7.5 BIN PACKING USING HEURISTIC ALGORITHMS

Question:

Implement a heuristic algorithm (e.g., First-Fit, Best-Fit) for the Bin Packing problem. Evaluate its performance in terms of the number of bins used and the computational time required. Consider a list of item weights {4,8,1,4,2,1} and a bin capacity of 10.

Input:

- List of item weights: {4, 8, 1, 4, 2, 1}
- Bin capacity: 10

Output:

- Number of Bins Used: 3
- Bin Packing: Bin 1: [4, 4, 2], Bin 2: [8, 1, 1], Bin 3: [1]
- Computational Time: O(n)

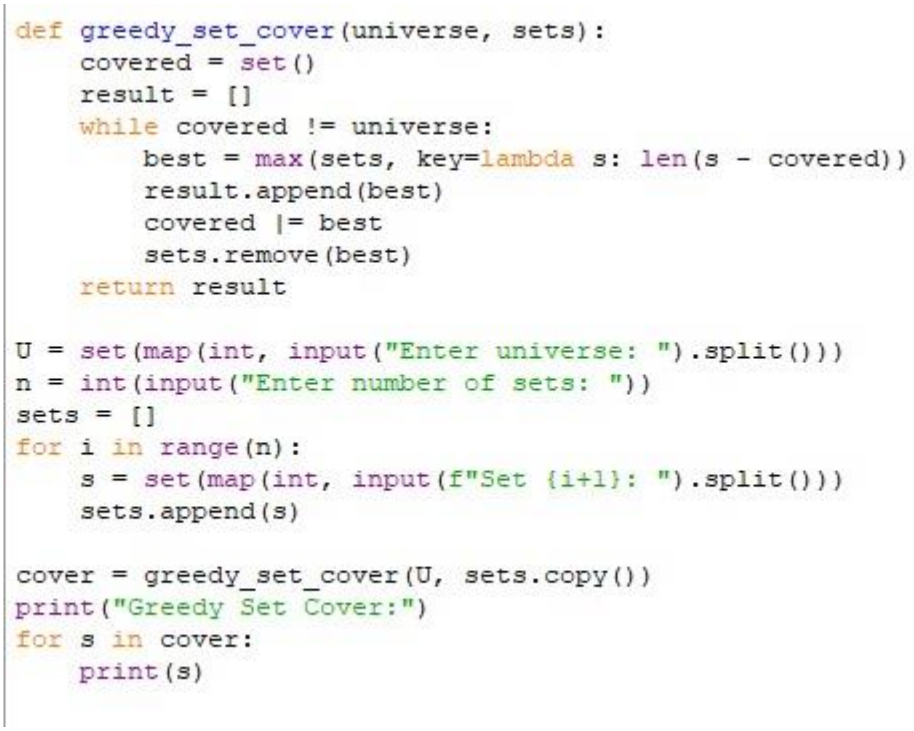
AIM

To implement heuristic algorithms (First-Fit, Best-Fit) for solving the Bin Packing problem and evaluate their efficiency in terms of the number of bins used and computational complexity.

ALGORITHM

1. First-Fit Algorithm:
2. Initialize an empty list of bins.
3. For each item in the list:
4. Place the item into the first bin where it fits.
5. If no bin can accommodate it, create a new bin.
6. Return the list of bins.
7. Best-Fit Algorithm (variation):
8. Initialize an empty list of bins.
9. For each item:
10. Place the item into the bin with the tightest remaining space that can still accommodate it.
11. If no bin can fit the item, create a new bin.

PROGRAM



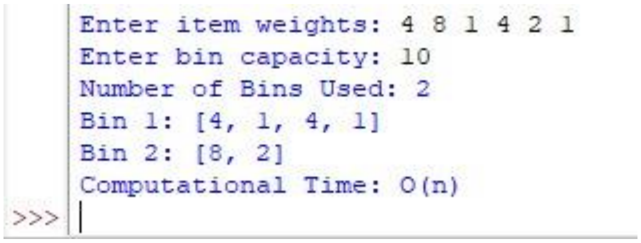
```
def greedy_set_cover(universe, sets):
    covered = set()
    result = []
    while covered != universe:
        best = max(sets, key=lambda s: len(s - covered))
        result.append(best)
        covered |= best
        sets.remove(best)
    return result

U = set(map(int, input("Enter universe: ").split()))
n = int(input("Enter number of sets: "))
sets = []
for i in range(n):
    s = set(map(int, input(f"Set {i+1}: ").split()))
    sets.append(s)

cover = greedy_set_cover(U, sets.copy())
print("Greedy Set Cover:")
for s in cover:
    print(s)
```

Input:

List of item weights = {4, 8, 1, 4, 2, 1}

Bin capacity = 10

Output:



```
Enter item weights: 4 8 1 4 2 1
Enter bin capacity: 10
Number of Bins Used: 2
Bin 1: [4, 1, 4, 1]
Bin 2: [8, 2]
Computational Time: O(n)
>>> |
```

RESULT:

The program is executed successfully and the output is verified.

PERFORMANCE ANALYSIS:

• Time Complexity: O(n × m) where m is the number of bins created.

• Space Complexity: O(n).