Sortiranje podataka

Sortiranje izborom

- Sortiranje izborom (Selection Sort) je vrlo jednostavan algoritam sortiranja
- Algoritam: u listi se nađe najmanji element i on mijenja mjesto s prvim elementom liste, postupak se ponavlja za listu od druge do zadnje pozicije itd, u svakom koraku je lista koja se sortira kraća za jedan element

- Primjer: 7 3 5 1 8 4 2 9 6
  1 | 3 5 7 8 4 2 9 6
  1 | 2 | 5 7 8 4 3 9 6
  1 | 2 | 3 | 7 8 4 5 9 6
  1 | 2 | 3 | 4 | 8 7 5 9 6
  1 | 2 | 3 | 4 | 5 | 7 8 9 6
  1 | 2 | 3 | 4 | 5 | 6 | 8 9 7
  1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 8
  1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

- Zapis algoritma: ulaz je nesortirana lista a_1, …, a_n , izlaz je sortirana lista

1. za i = 1, 2, …, n-1 radi korake 2 – 5
2. min = i
3. za j = i, .., n radi korak 4
4. ako je $a_j < a_{min}$ min = j
5. zamjeni(a_i, a_{min})

- Koraci 1, 2 i 5 se ponavljaju n-1 puta, koraci 3 i 4 se u prvom prolazu obavljaju n-1 puta, u drugom n-2, …, bez obzira na ulazni niz podataka. Jedini korak čiji broj ponavljanja se mijenja ovisno o ulaznom nizu podataka je korak 4. Najgori slučaj je da se korak 4 obavlja u svakom prolazu.
- Složenost problema je dakle:

$$T_{max}(n) = c_1 * n + \sum_{i=1}^{n-1} c_2 * i = c_1 * n + c_2 * \frac{n(n-1)}{2} = (c_1 - \frac{c_2}{2})n + \frac{c_2}{2} * n^2$$

- Dakle složenost u najgorem slučaju je $\Theta(n^2)$

- U najboljem slučaju, korak 4 se ne obavi nijednom (lista je već sortirana)
- Tada je konstanta uz sumu manja nego u najgorem slučaju, ali je ukupan broj operacija ista funkcija od n, pa je dakle i u najboljem slučaju složenost algoritma $\Theta(n^2)$
- Iz toga slijedi da je i prosječna složenost algoritma $\Theta(n^2)$

Sortiranje zamjenom

- (Exchange Sort) algoritam: prvi element niza se uspoređuje sa svakim iza sebe, kada se naiđe na manji element, oni zamjene mjesta i nastave se uspoređivati preostali elementi s novim prvim elementom. Nakon prvog prolaza na prvom mjestu je najmanji element. Postupak se ponavlja za drugi, itd
- Primjer: 7 3 5 1 8 4 2 9 6

  3 7 5 1 8 4 2 9 6

  1 7 5 3 8 4 2 9 6 do kraja niza se ne nađe manji

  1 | 7 5 3 8 4 2 9 6

  1 | 5 7 3 8 4 2 9 6

  1 | 3 7 5 8 4 2 9 6

  1 | 2 | 7 5 8 4 3 9 6

  1 | 2 | 5 7 8 4 3 9 6

  1 | 2 | 4 7 8 5 3 9 6

  1 | 2 | 3 | 7 8 5 4 9 6

  1 | 2 | 3 | 5 8 7 4 9 6

  1 | 2 | 3 | 4 | 8 7 5 9 6

1 | 2 | 3 | 4 | 7 8 5 9 6
1 | 2 | 3 | 4 | 5 | 8 7 9 6
1 | 2 | 3 | 4 | 5 | 7 8 9 6
1 | 2 | 3 | 4 | 5 | 6 | 8 9 7
1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 8
1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

- Zapis algoritma: ulaz je nesortirana lista a_1, …, a_n , izlaz je sortirana lista

1. za i = 1, .., n-1 radi korake 2 i 3
2. za j = i+1, …, n radi korak 3
3. ako je $a_i > a_j$ onda zamjeni (a_i, a_j)

- Složenost algoritma: vanjska petlja se izvrši n-1 puta, unutarnja u prvom prolazu n-1 puta, u drugom n-2, …
- Najgori slučaj: uvijek se izvrši zamjena u koraku 3

$$T_{max}(n) = c_1 * (n-1) + \sum_{i=1}^{n-1} c_2 * i = c_1 * (n-1) + c_2 * \frac{n(n-1)}{2}$$

- Dakle složenost je $\Theta(n^2)$

- U najboljem slučaju niti jednom nema zamjene mjesta u koraku 3, što znači da je konstanta uz sumu manja nego u najgorem slučaju, ali je ovisnost o n istog oblika, što znači da je i najbolja složenost $\Theta(n^2)$
- To naravno znači da je i prosječna složenost $\Theta(n^2)$

Sortiranje umetanjem

- Sortiranje umetanjem (Insertion Sort) se vrlo često koristi ako broj elemenata u nizu nije prevelik
- Algoritam: početna lista se dijeli na dva dijela, prvi već sortiran i drugi koji treba sortirati. Na početku je u prvom dijelu samo prvi element. U svakom koraku se uzima prvi element iz drugog dijela liste i umeće se u odgovarajuće mjesto u prvom dijelu liste. Elementi sortiranog dijela liste uspoređuju se od zadnjeg prema prvom

- Primjer:

7 | 3 5 1 8 4 2 9 6	uzme se 3
3 7 | 5 1 8 4 2 9 6	uzme se 5, usporedi s 7, 3
3 5 7 | 1 8 4 2 9 6	uzme se 1, usporedi s 7, 5, 3
1 3 5 7 | 8 4 2 9 6	uzme se 8, usporedi s 7, ne pomiče se
1 3 5 7 8 | 4 2 9 6	sad ide 4, usporedi s 8, 7, 5, 3
1 3 4 5 7 8 | 2 9 6	sad ide 2, usporedi se s 8, 7, 5, 4, 3, 1
1 2 3 4 5 7 8 | 9 6	9 se usporedi s 8 i ne pomiče
1 2 3 4 5 7 8 9 | 6	6 se usporedi s 9, 8, 7, 5
1 2 3 4 5 6 7 8 9	

- Zapis algoritma: ulaz je nesortirana lista a_1, …, a_n , izlaz je sortirana lista

1. izvrši korake 2-4 za j=2, …, n
2. izvrši korak 3 za i=1, …, j-1
3. ako je $a_i > a_j$ zamjeni(a_i, a_j)

- Složenost algoritma: korak 1 se uvijek izvrši n-1 puta, koraci 2 i 3 se u prvom prolazu izvrše jednom, pa dvaput, …, sve do n-1 puta
- Najgori slučaj je obrnuto sortirana lista kada se u svakom koraku moraju zamijeniti elementi:

$$T_{max}(n) = c_1 * (n-1) + \sum_{i=1}^{n-1} c_2 * i = c_1 * (n-1) + c_2 * \frac{n(n-1)}{2}$$

- Dakle složenost je $\Theta(n^2)$
- Najbolji slučaj je kad nema zamjena elemenata, pa je konstanta uz sumu manja, ali je opet složenost $\Theta(n^2)$
- Pa i ovdje vrijedi da je prosječna složenost $\Theta(n^2)$

Mjehuričasto sortiranje

- (Bubble Sort) – prolazi se redom po elementima liste i svaki se uspoređuje sa svojim sljedbenikom, ako je veći zamjene mjesta, time je na kraju prvog prolaza listom na zadnjem mjestu najveći element, postupak se ponavlja za listu skraćenu za zadnju poziciju

- Primjer:

7 3 5 1 8 4 2 9 6	1. i 2. element mijenjaju mjesta
3 7 5 1 8 4 2 9 6	2. i 3. element mijenjaju mjesta
3 5 7 1 8 4 2 9 6	3. <-> 4.
3 5 1 7 8 4 2 9 6	sad mjesta mijenjaju 5. i 6.
3 5 1 7 4 8 2 9 6	6. <-> 7.
3 5 1 7 4 2 8 9 6	8. <-> 9.
3 5 1 7 4 2 8 6 | 9	2. prolaz: kreće se od početka, kraća lista
3 1 5 7 4 2 8 6 | 9	4. <-> 5.
3 1 5 4 7 2 8 6 | 9	5. <-> 6.
3 1 5 4 2 7 8 6 | 9	7. <-> 8.
3 1 5 4 2 7 6 | 8 9	3. prolaz: iznova na kraćoj listi

1 3 5 4 2 7 6 | 8 9 3. <-> 4.

1 3 4 5 2 7 6 | 8 9 4. <-> 5.

1 3 4 2 5 7 6 | 8 9 6. <-> 7.

1 3 4 2 5 6 | 7 8 9 novi prolaz

1 3 2 4 5 | 6 7 8 9 novi prolaz

1 2 3 4 | 5 6 7 8 9 još 3. puta prolazi listu, iako je sortirana

- Zapis algoritma: ulaz je nesortirana lista a_1, …, a_n , izlaz je sortirana lista

1. ponavljaj korake 2-3 za i=1, …,n-1
2. ponavljaj korak 3. za j=1, …, n-i
3. ako je $a_j > a_{j+1}$ onda zamjeni(a_j, a_{j+1})

- Složenost algoritma: korak 1. se uvijek ponavlja n-1 puta, u prvom prolazi se koraci 2. i 3. ponavljaju n-1 puta, pa n-2, itd.
- Najgori slučaj: u svakom koraku se vrši zamjena, najbolji slučaj je kad nema nijedne zamjene; u oba slučaja isti je broj provjera i ovisnost o n, tj. i asimptotska ovisnost je ista:

$$T_{pros}(n) = c_1 * (n-1) + \sum_{i=1}^{n-1} c_2 * i = c_1 * (n-1) + c_2 * \frac{n(n-1)}{2} = \Theta(n^2)$$

- U primjeru je algoritam izvršio svih osam koraka iako je sortiranje bilo gotovo nakon pet koraka, tj, već u šestom koraku nije bilo nijedne zamjene mjesta
- Poboljšanje algoritma: provjerava se da li je bilo zamjene u tekućem koraku, ako nije, lista je već sortirana i algoritam završava

Sortiranje spajanjem

- Sortiranje spajanjem (Merge Sort) je algoritam sortiranja čija je složenost manja od kvadratne – O(n*lg n)
- Strategija podijeli-pa-vladaj
- Primjer:

(9, 2, 4, 6, 8, 1, 7, 5)

podijeli se na dvije liste

(9, 2, 4, 6) (8, 1, 7, 5)

podijeli se na 2 liste podijeli se na 2 liste

(9, 2) (4, 6) (8, 1) (7, 5)

podijeli na po 2 liste

(9) (2) (4) (6) (8) (1) (7) (5)

spajanje

(2, 9) (4, 6) (1, 8) (5, 7)

spajanje spajanje

(2, 4, 6, 9) (1, 5, 7, 8)

spajanje

(1, 2, 4, 5, 6, 7, 8, 9)

- Algoritam sortiranja spajanjem:

  ulaz je lista $a_1, \ldots, a_n$, izlaz sortirana lista

  1. podijeli listu na dva jednaka dijela
  2. sortiraj listu $a_1, \ldots, a_{n/2}$
  3. sortiraj listu $a_{n/2+1}, \ldots, a_n$
  4. spoji liste $a_1, \ldots, a_{n/2}$ i $a_{n/2+1}, \ldots, a_n$

- Algoritam spajanja dvije sortirane liste u jednu sortiranu listu:

  ulaz su dvije sortirane liste $b_1, \ldots, b_k$ i $c_1, \ldots, c_l$, izlaz je sortirana lista $a_1, \ldots, a_{k+l}$

  1. $i = 1, j = 1$
  2. ponavljaj korak 3 sve dok je $i \leq k$ i $j \leq l$
  3. ako je $b_i < c_j$ onda $a_{i+j-1} = b_i$, $i=i+1$, inače $a_{i+j-1} = c_j$, $j=j+1$
  4. ponavljaj korak 5 sve dok je $i \leq k$
  5. $a_{i+j-1} = b_i$, $i=i+1$
  6. ponavljaj korak 7 sve dok je $j \leq l$
  7. $a_{i+j-1} = c_j$, $j=j+1$

- Složenost algoritma spajanja: korak 1. se obavlja jednom, za svaki element u listama izvršit će se jedan od blokova naredbi iz koraka 3 ili koraka 5 ili koraka 7, koji svi imaju istu konstantnu složenost koja se može odozgo ograničiti
- Najgori i najbolji slučaj zahtijevaju isti broj operacija, pa su njihove složenosti iste, kao i prosječna složenost:

$$T_{pros} \leq c1 + (k + l)*c_2 + (k + l)*c_3 = c1 + (k + l)(c_2 + c_3)$$

$$T_{pros} = O(k + l)$$

- Složenost algoritma sortiranja spajanjem: za sve instance problema, složenost je ista (najbolji slučaj = najgori slučaj = prosječni slučaj)

$$T_{pros}(n) = 2 * T_{pros}(n/2) + T_{pros}(\text{spajanje } n/2 + n/2) =$$

$$= 2 * T_{pros}(n/2) + d_1 * n + d_2$$

- Rekurzija: $t_n = 2 * t_{n/2} + d_1 * n + d_2$

- Uvodi se supstitucija:

$$s_n = t_{2^n}$$

$$s_n = t_{2^n} = 2 * t_{\frac{2^n}{2}} + d_1 * n + d_2 = 2 * t_{2^{n-1}} + d_1 * n + d_2 =$$

$$= 2 * s_{n-1} + d_1 * 2^n + d_2$$

- Oduzme se jdba za n-1 član od jdbe za n-ti član:

$$s_n = 3 * s_{n-1} - 2 * s_{n-2} + 2^{n-1} * d_1$$

- Oduzme se izraz za n-1 član pomnožen s dva od izraza za n-ti član:

$$s_n = 5 * s_{n-1} - 8 * s_{n-2} + 4 * s_{n-3}$$

- Dobivena je homogena rekurzivna jdba, čija karakteristična jdba

$$x^3 - 5 * x^2 + 8 * x - 4 = 0$$

ima dvostruki korijen $x_{1,2} = 2$ i $x_3 = 1$, pa je opće rješenje

$$s_n = C_1 * 2^n + C_2 * n * 2^n + C_3 * 1^n$$

- Iz čega slijedi:

$$t_n = C_1 * n + C_2 * n * \log_2 n + C_3$$

- Što znači da je prosječna složenost algoritma O(n*lg n)
- Nedostatak ovog algoritma sortiranja: potrebno je dodatno polje

Sortiranje pomoću hrpe

- Sortiranje pomoću hrpe (Heap Sort) se zasniva na svojstvima posebnog apstraktnog tipa podataka - hrpi
- Potpuno binarno stablo T je hrpa (Heap) ako su ispunjeni uvjeti:

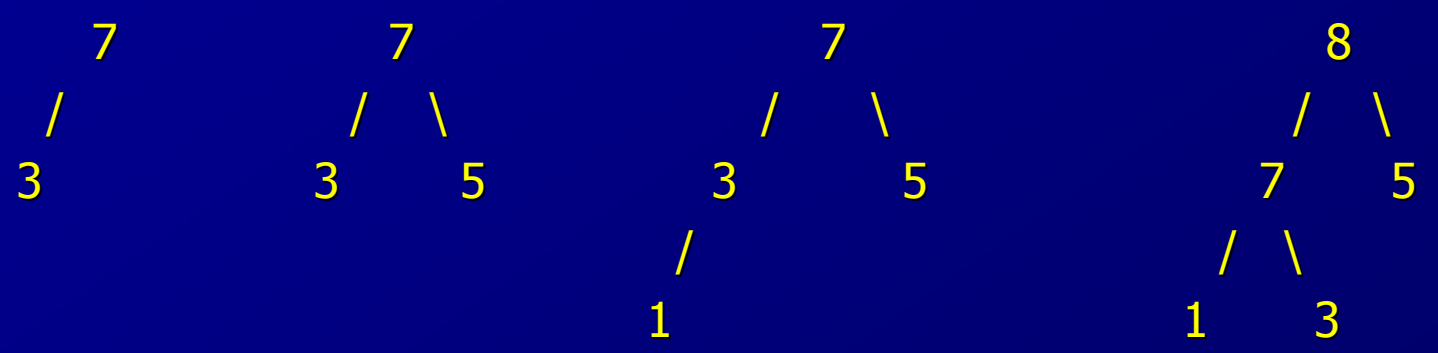
  - čvorovi od T su označeni podacima nekog tipa za koje je definiran totalni uređaj

  - neka je *i* bilo koji čvor od T. Tada je oznaka od *i* manja ili jednaka od oznake bilo kojeg djeteta od *i* – *minimalna hrpa*
- *ali može biti isto tako:*

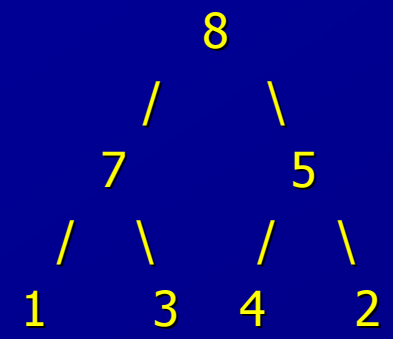
  - neka je *i* bilo koji čvor od T. Tada je oznaka od *i* veća ili jednaka od oznake bilo kojeg djeteta od *i* –*maksimalna hrpa*
- Uzlazno sortiranje koje koristi minimalnu hrpu zahtjeva dodatni memorijski prostor za spremanje sortirane liste, pa je bolje za taj slučaj upotrijebiti maksimalnu hrpu
- Analogno, za silazno sortiranje je bolje upotrijebiti minimalnu hrpu jer ne zahtjeva dodatni memorijski prostor za spremanje elemenata sortirane liste

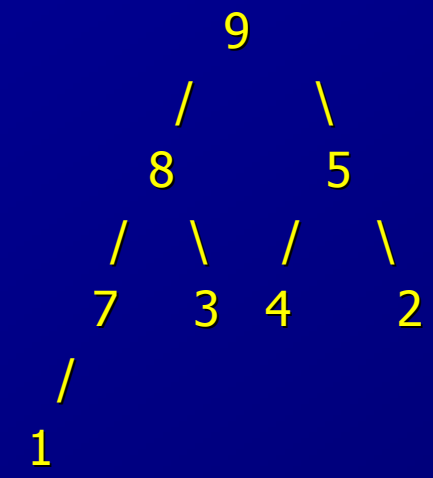
- Algoritam: od ulazne liste elemenata se kreira hrpa u kojoj je vrijednost roditelja veća od vrijednosti djece (maksimalna hrpa), pa će u njoj u korijenu biti najveći element
- Novi se element dodaje u najlijevije slobodno mjesto na posljednjoj razini stabla, te se uspoređuje njegova vrijednost s vrijednošću roditelja, ako je novi element veći, zamjenjuje mjesto s roditeljem. Zatim se uspoređuje vrijednost novog elementa koji je sad na pretposljednjoj razini s njegovim trenutnim roditeljem i ako je novi element veći, opet im se zamjenjuju mjesta, itd. sve dok se na nekoj razini ne nađe roditelj koji je veći od novog elementa ili novi element postane korijen stabla

- Primjer: 7 3 5 1 8 4 2 9 6




8
7
5
1
3
4


8
7
5
1
3
4
2


9
8
5
7
3
4
2
1


9
8
5
7
3
4
2
1
6

- Efikasna i jednostavna izvedba hrpe je ona pomoću polja. Ako se korijen stavi u element polja s indeksom 1 i elementi se slažu po razinama, tada će čvor hrpe na poziciji i imati lijevo dijete na $2i$ poziciji, a desno na $2i+1$
- Algoritam punjenja hrpe: ulazni podaci su hrpa H s elementima h_1, …, h_n i element x, a izlaz je hrpa H' s n+1 elemenata

  1. stavi x u polje na poziciju h[n+1], i=n+1
  2. dok je i > 1 i dok je h[i] > h[i//2] radi korak 3
  3. zamjeni(h[i],h[i//2]), i = i//2
- Složenost algoritma: broj elemenata u hrpi je n, jer je hrpa potpuno binarno stablo, njena visina je $1+\log_2 n$. Algoritam zamjenjuje element s elementom prethodne razine, pa se koraci 2 i 3 ponavljaju najviše $\log_2 n$ puta
- Slijedi da je složenost u najgorem slučaju

$$T_{max}(n) \leq c_1 + c_2 * \log_2 n = O(\lg n)$$

- Najbolji slučaj: dodani element odmah manji od svog roditelja:

$$T_{min}(n) \leq c_1 + c_2 = O(1)$$

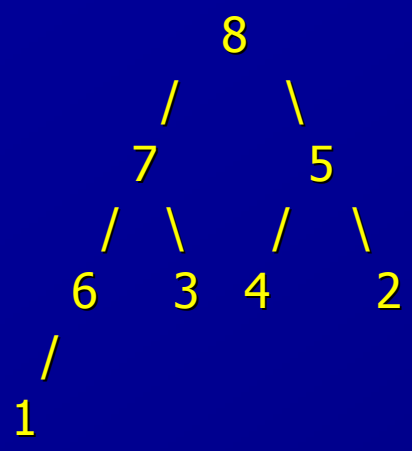
- Prosječan slučaj: pretpostavlja se da je jednako vjerojatno da će novododani element završiti na svakoj od razini stabla, tada je ta vjerojatnost jednaka $1/\log_2 n$, pa je složenost

$$T_{pros}(n) = c_1 + \sum_{i=1}^{\log_2 n} \frac{1}{\log_2 n} * i * c_2 = c_1 + \frac{1}{\log_2 n} * c_2 * \frac{\log_2 n * (\log_2 n + 1)}{2} =$$

$$= c_1 + \frac{c_2}{2} * (\log_2 n + 1) = O(\lg n)$$

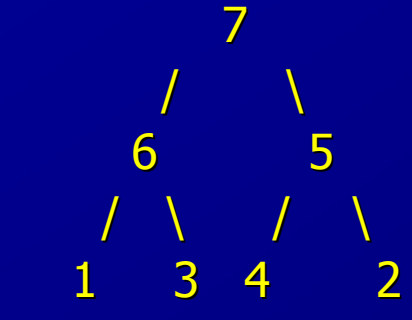
- algoritam sortiranja pomoću hrpe: korijen, koji je najveći element u hrpi, se zamjeni s posljednjim elementom u polju i broj elemenata hrpe se smanji za jedan, te se element koji je sad u korijenu uspoređuje i zamjenjuje, kad je potrebno, sa svojim potomcima sve dok se opet ne izgradi hrpa, a na posljednjoj poziciji u polju je zapisan najveći element
- Primjer: pražnjenje hrpe iz prethodnog primjera

9 ide na zadnje mjesto, 6 u korijen, preuredi se u hrpu



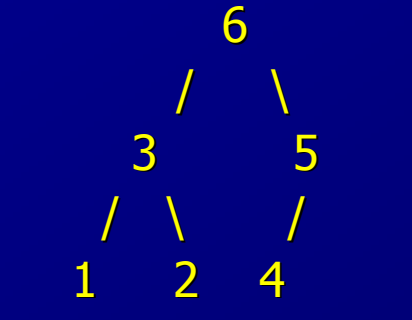
9

8 ide na predzadnje mjesto, 1 u korijen, preuredi se u hrpu



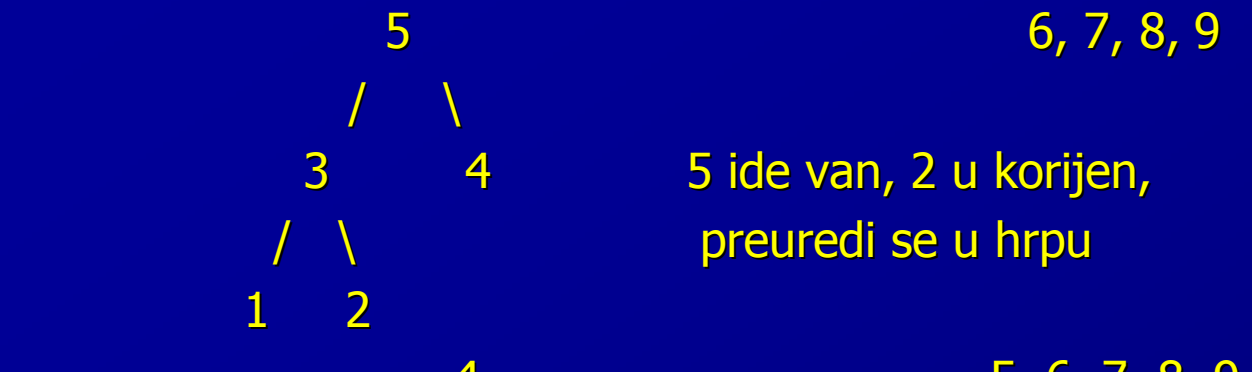
8, 9

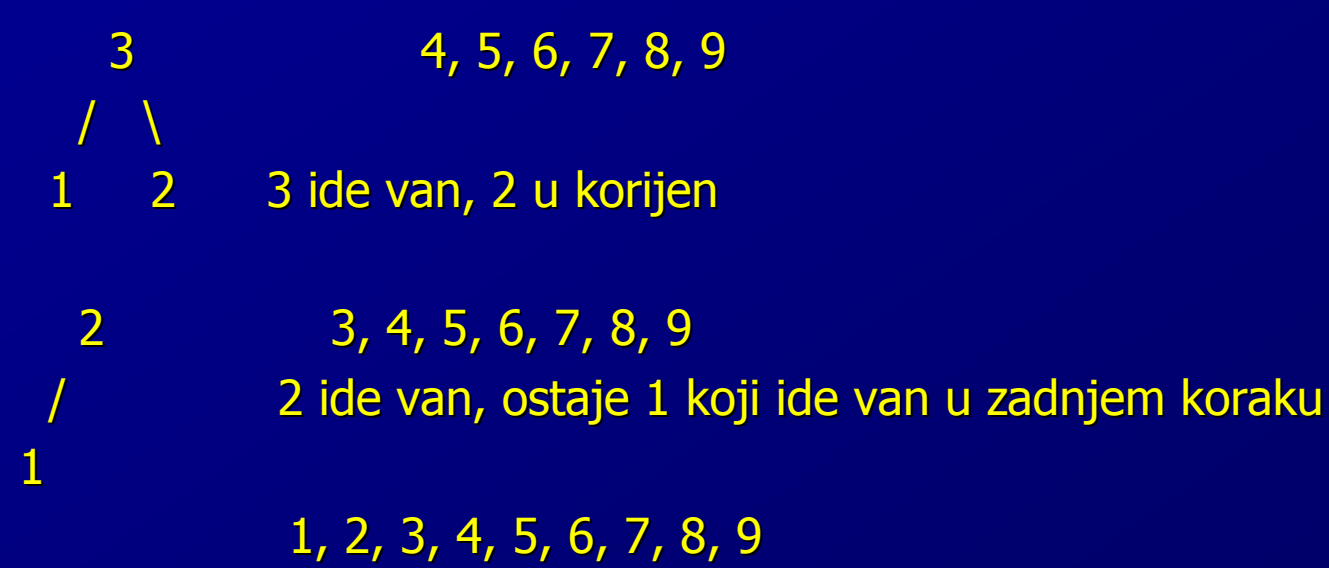
7 ide van, 2 u korijen, preuredi se u hrpu



7, 8, 9

6 ide van, 4 u korijen, preuredi se u hrpu


5
6, 7, 8, 9
3
4
5 ide van, 2 u korijen,
preuredi se u hrpu
1
2
4
5, 6, 7, 8, 9
3
2
4 ide van, 1 u korijen,
preuredi se u hrpu
1


3
4, 5, 6, 7, 8, 9
1
2
3 ide van, 2 u korijen
2
3, 4, 5, 6, 7, 8, 9
2 ide van, ostaje 1 koji ide van u zadnjem koraku
1
1, 2, 3, 4, 5, 6, 7, 8, 9

- Algoritam pražnjenja hrpe: ulaz je hrpa H s elementima h_1, …, h_n, izlaz je hrpa H' dobivena od H izbacivanjem korijena

  1. zamjeni(h[1],h[n]), izbaci h[n], i=1
  2. radi korake 3 i 4 sve dok je $2*i+1 \leq n$ i $h[i] < \max(h[2*i],h[2*i+1])$
  3. ako je $h[2*i] > h[2*i+1]$ zamjeni(h[i],h[$2*i$]), i=$2*i$
  4. inače zamjeni(h[i],h[$2*i$+1]), i=$2*i$+1

- Složenost algoritma: algoritam mijenja promatrani element s onim na sljedećoj razini, pa se koraci 2 – 4 mogu ponoviti najviše $\log_2 n$ puta

$$T_{max}(n) \leq c_1 + c_2 \log_2 n = O(\lg n)$$

- U najboljem slučaju je prebačeni element dobar korijen, pa je

$$T_{min}(n) = c_1 + c_2 = O(1)$$

- Prosječan slučaj: jednako je vjerojatno da će prebačeni element završiti na bilo kojoj razini, pa je rezultat isti kao kod punjenja hrpe

$$T_{pros}(n) = O(\lg n)$$

- Potpuni algoritam sortiranja pomoću hrpe: ulaz je lista a_1, …, a_n, izlaz je sortirana lista
  1. za i = 1, …, n radi korak 2
  2. zovi algoritam punjenja hrpe za element a_i
  3. za i = 1, …, n radi korak 4
  4. zovi algoritam pražnjenja hrpe
- Složenost algoritma: za n elemenata u listi, svi koraci se izvršavaju n puta, složenosti za korake 2 i 4 su određene ranije

$$T_{\max}(n) = c_1 + \sum_{i=1}^{n} T_{\max}^{punjenje}(i) + \sum_{u=1}^{n} T_{\max}^{pražnjenje}(i) \leq$$

$$\leq c_1 + \sum_{i=1}^{n} (d_1 + d_2 * \log_2 i) + \sum_{i=1}^{n} (e_1 + e_2 * \log_2 i) =$$

$$= c_1 + d_1 * n + n * \log_2 n * d_2 + e_1 * n + n * \log_2 n * e_2 =$$

$$c_1 + (d_1 + d_2) * n + (e_1 + e_2) * n * \log_2 n = O(n * \lg n)$$

- Složenost za prosječni slučaj za ubacivanje i pražnjenje hrpe su iste kao za najgori slučaj, pa je i ovdje prosječna složenost O(n*lg n)

- Složenost za najbolji slučaj

$$T_{\min}(n) = c_1 + \sum_{i=1}^{n} T_{\min}^{pražnjenje}(i) + \sum_{i=1}^{n} T_{\min}^{punjenje}(i) \leq c_1 + \sum_{i=1}^{n} d_1 + \sum_{i=1}^{n} e_1 =$$

$$= c_1 + (d_1 + e_1) * n = O(n)$$

Algoritam brzog sortiranja

- Algoritam brzog sortiranja (Quick Sort) je najbrži algoritam za sortiranje, koji u prosjeku treba Θ(n lg n) operacija uspoređivanja za sortiranje niza duljine n
- To je algoritam strategije podijeli-pa-vladaj, koji dijeli niz u dva podniza tako da izabere poseban element pivot (stožer) i onda preuredi niz tako da se svi elementi manji od pivota prebace na pozicije prije njega, a svi veći se nalaze iza pivot elementa, nakon ovog koraka pivot se nalazi na mjestu na kojem mora biti u sortiranom nizu
- Postupak se rekurzivno ponavlja za ta dva podniza (manji i veći od pivota), sve dok se ne dođe do podniza od 1 elementa
- Algoritam brzog sortiranja uspoređuje elemente u nizu pa spada u algoritme sortiranja uspoređivanjem
- rezultat je pokušaja ubrzavanja faze spajanja u algoritmu sortiranja spajanjem (Merge Sort); ovdje je spajanje u potpunosti izbjegnuto, jer nakon što su elementi u podnizovima sortirani, zbog uređenosti polja, svi su elementi iz drugog podniza veći od svakog elementa iz prvog podniza
- Izvedba algoritma ovisi o izboru pivot (stožer) elementa: najjednostavnija varijanta uzima prvi element za pivot, no on se može odrediti na bilo koji način koji se može izračunati u O(1) vremenu

- Razmotrimo slučaj kad je pivot prvi element u nizu. Za premještanje elemenata niza potrebna su dva kursora, prvi je iniciran na drugi element u nizu i raste, a drugi na zadnji i pada
- Kreće se s prvim kursorom koji se pomiče sve dok se ne nađe element veći od pivota, tada se kreće od drugog kursora koji se pomiče dok se ne nađe element manji od kursora
- Zamijene se mjesta ta dva elementa i nastavlja se traženje od prvog kursora
- Pretraživanje završava kad je prvi kursor iza drugoga, tada drugi kursor pokazuje na poziciju na kojoj će u sortiranom nizu biti pivot element, pa se zamijene mjesta tog elementa i pivot elementa
- Nakon toga se ponavlja postupak na dva podniza: za elemente prije pivot elementa i one iza pivot elementa
- Primjer: 7 3 5 1 8 4 2 9 6 , pivot je 7
- Kreće se od početka, prvi element veći od 7 je 8, sa stražnje strane prvi manji je 6, pa im se zamijene mjesta:

  7 3 5 1 6 4 2 9 8
- Sljedeći element veći od 7 je 9, odostraga prvi manji je 2. No drugi kursor je ispred prvog pa ovaj korak završava. Niz se dijeli na dva dijela, od početka do drugog kursora, te od prvog kursora do kraja

7 3 5 1 6 4 2 || 9 8

- Pivot i element na koji pokazuje drugi kursor zamijene mjesta, kako su svi elementi u prvoj listi manji od njega, on je na pravom mjestu i više ga se ne dira

  2 3 5 1 6 4 || 7 || 9 8

- Ponavljanje postupka na prvom podnizu: pivot je 2, sprijeda prvi veći od njega je 3, straga prvi manji je 1

  2 1 5 3 6 4 || 7 || 9 8

- Nastavljamo: sprijeda 5, straga 1, ali je prvi kursor iza drugoga, pa se mijenjaju pivot i element na koji pokazuje drugi kursor

  1 || 2 || 5 3 6 4 || 7 || 9 8

- Drugi podniz (zadnja 2 elementa): pivot je 9, prvi kursor ne nalazi veći od njega nego prelazi drugi kursor, pa 9 i 8 mijenjaju mjesta:

  1 || 2 || 5 3 6 4 || 7 || 8 || 9

- Jedini preostao niz je u sredini: pivot je 5, sprijeda nalazimo 6, straga 4

  1 || 2 || 5 3 4 6 || 7 || 8 || 9

- Sprijeda nalazimo 6 i sad je prvi kursor iza drugog: 5 i 4 zamijene mjesta

1 || 2 || 4 3 || 5 || 6 || 7 || 8 || 9

- Zadnji korak: niz od 2 elementa, pivot je 4, pretraga sprijeda ne nalazi veći od njega nego se prvi kursor pomiče iza drugog:

  1 || 2 || 3 || 4 || 5 || 6 || 7 || 8 || 9

- Algoritam brzog sortiranja: ulaz je lista a_1, … a_n i kursori *i* (početak) i *j* (kraj),a izlaz sortirana lista

  1. k = i+1, l = j
  2. sve dok je k <= l radi korake 3 – 5
  3. dok je (a_i >= a_k) k = k+1
  4. dok je (a_i <= a_l) l = l-1
  5. ako je k < l, zamjeni(a_k, a_l)
  6. ako je l > i zamjeni(a_i, a_l)
  7. ako je l > i quicksort(a, i, l-1)
  8. ako je k < j quicksort(a, k, j)

- Najgori slučaj za ovaj algoritam: već sortirana lista i lista sortirana obrnutim rasporedom. Tada se u svakom koraku lista dijeli na podliste od jednog i n-1 elemenata

- Korak 1. ima konstantnu složenost, koraci od 2. do 5. (za niz duljine n) imaju složenost $c_1 + c_2 * n$, a korak 6. ima konstantnu složenost

$$T_{max}(n) = T_{max}(n-1) + d_1 * n + d_2$$

- rekurzija

$$t_n = t_{n-1} + d_1 * n + d_2$$

- Oduzme se izraz za n-1 član od izraza za n-ti član i sredi

$$t_n = 2 * t_{n-1} - t_{n-2} + d_1$$

- Sada se opet oduzme jdba za n-1 član od jdbe za n-ti član i izlazi

$$t_n = 3 * t_{n-1} - 3 * t_{n-2} + t_{n-3}$$

- Karakteristična jdba

$$x^3 - 3 * x^2 + 3 * x - 1 = 0$$

- Jedan trostruki korijen $x_{1,2,3} = 1$, pa je opće rješenje rekurzije

$$t_n = C_1 * 1^n + C_2 * n * 1^n + C_3 * n^2 * 1^n$$

$$t_n = C_1 + C_2 * n + C_3 * n^2$$

- Dakle, u najgorem slučaju, složenost je $O(n^2)$

- Najbolji slučaj: kada je izabrani pivot srednja vrijednost elemenata dijela liste koji se promatra, pa se lista dijeli na dva dijela jednake duljine

$$T_{min}(n) \leq 2 * T_{min}(n/2) + d_1 * n + d_2$$

- Tj. rekurzija koja se rješava je

$$t_n \leq 2 * t_{n/2} + d_1 * n + d_2$$

- Uvodi se supstitucija

$$s_n = t_{2^n}$$

$$s_n = t_{2^n} = 2 * t_{\frac{2^n}{2}} + d_1 * 2^n + d_2 = 2 * t_{2^{n-1}} + d_1 * 2^n + d_2 =$$

$$= 2 * s_{n-1} + d_1 * 2^n + d_2$$

- Izraz za n-1 član se oduzme od izraza za n-ti član

$$s_n = 3 * s_{n-1} - 2 * s_{n-2} + d_1 * 2^{n-1}$$

- Od ovog izraza se oduzme izraz za n-1 član pomnožen s 2

$$s_n = 5 * s_{n-1} - 8 * s_{n-2} + 4 * s_{n-3}$$

- Karakteristična jdba $x^3 - 5 * x^2 + 8 * x - 4 = 0$ ima jednostruki korijen $x_1 = 1$ i dvostruki korijen $x_{2,3} = 2$, pa je opće rješenje rekurzije

$$s_n = C_1 * 1^n + C_2 * 2^n + C_3 * n * 2^n$$

$$s_n = C_1 + C_2 * 2^n + C_3 * n * 2^n$$

- Povratkom iz supstitucije izlazi

$$t_n = C_1 + C_2 * n + C_3 * n * \log_2 n$$

- Pa je dakle složenost u najboljem slučaju O(n*lg n)
- Složenost prosječnog slučaja: pretpostavlja se da je svaki raspored elemenata liste jednako vjerojatan. Brojat ćemo samo operacije uspoređivanja nad elementima polja (broj svih ostalih operacija ovisi o broju tih operacija, pa one određuju složenost algoritma). Također se pretpostavlja da su svi elementi u polju različiti, te da je jednaka vjerojatnost izbora svakog elementa za pivota
- Gleda se segment liste $a_m, \ldots, a_p$, kako su za izbor pivot elementa svi elementi iz segmenta jednako vjerojatni, prolaz algoritma uz izbor nekog a_i za pivota dijeli segment na dva dijela $a_m, \ldots, a_{m+i-2}$ i $a_{m+i}, \ldots, a_p$ s vjerojatnošću $1/(p-m+1)$
- Složenost u prosječnom slučaju (algoritam radi n+1 koraka prije rekurzivnog poziva) je dakle

$$T_{pros}(n) = n + 1 + \frac{1}{n} * \sum_{i=1}^{n} (T_{pros}(i-1) + T_{pros}(n-i))$$

- Rješava se rekurzija

$$t_n = \frac{1}{n} * \sum_{i=1}^{n} (t_{i-1} + t_{n-i}) + n + 1$$

- Kad se jdba pomnoži s n i srede dvije sume, slijedi

$$n * t_n = 2 * \sum_{i=0}^{n-1} t_i + n * (n+1)$$

- Od gornjeg izraza se oduzme izraz za n-1, nakon sređivanja slijedi

$$n * t_n = (n + 1) * t_{n-1} + 2 * n$$

- Dijeljenjem izraza s n(n+1) slijedi

$$\frac{t_n}{n+1} = \frac{t_{n-1}}{n} + \frac{2}{n+1}$$

- Rekurzivnim uvrštavanjem ovog izraza slijedi izraz

$$\frac{t_n}{n+1} = \frac{t_1}{2} + 2 * \sum_{i=3}^{n+1} \frac{1}{i}$$

- Poznato je da vrijedi:

$$\sum_{i=3}^{n+1} \frac{1}{i} \leq \ln(n+1) - \ln 2$$

- Uvrštavanjem i sređivanjem slijedi

$$t_n \leq 2 * (n + 1) * [\ln(n+1) - \ln 2 + k]$$

tj.

$$T_{pros} = O(n * \lg n)$$

- ovaj algoritam je stvarno efikasan i isplatljiv za sortiranje dugačkih nizova elemenata i uz pažljiv izbor pivot elementa
- Izvedba algoritma brzog sortiranja je znatno složenija od recimo sortiranja umetanjem, što znači da su konstante koje se javljaju u izrazu vremenske složenosti znatno veće od konstanti za sortiranje umetanjem. Zbog toga će za kratke nizove sortiranje umetanjem biti brže