Knapsack

Fie un şir de numere naturale S={s1, s2, …, sn} şi un număr natural K, cu K≥si pentru orice i între 1 şi n.

a) Să se scrie un algoritm pseudo-polinomial care găsește suma maximă, dar care să fie ≤K, ce poate fi formată din elemente din S (numere întregi, pozitive, luate cel mult o singură dată).(1p)

b) Să se găsească un algoritm aproximativ care calculează o sumă cel puțin pe jumătate de mare ca cea optimă dar rulează în timp O(n) și complexitate spațiu O(1). Mai exact: aveți voie să parcurgeți fiecare element din S cel mult o singură dată, respectiv aveți memorie alocată doar pentru 3 variabile de tip int (dintre care una este K) + varabile de tip stream (1p)

Rezolvare:

a)

```
#include <bits/stdc++.h>

using namespace std;

int main()
{
    int n,w;
    cin>>n>>k;
    int wh[k+5];
    int dp[n+5][k+5];

    memset(dp,0,sizeof(dp));

    for (int i=1; i<=n; i++)
        cin>>wh[i];

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

```
        {
            if (wh[i]>j)
                dp[i][j]=dp[i-1][j];
            else
                dp[i][j]=max(dp[i-1][j], dp[i-1][j-wh[i]]+wh[i]);
        }

    cout<<dp[n][w];
    return 0;
}
```

Complexitate timp: O(nk)

Complexitate spațiu: O(nk)

b)

```
#include <bits/stdc++.h>

using namespace std;

ifstream fin("data.in");

int main()
{
    int k, elem;
    int suma=0;
    cin>>k;
    while(fin>>elem)
    {
        if(elem+suma<=k)
            suma+=elem;
        else if(suma<elem)
            suma=elem;
    }
    cout<<suma;
    return 0;
}
```

Fie OPT suma calculată cu algoritmul optim. Întrucât trebuie sa găsim o sumă ce este cel puțin egală cu $\frac{1}{2}OPT$, distingem 2 cazuri:

Cazul 1. Suma ajunge la $s \geq \frac{1}{2}OPT$. Atunci condiția din ipoteză se îndeplinește.

Cazul 2. Suma este mai mică decât jumătate din soluția optimă și urmează un element care nu poate fi adăugat pentru a nu depăși k. Atunci acel element este mai mare decât $\frac{1}{2}OPT$, deci egalăm suma cu acel element și revenim la condiția din cazul 1.