Date of the Session:__/__/_____ **Time of the Session:_____to_____**

EX – 10 Sum of Subsets Problem using Back tracking

Prerequisites:

- Basics of Data Structures and C Programming.
- Basic knowledge about arrays.

Pre-Lab:

1) Dark Peace is volunteer in VISION 2K17.ChiefCO gave him a task to complete it before the CODEGEEKS which is on 11th march. ChiefCo asks DarkPeace to do the work as soon as possible. So, to complete the task as soon as possible Dark Peace asks you to help him. You will be given two set of length N and M. Your task is to find whether the subset is present in the first set whose sum of elements is equal to the member of the set. You must check for every member of the second set. Print Yes if subset is present and if not present print No and if the member exceeds the maximum sum of

Input

3 3

1 7 4

10 14 4

Output

No -1 Yes

Explanation

For first member no subset gives sum to 10 so we print No. For second since maximum sum is 12 and 14 is greater than 12 so we print -1. For last subset {4} exits whose sum is equal to 4 and hence we print Yes.

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

bool subset_sum_possible(int arr[], int n, int target_sum) {
  if (target_sum == 0) return true;

  bool dp[target_sum + 1];
  dp[0] = true;

  int i, j;
  for (i = 1; i <= target_sum; i++) {
```

```
        dp[i] = false;
    }

    for (i = 0; i < n; i++) {
        for (j = target_sum; j >= arr[i]; j--) {
            if (dp[j - arr[i]]) {
                dp[j] = true;
            }
        }
    }

    return dp[target_sum];
}

int main() {
    int first_set[] = {1, 7, 4};
    int second_set[] = {10, 14, 4};
    int N = sizeof(first_set) / sizeof(first_set[0]);
    int M = sizeof(second_set) / sizeof(second_set[0]);

    int i, target;

    int max_sum = 0;
    for (i = 0; i < N; i++) {
        max_sum += first_set[i];
    }

    for (i = 0; i < M; i++) {
        target = second_set[i];

        if (target > max_sum) {
            printf("-1 ");
        } else {
```

```
            if (subset_sum_possible(first_set, N, target)) {
                printf("Yes ");
            } else {
                printf("No ");
            }
        }
    }

    return 0;
}
```

In-Lab:

1) Geeta is working in a company, and she has n different projects to work on, where every project is scheduled to be done from startTime[i] to endTime[i], obtaining a profit of profit[i]. You are given the startTime, endTime and profit arrays, return the maximum profit you can take such that there are no two projects that she is working on in that given subset with overlapping time range. If she chooses a project that ends at time a then she will be able to start another project that starts at time b.

Input
startTime = [1,2,3,4,6], endTime = [3,5,10,6,9], profit = [20,20,100,70,60]
Output
150
Explanation: The subset chosen is the first, fourth and fifth project.
Profit obtained 150 = 20 + 70 + 60.

Source code:

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

typedef struct {
   int start, end, profit;
} Project;

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

int findLastNonOverlapping(Project projects[], int i) {
   int low = 0, high = i - 1;
   while (low <= high) {
      int mid = low + (high - low) / 2;
      if (projects[mid].end <= projects[i].start) {
         if (projects[mid + 1].end <= projects[i].start) {
            low = mid + 1;
         } else {
            return mid;
         }
      } else {
         high = mid - 1;
      }
   }
   return -1;
}
```

```
int findMaxProfit(int startTime[], int endTime[], int profit[], int n) {
   Project projects[n];
   for (int i = 0; i < n; i++) {
      projects[i].start = startTime[i];
      projects[i].end = endTime[i];
      projects[i].profit = profit[i];
   }
   qsort(projects, n, sizeof(Project), compare);

   int dp[n];
   dp[0] = projects[0].profit;

   for (int i = 1; i < n; i++) {
      dp[i] = dp[i - 1];
      int lastNonOverlap = findLastNonOverlapping(projects, i);
      if (lastNonOverlap != -1) {
         dp[i] = (dp[i] > projects[i].profit + dp[lastNonOverlap]) ? dp[i] : (projects[i].profit +
dp[lastNonOverlap]);
      } else {
         dp[i] = (dp[i] > projects[i].profit) ? dp[i] : projects[i].profit;
      }
   }

   return dp[n - 1];
}

int main() {
   int startTime[] = {1, 2, 3, 4, 6};
   int endTime[] = {3, 5, 10, 6, 9};
   int profit[] = {20, 20, 100, 70, 60};
   int n = sizeof(startTime) / sizeof(startTime[0]);

   printf("Maximum Profit: %d\n", findMaxProfit(startTime, endTime, profit, n));

   return 0;
}
```

2) The subset sum problem is an important problem in computer science. Below we will provide a simple algorithm for solving this problem. The challenge is to determine if there is some subset of numbers in an array that can sum up to some number S. These algorithms can both be implemented to solve Array Addition I and Array Addition.

Simple (Naive) solution
The algorithm for the exponential time, naive solution, is as follows: First we will generate every possible set (the power set), and then check if the sum of any of these sets equals the sum S. For example: arr = [1, 2, 3] sum = 5
All possible sets:[] [1] [2] [3] [1, 2] [1, 3] [2, 3] [1, 2, 3]
We can see that we can get a sum of 5 by adding the elements in the set [2, 3]. To generate all possible sets of an array, we will implement the following algorithm:
(1) The initial power set only contains the empty set: [[]]
(2) We loop through each element in the array and add it to every element in the powerset.
(3) Then we take the union of these two sets.
(4) Once we get the power set, we check to see if the sum equals our goal S.
Example
arr = [1, 2, 3] sum = 5 sets = [[]]
Step 1: Add 1 to the power set [[], [1]]
Step 2: Add 2 to the power set [[], [1], [1, 2], [2]]
Step 3: Add 3 to the power set [[], [1], [1, 2], [2], [1, 3], [2, 3], [1, 2, 3], [3]]

Source code:

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

void findSubsets(int arr[], int n, int sum) {
   int totalSubsets = 1 << n;
   for (int i = 0; i < totalSubsets; i++) {
      int currentSum = 0;
      printf("[");
      for (int j = 0; j < n; j++) {
         if (i & (1 << j)) {
            currentSum += arr[j];
            printf("%d ", arr[j]);
         }
      }
      printf("]\n");
      if (currentSum == sum) {
         printf("Found a subset with the required sum: %d\n", sum);
      }
   }
}
```

```
int main() {
    int arr[] = {1, 2, 3};
    int sum = 5;
    int n = sizeof(arr) / sizeof(arr[0]);
    findSubsets(arr, n, sum);
    return 0;
```

[@KLWKS_BOT] THANOS

Post-Lab:

1) Karthik was given a problem in an online interview, but he cannot solve the solution help him solve the question. There are n students whose ids vary between 0 to 9 digits; two students can have same id's. You will be given x numbers which also vary from 0 to 9. You need to find the total number of student id's subsets which contains all the x numbers.

Input
333
1
3
Output
6

Source code:

```
#include <stdio.h>
int count_subsets(int student_ids[], int n, int x_numbers[], int m) {
   int count[10] = {0};

   for (int i = 0; i < n; i++) {
      count[student_ids[i]]++;
   }

   for (int i = 0; i < m; i++) {
      if (count[x_numbers[i]] == 0) {
         return 0;
      }
   }
   int total_subsets = 1;
   for (int i = 0; i < m; i++) {
      total_subsets *= count[x_numbers[i]];
   }
   return total_subsets;
}
int main() {
   int student_ids[] = {3, 3, 3};
   int x_numbers[] = {1, 3};
   int n = sizeof(student_ids) / sizeof(student_ids[0]);
   int m = sizeof(x_numbers) / sizeof(x_numbers[0]);

   int result = count_subsets(student_ids, n, x_numbers, m);

   printf("%d\n", result);
   return 0;
}
```

2) You are offered N pay packages by various companies, Congratulations! You can only select 2 unique groups of packages each with a total of at least K. Select the 2 groups such that you choose minimum number of companies adding up to the total of K.

Input

arr[] = {2, 4, 5, 6, 7, 8}, K = 16

Output

6

Explanation:

The subsets {2, 6, 8} and {4, 5, 7} are the two smallest subsets with sum K (= 16).

Therefore, the sum of the lengths of both these subsets = 3 + 3 = 6.4

Source code:

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

#define MAX 100

int minCompanies(int arr[], int n, int K) {
   int dp[MAX] = {0};
   for (int i = 1; i <= K; i++) {
      dp[i] = MAX;
   }
   dp[0] = 0;

   for (int i = 0; i < n; i++) {
      for (int j = K; j >= arr[i]; j--) {
         if (dp[j - arr[i]] + 1 < dp[j]) {
            dp[j] = dp[j - arr[i]] + 1;
         }
      }
   }

   return dp[K];
}

int main() {
   int arr[] = {2, 4, 5, 6, 7, 8};
   int K = 16;
   int minCount = minCompanies(arr, sizeof(arr) / sizeof(arr[0]), K);

   printf("%d\n", minCount * 2);
   return 0;
}
```

Comments of the Evaluators (if Any)	Evaluator's Observation Marks Secured:_____out of [50]. Signature of the Evaluator Date of Evaluation: