Source Code(#1)

```
arr=list(map(int,input("Enter an Array: ").split()))
print("Your Array: ",arr)

print("\nInsertion Operation")
ele=int(input("Enter an element: "))
pos=int(input("Enter the position: "))
arr.insert(pos-1,ele)
print("After insertion: ",arr)

print("\nDeletion Operation")
pos=int(input("Enter a position to delete an element: "))
arr.remove(arr[pos-1])
print("After deletion your array: ",arr)
```

OUTPUT

```
Enter an Array: 1 7 4 2 5 3 8
Your Array:  [1, 7, 4, 2, 5, 3, 8]

Insertion Operation
Enter an element: 11
Enter the position: 2
After insertion:  [1, 11, 7, 4, 2, 5, 3, 8]

Deletion Operation
Enter a position to delete an element: 4
After deletion your array:  [1, 11, 7, 2, 5, 3, 8]
```

Source Code(#2)

```
def bubble(arr):
    for i in range(len(arr)):
        for j in range(i+1,len(arr)):
            if arr[i]>arr[j]:
                arr[i],arr[j]=arr[j],arr[i]
    return arr


arr=list(map(int,input("Enter an array: ").split()))
print("After Bubble sort: ",bubble(arr))
```

OUTPUT

```
Enter an array: 1 7 4 2 5 3 8
After Bubble sort:  [1, 2, 3, 4, 5, 7, 8]
```

Source Code(#3)

```
def linear_search(arr,key):
        n=len(arr)
        for pos in range(n):
                if arr[pos]==key:
                        return pos
        return -1


arr=list(map(int,input("Enter an array: ").split()))
print("Your array: ",arr)

print("\nLinear Search Start")
key=int(input("Enter the element: "))

result=linear_search(arr,key)
if result!=-1:
        print("Element is present at index: ",result)
else:
        print("Element is not present in array")
```

OUTPUT

```
Enter an array: 1 7 4 2 5 3 8
Your array:  [1, 7, 4, 2, 5, 3, 8]

Linear Search Start
Enter the element: 4
Element is present at index:  2
```

Source Code(#4)

```
def binary_search(arr, low, high, key):
    if high >= low:
        mid = (high + low) // 2
        if arr[mid] == key:
            return mid
        elif arr[mid] > key:
            return binary_search(arr, low, mid - 1, key)
        else:
            return binary_search(arr, mid + 1, high, key)
    else:
        return -1

arr=list(map(int,input("Enter an array: ").split()))
print("Your Array: ",arr)
arr.sort()

print("\nBinary Searching")
key=int(input("Enter the element: "))

result = binary_search(arr, 0, len(arr)-1, key)
if result!=-1:
    print("Element is present at index", result-1)
else:
    print("Element is not present in array")
```

OUTPUT

```
Enter an array: 1 7 4 2 5 3 8
Your Array:  [1, 7, 4, 2, 5, 3, 8]

Binary Searching
Enter the element: 4
Element is present at index 2
```

Source Code(#5)

```
def merge(arr,start,mid,end):
    merged=[0]*(end-start+1)
    idx1=start
    idx2=mid+1
    x=0

    while idx1<=mid and idx2<=end:
        if arr[idx1]<=arr[idx2]:
            merged[x]=arr[idx1]
            x+=1
            idx1+=1
        else:
            merged[x]=arr[idx2]
            x+=1
            idx2+=1
    while idx1<=mid:
        merged[x]=arr[idx1]
        x+=1
        idx1+=1

    while idx2<=end:
        merged[x]=arr[idx2]
        x+=1
        idx2+=1

    j=start
    for i in range(len(merged)):
        arr[j]=merged[i]
        j+=1

def merge_sort(arr,start,end):
    if start<end:
        mid = start+(end-start)//2
        merge_sort(arr,start,mid)
        merge_sort(arr,mid+1,end)
        merge(arr,start,mid,end)

arr=list(map(int,input("Enter the array: ").split()))
print("Entered Array: ",arr)
#function call
merge_sort(arr,0,len(arr)-1)
print("Sorted array: ",arr)
```

OUTPUT

```
Enter the array: 1 7 4 2 5 3 8
Entered Array:  [1, 7, 4, 2, 5, 3, 8]
Sorted array:  [1, 2, 3, 4, 5, 7, 8]
```

Source Code(#6)

```
def selection_sort(arr):
    n=len(arr)
    for i in range(n-1):
        minimum=i
        for j in range(i+1,n):
            if arr[minimum]>arr[j]:
                minimum=j
        arr[i],arr[minimum]=arr[minimum],arr[i]
    return arr

arr = list(map(int,input("Enter an Array: ").split()))
print("Entered Array: ",arr)
print("Sorted Array: ",selection_sort(arr))
```

OUTPUT

```
Enter an Array: 1 7 4 2 5 3 8
Entered Array:  [1, 7, 4, 2, 5, 3, 8]
Sorted Array:  [1, 2, 3, 4, 5, 7, 8]
```

Source Code(#7)

```
def pattern_matching(pat,txt):
        pl=len(pat)
        tl=len(txt)
        c=True

        for i in range(tl-pl+1):
                j=0
                while j<pl:
                        if txt[i+j]!=pat[j]:
                                Break
                        j+=1
                if j==pl:
                        print("Pattern Found in: ",i," index")
                c=False
        if c:
                print("Pattern Not Found")


str=input("Enter String: ")
pat=input("Enter Pattern: ")
pattern_matching(pat,str)
```

OUTPUT

```
Enter String: AAAGDAAAKDFAAA
Enter Pattern: AAA
Pattern Found in:  0  index
Pattern Found in:  5  index
Pattern Found in:  11  index
```

Source Code(#8)

```
def isSafe(mat,r,c):
    # Column check
    for i in range(len(mat)):
        if mat[i][c]=='Q':
            return False

    # diagonal check '\'
    i,j=r,c
    while i>=0 and j>=0:
        if mat[i][j]=='Q':
        return False
        i-=1
        j-=1

    # diagonal Check '/'
    i,j=r,c
    while i>=0 and j<len(mat):
        if mat[i][j]=='Q':
        return False
        i-=1
        j+=1

    return True

def printSolve(mat):
    for r in mat:
        print(str(r).replace(","," ").replace("\'"," "))
    print()

def NQueen(mat,r=0):
    if r==len(mat):
        printSolve(mat)
        return

    for i in range(len(mat)):
        if isSafe(mat,r,i):
            mat[r][i]='Q'
            NQueen(mat,r+1)
            mat[r][i]='-'

N=int(input("Enter N: "))
mat=[['-']*N for x in range(N)]
```

```
NQueen(mat)
```

OUTPUT

```
Enter N: 4
[ -     Q     -     - ]
[ -     -     -     Q ]
[ Q     -     -     - ]
[ -     -     Q     - ]

[ -     -     Q     - ]
[ Q     -     -     - ]
[ -     -     -     Q ]
[ -     Q     -     - ]
```

Source Code(#9)

```
def find(parent,i):
    if parent[i]==i:
        return i
    return find(parent,parent[i])

def union(parent,x,y):
    parent[x]=y
    return parent

def kruskals(g_nodes, g_from, g_to, g_weight):
    parent=[i for i in range(g_nodes+1)]
    E = [[i,j,k] for i,j,k in zip(g_from,g_to,g_weight)]
    i,e,res=0,0,0
    E.sort(key = lambda a:a[-1])
    while e<g_nodes-1:
        u,v,w = E[i]
        xr = find(parent,u)
        yr = find(parent,v)
        if xr!=yr:
        e+=1
        res+=w
        parent = union(parent,xr,yr)
        i+=1
    return res
g_nodes, g_edges = map(int, input().rstrip().split()) # scan numbers of nodes and
edges
g_from = [0] * g_edges
g_to = [0] * g_edges
g_weight = [0] * g_edges
for i in range(g_edges):
    g_from[i], g_to[i], g_weight[i] = map(int, input().rstrip().split())
res = kruskals(g_nodes, g_from, g_to, g_weight)
print(res)
```

OUTPUT

```
4 6
1 2 5
1 3 3
4 1 6
2 4 7
3 2 4
3 4 5
Result:  12
```

Source Code(#10)

```
def printJobScheduling(arr, t):
    n = len(arr)
    for i in range(n):
        for j in range(n - 1 - i):
            if arr[j][2] < arr[j + 1][2]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    result = [False] * t
    job = ['-1'] * t
    for i in range(len(arr)):
        for j in range(min(t - 1, arr[i][1] - 1), -1, -1):
            if result[j] is False:
                result[j] = True
                job[j] = arr[i][0]
                break
    print(job)


arr = [['a', 1, 3],['b', 3, 25],['c', 2, 1],['d', 1, 6],['e', 2, 30]]
print("Following is the maximum profit sequence of jobs")
printJobScheduling(arr, 3)
```

OUTPUT

```
Following is the maximum profit sequence of jobs
['d', 'e', 'b']
```

Source Code(#11)

```
def knapSack(W, wt, val, n):
    dp = [0 for i in range(W+1)]
    for i in range(1, n+1):
            for w in range(W, 0, -1):
                        if wt[i-1] <= w:
                                    dp[w] = max(dp[w], dp[w-wt[i-1]]+val[i-1])

    return dp[W]


p = [15,25,13,23]
w = [2,6,12,9]
c = 20
n = 4
print("Ans: ",knapSack(c, w, p, n))
```

OUTPUT

```
Ans: 63
```

Source Code(#12)

```
def TowerOfHanoi(n , source, destination, auxiliary):
        if n==1:
                print ("Move disk 1 from source",source,"to destination",destination)
                Return
        TowerOfHanoi(n-1, source, auxiliary, destination)
        print ("Move disk",n,"from source",source,"to destination",destination)
        TowerOfHanoi(n-1, auxiliary, destination, source)

n = 4
TowerOfHanoi(n,'A','B','C')
```

OUTPUT

```
Move disk 1 from source A to destination C
Move disk 2 from source A to destination B
Move disk 1 from source C to destination B
Move disk 3 from source A to destination C
Move disk 1 from source B to destination A
Move disk 2 from source B to destination C
Move disk 1 from source A to destination C
Move disk 4 from source A to destination B
Move disk 1 from source C to destination B
Move disk 2 from source C to destination A
Move disk 1 from source B to destination A
Move disk 3 from source C to destination B
Move disk 1 from source A to destination C
Move disk 2 from source A to destination B
Move disk 1 from source C to destination B
```

Source Code(#13)

```
from queue import Queue

q = Queue(maxsize = 3)

print(q.qsize())

q.put('a')
q.put('b')
q.put('c')
print("\nFull: ", q.full())

print("\nElements dequeued from the queue")
print(q.get())
print(q.get())
print(q.get())
print("\nEmpty: ", q.empty())

q.put(1)
print("\nEmpty: ", q.empty())
print("Full: ", q.full())
```

OUTPUT

```
0

Full:  True

Elements dequeued from the queue
a
b
c

Empty:  True

Empty:  False
Full:  False
```