PRACTICAL: 10

AIM: Implementation of solution of 0-1 Knapsack using branch and bound.

ALGORITHM: ___________________________

1. Sort all items in decreasing order of ratio of value per unit weight so that an upper bound can be computed using Greedy Approach.
2. Initialize maximum profit, maxProfit = 0
3. Create an empty queue, Q.
4. Create a dummy node of decision tree and enqueue it to Q. Profit and weight of dummy node are 0.
5. Do following while Q is not empty.

- Extract an item from Q. Let the extracted item be u.
- Compute profit of next level node. If the profit is more than maxProfit, then update maxProfit.
- Compute bound of next level node. If bound is more than maxProfit, then add next level node to Q.
- Consider the case when next level node is not considered as part of solution and add a node to queue with level as next, but weight and profit without considering next level nodes.

CODE: ___________________________

```
#include <bits/stdc++.h>
using namespace std;

struct Item
{
	float weight;
	int value;
};

struct Node
{
	int level, profit, bound;
	float weight;
};

bool cmp(Item a, Item b)
{
	double r1 = (double)a.value / a.weight;
	double r2 = (double)b.value / b.weight;
	return r1 > r2;
}

int bound(Node u, int n, int W, Item arr[])
{
	if (u.weight >= W)
		return 0;

	int profit_bound = u.profit;

	int j = u.level + 1;
	int totweight = u.weight;
```

```
	while ((j < n) && (totweight + arr[j].weight <= W))
	{
		totweight += arr[j].weight;
		profit_bound += arr[j].value;
		j++;
	}

	if (j < n)
		profit_bound += (W - totweight) * arr[j].value / arr[j].weight;
	return profit_bound;
}

int knapsack(int W, Item arr[], int n)
{
	sort(arr, arr + n, cmp);

	queue<Node> Q;
	Node u, v;
	u.level = -1;
	u.profit = u.weight = 0;
	Q.push(u);
	int maxProfit = 0;
	while (!Q.empty())
	{
		u = Q.front();
		Q.pop();
		if (u.level == -1)
			v.level = 0;
		if (u.level == n-1)
			continue;
		v.level = u.level + 1;
		v.weight = u.weight + arr[v.level].weight;
		v.profit = u.profit + arr[v.level].value;

		if (v.weight <= W && v.profit > maxProfit)
			maxProfit = v.profit;

		v.bound = bound(v, n, W, arr);

		if (v.bound > maxProfit)
			Q.push(v);
		v.weight = u.weight;
		v.profit = u.profit;
		v.bound = bound(v, n, W, arr);
		if (v.bound > maxProfit)
			Q.push(v);
	}

	return maxProfit;
}

int main()
{
	int W = 10; // Weight of knapsack
	Item arr[] = {{2, 40}, {3.14, 50}, {1.98, 100},
				{5, 95}, {3, 30}};
```

```
        int n = sizeof(arr) / sizeof(arr[0]);

        cout << "Maximum possible profit = "
             << knapsack(W, arr, n);

        return 0;
}
```

OUTPUT: ___________________________

```
Maximum possible profit = 235
```

TIME COMPLEXITY:-

θ(nw)