5.3.

Če je problem optimizacijski, lahko za problem uporabimo genetski algoritem, ki sam hoče konvergirati s fitnes funkcijo, namesto da ga randomiziramo kot dela monte carlo.

a) minimalno vpeto drevo:

Če graf nima negativnih povezav:
Dijkstrov algoritem O(m * log(n)) - v NP in imajo polinomski čas

Algoritem (s predavanj):

```
Dijkstra(G,s){
        s = {};
        for(all v in G){ d[v] = Integer.MAX_VALUE;, pq.insert(v);}
        pq.decreaseKey(s,0)
        while(! pq.isEmpty()){
                u = pq.delMin()
                for all x in (u,x){
                        if(d[x) > d[x + w(u,x=)]){
                                pq.decreaseKey(x, d[u]+w(u,x)); //relaksacija
                        }
                }
        }
}
```

Če pa graf ima negativne povezave, pa lahko uporabnimo Belman-Ford alg:

-inicializiramo cene vseh vozlišč na neskončno, izvorno na 0
-nkrat ponovimo:
    za vsako povezavo U,V poženemo relaksacijo
-še enkrat poženemo relaksacijo da zvemo ali je v grafu negativen cikelj

b) subset sum problem: NP-polni

Rekurzivni algoritem je eksponenten. Obstaja pseudo polinomski čas z dinamičnim programiranjem, lahko pa naredimo aproksimacijski alg rešljiv v polinomskem času če so števila nenegativna:
https://en.wikipedia.org/wiki/Subset_sum_problem

```
S  = {0}
for i in range(1,N):
        T = []
        for(y in S):
                T.append(x[i] + y)
        U = union(T,S)
        U = sorted(U)
        S = []
        y = min(U)
        S.append(y)
        for(z in U):
                if y + c/N < z <= s:
                        y =z
                        S.append(z)
if (S.contains(range(1,c))):
        print("yes")
else:
        print("no")
```

c) min partition difference:

Časovna kompleksnost z rekurzijo: O(2^n)- optimizacijski problem -NP-težek

spodnji algoritem vrne [[2, 3, 1, 5], [6, 4]] saj je razlika vsot množic 0

```
import random
import math

_ITERATIONS = 100000
_SET = [2, 3, 1, 5, 6, 4]

def getsumDifference(array_2d):
    # returns difference between all arrays - O(n^2)
    diff = 0
    for i in range(len(array_2d)):
        sum_arr1 = sum(array_2d[i])
        for j in range(len(array_2d)):
            sum_arr2 = sum(array_2d[j])
            diff += abs(sum_arr1 - sum_arr2)
    return diff


def main():
    MIN_DIFF = math.inf
    MIN_SET = {}

    for i in range(_ITERATIONS):  # n iterations
        rand_num_sets = random.randint(0, len(_SET) - 1)
        for j in range(rand_num_sets):  # split on random number of sets
            array_2d = []
            for i in range(0, len(_SET), rand_num_sets):
                array_2d.append(_SET[i: i + rand_num_sets])  # same chunks
            diff = getsumDifference(array_2d)
            if diff < MIN_DIFF:
                MIN_DIFF = diff
                MIN_SET = array_2d
    print(MIN_SET)


main()
```