Theorem 0.1. *je dana postupnost kladnych aj zapornych cisel $a_1, ..., a_n$. chceme najst taku podpostupnost, ktora bude mat najvacsi sucet.*

Proof. zamyslime sa najprv nad trivialnym riesenim. kedze nevieme kde hladana podpostupnost bude a ani aka dlha bude, musime postupne vyskusat vsetky mozne zaciatky (n moznosti) a vsetky mozne dlzky (toto sa bude menit s polohou zaciatku).

takze pre zaciatok hned v a_1 mame n moznych dlzok (mozeme ostat stat, mozeme zobrat este nasledujuci clen atd ... teoreticky mozeme ist az na koniec celej postupnosti), pre zaciatok v a_2 mame $n-1$ moznych dlzok atd ... az pre zaciatok v a_{n-1} mame 2 mozne dlzky a pre a_n mame jednu. pozorne oko si iste vsimlo, ze celkovo mame teda $1+2+3+...+n$ moznosti pre polohu hladanej podpostupnosti. no a kazdy vie, ze

$$1+2+3+...+n = \frac{n(n+1)}{2} = O(n^2)$$

my by sme chceli lepsi algoritmus. skusime metodu rozdeluj a panuj.

cielom tejto metody je dostat rekurziu s casovou zlozitostou

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log n)$$

teda z problemu dlzky n spravit dva polovicnej velkosti, pricom spajanie ciastkovych rieseni pre polovicne problemy a ine rezijne naklady nas budu stat linearne vela casu ... $O(n)$.

idea je jednoducha: z danej postupnosti dlzky n vyrobime dve polovicne (lavu a pravu) ... prva polovica prvkov do jednej, druha do druhej (na toto treba $O(n)$ operacii). potom riesime polovicne postupnosti zvlast ... $2T\left(\frac{n}{2}\right)$ ... a az budeme mat vysledky pre ne, potom z nich zlozime riesenie pre celu postupnost.

povedzme, ze sme uz dostali riesenia pre polovicne postupnosti ... teda dostali sme trojicu cisel (x_l, y_l, z_l) z lavej polovice hovoriacu, ze riesenim je podpostupnost so suctom z_l zacinajuca v x_l a konciaca v y_l. podobnu trojicu (x_p, y_p, z_p) dostaneme aj z pravej polovice.

teraz treba tieto ciastkove riesenia skombinovat do riesenia pre celu postupnost. na zaciatok mozeme porovnat z_l so z_p a zistit, ktore ciastkove riesenie je lepsie. treba vsak pamatat na to, ze rozdelenim postupnosti na

dve sme este nepozreli postupnosti, ktore by prechadzali tymto predelom a to musime teraz napravit. toto sa moze zdat zaludne ale vlastne vobec nie je.

pozrite sa este raz na trivialne riesenie. tam bol problem v tom, ze sme nevedeli kde zacat ani kde skoncit. teraz ale predsa vieme kde zaciname .... presne v strede ... zostava iba zistit, kde mame skoncit.

presnejsie, zo stredu pojdeme nalavo aj napravo krok za krokom a pre kazdy prvok si budeme ukladat velkost podpostupnosti, keby sme zastavili na danom prvku. keby sme mali nasledujucu postupnost (|| oznacuje predel)

$$3,\ 14,\ -1,\ -5,\ 9,\ -2,\ -6 \mid\mid 5,\ 3,\ 5,\ 89,\ -7,\ -92,\ 3$$

potom by sme si cestou nalavo od predelu postupne pamatali

$$\begin{aligned} L_1 &= -6 \\ L_2 &= -6 - 2 = -8 \\ L_3 &= -6 - 2 + 9 = 1 \\ L_4 &= -4 \\ L_5 &= -5 \\ L_6 &= 9 \\ L_7 &= 12 \end{aligned}$$

vidime, ze sa nam oplati ist az na koniec. nieco podobne by sme mali aj pre pravu stranu

$$\begin{aligned} P_1 &= 5 \\ P_2 &= 5 + 3 = 8 \\ P_3 &= 13 \\ P_4 &= 102 \\ P_5 &= 95 \\ P_6 &= 3 \\ P_7 &= 6 \end{aligned}$$

tu sa uz neoplati ist az po koniec, najlepsie je zobrat len 4 cleny od stredu napravo.

pre najvacsiu postupnost prechadzajucu predelom potom staci tieto dve casti spojit ... teda v nasom pripade by vysledna zacinala na lavom konci a isla by po 4. clen od stredu napravo.

este si uvedomte, ze toto cele nam trva iba $O(n)$.

dobre, teraz ostava uz len to dat dokopy. mame riesenie pre lavu polovicu, pravu polovicu a tiez pre podpostupnosti prechadzajuce stredom. ostava uz len vybrat si najlepsiu.

velmi hruby pseudokod:

```
function naj(S)

if S={a_i} then return (i,i,a_i) else
        daj 1. polovicu prvkov S do L    ...     O(n)
        daj 2. polovicu prvkov S do P    ...     O(n)
        lave_riesenie  = naj(L)          ...     T(n/2)
        prave_riesenie = naj(P)          ...     T(n/2)
        najdi naj podpostupnost
                prechadzajucu stredom S  ...     O(n)
        vyber najlepsie riesenie
                z 3 kandidatov           ...     O(1)
```

premyslite si implementacne detaily, pripadne si to skuste naprogramovat (tak zistite, ci tomu naozaj rozumiete).

□