Algoritem

Vsak dobro definiran računski postopek, s katerim rešimo kak problem Vhodne podatke spremeni v izhodne. Pozorni moramo biti na 4 stvari:

1. Ali pravilno deluje?
2. Kakšna je časovna zahtevnost?
3. Kakšna je prostorska zahtevnost (velikost problema)?
4. Ali se da narediti bolje?

Delujoče ne implicira pravilno.

Pravilnost delovanja

Za primer bomo vzeli algoritem z vstavljanjem ali *insertion sort*. Algoritem je učinkovit za delo z malo podatki.

```
1:  for j ← 2 to A.length do
2:      key ← A[j]
3:      i ← j − 1
4:      while i > 0 and A[i] > key do
5:          A[i + 1] ← A[i]
6:          i ← i − 1
7:      end while
8:      A[i + 1] ← key
9:  end for
```

Kako algoritem deluje

Naj bo $A = \{5, 2, 4, 6, 1, 3\}$ seznam števil. Indeks i označuje trenutno karto iz seznama A , ki bo evaluirana. Na začetku vsake iteracije for loopa je podseznam elementov $A[1 : i - 1]$ sestavljen iz trenutno sortiranih kart. Podseznam $A[i + 1 : n]$ (n = število elementov) je sestavljen iz kart seznama A,ki še niso nujno sortirane.

Te lastnosti opišemo formalno z *zančno invarianco*: to je *izjava o stanju spremenljivk v algoritmu, ki velja za vse ponovitve (iteracije)*. Pomaga nam razumeti zakaj je nek algoritem korekten. Ko uporabljamo ta pristop opisa, moramo biti pozorni na tri stvari:

1. **initializaction** : drži pred prvo iteracijo loopa
2. **maintenance** : če drži pred začetkom iteracije loopa, potem bo veljalo pred naslednjo iteracijo
3. **termination** : loop se ustavi in s tem poda podatek o tem, zakaj je algoritem veljaven.

Zadnji korak je najbolj pomemben, saj pove kdaj se algoritem ustavi. Z matematično indukcijo dokazujemo, da nekaj velja v neskončnost, z zančno invarianco pa se indukcija konča, ko se loop terminira.

Pravilnost delovanja z zančno invarianco

Dokaz za loop lahko naredimo z indukcijo po zunanji zanki. To pomeni, če imamo znotraj loopa več loopov, gledamo samo zunanjega. BŠZS predpostavimo, da so elementi med sabo različni.

1. Inicializacija

Pokažemo, da velja pred prvo iteracijo loopa. Zdaj velja, da je podseznam $A[1 : i - 1]$ sestavljen iz samo enega elementa, ki je po sebi že urejen.

2. Maintenance

Pokažemo, da velja lastnost za vse nadaljne iteracije loopa. Loop deluje tako, da premakne elemente $A[i - 1]$, $A[i - 2]$, $A[i - 3]$ itd. eno mesto v desno, dokler ne najde pravega položaja za element $A[i]$.

Z drugimi besedami, element $A[i]$ premika v levo, dokler ne najde pravega mesta zanj.

Ko najde mesto za i-ti element, velja, da so v podseznamu $A[1:i]$ ista števila, kot prej, a v urejenem vrstnem redu, podseznam $A[i+1:n]$ pa ostaja še nedotaknjen in neurejen (ta del je nekako zančna invarianca; to glede urejenih števil).

3. Termination
Pokažemo, kdaj se loop ustavi. Začne se z $i = 2$ in se veča za 1 vsako iteracijo. Ko je enkrat $i > n$, se loop ustavi. To velja torej, ko je $i = n + 1$.

$$\begin{aligned} i &= n+1 \\ i-1 &= n \end{aligned}$$

Če vstavimo n v zančno invarianco, vidimo, da velja, da je podseznam $A[1:n]$ (kar je enako $A[1:i-1]$) sestavljen it vseh elementov iz $A[1:n]$, a v urejenem vrstnem redu, torej so vsi elementi po koncu loopa urejeni.

Na začetku vsake iteracije for loopa je podseznam $A[1:i-1]$ sestavljen iz elementov prvotno v $A[1:i-1]$, a v sortiranem vrstnem redu. To implicira, da je naš algoritem korekten. □

Dokaz z indukcijo

Delovanje lahko pokažemo tudi z indukcijo.

1. **Baza indukcije:** če je samo en element (a_1), je polje urejeno.
2. **Hipoteza:** predpostavimo, da algoritem uredi elemente:

$$a_1, a_2, ..., a_{j-1} \to a'_1, a'_2, ..., a'_{j-1}$$

3. **Indukcijski korak:** imamo neko polje elementov $a_1, a_2, ..., a_j$. Prvih $j-1$ po predpostavki znamo urediti v $a'_{\{i\}}$. While zanka se izvaja dokler je `key` manjši (key je `A[j]`, torej prvi pivot, primerjamo ga z vsakim naslednjim `A[i]`). Ustavi se pri i-tem elementu in vstavi na to mesto. Tedaj so elemeti $a'_1 < a'_2 < ... < a'_i <$ key urejeni. ... in some bollocks.