Lab 6: Implementation of sum of subset problem using backtracking

Objective:

To implement the sum of subset problem using backtracking method.

Theory:

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece ate a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time. It is a form of recursive depth first search.

Algorithm :

1. Create a recursive function that takes the following parameters:
    - The current subset
    - The current sum
    - The target sum
    - The index of the current element being considered
2. If the current sum equals the target sum, we've found a valid subset.
3. If the current sum exceeds the target sum or we've considered all elements, backtrack
4. For the current element, we have two choices:
    - Include it in the subset
    - Exclude it from the subset
5. Recursively try both choices

Observation

```cpp
#include <iostream>
#include <vector>
#include "gettime.h"
using namespace std;

class SubsetSumSolver
{
private:
    vector<int> set;
    int targetSum;
    vector<vector<int>> solutions;

    void backtrack(vector<int> &currentSubset, int currentSum, int index)
    {
        if (currentSum == targetSum)
        {
            solutions.push_back(currentSubset);
            return;
        }
        if (currentSum > targetSum || index >= set.size())
        {
            return;
        }
        currentSubset.push_back(set[index]);
        backtrack(currentSubset, currentSum + set[index], index + 1);
        currentSubset.pop_back();
        backtrack(currentSubset, currentSum, index + 1);
    }

public:
    SubsetSumSolver(const vector<int> &s, int target)
    {
        this->set = s;
        this->targetSum = target;
    }

    vector<vector<int>> solve()
    {
        vector<int> currentSubset;
        backtrack(currentSubset, 0, 0);
        return solutions;
    }
};
```

```
class SubsetSumExecutor
{
public:
    static void execute(int size)
    {
        vector<int> set;
        int targetSum = 50;
        for (int i = 0; i < size; i++)
        {
            set.push_back(rand() % 10);
        }
        SubsetSumSolver solver(set, targetSum);
        vector<vector<int>> solutions;
        long long time = getTime([&]()
                                 { solutions = solver.solve(); });
        cout << "Size = " << size << "Time = " << time << endl;
    }
};

bool flag = 0;
void PrintSubsetSum(int i, int n, int set[], int targetSum,
                    vector<int> &subset)
{
    if (targetSum == 0)
    {

        flag = 1;
        cout << "[ ";
        for (int i = 0; i < subset.size(); i++)
        {
            cout << subset[i] << " ";
        }
        cout << "]";
        return;
    }

    if (i == n)
    {
        return;
    }

    PrintSubsetSum(i + 1, n, set, targetSum, subset);

    if (set[i] <= targetSum)
```

```
    {

        subset.push_back(set[i]);

        PrintSubsetSum(i + 1, n, set, targetSum - set[i],
                       subset);

        subset.pop_back();
    }
}

int main()
{
    vector<int> sizes;
    int start_size = 15;
    int increment = 1;
    sizes.push_back(start_size);
    srand(time(NULL));
    SubsetSumExecutor::execute(start_size);
    for (size_t i = 0; i < 4; i++)
    {
        start_size += increment;
        sizes.push_back(start_size);
        SubsetSumExecutor::execute(start_size);
    }

    return 0;
}
```

Output:

```
Size = 15Time = 2002600
Size = 16Time = 5044300
Size = 17Time = 8994800
Size = 18Time = 236226000
Size = 19Time = 780274600
```

Conclusion

We implemented backtracking algorithm to solve subset sum problem in C++.