GREEDY TECHNIQUE

```
// Q. Fractional Knapsack
// You are given n items in a basket.
// Each item => {weight in kg, value in rupees}.
// You want to carry these items to a market and sell them. Selling the entire
// item gives 'value' as a profit.
// You have limited space in your truck -> W kg.
// You can carry a fraction of an item in the truck. Not necessary to take every
// item in the truck (some items can be left out in the basket).
// Find out the maximum profit you can earn with the given items.

// 3 items -> [{20 kg, Rs. 100}, {50 kg, Rs.50}, {10 kg, Rs.60}]
// truck capacity -> 40 kg. -> 40-20 = 20 kg -20 kg = 0
// Find out Max profit.
// Assignment -> [20 kg Item 1, 20 kg Item 2] -> Profit = 20*5 + 20*2 = Rs. 140

// Find value-per-unit-kg ratio for each item.
// Principle: We will not take any item with lesser v/w ratio till an item with
higher v/w ratio is remaining (not filled in the truck).
// Why is this Optimal? If a higher ratio item is in the basket and a lower ratio
// item is laid on the truck, they can be swapped, so that truck can earn higher
// profit with the same weight utilization.

// Demo
// Initial -> [{5 kg, 30 rupees , 6 rupees/kg}, {2 kg, 10 rupees, 5 rupees/kg}]
// W = 4kg
// Truck -> [ Rs.6, Rs.6, Rs.6, Rs.5] = Rs.23, Basket -> {Rs.5, Rs.6, Rs.6}
// Above arrangement is NOT OPTIMAL, swapping is possible.
// After swap -> [ Rs.6, Rs.6, Rs.6, Rs.6] = Rs. 24, Basket-> {Rs.5, Rs.5, Rs.6}
```

```cpp
//Range of v, w -> [1, 1000]
struct Item{
    int v;
    int w;
    bool operator < (Item item){
        return (v * item.w) < (item.v * w);
        // If the above function returns true, the current item will be shifted
to the left in the vector
    }
};

// 2 items a and b
// (a.v / a.w) < (b.v / b.w)
// Same as (a.v * b.w) < (b.v * a.w)
float max_profit(vector<Item> items, int W){
    // Sort in descending order
    sort(items.rbegin(), items.rend());

    int n = items.size();
    float profit = 0;
    for(int i=0; i<n; i++){

        // Capacity of i-th item to be put into truck
        int cap = min(W, items[i].w);

        W -= cap; //Reducing Truck's capacity
        items[i].w -= cap;  // Deducting item's availability

        // Adding (weight taken) * (value per unit weight)
        profit = profit + cap * ( ((float)items[i].v) / items[i].w );

        // Break out when truck is full
        if(W == 0)
            break;
    }

    return profit;
}
```

```
int main(){
    //n
    int n;
    cin>>n;
    vector<Item> items(n);

    // n lines
    // value weight
    for(int i=0; i<n; i++){
        cin>>items[i].v >> items[i].w;
    }

    //W -> truck capacity
    int W;
    cin>>W;

    cout<<max_profit(items, W);
}
```

DP

//fibonacci

```
int fibonacci(int n){

  if(n<=1)
    return n;

  if(dp[n]!=-1)
    return dp[n];

  return dp[n]=fibonacci(n-1)+fibonacci(n-2);

}
```

//factorial

```
int factorial(int n){

  if(n<=1)
    return 1;
```

```
  if(dp[n]!=-1)
    return dp[n];

  return dp[n]=factorial(n-1)*n;

}
```

//binomial coeff

```
int f(int n,int k){

  if(k==0 || k==n)
    return 1;

  if(dp[n][k]!=-1)
    return dp[n][k];

  return dp[n][k]=f(n-1,k)+f(n-1,k-1);

}
```

//0-1 knapsack

```
int f(int i,int c){

  if(i==n)
    return 0;

  if(dp[i][c]!=-1)
    return dp[i][c];

  int op1=f(i+1,c);
  int op2=-INF;

  if(wt[i]<=c)
    op2=f(i+1,c-wt[i])+value[i];

  return dp[i][c]=max(op1,op2);

}
```

Problems on STL/Greedy:

1. https://codeforces.com/problemset/problem/920/B
2. https://codeforces.com/problemset/problem/799/B
3. https://codeforces.com/problemset/problem/81/A
4. https://codeforces.com/problemset/problem/903/C