In [31]:
```
#linear search
def ls(arr,k):
    for i in range(len(arr)):
        if k==arr[i]:
            print('Found at position: ',i+1)
            return 1
    return-1
arr=[56,34,25,7,1,20,6,7]
print(ls(arr,1))
```

```
Found at position:  5
1
```

In [30]:
```
#binary searh
def bis(arr,k):
    s=0
    e=len(arr)-1
    while(s<=e):
        m=(s+e)//2
        if arr[m]==k:
            print("found !!")
            return m+1
        elif k<arr[m]:
            e=m-1
        else:
            s=m+1
    return -1
arr=[56,34,25,7,1,20,6,7]
print(bis(arr,7))
```

```
found !!
4
```

In [29]:
```
#Bubble sort
def bs(arr):
    n=len(arr)
    for i in range(n):
        for j in range(n-i-1):
            if arr[j]>arr[j+1]:
                arr[j],arr[j+1]=arr[j+1],arr[j]
    return arr
arr=[56,34,25,7,1,20,6,7]
print(bs(arr))
```

```
[1, 6, 7, 7, 20, 25, 34, 56]
```

In [28]:
```
#selection sort
def ss(arr):
    n=len(arr)
    for i in range(n):
        min_i=i
        for j in range(i+1,n):
            if arr[j]<arr[min_i]:
                min_i=j
        arr[i],arr[min_i]=arr[min_i],arr[i]
    return arr
arr=[56,34,25,7,1,20,6,7]
print(ss(arr))
```

```
[1, 6, 7, 7, 20, 25, 34, 56]
```

In [33]:
```
#sequential sort
def sequential_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(i + 1, n):
            if arr[i] > arr[j]:
                arr[i], arr[j] = arr[j], arr[i]
    return arr

# Example usage
arr = [64, 25, 12, 22, 11]
print("Sequential Sort:", sequential_sort(arr))
```

```
Sequential Sort: [11, 12, 22, 25, 64]
```

In [27]:
```
#string matching
def pm(text,pattern):
    n=len(text)
    m=len(pattern)
    for i in range(n-m+1):
        j=0
        while j<m and text[i+j]==pattern[j]:
            j+=1
        if j==m:
            print('pattern matched')
            return i
    return 'pattern did not match'
t='I LOVE INDIA'
p='INDIA'
print(pm(t,p))
```

```
pattern matched
7
```

In [26]:
```
#merge sort
def Mergesort(a,l,h):
    if(l<h):
        mid=(l+h)//2
        Mergesort(a,l,mid)
        Mergesort(a,mid+1,h)
        Merge(a,l,mid,h)

def Merge(a,l,m,h):
    i=l
    j=m+1
    b=[]
    while(i<=m and j<=h):
        if (a[i]<=a[j]):
            b.append(a[i])
            i+=1
        else:
            b.append(a[j])
            j+=1
    while j<=h:
            b.append(a[j])
            j+=1
    while i<=m:
            b.append(a[i])
            i+=1
    pos=0
    for k in range(l,h+1):
            a[k]=b[pos]
            pos+=1

a=[56,34,25,6,1,20,6,7]
Mergesort(a,0,len(a)-1)
print(a)
```

```
[1, 6, 6, 7, 20, 25, 34, 56]
```

In [25]:
```
#Quick sort
def Quicksort(a,i,j):
    if(i<j):
        l=partition(a,i,j)
        Quicksort(a,i,l-1)
        Quicksort(a,l+1,j)
def partition(a,l,h):
    pivot=a[l]
    i=l+1
    j=h
    while True:
        while i<=j and (a[i])<=pivot:
            i+=1
        while i<=j and (a[j]>pivot):
            j-=1
        if (i>j):
            break
        else:
            a[i],a[j]=a[j],a[i]
    a[j],a[l]=a[l],a[j]
    return j

a=[56,34,25,6,1,20,6,7]
Quicksort(a,0,len(a)-1)
print(a)
```

```
[1, 6, 6, 7, 20, 25, 34, 56]
```

In [24]:
```
#prim's algorithm
n=int(input("enter the number of vertices: "))
g=[[0,9,75,0,0],
   [9,0,95,19,42],
   [75,95,0,51,66],
   [0,19,51,0,31],
   [0,42,66,31,0]]
selected=[0 for i in range(n)]
selected[0]=True
noe=0
print("edge\tweight")
while(noe<n-1):
    mini=999999
    x=0
    y=0
    for i in range(n):
        if selected[i]:
            for j in range(n):
                if not selected[j] and g[i][j]:
                    if mini>g[i][j]:
                        mini=g[i][j]
                        x=i
                        y=j
    print(str(x)+'-'+str(y)+':\t'+str(g[x][y]))
    selected[y]=True
    noe+=1
```

```
enter the number of vertices: 5
edge    weight
0-1:    9
1-3:    19
3-4:    31
3-2:    51
```

In [23]:
```python
#Kruskal's Algorithm
def ka(ver,edge):
    parent=list(range(ver))
    rank=[0]*ver

    def find(x):
        if parent[x]!=x:
            parent[x]=find(parent[x])
        return parent[x]

    def union(x,y):
        rootx,rooty=find(x),find(y)
        if rootx!=rooty:
            if rank[rootx]>rank[rooty]:
                parent[rooty]=rootx
            elif rank[rootx]<rank[rooty]:
                parent[rootx]=rooty
            else:
                parent[rooty]=rootx
                rank[rootx]+=1
    mst=[]
    for u,v,w in sorted(edge, key=lambda e:e[2]):
        if find(u)!=find(v):
            union(u,v)
            mst.append((u,v,w))
    return mst

ver = 9
edges = [
   (7, 6, 1),
    (8, 2, 2),
    (6, 5, 2),
    (0, 1, 4),
    (2, 5, 4),
    (8, 6, 6),
    (2, 3, 7),
    (7, 8, 7),
    (0, 7, 8),
    (1, 2, 8),
    (3, 4, 9),
    (5, 4, 10),
    (1, 7, 11),
    (3, 5, 14)
]
mst = ka(ver, edges)
for u, v, w in mst:
    print(f'{u}-{v}: {w}')
```

```
7-6: 1
8-2: 2
6-5: 2
0-1: 4
2-5: 4
2-3: 7
0-7: 8
3-4: 9
```

In [22]:
```python
#dijkstras algorithm
def dij(graph,srt):
    n=len(graph)
    dis=[float('inf')]*n
    dis[srt]=0
    visited=[False]*n

    for _ in range(n):
        min_d=float('inf')
        min_i=-1
        for i in range(n):
            if not visited[i] and dis[i]<min_d:
                min_d=dis[i]
                min_i=i
        if min_i==-1:
            break
        visited[min_i]=True

        for j in range(n):
            if not visited[j] and graph[min_i][j]>0:
                new=dis[min_i]+ graph[min_i][j]
                if dis[j]>new:
                    dis[j]=new
    return dis

g=[[0, 0, 1, 2, 0, 0, 0],
         [0, 0, 2, 0, 0, 3, 0],
         [1, 2, 0, 1, 3, 0, 0],
         [2, 0, 1, 0, 0, 0, 1],
         [0, 0, 3, 0, 0, 2, 0],
         [0, 3, 0, 0, 2, 0, 1],
         [0, 0, 0, 1, 0, 1, 0]]
srt=0
dis=dij(g,srt)
print("vertex\t distance from source")
for v,d in enumerate(dis):
  print(f"{v}\t {d}")
```

```
vertex   distance from source
0        0
1        3
2        1
3        2
4        4
5        4
6        3
```

In [21]:
```python
#BFS
graph={
    '5' : ['3','7'],
    '3' : ['2', '4'],
    '7' : ['8'],
    '2' : [],
    '4' : ['8'],
    '8' : []
}
visited=[]
queue=[]
def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)
    while queue:
        m=queue.pop(0)
        print(m, end=' -> ')
        for n in graph[m]:
            if n not in visited:
                visited.append(n)
                queue.append(n)
bfs(visited, graph, '5')
```

```
5 -> 3 -> 7 -> 2 -> 4 -> 8 ->
```

In [20]:
```
#DFS
graph={
    '5' : ['3','7'],
    '3' : ['2', '4'],
    '7' : ['8'],
    '2' : [],
    '4' : ['8'],
    '8' : []
}
visited=set()
def dfs(visited,graph,node):
    if node not in visited:
        print(node, end=' -> ')
        visited.add(node)
        for n in graph[node]:
            dfs(visited,graph,n)
dfs(visited,graph,'5')
```

```
5 -> 3 -> 2 -> 4 -> 8 -> 7 ->
```

In [34]:
```
#topological sorting
def topoSortUtil(a, adj, vis, stack):
    vis[a] = True
    for i in adj[a]:
        if not vis[i]:
            topoSortUtil(i, adj, vis, stack)
    stack.append(a)

def topoSort(adj, V):
    stack = []
    vis = [False] * V
    for i in range(V):
        if not vis[i]:
            topoSortUtil(i, adj, vis, stack)
    print('Topological sorting:', *reversed(stack))

V = 6
edges = [[5, 2], [5, 0], [4, 0], [4, 1], [2, 3], [3, 1]]
adj = [[] for _ in range(V)]
for u, v in edges:
    adj[u].append(v)
topoSort(adj, V)
```

```
Topological sorting: 5 4 2 3 1 0
```

In [35]:
```python
#horspool
def horspool(text,pattern):
    n=len(text)
    m=len(pattern)
    shift_table={pattern[i]:m-1-i for i in range(m-1)}
    shift_table[pattern[m-1]]=m
    print(shift_table)

    i=0
    while i<n-m+1:
        print('checking position at: ',i)
        j=m-1
        while j>=0 and text[i+j]==pattern[j]:
            j-=1
        if j<0:
            print("pattern found at: ")
            return i

        shift=shift_table.get(text[i+m-1],m)
        print(f'Character {text[i + m - 1]} not match,shift by position: {shift}')
        i+=shift
    print("not found")
    return -1

text='I Love India'
pattern='India'
horspool(text,pattern)
```

```
{'I': 4, 'n': 3, 'd': 2, 'i': 1, 'a': 5}
checking position at:  0
Character v not match,shift by position: 5
checking position at:  5
Character d not match,shift by position: 2
checking position at:  7
pattern found at:
```

Out[35]: 7

In [19]:

```python
#2-3 tree
class Node:
    def __init__(self, k, c=None):
        self.k = k
        self.c = c if c is not None else []

def insert(n, k):
    if not n:
        return Node([k])

    if not n.c:
        n.k.append(k)
        n.k.sort()
        if len(n.k) == 3:
            mid = n.k[1]
            return Node([mid], [Node(n.k[:1]), Node(n.k[2:])])
        return n

    i = len([x for x in n.k if k > x])
    n.c[i] = insert(n.c[i], k)

    if len(n.c[i].k) == 3:
        mid = n.c[i].k[1]
        return Node([mid], [n.c[i].c[:2], n.c[i].c[2:]])

    return n

def print_levels(r, l=0):
    print(f"Level {l}: {r.k}")
    for c in r.c:
        print_levels(c, l + 1)

tree = None
for i in [10, 20, 5, 6, 12, 30, 7, 1]:
    tree = insert(tree, i)

print_levels(tree)
```

```
Level 0: [10]
Level 1: [6]
Level 2: [1, 5]
Level 2: [7]
Level 1: [20]
Level 2: [12]
Level 2: [30]
```

In [8]:
```python
#heap sort
def heapify(arr,n,i):
    largest=i
    l=2*i+1
    r=2*i+2
    if l<n and arr[l]>arr[largest]:
        largest=l
    if r<n and arr[r]>arr[largest]:
        largest=r
    if largest!=i:
        arr[largest], arr[i]=arr[i],arr[largest]
        heapify(arr,n,largest)
def heap(arr):
    n=len(arr)
    for i in range(n//2-1,-1,-1):
        heapify(arr,n,i)
        print("heapify: ",arr)
    for i in range(n-1,0,-1):
        arr[i],arr[0]=arr[0],arr[i]
        heapify(arr,i,0)
        print("sorting: ",arr)

arr = [12,56, 11, 13, 5, 6, 7,26]
print("Original array:", arr)
heap(arr)
print("Sorted array:", arr)
```

```
Original array: [12, 56, 11, 13, 5, 6, 7, 26]
heapify:  [12, 56, 11, 26, 5, 6, 7, 13]
heapify:  [12, 56, 11, 26, 5, 6, 7, 13]
heapify:  [12, 56, 11, 26, 5, 6, 7, 13]
heapify:  [56, 26, 11, 13, 5, 6, 7, 12]
sorting:  [26, 13, 11, 12, 5, 6, 7, 56]
sorting:  [13, 12, 11, 7, 5, 6, 26, 56]
sorting:  [12, 7, 11, 6, 5, 13, 26, 56]
sorting:  [11, 7, 5, 6, 12, 13, 26, 56]
sorting:  [7, 6, 5, 11, 12, 13, 26, 56]
sorting:  [6, 5, 7, 11, 12, 13, 26, 56]
sorting:  [5, 6, 7, 11, 12, 13, 26, 56]
Sorted array: [5, 6, 7, 11, 12, 13, 26, 56]
```

In [9]:
```python
#binomial coefficient
def binomial_coefficient(n, k):
    dp = [[0] * (k + 1) for _ in range(n + 1)]

    for i in range(n + 1):
        for j in range(min(i, k) + 1):
            if j == 0 or j == i:
                dp[i][j] = 1
            else:
                dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j]
    return dp[n][k]
n = 5
k = 2
print(f"C({n}, {k}) =", binomial_coefficient(n, k))
```

```
C(5, 2) = 10
```

In [4]:
```python
#floyd's algorithm
import numpy as np
def floyds(graph):
    n=len(graph)
    dist=np.array(graph, dtype=float)
    print('initial matrix: ')
    print(dist)
    print('-------------------------------')

    for k in range(n):
        print("processing intermediate vertex: ",k+1)
        for i in range(n):
            for j in range(n):
                if dist[i][j]> dist[i][k]+dist[k][j]:
                    dist[i][j]= dist[i][k]+dist[k][j]
                    print(f"updated distance from {i+1} to {j+1}: {dist[i][j]}")
        print(dist)
        print('-------------------------------')
    print("final matrix: ")
    print(dist)

graph=[[0, 3, np.inf, 5],
     [2, 0, np.inf, 4],
     [np.inf, 1, 0, np.inf],
     [np.inf, np.inf, 2, 0]]
floyds(graph)
```

```
initial matrix:
[[ 0.  3. inf  5.]
 [ 2.  0. inf  4.]
 [inf  1.  0. inf]
 [inf inf  2.  0.]]
-------------------------------
processing intermediate vertex:  1
[[ 0.  3. inf  5.]
 [ 2.  0. inf  4.]
 [inf  1.  0. inf]
 [inf inf  2.  0.]]
-------------------------------
processing intermediate vertex:  2
updated distance from 3 to 1: 3.0
updated distance from 3 to 4: 5.0
[[ 0.  3. inf  5.]
 [ 2.  0. inf  4.]
 [ 3.  1.  0.  5.]
 [inf inf  2.  0.]]
-------------------------------
processing intermediate vertex:  3
updated distance from 4 to 1: 5.0
updated distance from 4 to 2: 3.0
[[ 0.  3. inf  5.]
 [ 2.  0. inf  4.]
 [ 3.  1.  0.  5.]
 [ 5.  3.  2.  0.]]
-------------------------------
processing intermediate vertex:  4
updated distance from 1 to 3: 7.0
updated distance from 2 to 3: 6.0
[[0. 3. 7. 5.]
 [2. 0. 6. 4.]
 [3. 1. 0. 5.]
 [5. 3. 2. 0.]]
-------------------------------
final matrix:
[[0. 3. 7. 5.]
 [2. 0. 6. 4.]
 [3. 1. 0. 5.]
 [5. 3. 2. 0.]]
```

In [1]:

```python
#TSP
def tsp(graph):
    n=len(graph)
    visited=[False]*n
    route=[0]
    cost=0
    current=0

    for _ in range(n-1):
        visited[current]=True
        print(f"current node: {current}")
        new_n,min_cost=min(((j, graph[current][j]) for j in range (n) if not visited[j]),
                           key=lambda x :x[1],default=(None, float('inf')))
        if new_n is not None:
            print(f"moving to {new_n} with mincost of {min_cost}")
            route.append(new_n)
            cost+=min_cost
            current=new_n
    cost+=graph[current][route[0]]
    route.append(route[0])
    print('returning to the start node')

    return cost, route

graph = ([
    [12, 30, 33, 10, 45],
    [56, 22, 9, 15, 18],
    [29, 13, 8, 5, 12],
    [33, 28, 16, 10, 3],
    [1, 4, 30, 24, 20]
])
min_cost, route=tsp(graph)
print(f"min cost is {min_cost}")
print(f"Best Route is: {'->'.join(map(str,route))}")
```

```
current node: 0
moving to 3 with mincost of 10
current node: 3
moving to 4 with mincost of 3
current node: 4
moving to 1 with mincost of 4
current node: 1
moving to 2 with mincost of 9
returning to the start node
min cost is 55
Best Route is: 0->3->4->1->2->0
```