Algoritem Timsort

Algoritmi in podatkovne strukture 2

Algoritem Timsort

- stabilen algoritem za urejanje zaporedij (tabel)
  - ohrani medsebojni vrstni red enakih elementov
  - pomembno pri urejanju tabel elementov sestavljenih tipov (objektov oz. struktur)
- podoben naravnemu algoritmu z zlivanjem (angl. natural merge sort)
  - enota zlivanja je četa
- osredotočili se bomo na različico, opisano v N. Auger et al., On the worst-case complexity of TimSort, ESA 2018
  - možne številne optimizacije
  - obstajajo različne implementacije (python, java, ...)

Četa

- strogo naraščajoče ali strogo padajoče strnjeno podzaporedje znotraj vhodnega zaporedja
- primer zaporedja s šestimi četami

  6, 11, 14, 10, 1, 3, 7, 12, 15, 2, 8, 17, 13, 9, 4, 5, 16

Časovna zahtevnost

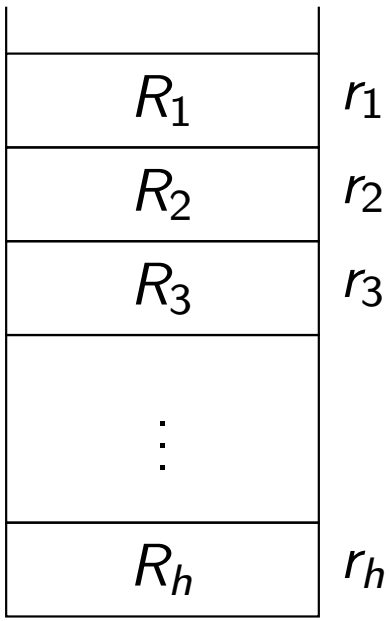
- $O(n \log \rho)$, kjer je ρ število čet v vhodnem zaporedju
- $O(n \log n)$ v najslabšem primeru
  - n čet
- $O(n)$ v najboljšem primeru
  - ena sama četa

Pregled

- potuje po vhodnem zaporedju od začetka do konca in vzdržuje sklad četa
- vsakokrat doda po eno četo na sklad
- če so izpolnjeni določeni pogoji, zlije bodisi vrhnji dve četi na skladu ali pa drugo in tretjo od vrha
- če na koncu ostane še kaj četa na skladu, se zlijejo od vrha navzdol (vsakokrat po dve vrhnji četi)

Psevdokoda — izhodišče

- h: višina sklada (število čet na skladu)
- R_1, R_2, R_3, …, R_h: čete od vrha proti dnu
- r_i: dolžina čete R_i



Psevdokoda

$\mathcal{R} \leftarrow$ prazen sklad
while v S je še kakšna četa **do**
    odstrani četo r iz S in jo postavi na $\mathcal{R}$
    while true **do**
        if $h \geq 3 \wedge r_1 > r_3$ **then** zlij R_2 in R_3
        else if $h \geq 2 \wedge r_1 \geq r_2$ **then** zlij R_1 in R_2
        else if $h \geq 3 \wedge r_1 + r_2 \geq r_3$ **then** zlij R_1 in R_2
        else if $h \geq 4 \wedge r_2 + r_3 \geq r_4$ **then** zlij R_1 in R_2
        else break
        end if
    end while
end while
while $h > 1$ **do**
    zlij četi R_1 in R_2
end while

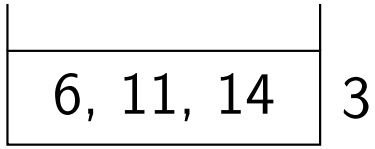
Primer izvajanja

- vhodno zaporedje

  6, 11, 14, 10, 1, 3, 7, 12, 15, 2, 8, 17, 13, 9, 4, 5, 16

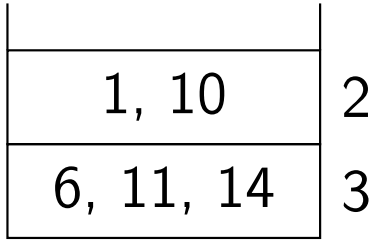
Primer izvajanja

- na sklad postavi četo 6, 11, 14



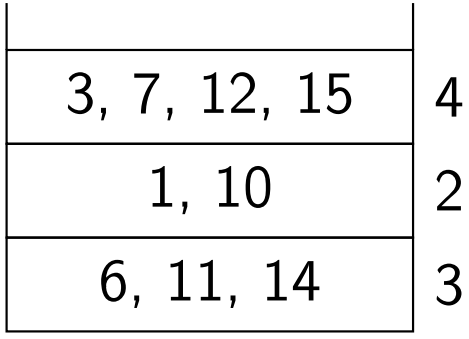
Primer izvajanja

- na sklad postavi četo 10, 1

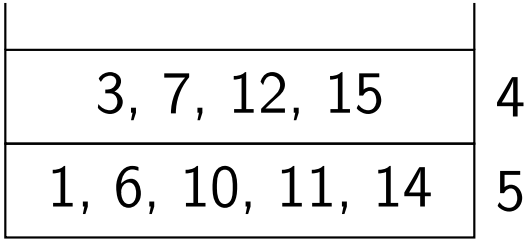


Primer izvajanja

- na sklad postavi četo 3, 7, 12, 15

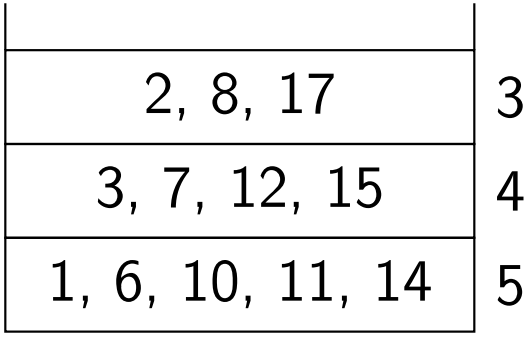


- $r_1 > r_3 \implies$ zlij R_2 in R_3

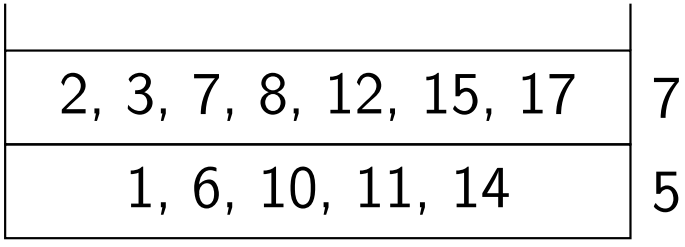


Primer izvajanja

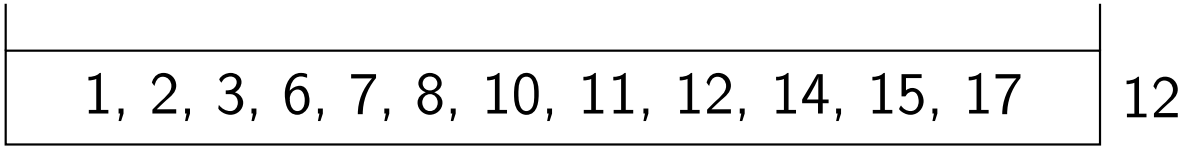
- na sklad postavi četo 2, 8, 17



- $r_1 + r_2 \geq r_3 \implies$ zlij R_1 in R_2

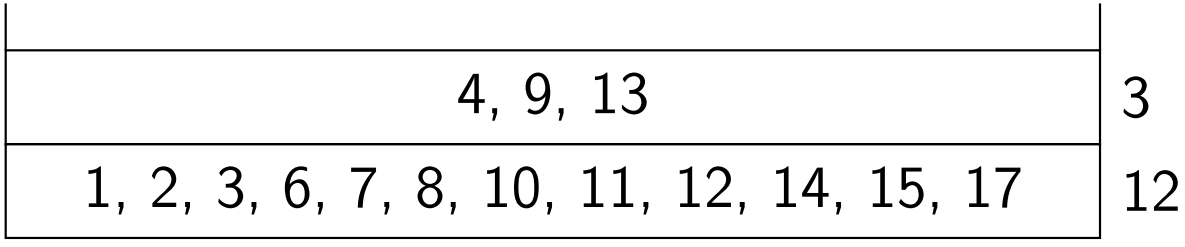


- $r_1 \geq r_2 \implies$ zlij R_1 in R_2



Primer izvajanja

- na sklad postavi četo 13, 9, 4



Primer izvajanja

- na sklad postavi četo 5, 16

5, 16	2
4, 9, 13	3
1, 2, 3, 6, 7, 8, 10, 11, 12, 14, 15, 17	12

- končna zanka: zlij R_1 in R_2

4, 5, 9, 13, 16	5
1, 2, 3, 6, 7, 8, 10, 11, 12, 14, 15, 17	12

- končna zanka: zlij R_1 in R_2

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17	17

Implementacija

- na skladu je smiselno hraniti samo dolžine čet
- čete hranimo in zlivamo v vhodni tabeli
- pri zlivanju »recikliramo« isto pomožno tabelo
- previdno pri zlivanju (in obračanju) padajočih čet (stabilnost!)
- obstaja veliko optimizacij
- najbolj tipična: z algoritmom navadnega vstavljanja (učinkovit pri majhnih vhodih) dosežemo, da nimamo zelo kratkih čet
  - daljše čete $\Longrightarrow$ manj čet $\Longrightarrow$ hitrejše izvajanje