PRACTICAL NO. – 06

NAME : Krish Parothi

SECTION/BATCH : A4/B3

ROLL NO. : 49

SUBJECT : DAA

GFG PROFILE ID : Krish Parothi - Ramdeobaba University, Nagpur-440013 | GeeksforGeeks Profile

AIM : Construction of OBST

Problem Statement: Smart Library Search Optimization

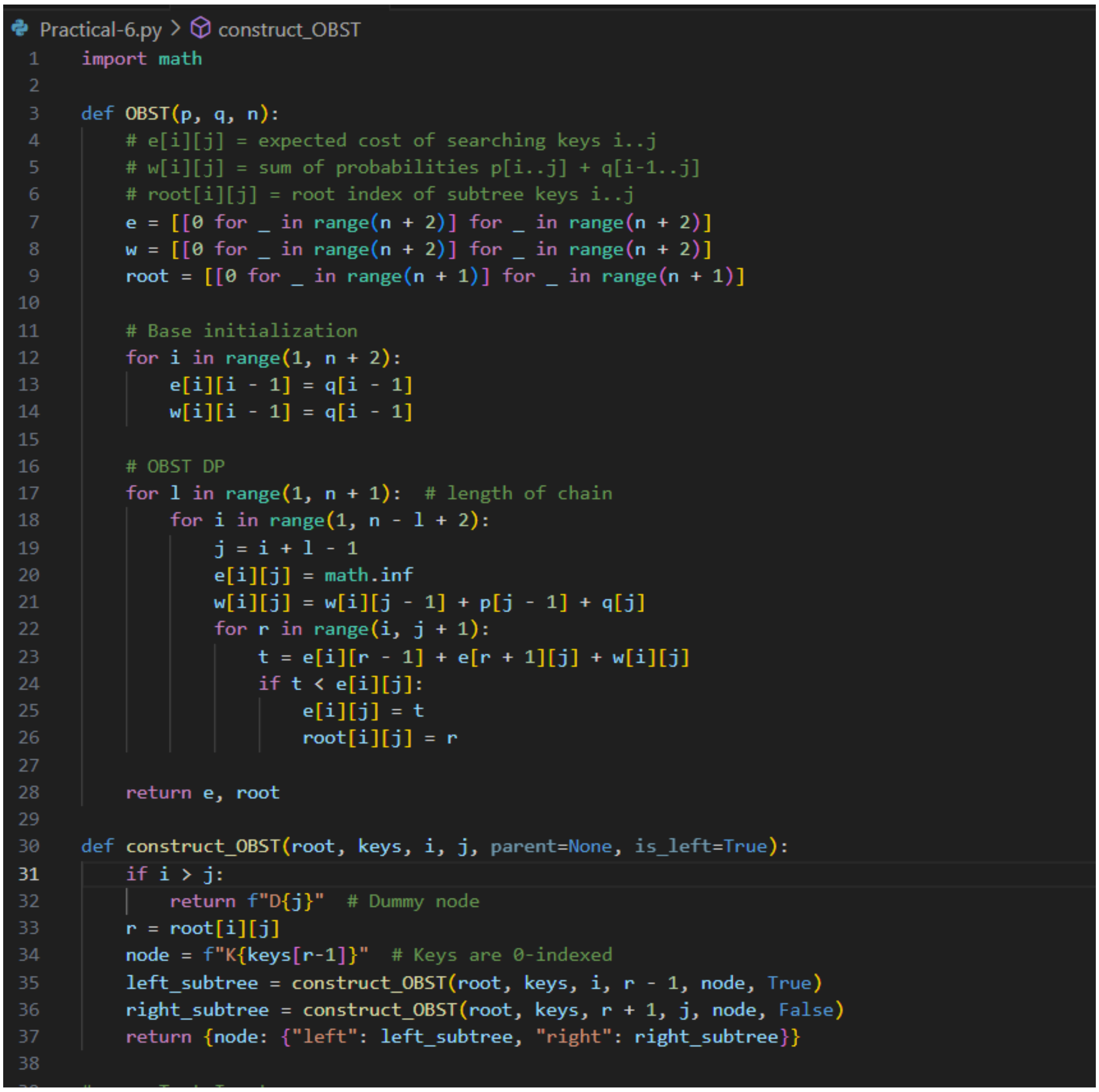
TASK : 01

Scenerio : A university digital library system stores frequently accessed books using a binary search mechanism. The library admin wants to minimize the average search time for book lookups by arranging the book IDs optimally in a binary search tree. Each book ID has a probability of being searched successfully and an associated probability for unsuccessful searches (when a book ID does not exist between two keys). Your task is to determine the minimum expected cost of searching using an Optimal Binary Search Tree (OBST).

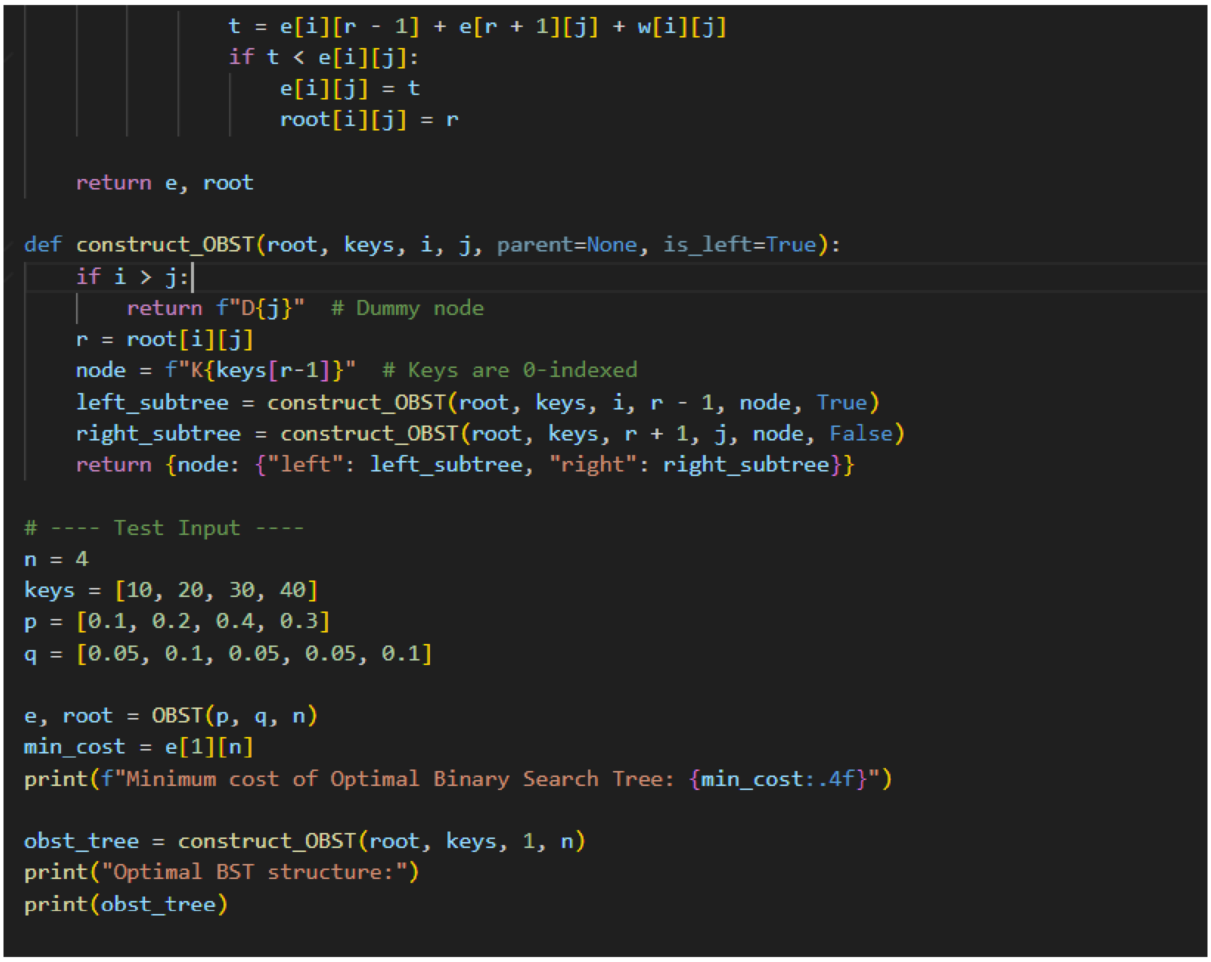
CODE : IN TEXT FORMAT

```
import math


def OBST(p, q, n):
    e = [[0 for _ in range(n + 2)] for _ in range(n + 2)]
    w = [[0 for _ in range(n + 2)] for _ in range(n + 2)]


    # Base initialization
    for i in range(1, n + 2):
        e[i][i - 1] = q[i - 1]
        w[i][i - 1] = q[i - 1]

    # OBST DP
    for l in range(1, n + 1):  # l = length of chain
        for i in range(1, n - l + 2):
            j = i + l - 1
            e[i][j] = math.inf
            w[i][j] = w[i][j - 1] + p[j - 1] + q[j]
            for r in range(i, j + 1):
                t = e[i][r - 1] + e[r + 1][j] + w[i][j]
                if t < e[i][j]:
                    e[i][j] = t

    return e[1][n]

# ---- Test Input ----
n = 4
keys = [10, 20, 30, 40]
p = [0.1, 0.2, 0.4, 0.3]
q = [0.05, 0.1, 0.05, 0.05, 0.1]

min_cost = OBST(p, q, n)
print(f"Minimum cost of Optimal Binary Search Tree: {min_cost:.4f}")
```

<u>CODE AND OUTPUT SCREENSHOT</u> :

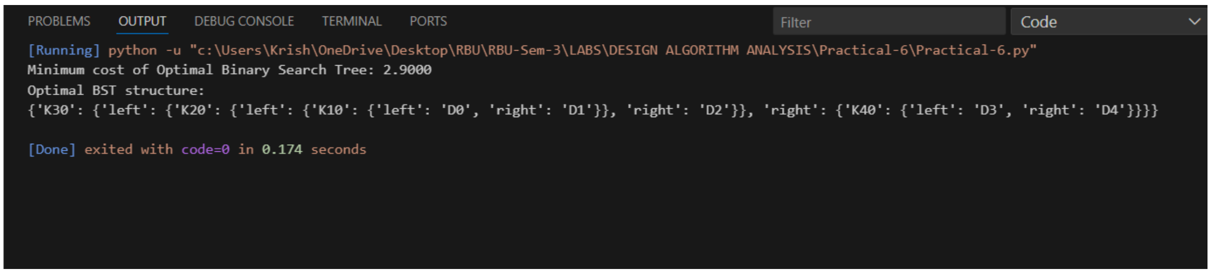


```python
import math

def OBST(p, q, n):
    # e[i][j] = expected cost of searching keys i..j
    # w[i][j] = sum of probabilities p[i..j] + q[i-1..j]
    # root[i][j] = root index of subtree keys i..j
    e = [[0 for _ in range(n + 2)] for _ in range(n + 2)]
    w = [[0 for _ in range(n + 2)] for _ in range(n + 2)]
    root = [[0 for _ in range(n + 1)] for _ in range(n + 1)]

    # Base initialization
    for i in range(1, n + 2):
        e[i][i - 1] = q[i - 1]
        w[i][i - 1] = q[i - 1]

    # OBST DP
    for l in range(1, n + 1):  # length of chain
        for i in range(1, n - l + 2):
            j = i + l - 1
            e[i][j] = math.inf
            w[i][j] = w[i][j - 1] + p[j - 1] + q[j]
            for r in range(i, j + 1):
                t = e[i][r - 1] + e[r + 1][j] + w[i][j]
                if t < e[i][j]:
                    e[i][j] = t
                    root[i][j] = r

    return e, root

def construct_OBST(root, keys, i, j, parent=None, is_left=True):
    if i > j:
        return f"D{j}"  # Dummy node
    r = root[i][j]
    node = f"K{keys[r-1]}"  # Keys are 0-indexed
    left_subtree = construct_OBST(root, keys, i, r - 1, node, True)
    right_subtree = construct_OBST(root, keys, r + 1, j, node, False)
    return {node: {"left": left_subtree, "right": right_subtree}}

```



```python
                t = e[i][r - 1] + e[r + 1][j] + w[i][j]
                if t < e[i][j]:
                    e[i][j] = t
                    root[i][j] = r

    return e, root

def construct_OBST(root, keys, i, j, parent=None, is_left=True):
    if i > j:
        return f"D{j}"  # Dummy node
    r = root[i][j]
    node = f"K{keys[r-1]}"  # Keys are 0-indexed
    left_subtree = construct_OBST(root, keys, i, r - 1, node, True)
    right_subtree = construct_OBST(root, keys, r + 1, j, node, False)
    return {node: {"left": left_subtree, "right": right_subtree}}

# ---- Test Input ----
n = 4
keys = [10, 20, 30, 40]
p = [0.1, 0.2, 0.4, 0.3]
q = [0.05, 0.1, 0.05, 0.05, 0.1]

e, root = OBST(p, q, n)
min_cost = e[1][n]
print(f"Minimum cost of Optimal Binary Search Tree: {min_cost:.4f}")

obst_tree = construct_OBST(root, keys, 1, n)
print("Optimal BST structure:")
print(obst_tree)
```

OUTPUT:

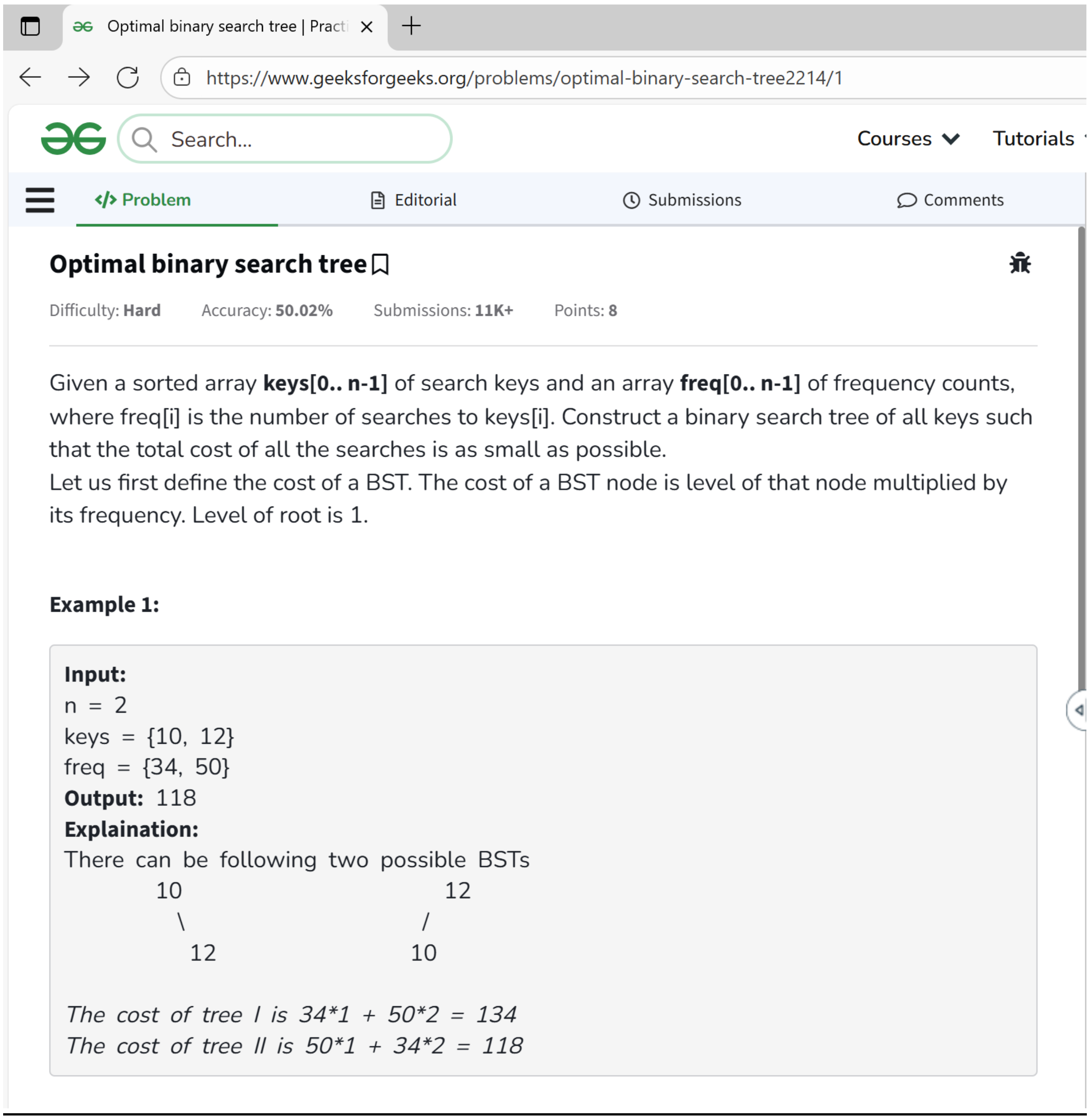


```
[Running] python -u "c:\Users\Krish\OneDrive\Desktop\RBU\RBU-Sem-3\LABS\DESIGN ALGORITHM ANALYSIS\Practical-6\Practical-6.py"
Minimum cost of Optimal Binary Search Tree: 2.9000
Optimal BST structure:
{'K30': {'left': {'K20': {'left': {'K10': {'left': 'D0', 'right': 'D1'}}, 'right': 'D2'}}, 'right': {'K40': {'left': 'D3', 'right': 'D4'}}}}

[Done] exited with code=0 in 0.174 seconds
```

<u>TASK : 02</u>

<u>GFG QUESTION ID</u> : https://www.geeksforgeeks.org/problems/optimal-binary-search-tree2214/1

<u>QUESTION SCREENSHOT</u>


Optimal binary search tree | Pract
https://www.geeksforgeeks.org/problems/optimal-binary-search-tree2214/1
Search...
Courses
Tutorials
Problem
Editorial
Submissions
Comments

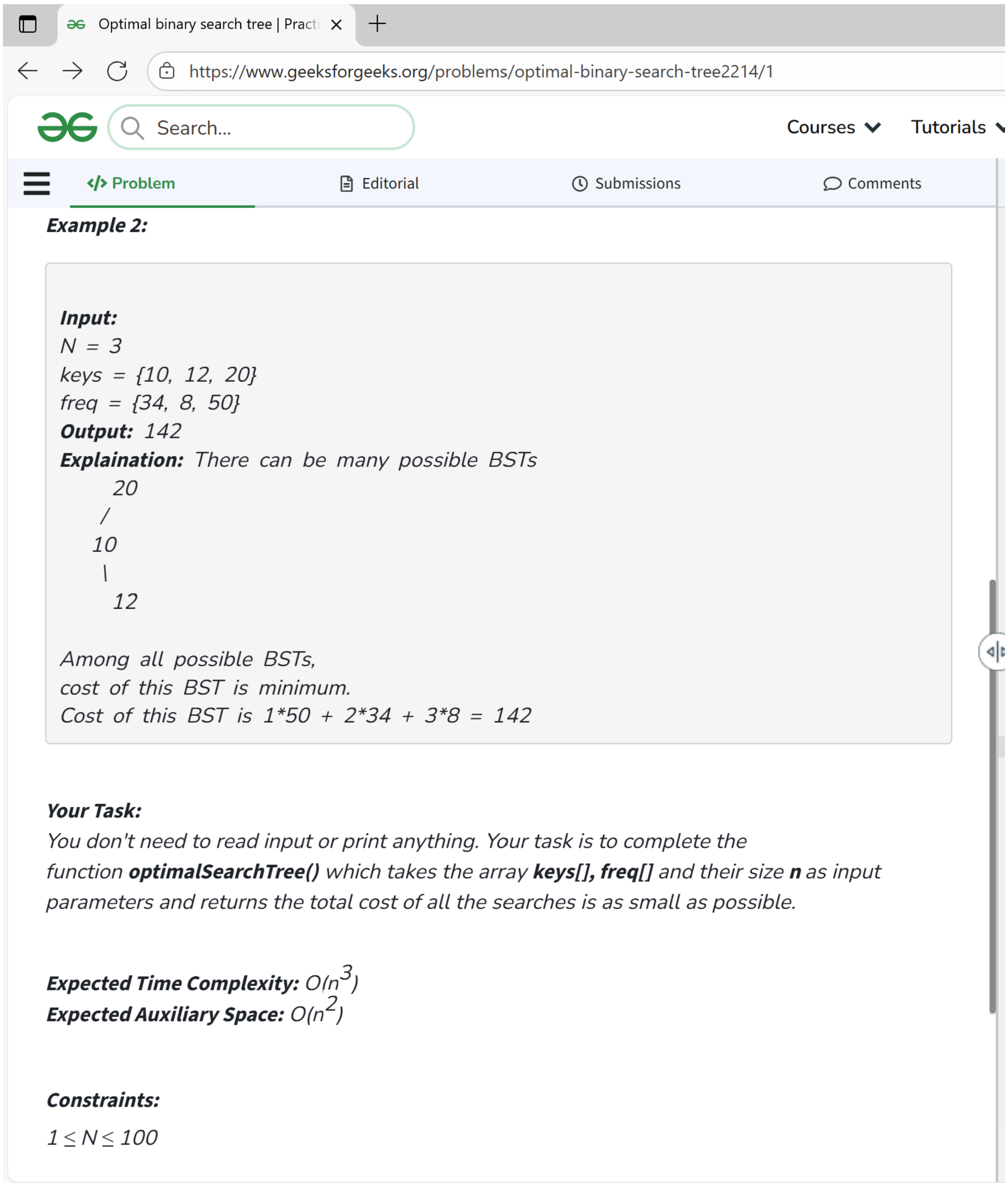
Optimal binary search tree

Difficulty: **Hard** Accuracy: **50.02%** Submissions: **11K+** Points: **8**

Given a sorted array **keys[0.. n-1]** of search keys and an array **freq[0.. n-1]** of frequency counts, where freq[i] is the number of searches to keys[i]. Construct a binary search tree of all keys such that the total cost of all the searches is as small as possible.
Let us first define the cost of a BST. The cost of a BST node is level of that node multiplied by its frequency. Level of root is 1.

Example 1:

```
Input:
n = 2
keys = {10, 12}
freq = {34, 50}
Output: 118
Explaination:
There can be following two possible BSTs
         10                       12
           \                     /
             12                10

The cost of tree I is 34*1 + 50*2 = 134
The cost of tree II is 50*1 + 34*2 = 118
```


Optimal binary search tree | Practi
https://www.geeksforgeeks.org/problems/optimal-binary-search-tree2214/1
Search...
Courses
Tutorials
Problem
Editorial
Submissions
Comments

Example 2:

```
Input:
N = 3
keys = {10, 12, 20}
freq = {34, 8, 50}
Output: 142
Explaination: There can be many possible BSTs
     20
    /
   10
    \
     12

Among all possible BSTs,
cost of this BST is minimum.
Cost of this BST is 1*50 + 2*34 + 3*8 = 142
```

Your Task:
*You don't need to read input or print anything. Your task is to complete the function **optimalSearchTree()** which takes the array **keys[], freq[]** and their size **n** as input parameters and returns the total cost of all the searches is as small as possible.*

Expected Time Complexity: $O(n^3)$
Expected Auxiliary Space: $O(n^2)$

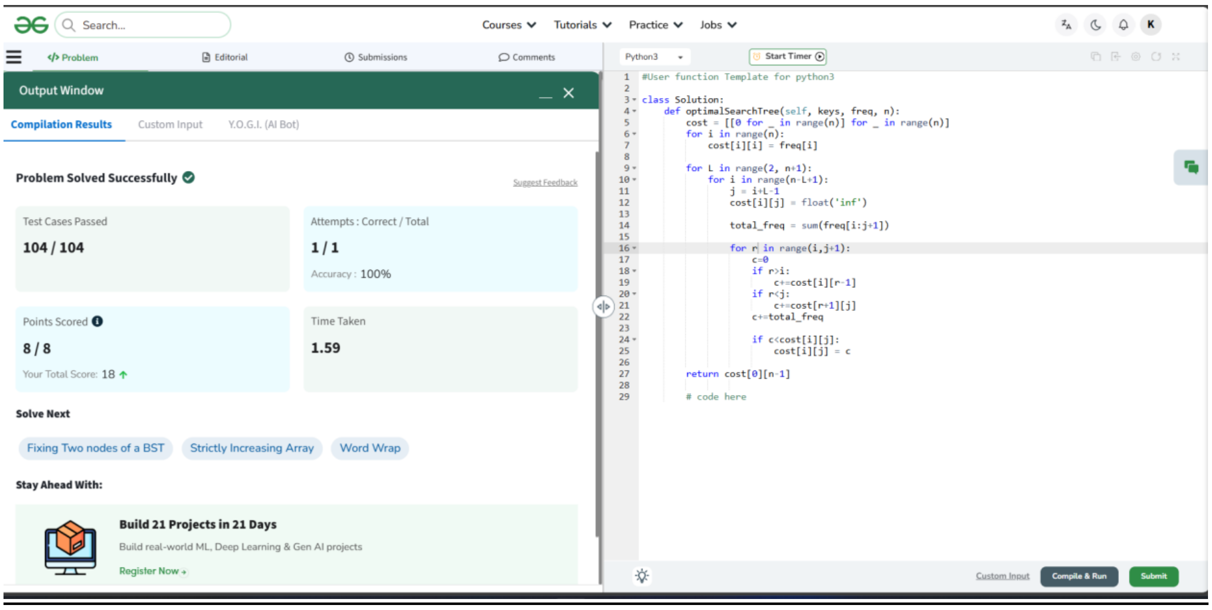
Constraints:
$1 \le N \le 100$

CODE : IN TEXT FORMAT

```
class Solution:
    def optimalSearchTree(self, keys, freq, n):
        cost = [[0 for _ in range(n)] for _ in range(n)]


        for i in range(n):
```

```
    cost[i][i] = freq[i]

for L in range(2, n + 1):
    for i in range(n - L + 1):
        j = i + L - 1
        cost[i][j] = float('inf')

        total_freq = sum(freq[i:j + 1])

        for r in range(i, j + 1):
            c = 0
            if r > i:
                c += cost[i][r - 1]
            if r < j:
                c += cost[r + 1][j]
            c += total_freq

            if c < cost[i][j]:
                cost[i][j] = c

return cost[0][n - 1]
```

SUBMISSION SCREENSHOT



Problem Solved Successfully

Test Cases Passed
104 / 104

Attempts : Correct / Total
1 / 1
Accuracy : 100%

Points Scored
8 / 8
Your Total Score: 18

Time Taken
1.59

Solve Next

Fixing Two nodes of a BST | Strictly Increasing Array | Word Wrap

Stay Ahead With:

Build 21 Projects in 21 Days
Build real-world ML, Deep Learning & Gen AI projects
Register Now

```python
#User function Template for python3

class Solution:
    def optimalSearchTree(self, keys, freq, n):
        cost = [[0 for _ in range(n)] for _ in range(n)]
        for i in range(n):
            cost[i][i] = freq[i]

        for L in range(2, n+1):
            for i in range(n-L+1):
                j = i+L-1
                cost[i][j] = float('inf')

                total_freq = sum(freq[i:j+1])

                for r in range(i,j+1):
                    c=0
                    if r>i:
                        c+=cost[i][r-1]
                    if r<j:
                        c+=cost[r+1][j]
                    c+=total_freq

                    if c<cost[i][j]:
                        cost[i][j] = c

        return cost[0][n-1]

        # code here
```