Experiment **#14**		Student ID	
Date		Student Name	[@KLWKS_BOT THANOS]

Experiment Title: Implementation of Programs on Branch and Bound Technique Problems.

Aim/Objective: To understand the concept and analyse the algorithm's time complexities with implementation of basic programs on Branch and Bound Technique Problems.

Description: The students will be able to understand problems and implement programs using Branch and Bound Technique.

Pre-Requisites:

Knowledge: Branch and Bound Technique in C/C++/Java/Python

Tools: Code Blocks/Eclipse IDE

Pre-Lab:

Find if there exists a subset of a given set of n integers that sums up to a given target S using Branch and Bound.

Input:

- n (number of integers)
- An array of integers
- Target sum S.

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

Procedure:

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

bool subset_sum_branch_and_bound(int n, int arr[], int target, int *result, int
*result_size) {
  for (int i = 0; i < (1 << n); i++) {
    int sum = 0, idx = 0;
    for (int j = 0; j < n; j++) {
      if (i & (1 << j)) {
            sum += arr[j];
            result[idx++] = arr[j];
          }
        }
        if (sum == target) {
          *result_size = idx;
          return true;
        }
    }
    return false;
}

int main() {
    int n, target;
    printf("Enter the number of integers: ");
    scanf("%d", &n);

    int arr[n], result[n], result_size = 0;
    printf("Enter the integers: ");
    for (int i = 0; i < n; i++) scanf("%d", &arr[i]);

    printf("Enter the target sum: ");
    scanf("%d", &target);

    if (subset_sum_branch_and_bound(n, arr, target, result, &result_size)) {
        printf("Yes\nSubset: ");
        for (int i = 0; i < result_size; i++) printf("%d ", result[i]);
        printf("\n");
    } else {
        printf("No\n");
    }

    return 0;
}
```

In Lab:

You are given weights and values of N items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. Note that we have only one quantity of each item. In other words, given two integer arrays value [0..N-1] and weight [0..N-1] which represent values and weights associated with N items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of value [] such that sum of the weights of this subset is smaller than or equal to W. You cannot break an item, either pick the complete item or do not pick it.

Input:

N = 4

W = 15

values [] = {10, 10, 12, 18}

weight [] = {2, 4, 6, 9}

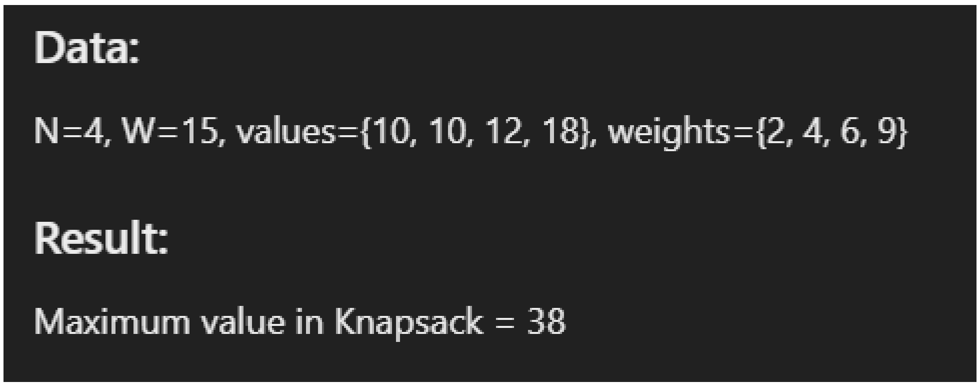
Output: 38

Procedure/Program:

```
#include <stdio.h>

int knapsack(int N, int W, int values[], int weights[]) {
  int dp[N + 1][W + 1];

  for (int i = 0; i <= N; i++) {
    for (int w = 0; w <= W; w++) {
      dp[i][w] = 0;
    }
  }
```

```
    for (int i = 1; i <= N; i++) {
        for (int w = 0; w <= W; w++) {
            if (weights[i - 1] <= w) {
                dp[i][w] = (dp[i - 1][w] > values[i - 1] + dp[i - 1][w - weights[i - 1]])
                            ? dp[i - 1][w]
                            : values[i - 1] + dp[i - 1][w - weights[i - 1]];
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }

    return dp[N][W];
}

int main() {
    int N = 4;
    int W = 15;
    int values[] = {10, 10, 12, 18};
    int weights[] = {2, 4, 6, 9};

    printf("Maximum value in Knapsack = %d\n", knapsack(N, W, values, weights));
    return 0;
}
```

- **Data and Results:**



Data:

N=4, W=15, values={10, 10, 12, 18}, weights={2, 4, 6, 9}

Result:

Maximum value in Knapsack = 38

Post-Lab:

Imagine you are a thief trying to maximize the value of items you can steal from a house. However, you have a limited capacity (knapsack size) to carry the stolen items. Each item has a certain weight and a corresponding value. Your goal is to find the most valuable combination of items to steal without exceeding the knapsack capacity.

Input

Values: {60, 100, 120}

Weights: {10, 20, 30}

Knapsack Capacity: 50

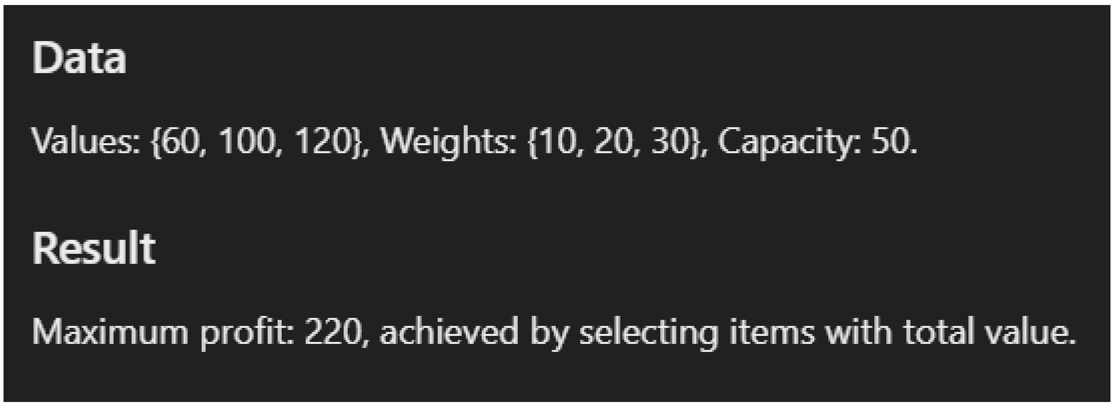
Output

Maximum profit: 220

Procedure/Program:

```
#include <stdio.h>


int knapsack(int values[], int weights[], int n, int capacity) {
    int dp[n + 1][capacity + 1];


    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= capacity; w++) {
            dp[i][w] = 0;
        }
    }

    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= capacity; w++) {
            if (weights[i - 1] <= w) {
                dp[i][w] = (dp[i - 1][w] > dp[i - 1][w - weights[i - 1]] + values[i - 1]) ?
                            dp[i - 1][w] : (dp[i - 1][w - weights[i - 1]] + values[i - 1]);
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }

    return dp[n][capacity];
}

int main() {
    int values[] = {60, 100, 120};
    int weights[] = {10, 20, 30};
    int capacity = 50;
    int n = sizeof(values) / sizeof(values[0]);
    printf("Maximum profit: %d\n", knapsack(values, weights, n, capacity));


    return 0;
}
```

- **Data and Results:**



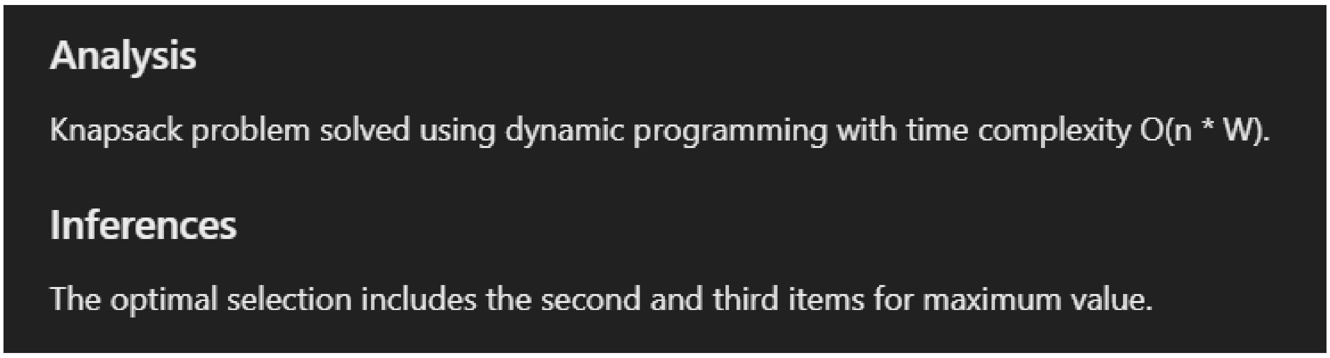
Data

Values: {60, 100, 120}, Weights: {10, 20, 30}, Capacity: 50.

Result

Maximum profit: 220, achieved by selecting items with total value.

- **Analysis and Inferences:**



Analysis

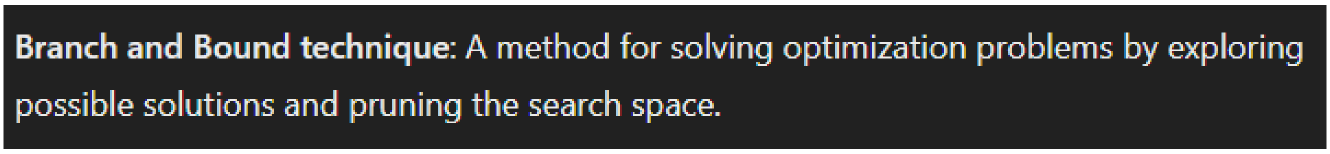
Knapsack problem solved using dynamic programming with time complexity O(n * W).

Inferences

The optimal selection includes the second and third items for maximum value.

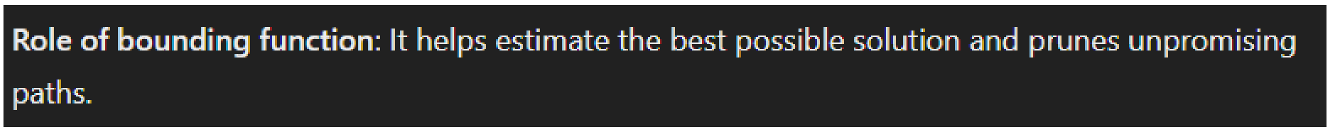
- **Sample VIVA-VOCE Questions :**

1. What is the Branch and Bound technique?



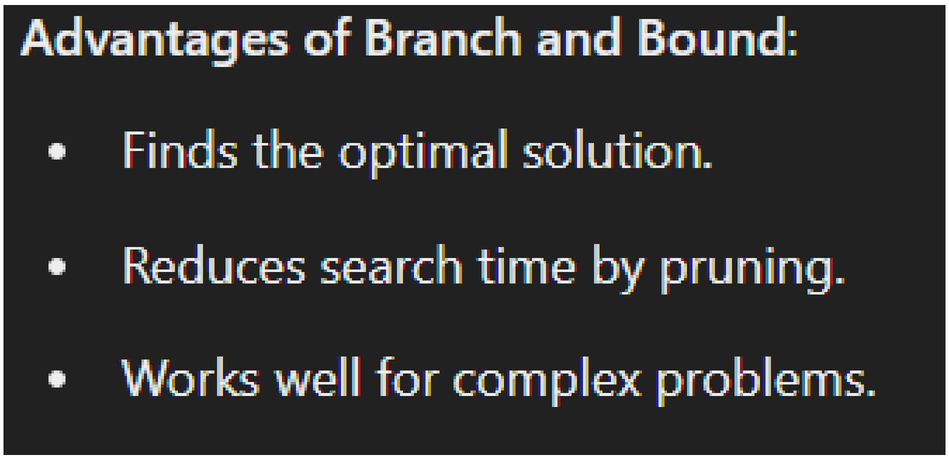
Branch and Bound technique: A method for solving optimization problems by exploring possible solutions and pruning the search space.

2. What is the role of the bounding function in Branch and Bound?



Role of bounding function: It helps estimate the best possible solution and prunes unpromising paths.

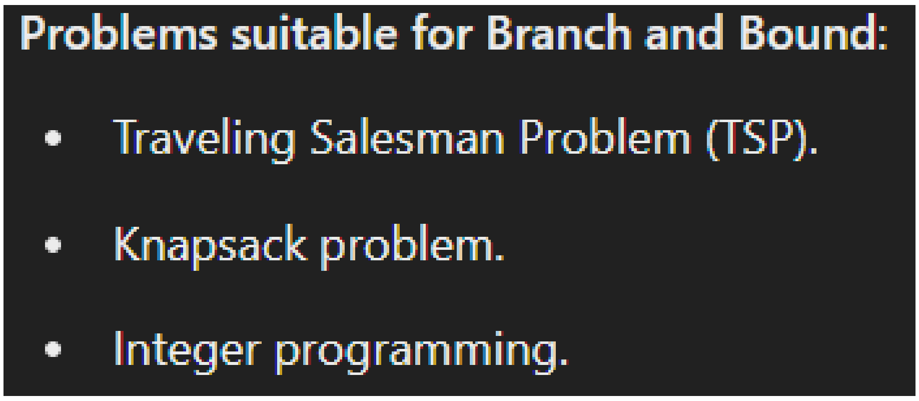
3. What are the advantages of using the Branch and Bound technique?



Advantages of Branch and Bound:

- Finds the optimal solution.
- Reduces search time by pruning.
- Works well for complex problems.

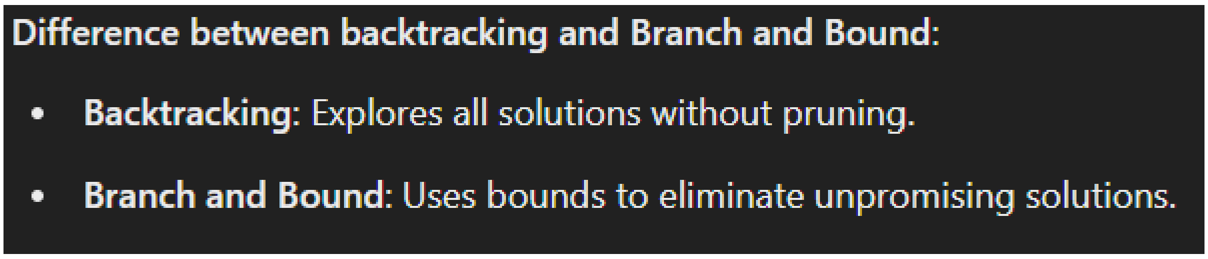
4. What types of problems are suitable for solving using Branch and Bound?



Problems suitable for Branch and Bound:

- Traveling Salesman Problem (TSP).
- Knapsack problem.
- Integer programming.

5. Difference between backtracking and branch and bound?

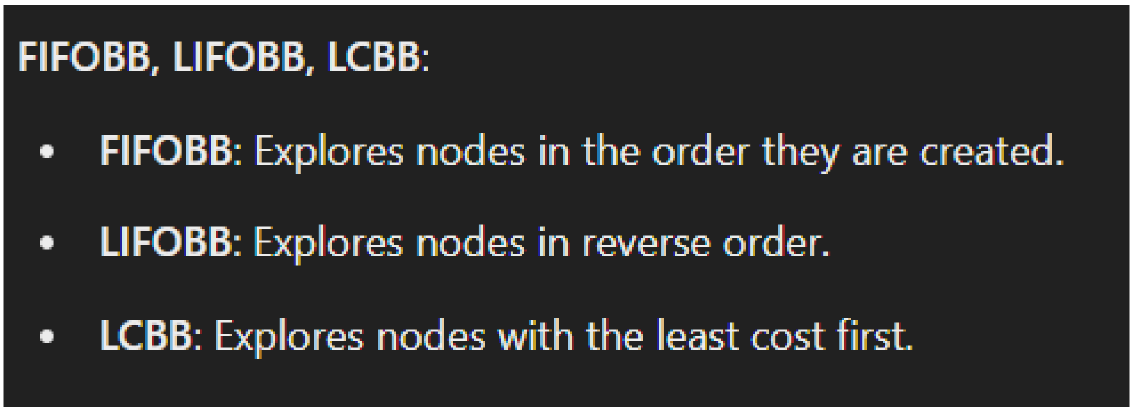


Difference between backtracking and Branch and Bound:

- **Backtracking:** Explores all solutions without pruning.
- **Branch and Bound:** Uses bounds to eliminate unpromising solutions.

Experiment **#14**		Student ID	
Date		Student Name	[@KLWKS_BOT THANOS]

6. Define FIFOBB, LIFOBB and LCBB techniques with examples?



FIFOBB, LIFOBB, LCBB:

- **FIFOBB**: Explores nodes in the order they are created.
- **LIFOBB**: Explores nodes in reverse order.
- **LCBB**: Explores nodes with the least cost first.

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.