# Úloha č. 4
## Vesmírný reportér

Rozmysli, popiš a naprogramuj! 10 b

Hledáme trojice robůtků jejihž váha je rovna zadanému číslu. Situace je trochu stížena tím, že vypisujeme indexy, ale to není velká komplikace.

## Naivní řešení O($N^3$)

Projdeme všechny možné trojice z pole a u každého si uděláme test na typ trojúhelníku. Toto řešeni je ovšem příliš pomalé pro největší vstup.

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

int main()
{
    int n, x; ll m;
    while(cin >> n >> m && n ){
        vector<int> v;
        for(int i = 1; i <= n; i ++){
            cin>>x;v.push_back(x);
        }
        for(int a = 0; a < n; a ++)
            for(int b = a+1; b < n; b ++)
                for(int c = b+1; c < n; c ++){
                    if(v[a] + v[b] + v[c] == m){
                        cout << a+1 << ' ' << b+1 << ' ' << c+1 << endl;
                    }
                }
        cout << endl;
    }
}
```

# Řešení $O(N^2 \log(N))$

Pokud si vstup nejprve seřadíme, potom můžeme pro každou dvojici $a$ $b$ si dopočítat hodnotu $c = m - a - b$. Tuto hodnotu můžeme vyhledávat metodou půlení intervalů v čase $O(\log(n))$.

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

int findIndexOf(vector<pair<int, int> > & v, int from, int to, int x){
    int med;
    while (from < to){
        med = (from+to)/2;
        if (v[med].first < x) from = med + 1;
        else if(v[med].first > x ) to = med;
        else return v[med].second;
    }
    return -1;
}

int main()
{
    int n, x; ll m;
    while(cin >> n >> m && n) {
        vector<pair<int, int> > v;
        for(int i = 1; i <= n; i ++){
            cin>>x;v.push_back({x, i});
        }
        sort(v.begin(), v.end());
        for(int a = 0; a < n; a ++)
            for(int b = a+1; b < n; b ++) {
                int valOfC = m-v[a].first-v[b].first;
                int res = findIndexOf(v, b+1, n, valOfC);
                if(res >= 0) {
                    cout << v[a].second << ' ' << v[b].second
                        << ' ' << res << endl;
                }
            }
        cout << endl;
    }
}
```

## Řešení O($N^2$)

Problém nalezení dvojice v seřazeném $n$-prvkovém poli, která bude mít hledaný součet se dá řešit v čase O(n). Namísto hloupého projíždění každé dvojice si můžeme dva ukazatele rozmístit na začátek a konec pole. První ukazatel nám ukazuje na nejmenší položku a druhý na nejvyšší. Pokud je jejich součet vynižší než hledaná hodnota posuneme první ukazatel na další položku a tím součet můžeme jedině zvýšit. A máme zaručeno že jsme nic nepřeskočili, protože první položka byla tak malá, že ani s nejvyšší součet nedosáhl hledané hodnoty. Analogicky pokud je součet vyšší posuneme druhý ukazatel a tím součet snížíme. Pokud se ukazatelé potkají, alegoritmus končí. Tuto techniku pužijeme i na náš problém. Pro každou položku $a$ pole budeme hledat dvojici se součtem $m - a$.

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

int main()
{
    int n, x, c; ll m;
    while(cin >> n >> m && n){
        vector<pair<int, int> > v;
        for(int i = 1; i <= n; i ++){
            cin>>x;v.push_back({x, i});
        }
        sort(v.begin(), v.end());
        for(int a = 0; a < n; a ++){
            c = n - 1;
            for(int b = a+1; b < n; b ++) {
                int valOfC = m-v[a].first-v[b].first;
                while(v[c].first > valOfC && c > b) c --;
                if(c <= b) continue;
                if(v[c].first == valOfC) {
                    cout << v[a].second << ' ' << v[b].second
                        << ' ' << v[c].second << endl;
                }
            }
        }
        cout << endl;
    }
}
```