Problem 1: Israk spending his free time in his rooftop. So, he starts counting the number of floors of buildings around him. He gives those number to his elder brother and ask him to find the longest building and the shortest building.

Input:

7

B1 6

B2 6

B3 10

B4 8

B5 4

B6 5

B7 3

Output:

Longest B3

Shortest B5

Algorithmic Justification: The problem is to find the longest and shortest buildings in an array on the number of floors. The overall structure of dividing the problem into smaller sub problems, solving them recursively, and combining the results to get the final solution aligns with the key principles of divide-and-conquer algorithms. and it takes O(n log n). in this case, it comes from the fact that the array of buildings is divided into halves at each level of recursion (log n levels), and at each level, the merge or combine step takes linear time O(n). But if we don't use DnC and use linear search algorithm it will take O(n). That's why we use DnC approach.

```cpp
//        ***************  Author :  Khan Israk
Ahmed  ************************\
//        ***************  Date:    27-08-2023     ************************\


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

struct Building
{
    string name;
    int floors;
};

struct LongShortBuilding
{
    int longest;
    int shortest;
};

LongShortBuilding find(Building arr[], int left, int right)
{
    LongShortBuilding result;

    if (left == right) // / No building
    {
        result.longest = left;
        result.shortest = right;
        return result;
    }
    else if (right - left == 1) //// Base case 1
    {
        if (arr[left].floors > arr[right].floors)
        {
            result.longest = left;
            result.shortest = right;
        }
        else
        {
            result.longest = right;
            result.shortest = left;
        }
    }
    else
    {
        int mid = (left + right) / 2;

        LongShortBuilding leftSide = find(arr, left, mid);
        LongShortBuilding rightSide = find(arr, mid + 1, right);
```

```
        if (arr[leftSide.shortest].floors < arr[rightSide.shortest].floors)
        {
            result.shortest = leftSide.shortest;
        }
        else
        {
            result.shortest = rightSide.shortest;
        }

        if (arr[leftSide.longest].floors > arr[rightSide.longest].floors)
        {
            result.longest = leftSide.longest;
        }
        else
            result.longest = rightSide.longest;
    }

    return result;
}

int main()
{
    int numOfBuildings;
    cin >> numOfBuildings;

    Building buildings[numOfBuildings];

    for (int i = 0; i < numOfBuildings; ++i)
    {
        cin >> buildings[i].name;
        cin >> buildings[i].floors;
    }

    LongShortBuilding result = find(buildings, 0, numOfBuildings - 1);

    cout << "Longest: " << buildings[result.longest].name << endl;
    cout << "Shortest: " << buildings[result.shortest].name << endl;

    return 0;
}
/*
7
B1 6
B2 6
B3 10
B4 8
B5 4
```

```
B6 5
B7 3
*/
```

Problem 2: Israk playing an open world adventurer game. Where he's an adventurer. He stated a quest where he has to retrieve the most valuable treasures from an ancient enchanted dungeon. The dungeon contains mystical artifacts, each guarded by a mage guardian. The artifacts not only have weight and value but are also associated with specific enchantments that can either enhance or diminish their overall worth. Also, his backpack has a weight limit. Now how can I maximize the total value while staying within the weight limit.

Input:

6

20

A1 8 15 3

A2 6 20 2

A3 10 25 4

A4 5 10 -1

A5 12 30 5

A6 7 18 -3

3

Output

A3 A1

Total Value with enchantment 40

Algorithmic Justification: Greedy approach can provide optimal solutions in certain scenario but it may not provide the optimal solution in every scenario. However, Greedy approach is generally simpler to implement rather than Dynamic programming. That's why it will be the write choice to use greedy in this problem.

```
//        ***************  Author :  Khan Israk
Ahmed  ************************\

//        ***************  Date:    18-08-2023     ************************\


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

struct Artifact
{
    string name;
    int weight;
    int value;
    int enchantment;
};

int maximizeTotalValue(vector<Artifact> artifacts, int capacity, int
desiredEnchantment)
{
    auto cmp = [&](Artifact a, Artifact b)
    {
        double ratio_A = (double)a.value / a.weight;
        double ratio_B = (double)b.value / b.weight;
        return ratio_A > ratio_B;
    };
    sort(artifacts.begin(), artifacts.end(), cmp);

    int remainingCapacity = capacity;
    int totalValue = 0;

    for (auto a : artifacts)
    {
        if (a.weight <= remainingCapacity && a.enchantment >=
desiredEnchantment)
        {
            totalValue += a.value;
            remainingCapacity -= a.weight;

            cout << a.name << " ";
        }
    }
    cout << endl;
    return totalValue;
}

int main()
{
    int numArtifacts;
```

```
    cin >> numArtifacts;

    int weightLimit;
    cin >> weightLimit;

    vector<Artifact> artifacts(numArtifacts);

    for (int i = 0; i < numArtifacts; ++i)
    {
        getchar();
        cin >> artifacts[i].name;
        cin >> artifacts[i].weight;
        cin >> artifacts[i].value;
        cin >> artifacts[i].enchantment;
    }

    int desiredEnchantment;
    cin >> desiredEnchantment;

    int result = maximizeTotalValue(artifacts, weightLimit,
desiredEnchantment);

    cout << "Total Value with enchantment  " << result << endl;

    return 0;
}
/*
6
20
A1 8 15 3
A2 6 20 2
A3 10 25 4
A4 5 10 -1
A5 12 30 5
A6 7 18 -3
3
*/
```

Problem 3: Israk is teaching his nephew primary math. Suddenly, his nephew asked for help in solving a mathematical problem. The problem is related to finding a targeted number which you can find from adding some given numbers. As an algo learner how can he solve this problem?

Input:

6

2 5 15 8 4 3

16

Output

Subsets with the given sum: (3 8 5)

Algorithmic Justification: In this problem we use Dynamic programming. Because DP divide the problem into smaller subproblems and store their solutions in a dp table and overlapping subproblems. And in the end provide us the optimal solution for the problem. And it also reduced the time complexity. In this problem the time complexity is O(n * target).

```cpp
//        ***************  Author :  Khan Israk
Ahmed  ************************\
//        ***************  Date:    27-08-2023     ************************\


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

void printSubset(const bool subset[][1001], int n, int sum, int set[])
{
    cout << "Subsets with the given sum: ";
    for (int i = 0; i <= n; i++)
    {
        if (subset[i][sum])
        {
            cout << "( ";
            int currentSum = sum;
            for (int j = i; j > 0 && currentSum > 0; j--)
            {
                if (!subset[j - 1][currentSum])
                {
                    cout << set[j - 1] << " ";
                    currentSum -= set[j - 1];
                }
            }
            cout << ") ";
        }
    }
    cout << endl;
}

bool subsetSum(int set[], int n, int sum)
{
    bool subset[n + 1][1001];

    for (int i = 0; i <= n; i++)
    {
        subset[i][0] = true;
    }

    for (int i = 1; i <= sum; i++)
    {
        subset[0][i] = false;
    }

    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= sum; j++)
        {
```

```
            if (j < set[i - 1])
            {
                subset[i][j] = subset[i - 1][j];
            }
            if (j >= set[i - 1])
            {
                subset[i][j] = subset[i - 1][j] || subset[i - 1][j - set[i -
1]];
            }
        }
    }

    if (subset[n][sum])
    {
        printSubset(subset, n, sum, set);
        return true;
    }
    else
    {
        return false;
    }
}

int main()
{
    int n;
    cin >> n;

    int set[n];
    for (int i = 0; i < n; i++)
    {
        cin >> set[i];
    }

    int sum;
    cin >> sum;

    if (subsetSum(set, n, sum) == false)
    {
        cout << "No subset with the given sum";
    }

    return 0;
}

/*
6
2 5 15 8 4 3
```

```
9
*/
```