Started on	Monday, 21 April 2025, 1:25 PM
State	Finished
Completed on	Monday, 21 April 2025, 1:57 PM
Time taken	32 mins 11 secs
Grade	**100.00** out of 100.00

Question **1**

Correct

Mark 20.00 out of 20.00

Write a python program to implement knight tour problem using backtracking

For example:

Input	Result
5	Found a solution 01 20 11 14 03 10 15 02 19 12 21 24 13 04 07 16 09 06 23 18 25 22 17 08 05

Answer: (penalty regime: 0 %)

Reset answer

```
                return True
            board[x_new][y_new]=0
    return False

def is_safe(x, y):
    return 0 <= x < BOARD_SIZE and 0 <= y < BOARD_SIZE and board[x][y] == 0


def print_solution():
    for row in board:
        for col in row:
            print("0" + str(col) if col < 10 else col, end=" ")
        print()


board[0][0] = 1      # First move is at (0, 0)

if solve_knights_tour(0, 0, 2):
    print("Found a solution")
    print_solution()
else:
    print("Could not find a solution")
```

	Input	Expected	Got	
✓	5	Found a solution 01 20 11 14 03 10 15 02 19 12 21 24 13 04 07 16 09 06 23 18 25 22 17 08 05	Found a solution 01 20 11 14 03 10 15 02 19 12 21 24 13 04 07 16 09 06 23 18 25 22 17 08 05	✓

Passed all tests! ✓

Correct

Marks for this submission: 20.00/20.00.

Question **2**

Correct

Mark 20.00 out of 20.00

Write a Python program for Bad Character Heuristic of Boyer Moore String Matching Algorithm

For example:

Input	Result
ABAAAABCD ABC	`Pattern occur at shift = 5`

Answer: (penalty regime: 0 %)

Reset answer

```
    badChar = [-1] * NO_OF_CHARS
    for i in range(size):
        badChar[ord(string[i])] = i
    return badChar
def search(txt, pat):
    m = len(pat)
    n = len(txt)
    badChar = badCharHeuristic(pat, m)
    s = 0
    while(s <= n-m):
        j = m-1
        while j>=0 and pat[j] == txt[s+j]:
            j -= 1
        if j<0:
            print("Pattern occur at shift = {}".format(s))
            s += (m-badChar[ord(txt[s+m])] if s+m<n else 1)
        else:
            s += max(1, j-badChar[ord(txt[s+j])])
def main():
    txt = input()                     #"ABAAABCD"
    pat = input()                     #"ABC"
    search(txt, pat)
```

	Input	Expected	Got	
✓	ABAAAABCD ABC	`Pattern occur at shift = 5`	`Pattern occur at shift = 5`	✓

Passed all tests! ✓

Correct

Marks for this submission: 20.00/20.00.

Question **3**

Correct

Mark 20.00 out of 20.00

Write a python program to implement KMP (Knuth Morris Pratt).

For example:

Input	Result
ABABDABACDABABCABAB ABABCABAB	Found pattern at index 10

Answer: (penalty regime: 0 %)

Reset answer

```
def KMPSearch(pat, txt):
    ############## Add your code here #################
    lp=len(pat)
    ls=len(txt)
    lps=[0]*lp
    computeLPSArray(pat,lp,lps)
    i=0
    j=0

    while(i!=ls):
        if txt[i]==pat[j]:
            i+=1
            j+=1
        else:
            j=lps[j-1]
        if j==lp:
            print("Found pattern at index",i-j)
            j=lps[j-1]
        elif j==0:
            i+=1

def computeLPSArray(pat, M, lps):
```

	Input	Expected	Got	
✓	ABABDABACDABABCABAB ABABCABAB	Found pattern at index 10	Found pattern at index 10	✓
✓	SAVEETHAENGINEERING VEETHA	Found pattern at index 2	Found pattern at index 2	✓

Passed all tests! ✓

Correct

Marks for this submission: 20.00/20.00.

Question **4**

Correct

Mark 20.00 out of 20.00

Write a recursive python function to perform merge sort on the unsorted list of float values.

For example:

Test	Input	Result
mergesort(li)	5 3.2 1.5 1.6 1.7 8.9	[1.5, 1.6, 1.7, 3.2, 8.9]
mergesort(li)	6 3.1 2.3 6.5 4.5 7.8 9.2	[2.3, 3.1, 4.5, 6.5, 7.8, 9.2]

Answer: (penalty regime: 0 %)

```
    left_half = mergesort(arr[:mid])
    right_half = mergesort(arr[mid:])
    return merge(left_half, right_half)

def merge(left, right):
    merged = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
    merged.extend(left[i:])
    merged.extend(right[j:])
    return merged

n = int(input())
li = [float(input()) for _ in range(n)]
sorted_list = mergesort(li)
print(sorted_list)
```

	Test	Input	Expected	Got	
✓	mergesort(li)	5 3.2 1.5 1.6 1.7 8.9	[1.5, 1.6, 1.7, 3.2, 8.9]	[1.5, 1.6, 1.7, 3.2, 8.9]	✓

	Test	Input	Expected	Got	
✓	`mergesort(li)`	6 3.1 2.3 6.5 4.5 7.8 9.2	[2.3, 3.1, 4.5, 6.5, 7.8, 9.2]	[2.3, 3.1, 4.5, 6.5, 7.8, 9.2]	✓
✓	`mergesort(li)`	4 3.1 2.3 6.5 4.1	[2.3, 3.1, 4.1, 6.5]	[2.3, 3.1, 4.1, 6.5]	✓

Passed all tests! ✓

Correct

Marks for this submission: 20.00/20.00.

Question **5**

Correct

Mark 20.00 out of 20.00

Create a python program to implement Hamiltonian circuit problem using Backtracking.

For example:

Result
Solution Exists: Following is one Hamiltonian Cycle 0 1 2 4 3 0

Answer: (penalty regime: 0 %)

Reset answer

```
            return False
    def hamCycle(self):
        path = [-1] * self.V
        path[0] = 0

        if self.hamCycleUtil(path,1) == False:
            print ("Solution does not exist\n")
            return False

        self.printSolution(path)
        return True

    def printSolution(self, path):
        print ("Solution Exists: Following",
                 "is one Hamiltonian Cycle")
        for vertex in path:
            print (vertex, end = " ")
        print (path[0], "\n")
g1 = Graph(5)
g1.graph = [ [0, 1, 0, 1, 0], [1, 0, 1, 1, 1],
            [0, 1, 0, 0, 1,],[1, 1, 0, 0, 1],
            [0, 1, 1, 1, 0], ]
```

	Expected	Got	
✓	Solution Exists: Following is one Hamiltonian Cycle 0 1 2 4 3 0	Solution Exists: Following is one Hamiltonian Cycle 0 1 2 4 3 0	✓

Passed all tests! ✓

Correct

Marks for this submission: 20.00/20.00.