Experiment **#16**		Student ID	
Date		Student Name	[@KLWKS_BOT THANOS]

Experiment Title: Implementation of Programs on Non-Deterministic Algorithms - II.

Aim/Objective: To understand the concept and implementation of Basic programs on Non-Deterministic Algorithms.

Description:

The students will understand and able to implement programs on Non-Deterministic Algorithms.

Pre-Requisites:

Knowledge: Non-Deterministic Algorithms in C/C++/PythonTools: Code Blocks/Eclipse IDE

Pre-Lab:

Given a set of integers and a target sum S, determine whether there exists a subset whose sum equals S.

Input: An array of integers and a target sum S.

Output: Yes/No and the subset if it exists.

- **Procedure/Program:**

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

bool isSubsetSum(int arr[], int n, int sum, int subset[]) {
  bool dp[n+1][sum+1];

  for (int i = 0; i <= n; i++) {
    dp[i][0] = true;
  }
   for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= sum; j++) {
      if (arr[i-1] <= j) {
        dp[i][j] = dp[i-1][j] || dp[i-1][j-arr[i-1]];
      } else {
        dp[i][j] = dp[i-1][j];
      }
    }
  }

  if (!dp[n][sum]) {
```

```
        return false;
    }
  int index = 0;
    int i = n, j = sum;
    while (i > 0 && j > 0) {
        if (dp[i][j] != dp[i-1][j]) {
            subset[index++] = arr[i-1];
            j -= arr[i-1];
        }
        i--;
    }

    return true;
}
int main() {
    int n, sum;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements of the array: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the target sum: ");
    scanf("%d", &sum);
     int subset[n];
    if (isSubsetSum(arr, n, sum, subset)) {
        printf("Yes\nSubset: ");
        for (int i = 0; subset[i] != 0; i++) {
            printf("%d ", subset[i]);
        }
        printf("\n");
    } else {
        printf("No\n");
    }

    return 0;
}
```

In-Lab:

Given a Boolean formula in Conjunctive Normal Form (CNF), determine whether it is satisfiable (i.e., if there exists an assignment of variables such that the formula evaluates to true).

Input: A Boolean formula in CNF.

Output: Satisfiable (Yes/No) and the satisfying assignment if it exists.

- **Procedure/Program:**

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

#define MAX_VARS 20
#define MAX_CLAUSES 50

int variables[MAX_VARS];
int num_vars, num_clauses;
int clauses[MAX_CLAUSES][MAX_VARS];
int clause_sizes[MAX_CLAUSES];

bool is_satisfied() {
   for (int i = 0; i < num_clauses; i++) {
      bool clause_satisfied = false;
      for (int j = 0; j < clause_sizes[i]; j++) {
         int literal = clauses[i][j];
         int var = abs(literal) - 1;
         if ((literal > 0 && variables[var] == 1) || (literal < 0 && variables[var] == 0)) {
            clause_satisfied = true;
            break;
         }
      }
      if (!clause_satisfied) {
         return false;
      }
   }
   return true;
}

bool solve(int var_index) {
   if (var_index == num_vars) {
      return is_satisfied();
   }
   variables[var_index] = 1;
   if (solve(var_index + 1)) return true;
   variables[var_index] = 0;
   if (solve(var_index + 1)) return true;
```

```
    return false;
}

int main() {
    printf("Enter the number of variables: ");
    scanf("%d", &num_vars);

    printf("Enter the number of clauses: ");
    scanf("%d", &num_clauses);

    printf("Enter the clauses (literals separated by spaces, end with 0):\n");
    for (int i = 0; i < num_clauses; i++) {
        int literal;
        clause_sizes[i] = 0;
        while (true) {
            scanf("%d", &literal);
            if (literal == 0) break;
            clauses[i][clause_sizes[i]++] = literal;
        }
    }

    if (solve(0)) {
        printf("Satisfiable (Yes)\n");
        printf("Satisfying Assignment: ");
        for (int i = 0; i < num_vars; i++) {
            printf("%d ", variables[i] ? (i + 1) : -(i + 1));
        }
        printf("\n");
    } else {
        printf("Satisfiable (No)\n");
    }

    return 0;
}
```

- **Data and Results:**

Data:

The input includes variables, clauses, and literals, ending with 0.

Result:

The output states whether the formula is satisfiable (Yes/No) and provides the satisfying assignment if it exists.

- **Analysis and Inferences:**

Analysis:

The solution uses backtracking to explore all possible assignments to determine satisfaction.

Inferences:

A formula in CNF is satisfiable if all its clauses evaluate to true under some assignment.

Post Lab:

Determine if a given set of integers can be partitioned into two subsets such that the sum of elements in both subsets is the same.

Input: An array of integers.

Output: Yes/No and the subsets if possible.

- **Procedure/Program:**

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

bool canPartition(int arr[], int n, int subsets[2][50], int *len1, int *len2) {
   int totalSum = 0;
   for (int i = 0; i < n; i++)
      totalSum += arr[i];

   if (totalSum % 2 != 0)
      return false;
```

```
int target = totalSum / 2;
bool dp[n + 1][target + 1];
for (int i = 0; i <= n; i++)
   dp[i][0] = true;
for (int j = 1; j <= target; j++)
   dp[0][j] = false;

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

if (!dp[n][target])
   return false;

int subset1[50], subset2[50];
*len1 = 0;
```

```
*len2 = 0;


int i = n, j = target;
while (i > 0 && j > 0) {
   if (dp[i - 1][j]) {
      i--;
   } else {
      subset1[(*len1)++] = arr[i - 1];
      j -= arr[i - 1];
      i--;
   }
}


for (int k = 0; k < n; k++) {
   bool found = false;
   for (int l = 0; l < *len1; l++) {
      if (arr[k] == subset1[l]) {
         found = true;
         break;
      }
   }
   if (!found)
```

```
        subset2[(*len2)++] = arr[k];
    }

    for (int k = 0; k < *len1; k++)
        subsets[0][k] = subset1[k];
    for (int k = 0; k < *len2; k++)
        subsets[1][k] = subset2[k];

    return true;
}

int main() {
    int arr[] = {1, 5, 11, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    int subsets[2][50];
    int len1 = 0, len2 = 0;

    if (canPartition(arr, n, subsets, &len1, &len2)) {
        printf("Yes\n");
        printf("Subset 1: ");
        for (int i = 0; i < len1; i++)
            printf("%d ", subsets[0][i]);
```

```
        printf("\nSubset 2: ");
        for (int i = 0; i < len2; i++)
            printf("%d ", subsets[1][i]);
    } else {
        printf("No\n");
    }

    return 0;
}
```

- **Data and Results:**

Data

Input: Array of integers.

Example: `{1, 5, 11, 5}`

Result

Yes, subsets with equal sum exist: Subset 1: `{1, 11}`, Subset 2: `{5, 5}`

- **Analysis and Inferences:**

Analysis

The sum of the array is even, allowing partitioning.

Inferences

Dynamic programming can efficiently solve the partition problem for subsets.

- **Sample VIVA-VOCE Questions :**

1. What are the advantages of using non-deterministic algorithms?

- Can explore multiple solutions simultaneously.
- Potentially faster for certain problems (e.g., NP-complete).
- Useful for problems where the exact solution is not critical.

2. What are the components of an AND/OR graph?

- **Nodes**: Represent states or conditions.
- **Edges**: Represent actions or decisions.
- **AND Nodes**: Require all child nodes to be satisfied.
- **OR Nodes**: Require at least one child node to be satisfied.

3. What are some techniques for solving problems using AND/OR graphs?

- **Backtracking**: Exploring paths recursively.
- **Heuristic Search**: Using strategies like A* to find optimal paths.
- **Dynamic Programming**: Breaking problems into sub-problems and storing results.

4. Are non-deterministic algorithms always more efficient than deterministic algorithms?

- Not always more efficient.
- Depends on the problem type (e.g., NP-complete problems).
- Non-deterministic algorithms may require more resources in some cases.

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.