary withletter as keys and binary
value and value.
# function get_codes() is for encoding purposesdef
get_codes(root):
    if root is None:return
        {}
    frequency, characters = root.value
    char_dict = dict([(i, '') for i in list(characters)])left_branch =
    get_codes(root.get_left_child())
    for key, value in left_branch.items(): char_dict[key] += '0' +
        left_branch[key]
    right_branch = get_codes(root.get_right_child())for key, value
    in right_branch.items():
        char_dict[key] += '1' + right_branch[key]

    return char_dict

# when we've got the dictionary of binary values and huffman tree, treeencoding is simple
def huffman_encoding_func(data):if data
    == '':
        return None, '' tree =
```

```
        build_tree(data)
        dict = get_codes(tree.root)codes =
        "
        for char   in   data: codes +=
                dict[char]
        return tree, codes

# The function traverses over the encoded data and checks if a certain pieceof binary code could
actually be a letter
def huffman_decoding_func(data, tree):if data ==
        ":
                return "
        dict = get_codes(tree.root)
        reversed_dict = {}
        for value, key in dict.items():
                reversed_dict[key] = value
        start_index = 0
        end_index = 1 max_index
        = len(data)s = "


        return s



def main():
        print("Welcome to Huffman Encoding and Decoding!")
        while(True):
                choice = int(input("Select 1 for Encoding, 2 for Decoding, 3 to exit:
"))

                if (choice == 1):
                        senctence = input("Enter your sentence to encode: ")
                        print("Encoding process: ")
                        print("The content of the data is: {}".format(senctence))tree, encoded_data
                        = huffman_encoding_func(senctence) print("The content of the encoded
                        data is:
{}".format(encoded_data))

                elif(choice == 2):
                        encoded_data = input("Enter huffman code to decode: ")
                        print("Decoding process: ")
                        print("The content of the encoded data is:
{}".format(encoded_data))
                        decoded_data = huffman_decoding_func(encoded_data, tree)print("The
                        content of the encoded data is:
{}".format(decoded_data))
```

```
            else:
                break

# Using the special variable _name if__name
=="__main_":
    main()
```

OUTPUT:

```
Welcome to Huffman Encoding and Decoding!
Select 1 for Encoding, 2 for Decoding, 3 to exit:
1Enter your sentence to encode: she sells
Encoding process:

The content of the data is: she sells

The content of the encoded data is: 00010110110011101000
Select 1 for Encoding, 2 for Decoding, 3 to exit: 2

Enter huffman code to decode: 00010110110011101000
Decoding process:

The content of the encoded data is: 00010110110011101000
The content of the encoded data is: she sells

Select 1 for Encoding, 2 for Decoding, 3 to exit: 3
```

CODE:

```
result = []
def isSafe(board, row, col):

    # Check this row on left sidefor i
    in range(col):
        if (board[row][i]):
            return False

    # Check upper diagonal on left sidei =
    row
    j = col
    while i >= 0 and j >= 0:
        if(board[i][j]):
            return Falsei -
        = 1
        j -= 1

    # Check lower diagonal on left sidei =
    row
    j = col
    while j >= 0 and i < n:
        if(board[i][j]):
            return Falsei =
        i + 1
        j = j - 1
    return True

def solveNQUtil(board, col):if
    (col == n):
        v = []
        for i in board:
            for j in range(len(i)):if i[j]
                == 1:
```

```
                        v.append(j+1)
            result.append(v)
            return True
        res = False
        for i in range(n):
            if (isSafe(board, i, col)):
                board[i][col] = 1
                res = solveNQUtil(board, col + 1) or res
                board[i][col] = 0
        return res


def printboard(l1):
        print(l1)
        board = [[0 for j in range(n)] for i in range(n)]for row in
        range(len(l1)):
            col = l1[row]
            board[row][col-1] = 1

        for i   in   range(n): for j in
            range(n):
                print(board[i][j] ,end=" ")
            print("")

# Driver Code
n = int(input("Enter size of chess board: ")) board = [[0 for
j in range(n)] for i in range(n)]solveNQUtil(board, 0)
print("No of solutions found for", n , " Queen problem : ", len(result))for i in
range(len(result)):
        a = input("Press enter to see solutions!")
        printboard(result[i])
```

OUTPUT:

```
======================== RESTART: C:/Users/Admin/nqueen.py
Enter size of chess board: 4
No of solutions found for 4  Queen problem :  2
Press enter to see solutions!
[3, 1, 4, 2]
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
Press enter to see solutions!
[2, 4, 1, 3]
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0
```

Code:

```
a = list(map(int, input("Enter elements : ").split()))
b = list(map(int, input("Enter weights : ").split()))
#a = [15,12,10,14]
#b = [4,2,2,3]
BagSize = int(input("Enter Bag Size : "))
FilledBagSize = 0

flst = [[a[i], b[i], (a[i]/b[i])] for i in range(0, len(a))]
flst = sorted(flst, key = lambda x : x[-1], reverse=True)

lst = []
for i in flst:
    if FilledBagSize < BagSize :
        if (FilledBagSize + i[1]) > BagSize :
            temp = BagSize - FilledBagSize
            profit = (temp / i[1]) * i[0]
            lst.append([i[0], str(temp)+"/"+str(i[1]), profit])
            FilledBagSize += temp
        else:
            lst.append([i[0], str(i[1]), i[0]])
            FilledBagSize += i[1]

sum = 0
for i in range(len(lst)):
    sum += lst[i][-1]
print("Selected items : ", lst)
print("Maximum Profit :",sum)
```

Output: