ASSIGNMENT-1

```python
from collections import deque

# Function to perform Depth First Search
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end=" ")

    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)

# Function to perform Breadth First Search
def bfs(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)

    while queue:
        node = queue.popleft()
        print(node, end=" ")

        for neighbor in graph[node]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)

# Function to construct the graph based on user input
def construct_graph():
    graph = {}
    n = int(input("Enter the number of nodes: "))

    for i in range(1, n+1):
        node = input("Enter node {}: ".format(i))
        edges = input("Enter the child nodes of {}: ".format(node))
        graph[node] = edges.split()
```

```
    return graph

# Example usage:
print("Construct the graph:")
graph = construct_graph()

start_node = input("Enter the starting node: ")

print("DFS traversal:")
dfs(graph, start_node)

print("\nBFS traversal:")
bfs(graph, start_node)
```

ASSIGNMENT-5

```
QnA = {

    "Hi":"Hello sir, how may I help you ?",
    "I want to buy a watch":"Sir / Madam, which type of
watch do you want to buy ?",
    "Which types of watches are available at your store
?":"Sir / Madam, we have smart watches, formal watches,
casual watches as well as semi-casual watches",
    "I would like to buy a smart watch":"Good Choice, I
would like to ask you some questions, so that I would get
you some of the best watches",
    "Yes, sure":"Your age and gender",
    "I am 21 years old and I am an male individual":"Ok Sir,
here are some of the watches",
    "I would like to buy this watch":"Ok Sir, Thank You for
your purchase",
    "Thank You for your service too":"Welcome sir"


}

while True:
    Ques = input()

    if(Ques == "quit"):
        break

    else:
        print(QnA[Ques])
```

Assign selectionSorting

```
def selectionSort(array, size):
    for ind in range(size):
        min_index = ind

        for j in range(ind + 1, size):

            # select the minimum element in every iteration
            if array[j] < array[min_index]:
                min_index = j

        (array[ind], array[min_index]) = (array[min_index],
array[ind])

arr_size = int(input("Enter size of array: "))
arr = []

for i in range(arr_size):
    user_input = int(input("Enter the value:- "))
    arr.append(user_input)

print("\nUnsorted Array:- {}".format(arr))
size = len(arr)

selectionSort(arr, size)
print("\nSorted Array:- {}".format(arr))
```

ASSIGN 4

```
def safe(arr,x,y,n):
    #checking if queen in same column
    for row in range(x):
        if arr[row][y] == 1:

            return False

    #checking queen in left diagonal
    row = x
    col = y
    while row>=0 and col>=0:
        if arr[row][col] == 1:
            # If a queen is already present in the diagonal,
return False
            return False
        row = row-1
        col = col-1

    #checking queen in right diagonal
    row = x
    col = y
    while row>=0 and col<n:
        if arr[row][col] == 1:
            # If a queen is already present in the diagonal,
return False
            return False
        row = row-1
        col = col+1

    # if there is no conflict, it is safe to place the queen
    return True

def Nqueen(arr,x,n,count):
    #count = 0
    # If all queens are placed, increment the count of solutions
and print the board
    if x>=n:
        count[0] +=1
        print("Output", count[0], ":")
```

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

    # placing queen in each column of the current row
    for col in range(n):
        if safe(arr,x,col,n):                # Check if it is
safe to place queen at current position
            arr[x][col] = 1                  # Place queen at
current position
            Nqueen(arr,x+1,n,count)          # recursive function
            arr[x][col] = 0                  # If no solution is
found in this path, backtrack by removing the queen

    return

def main():
    n = int(input("Enter number of queens:  "))
    # Create an empty board of size n x n
    arr = [[0]* n for i in range(n)]
    count = [0]

    # Placing the queens on the board recursively
    Nqueen(arr,0,n,count)
    print("Number of possible solutions",count[0])

if __name__ == '__main__':
    main()
```