Experiment 1:

Employees in an organization need to be grouped for a tournament based on their ages. Sort the ages using Merge sort and find the time required to perform the sorting.

```
import random

def gen_data(a, n):
    for i in range(n):
        x=random.randint(0,100)
        a.append(x)

def merge_sort(a,low,high):
    if high-low<1:
        return
    mid=int( (low+high)/2 )
    merge_sort(a,low,mid)
    merge_sort(a,mid+1,high)
    merge(a,low,mid,high)

def merge(a,low,mid,high):
    left=a[low:mid+1]
    right=a[mid+1:high+1]
    k=low
    i=0
    j=0
    while(i<len(left) and j<len(right)):
        if left[i]<right[j]:
            a[k]=left[i]
            i=i+1
        else:
            a[k]=right[j]
            j=j+1
        k=k+1

    if i<len(left):
        while i<len(left):
            a[k]=left[i]
            i=i+1
            k=k+1
    else:
        while j<len(right):
            a[k]=right[j]
```

```
        j=j+1
        k=k+1

#------driver code------
a=[ ]
print("enter size: ")
n=int(input())
gen_data(a,n)
print(" the numbers are :")
print (a)
merge_sort(a,0,n)
print(" the sorted numbers are:")
print (a)
```

Output-1:

```
enter size:
10
the numbers are :
[67, 88, 71, 83, 37, 97, 35, 27, 76, 26]
the sorted numbers are:
[26, 27, 35, 37, 67, 71, 76, 83, 88, 97]
```

Output-2:

```
enter size:
50
the numbers are :
[78, 63, 41, 68, 98, 38, 95, 16, 75, 36, 34, 57, 77, 97, 63, 54, 66, 92, 78, 97, 0, 72, 45, 78, 33, 14, 75,
74, 98, 97, 10, 12, 15, 0, 47, 64, 58, 59, 34, 49, 43, 67, 6, 80, 41, 6, 81, 91, 59, 36]
the sorted numbers are:
[0, 0, 6, 6, 10, 12, 14, 15, 16, 33, 34, 34, 36, 36, 38, 41, 41, 43, 45, 47, 49, 54, 57, 58, 59, 59, 63, 63,
64, 66, 67, 68, 72, 74, 75, 75, 77, 78, 78, 78, 80, 81, 91, 92, 95, 97, 97, 97, 98, 98]
```

Experiment 2:

Students in a department need to be selected for a high jump competition based on their height (integer values only). Sort the heights of students using Quick sort and find the time required for the Sorting.

```
import random
import time

def gen_data(a,n):
    for i in range(n):
        x=random.randint(0,100)
        a.append(x)

def partition(a,left,right):
    key=a[left]
    i=left+1
    j=right
    while True:
        while a[i]<key and i<right:
            i=i+1
        while a[j]>key and j>left:
            j=j-1
        if i<j:
            a[i],a[j]=a[j],a[i]
            i=i+1
            j=j-1
        else:
            break

    a[left],a[j]=a[j],a[left]
    return j

def quick_sort(a,low,high):
    if low<high:
        k=partition(a,low,high)
        quick_sort(a,low,k-1)
        quick_sort(a,k+1,high)



#----------driver code---------
a=[ ]
```

```
print("enter size: ")
n=int(input())
gen_data(a,n)
print("the numbers are:")
print (a)
start=time.time()
quick_sort(a,0,n-1)
end=time.time()
print("the sorted numbers are:")
print (a)
print("Time required to sort :",(end-start))
```

Output:

```
enter size: 10
the numbers are:
[46, 90, 95, 83, 14, 32, 31, 69, 74, 3]
the sorted numbers are:
[3, 14, 31, 32, 46, 69, 74, 83, 90, 95]
```

Experiment 3:

Print all the nodes reachable from a given starting node in a digraph using BFS method.

```
def bfs(graph,start,path):
    q=[start]
    path.append(start)
    while q:
        node=q.pop(0)
        for v in graph[node]:
            if not v in path:
                path.append(v)
                q.append(v)
#--------------------driver code--------------------
#//first write the example graph and then take input to the program from the same graph.
graph={'A':['B','C'],'B':['C'],'C':['D','E'],'D':['E'],'E':['A']}
#graph={'A':['B','C'],'B':['E','G'],'C':['F'],'D':['A','B','C','E'],'E':['F','D'],'F':[],'G':['E','F']}
#graph={1:[2,3,4],2:[6,3,1],3:[1,2,6,5,4],4:[1,3,5],5:[3,4],6:[2,3]}
#graph={40:[20,10],20:[10,30,50,60],50:[70],70:[10],10:[30],30:[60],60:[70]}
#graph={1:[2,3],2:[1],3:[1],4:[5],5:[4],6:[]}

path=[ ]
bfs(graph,'A',path)
print( path)
```

Output:

```
['A', 'B', 'C', 'D', 'E']
```

Experiment 4:

Sort a given set of elements using the Heap sort method.

```
import random

def gen_data(a,n):
    for i in range(n):
        a.append(random.randint(0,100))

def heapify(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2
    if l < n and arr[i] < arr[l]:
        largest = l

    if r < n and arr[largest] < arr[r]:
        largest = r
    if largest != i:
        arr[i],arr[largest] = arr[largest],arr[i]
        heapify(arr, n, largest)
 def heapSort(arr):
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)
 # Driver code to test above
a=[ ]
n=int(input('Enter size :'))
gen_data(a,n)
print('Given elements are:')
print (a)
heapSort(a)
n = len(a)
print ("Sorted array is")
print (a)
```

Output-1:
Enter size : 8
Given elements are:
[69, 48, 62, 49, 44, 19, 2, 91]
Sorted array is
[2, 19, 44, 48, 49, 62, 69, 91]

Output-2:
Enter size :10
Given elements are:
[84, 56, 35, 35, 12, 80, 47, 61, 2, 23]
Sorted array is
[2, 12, 23, 35, 35, 47, 56, 61, 80, 84]

Experiment 5:

Implement Horspool algorithm for String Matching.

```
def horspool(text,pattern):
    m=len(text)
    n=len(pattern)
    if m<n:
        return -1
    shift=[n]*250
    for i in range(n-1):
        shift[ord(text[i])]=n-i-1
    pos=0
    while pos<=m-n:
        j=n-1
        while j>=0 and pattern[j]==text[pos+j]:
            j=j-1
        if j==-1:
            return pos
        else:
            pos=pos+shift[ord(pattern[pos+n-1])]
    return -1
#------------Driver code--------------
text=input("Enter text string: ")
pattern=input("Enter pattern string: ")

pos=horspool(text,pattern)

if(pos>=0):
    print("Pattern string found at position:",pos+1)
else:
    print("Pattern string not found")
```

Output-1:

```
Enter text string: malnad college of engineering
Enter pattern string: college
Pattern string found at the position: 8
```

Experiment 6:

Consider n cities. The shortest path between every pair of cities needs to be determined. Implement Floyd's algorithm for the All-Pairs- Shortest-Paths problem.

```
def all_pair_sort(dist,n):
  for k in range(n):
    for i in range(n):
      for j in range(n):
        dist[i][j]=min(dist[i][j],(dist[i][k]+
                  dist[k][j]))
#-------------------driver code---------------------
#//first write the example graph and then take input to the program from the same graph.
graph=[[0,100,3,100],[2,0,100,100],[100,7,0,1],[6,100,100,0]]
n=4
all_pair_sort(graph,n)
for i in range(n):
   print (graph[i])
```

Output-1:

```
[0, 10, 3, 4]
[2, 0, 5, 6]
[7, 7, 0, 1]
[6, 16, 9, 0]
```

Experiment 7:

There are n different routes from hostel to college. Each route incurs some cost. Find the minimum cost route to reach the college from hostel using Prim's algorithm.

```
def min_edge(edge,v,vt):
    min=999
    for i in range(len(edge)):
        x=edge[i][0]
        y=edge[i][1]
        w=edge[i][2]
        if(x in v and y in vt)or(x in vt and y in v):
            if w<min:
                min=w                pos=I       return pos
def prims(edge,v):
    vt=[]
    et=[]
    vert=v.pop(0)
    vt.append(vert)
    n=len(v)
    for i in range(n):
        pos=min_edge(edge,v,vt)
        x=edge[pos][0]
        y=edge[pos][1]
        b=edge[pos]
        del edge[pos]
        et.append(b)
        if x in vt:
            vt.append(y)
            v.remove(y)
        else:
            vt.append(x)
            v.remove(x)
    return et
#------------driver code--------------
#//first write the example graph and then take input to the program from the same graph.
edge=[['a','b',5],['b','d',6],['c','d',4],['a','c',7],['a','e',2],['b','e',3],['d','e',5],['c','e',4]]
vert=['a','b','c','d','e']
st_edge=[ ]
st_edge=prims(edge,vert)
print (st_edge)
```

Output-1:

```
[['a','e',2],['b','e',3],['c','e', 4],['c','d',4]]
```

Experiment 8:

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

```
def find_set(all_set,key):
    for i in range(len(all_set)):
        if key in all_set[i]:
            return i
def kruskal(edge,v):
    edge.sort(key=lambda x:x[2])
    all_set=[ ]
    for k in vert:
        a=[]
        a.append(k)
        all_set.append(a)
    st_edge=[]
    i=0
    ct=0
    while ct<n-1:
        x=edge[i][0]
        y=edge[i][1]
        s1=find_set(all_set,x)
        s2=find_set(all_set,y)
        if s1!=s2:
            a=[]
            a.append([x,y,edge[i][2]])
            st_edge.append(a)
            all_set[s1]=all_set[s1]+all_set[s2]
            del all_set[s2]
            ct=ct+1
        i=i+1
    return st_edge
#--------driver code------
#//first write the example graph and then take input to the program from the same graph.
edge=[['a','b',5],['b','d',6],['c','d',4],['a','c',7],['a','e',2],['b','e',3],['d','e',5],['c','e',4]]
vert=['a','b','c','d','e']
n=len(vert)
st_edge=kruskal(edge,vert)
print (st_edge)
```

Output-1:

```
[[['a', 'e', 2]], [['b', 'e', 3]], [['c', 'd', 4]], [['c', 'e', 4]]]
```

Experiment 9:

Consider the distance between Hassan and n different cities. Every city can be reached from Hassan directly or by using intermediate cities whichever costs less. Find the shortest distance from Hassan to other cities using Dijkstra's algorithm.

```
def near_vertex(d,vt):
    min=999
    for i in range(len(d)):
        if i not in vt:
            if d[i]<min:
                min=d[i]
                v=i
    return v

def path(v,pv,a):
    if pv[v]!=-1:
        a.append(pv[v])
        path(pv[v],pv,a)

def print_paths(d,pv):
    for i in range(len(d)):
        print
        print("path to vertex:",i)
        a=[]
        path(i,pv,a)
        a.reverse()
        a.append(i)
        print(a)

def dijkstra(graph):
    n=len(graph)
    d=[]
    pv=[]
    vt=[]

    for i in range(n):
        d.append(999)
        pv.append(-1)
    s=0
    d[s]=0
    for i in range(n):
        v=near_vertex(d,vt)
```

```
        vt.append(v)
        adj_vertex=graph[v]
        for val in adj_vertex:
            if val[0] not in vt:
                if d[v]+val[1]<d[val[0]]:
                    d[val[0]]=d[v]+val[1]
                    pv[val[0]]=v
    print_paths(d,pv)

#---------Driver code----------------
#//first write the example graph and then take input to the program from the same graph.
#Take the input from graph
graph={0:[[3,7]],1:[[2,4]],2:[[4,6]],3:[[1,2],[2,5]],
       4:[[3,4]]}

dijkstra(graph)
```

Output-1:

```
path to vertex: 0
[0]
path to vertex: 1
[0, 3, 1]
path to vertex: 2
[0, 3, 2]
path to vertex: 3
[0, 3]
path to vertex: 4
[0, 3, 2, 4]
```

Experiment 10:

Implement N Queens problem using Back Tracking.

```
from math import *
x={ }
n=4
def place(k,i):
    if(i in x.values()):
        return False
    j=1
    while j<k:
        a=x[j]
        if abs(a-i)==abs(j-k):
            return False
        j=j+1
    return True
def clearBlocks(k):
    for i in range(k,n+1):
        x[i]=None
def Nqueens(k):
  for i in range(1,n+1):
    clearBlocks(k)
     if place(k,i):
        x[k]=i
        if k==n:
          for j in x:
            print("Place Queen at position :",x[j])
          print(".....................")
        else:
          Nqueens(k+1)
#--------------Driver code--------------
Nqueens(1)
```

Output:

```
Place Queen at position : 2
Place Queen at position : 4
Place Queen at position : 1
Place Queen at position : 3
.....................
Place Queen at position : 3
Place Queen at position : 1
Place Queen at position : 4
Place Queen at position : 2
.....................
```