**STRUKTURE PODATAKA**

**I ALGORITMI 2**

ODGOVORI NA PITANJA ZA

USMENI DEO ISPITA

JANUAR 2021

Git: <https://github.com/NikolaVetnic/SPA2>

Sortiranje niza grubom silom i umetanjem.

01

**SORTIRANJE NIZOVA:** sortiranje po prirodnom uređenju koje postoji kada objekti implementiraju *Comparable* interfejs, sortiranje po eksternom kriterijumu koji je definisan komparatorom.

**SORTIRANJE GRUBOM SILOM:** dva elementa a[i] i a[j] su u inverziji ako je i < j i a[i] > a[j], pa se sortiranje grubom silom (brute force) svodi na poređenje svaka dva elementa ponaosob i njihovu razmenu ukoliko su u inverziji. Ukupan broj poređenja za elemenata je n elemenata je n(n – 1)/2 pa je vremenska složenost O(n2) – u najgorem slučaju operacija je kvadratne složenosti veličine ulaza.

<https://github.com/NikolaVetnic/SPA2/tree/master/src/p01_elementary_sorts>

**SORTIRANJE UMETANJEM:** ideja je da se niz sastoji od sortiranog (sa leve strane) i nesortiranog dela, gde se prvi element iz nesortiranog ubacuje na odgovarajuće mesto u sortiranom (čime se sortirani uvećava za element), što se ponavlja dok se ne sortira čitav niz; početak je od prvog elementa koji predstavlja sortiran, a ostatak nesortiran deo. Postupak: neka je deo niza [0, i-1] sortiran – u svakom koraku se povećava za jedan, pa se i kreće u [1, arr.length – 1]; prvi nesortirani element je arr[i]; ukoliko je arr[i] > arr[i-1] ne radimo ništa budući da je on zapravo već sortiran, u suprotnom tražimo indeks j < i takav da je arr[j] ≤ arr[i] – arr[i] tada dolazi na poziciju j+1, elemente od arr[j+1] do arr[i-1] pomeramo za jedno mesto nadesno (što se vrši dok tražimo indeks j), pri čemu čuvamo referencu na arr[i] (jer će ga u pomeranju prebrisati arr[i-1]).

<https://github.com/NikolaVetnic/SPA2/tree/master/src/p01_elementary_sorts>

Sortiranje niza izabiranjem i razmenom.

02

**SORTIRANJE IZABIRANJEM:** niz je sastavljen iz dva dela – sortiranog (sa desne strane) i nesortiranog, čime su svi elementi iz nesortiranog manji od svih iz sortiranog dela niza; traži se maksimalni element nesortiranog dela postavlja se na njegov kraj. Druga varijanta sortiranja deli niz na sortirani deo sa leve i nesortirani sa desne strane; traži se minimum desne strane i postavlja na početak nesortiranog dela niza.

<https://github.com/NikolaVetnic/SPA2/tree/master/src/p01_elementary_sorts>

**STABILNA I NESTABILNA SORTIRANJA:** postupak sortiranja je stabilan ako za svako i i j, takvo da je a[i] = a[j] u polaznom nizu, element a[i] u sortiranom nizu bude pre a[j] – relativni redosled identičnih elemenata biva očuvan kod stabilnog sortiranja. Po *default*-u sortiranje umetanjem je stabilno a izabiranjem nije (iako se može napraviti da bude). Stabilna sortiranja dozvoljavaju sortiranje po dva kriterijuma, i to prvo po sekundarnom, a onda po primarnom – sortiranje po sekundarnom ostaje očuvano zahvaljujući stabilnosti sortiranja.

**SORTIRANJE RAZMENOM:** zasniva se na ideji da će, ako svaka dva susedna elementa koja su u inverziji zamene mesta, maksimalni element „isplivati“ na kraj niza (kraj nesortiranog dela varira od n – 1 kada je nesortirani deo čitav niz, do 1 kada nesortirani deo niza čine dva elementa); inverzni postupak ima analogni efekat na minimalni element (početak nesortiranog dela varira od 0 do n – 2). Sortiranje se može unaprediti tako što se sortiranje prekine posle iteracije u kojoj nije bilo razmene elemenata.

<https://github.com/NikolaVetnic/SPA2/tree/master/src/p01_elementary_sorts>

Sortiranje niza *Shell sort*-om.

03

**SHELL SORT:** (*Donald Shell*, 1959) osnovna ideja – 1) ukupan broj parova u inverziji se može posmatrati kao mera neuređenosti niza, 2) kod elementarnih sortiranja pri svakom poređenju susednih elemenata ukupan broj inverzija se smanjuje za jedan, 3) razmena nesusednih elemenata u nizu može smanjiti više od jedne inverzije. Niz A dužine n se deli na podnizove A0, A1… Ak-1 za neko k na sledeći način: A0 = (0, k, 2k…), A1 = (1, k+1, 2k+1…) … Ak-1 = (k-1, 2k-1…) – drugim rečima, podniz Aj sadrži indekse koji pri deljenju sa k daju ostatak j; podnizovi se sortiraju umetanjem; postupak se ponavlja za vrednosti k koje se smanjuju od neke početne vrednosti do 1 sa nekim korakom; kada je k = 1 dešava se sortiranje umetanjem nad celim nizom.

Kako varirati k – 1) sam *Shell* predložio je vrednosti , … … 1, gde je n veličina niza (*Shell sort* se značajno ubrzava kada k varira tako da bude stepen broja 2 uvećan ili umanjen za 1), 2) (*Tokuda*, 1992) sortiranje daje najbolje performanse kada se k sukcesivno smanjuje 2.25 puta; 3) (*Ciura*, 2001) sortiranje daje najbolje performanse kada k uzima redom vrednosti { 701, 301, 132, 57, 23, 10, 4, 1 }.

<https://github.com/NikolaVetnic/SPA2/tree/master/src/p01_elementary_sorts>

Sortiranje niza *Comb sort*-om.

04

**COMB SORT:** zasniva se na razmeni nesusednih elemenata u nizu, predstavlja unapređenje *Bubble sort*-a (sortiranje razmenom); kod sortiranja razmenom veći elementi „brzo prilaze“ kraju niza, dok manji „sporo prilaze“ početku niza – osnovna ideja je eliminisati „efekat kornjača“ („sporih“ elemenata) razmenom nesusednih elemenata u nizu.

Algoritam: 1) za svaki element proveravamo da li je u inverziji sa elementom udaljenim k pozicja udesno i ako jeste razmenimo ih, 2) ponavljamo *Comb* prolaz smanjujući k, 3) niz je sortiran kada je k = 1 i nije bilo razmena elemenata. Drugim rečima, *Comb* prolaz se za k = 1 ponavlja sve dok postoje elementi u inverziji što je zapravo *Bubble sort*.

Kako varirati k – eksperimentalno je utvrđeno da *Comb sort* daje najbolje performanse kada se k sukcesivno smanjuje 1.3 puta, ali se preskoče k = 9 i k = 10 i uzme se k = 11.

<https://github.com/NikolaVetnic/SPA2/tree/master/src/p01_elementary_sorts>

Sortiranje niza *Heap sort*-om.

05

**HEAP SORT:** unapređenje sortiranja izabiranjem linearitamske složenosti (O(n logn)).

Algoritam: 1) neka je A niz dužine n, 2) transformišemo A tako da ime strukturu *heap*-a (maksimum je tada prvi u nizu), 3) razmenimo maksimum sa elementom na kraju niza, 4) povratimo strukturu *heap*-a na podnizu A[0 .. n-1] i maksimum zamenimo sa elementom na poz. n – 2, 5) ponavljamo postupak do podniza A[0 .. 1] kada maksimum zamenimo sa susednim elementom.

*Heap* osobina: niz ima *heap* osobinu ukoliko je svaki element veći ili jednak svojim sinovima, gde sinovi roditelja indeksa p imaju indekse 2p + 1 i 2p + 2, a roditelj sina indeksa s ima indeks (s – 1)/2 (celobrojno deljenje); kod niza dužine n elementi sa idx u opsegu [0, (n – 2)/2] su očevi. Počevši od poslednjeg ka prvom ocu uspostavljamo *heap* strukturu tako što: 1) proverimo da li je otac manji od većeg sina, 2) ako jeste, zamenimo oca sa većim sinom, 3) ponavljamo **1)** i **2)** dokle god otac ne postane veći od sinova.

Sortiranje niza *Heap sort*-om.

05

**HEAP SORT:** unapređenje sortiranja izabiranjem linearitamske složenosti (O(n logn)).