Experiment Title: Implementation of Programs on Greedy method-III.

Aim/Objective: To understand the concept and implementation of Basic programs on Greedy method - Scenario 3.

Description: The students will understand and able to implement programs on Greedy method - Scenario 3.

Pre-Requisites:

Knowledge: Greedy method - Scenario 3 in C/C++/PythonTools: Code Blocks/Eclipse IDE

Pre-Lab:

Given a set of jobs where each job has a deadline and a profit, schedule the jobs to maximize the total profit.

Input: A list of jobs with their profits and deadlines.

Output: The sequence of jobs and the maximum profit.

Example:

Input:

Jobs = [(5, 2), (4, 1), (6, 2), (3, 1)], where (profit, deadline)

Output: Job sequence: [1, 3]

Maximum profit: 9

- **Procedure/Program:**

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int profit;
    int deadline;
    int job_id;
} Job;

int compare(const void *a, const void *b) {
    return ((Job *)b)->profit - ((Job *)a)->profit;
}
```

```
void jobScheduling(Job jobs[], int n) {
    qsort(jobs, n, sizeof(Job), compare);

    int max_deadline = 0;
    for (int i = 0; i < n; i++) {
        if (jobs[i].deadline > max_deadline) {
            max_deadline = jobs[i].deadline;
        }
    }

    int slots[max_deadline];
    for (int i = 0; i < max_deadline; i++) {
        slots[i] = -1;
    }

    int total_profit = 0;
    int job_count = 0;

    int job_sequence[n];

    for (int i = 0; i < n; i++) {
        for (int t = jobs[i].deadline - 1; t >= 0; t--) {
            if (slots[t] == -1) {
                slots[t] = i;
                total_profit += jobs[i].profit;
                job_sequence[job_count++] = jobs[i].job_id;
                break;
            }
        }
    }

    printf("Job sequence: ");
    for (int i = 0; i < job_count; i++) {
        printf("%d ", job_sequence[i]);
    }
    printf("\nMaximum profit: %d\n", total_profit);
}
```

```
int main() {
    Job jobs[] = {{5, 2, 1}, {4, 1, 2}, {6, 2, 3}, {3, 1, 4}};
    int n = sizeof(jobs) / sizeof(jobs[0]);

    jobScheduling(jobs, n);

    return 0;
}
```

- **Data and Result:**

Data

Job list: profits, deadlines, and job identifiers for scheduling optimization.

Result

Job sequence: [1, 3], maximum profit: 9, schedule completed.

- **Inference Analysis:**

Analysis

Sorting jobs by profit and fitting them within deadlines maximizes profit.

Inferences

Profit maximization relies on efficient job scheduling and deadline management.

In-Lab:

Given an array of integers, partition the array into two subsets such that the sum of the elements in both subsets is as equal as possible.

Input: An array of integers.

Output: The two subsets with the closest possible sums.

Example:

Input: [1, 5, 11, 5]

Output: Subsets: [11, 5], [5, 1]

- **Procedure/Program:**

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

void findPartition(int arr[], int n) {
   int total_sum = 0;
   for (int i = 0; i < n; i++) {
      total_sum += arr[i];
   }

   int target = total_sum / 2;

   bool dp[target + 1];
   for (int i = 0; i <= target; i++) {
      dp[i] = false;
   }
   dp[0] = true;

   for (int i = 0; i < n; i++) {
      for (int j = target; j >= arr[i]; j--) {
         dp[j] = dp[j] || dp[j - arr[i]];
      }
   }

   int subset_sum = 0;
   for (int i = target; i >= 0; i--) {
```

```
        if (dp[i]) {
            subset_sum = i;
            break;
        }
    }

    printf("Subset 1: ");
    int sum = subset_sum;
    for (int i = n - 1; i >= 0; i--) {
        if (sum >= arr[i] && dp[sum - arr[i]]) {
            printf("%d ", arr[i]);
            sum -= arr[i];
        }
    }

    printf("\nSubset 2: ");
    bool used[n];
    for (int i = 0; i < n; i++) {
        used[i] = false;
    }

    sum = subset_sum;
    for (int i = n - 1; i >= 0; i--) {
        if (sum >= arr[i] && dp[sum - arr[i]] && !used[i]) {
            used[i] = true;
            sum -= arr[i];
        }
    }

    for (int i = 0; i < n; i++) {
        if (!used[i]) {
            printf("%d ", arr[i]);
        }
    }

    printf("\n");
}
int main() {
```

```
    int arr[] = {1, 5, 11, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    findPartition(arr, n);

    return 0;
}
```

- **Data and Results:**

Data:

Input: Array: [1, 5, 11, 5]

Result:

Subset 1: [11, 5], Subset 2: [5, 1]

- **Analysis and Inferences:**

Analysis:

Subset sums are as close as possible, minimizing the difference.

Inferences:

Partitioning array reduces the difference between subset sums effectively.

Post-Lab:

A store offers a loyalty program where customers accumulate points for each purchase. Customers can redeem their points for discounts on future purchases. Implement a greedy algorithm to maximize the total discount for a customer based on the points they have accumulated.

Input: A list of purchases and their associated point values.

Output: The total discount the customer can get.

Example:

Input:

Purchases = [10, 20, 30], Points per purchase = [5, 8, 12]

Output: Maximum discount the customer can get.

- **Procedure/Program:**

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int purchase;
    int points;
} Purchase;

int compare(const void *a, const void *b) {
    return ((Purchase*)b)->points - ((Purchase*)a)->points;
}

int maximize_discount(int purchases[], int points_per_purchase[], int n) {
    Purchase purchase_info[n];
    for (int i = 0; i < n; i++) {
        purchase_info[i].purchase = purchases[i];
        purchase_info[i].points = points_per_purchase[i];
    }

    qsort(purchase_info, n, sizeof(Purchase), compare);
```

```
 int total_discount = 0;
   for (int i = 0; i < n; i++) {
      total_discount += purchase_info[i].points;
   }

   return total_discount;
}

int main() {
   int purchases[] = {10, 20, 30};
   int points_per_purchase[] = {5, 8, 12};
   int n = sizeof(purchases) / sizeof(purchases[0]);

   int discount = maximize_discount(purchases, points_per_purchase, n);
   printf("Maximum discount the customer can get: %d\n", discount);

   return 0;
}
```

- **Data and Results:**

Data:

- Purchases: [10, 20, 30]
- Points per purchase: [5, 8, 12]

Result:

- Maximum discount the customer can get: 25 points.

- **Analysis and Inferences:**

Analysis:

- Sorted purchases by points: [12, 8, 5].
- Summing points yields total discount of 25 points.

Inferences:

- Larger point values contribute more to the total discount.

- **Sample VIVA-VOCE Questions:**

1. What is the Huffman coding algorithm, and how does it use Greedy techniques to compress data?

- A compression algorithm using a binary tree.
- Greedy selects the least frequent characters first.

2. Can you think of any situations where a Greedy algorithm might fail to produce an optimal solution?

- Fails when local decisions don't lead to global optimum (e.g., 0/1 Knapsack).

3. What is the time complexity of Huffman coding algorithm?

- O(n log n).

4. Find out the time complexity of BFS and DFS algorithm?

- $O(V + E)$, where `V` is vertices and `E` is edges.

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.