Quicksort

Quicksort[1] je jako zanimljiv algoritam za sortiranje niza. Verovatno ste svi do sada čuli da je to najbrži algoritam za sortiranje niza. U prvom delu ovog dokumenta smo razmatrali složenost jednoj od najjednostavnijih algoritama za sortiranje. Složenost quicksort-a, u najgorem slučaju, je $O(n^2)$ gde je n broj elemenata niza. Iako je spor u najgorem slučaju, on predstavlja najbolji izbor za ovaj problem jer je neverovatno brz u srednjem tj. očekivanom vremenu sortiranja: očekivano vreme sortiranje je $O(n \log n)$, dok je konstanta sakrivena iz ove složenosti jako mala.

Sa druge strane, pored ovog algoritma postoji i **sortiranje razdvajanje** koje se zanisniva na metodi **podeli pa vladaj**. Ovaj algoritam je u stranoj literaturi poznat kao **merge sort**. Složenost je $O(n \log n)$ u najgorem slučaju. Međutim, u očekivanom vremenu quicksort je dosta brži. Sličan slučaj imamo i sa **radix sort-om**.

Kako bi bili sigurni da će quicksort raditi u očekivanom vremenu, pre sortiranja niza treba napraviti slučajnu permutaciju a zatim nju sortirati. Na taj način možete izbeći one najgore slučajeve i vreme vratiti na očekivano.

Kako možemo kreirati slučajnu permutaciju? Označimo dati niz sa a, njegovu dužinu sa n. Naintuitivniji pristup je dodeliti svakom elementu niza slučajan broj $b\,[i]$, a zatim sortirati niz a ali po vrednostima niza b. Problem kod ovog metoda je kako obezbediti da su dodeljene vrednosti jedinstvene. Kako se i sami slučajni brojevi generišu pseudo-slučajnim algoritmom ovde se možemo zadovoljiti i jako malom verovatnoćom pojavljivanje istih vrednosti. Može se pokazati da ukoliko vrednosti biramo na slučajan način iz segmenta $[1, n^3]$, verovatnoća da su svi elementi različiti jednaka je $1 - \frac{1}{n}$. Složenost ovog pristupa je jednak složenosti sortiranje.

Najpraktičniji način generisanja slučajne permutacijije prikazan je u *Algoritmu 6*. Složenost algoritma je linearna. Dokaz korektnosti algoritma nećemo izložiti ovde. Za one koji žele više informacija, mogu pogledati [1].

```
=================================================================
01    for i = 1 to n do
02        j = random (i, n);
03        zameni (a [i], a [j]);
=================================================================
```

Algoritam 9

Zanimljiva napomena je da ukoliko se indeks j inicijalizuje kao $random(1, i)$ ne dobija se slučajna permutacija. Dokaz ove činjenice ostavljamo čitaocu.

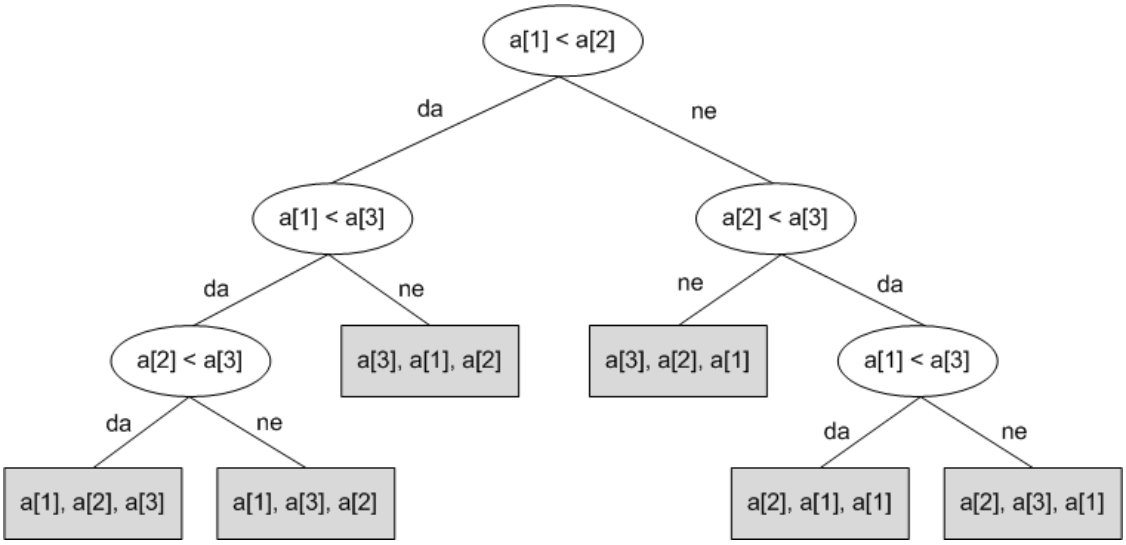
Priču o quicksort-u završićemo dokazom njegove optimalnosti. Do sada samo uvek računali složenost algoritama i pokazivali kako je neki pristup bolji od drugog. Međutim, nekada je potrebno postaviti

[1] Ovde nećemo objašnjavati kako sam algoritam radi. Opis algoritma možete naći skoro u bilo kojoj knjizi koja se bavi programiranjem.

pitanje: da li postoji brži algoritam? Kako bi odgovorili na ovo pitanje potrebno je naći donju granicu složenosti algoritma za konkretan problem. Ovo je nemoguće odrediti za generalni slučaj, jer se dokaz odnosi na sve moguće algoritme. Neohpodno je napre specificirati model koji odgovara porizvoljnom algoritmu, a zatim dokazati da je vremenska složenost proizvoljnog algoritma obuhvaćenog tim modelom veća ili jednaka od donje granice.

U daljem delu odeljka pokazaćemo da je optimalna složenost algoritama za sortiranje niza upravo $O(n \log n)$. Model koji ćemo koristiti pri dokazivanju se zove **stablo odlučivanja**. Stablo odlučivanja modelira izračunavanje koje se najvećim delom zasniva na upoređivanju. Definiše se kao potpuno binarno stablo sa dve vrste čvorova: unutrašnjim čvorovima i listovima. Svakom unutrašnjem čvoru odgovara jedno "pitanje" sa dva moguća odgovora, pri čemu je svaki od njih pridružen grani koja izlazi iz tog čvora. Svakom listu je pridružen jedan od mogućih izlaza algoritma. Izračunavanje počinje od korena stabla. U svakom čvoru se postavlja pitanje u vezi sa ulazom i u zavisnosti od odgovora nastavlja levim ili desnim potomkom. Kada se dostigne list, izlaz pridužen njemu je izlaz izračunavanja. Kako bi ova priča bila jasnija pogledati *Sliku 4* na kojoj je prikazano sortiranje niza dužine 3 pomoću stabla odlučivanja.

Vremenska složenost algoritma definisanog preko stabla odlučivanja jednaka je njegovoj visini, odnosno maksimalnom broju pitanja (operacija) koje je potrebno postaviti kako bi se rešenje jednoznačno odredilo.



Slika 5. Primer stabla odlučivanja za sortiranje niza a dužine $n = 3$

Teorema Proizvoljno stablo odlučivanja za sortiranje n elemenata niza ima visinu $n \log n$.

Dokaz: Ulaz u algoritam je niz a dužine n. Izlaz je niz u sortiranom poretku, drugim rečima permutacija ulaza. Kako ulaz može biti u porivoljnom poretku, svaka permutacija brojeva $(1,2, \dots, n)$ treba da se pojavi kao mogući izlaz stabla odlučivanja. Prema tome, za svaku permutaciju treba da postoji bar jedan list. Permutacija ima $n!$, pa pošto je stablo binarno, njegova visina je najmanje $\log_2(n!)$. Koristeći Stirlingovu formulu

$$n! \approx \sqrt{2\,\pi\,n}\left(\frac{n}{e}\right)^n$$

dobijamo da je $\log_2(n!) \approx n \log n$. □

Ovaj dokaz donje granice pokazuje da svaki algoritam za sortiranje zasnovan na upoređivanju ima složenost veću ili jednaku od $n \log n$. Sortiranje je moguće izvršiti i korišćenjem nekih drugih osobina. Primera radi ukoliko bi ograničenje brojeve bilo malo (malo u smislu da možemo definisati niz te veličine**), sortiranje razvrstanjem (eng. counting sort)** bi vratilo rezultat u linearnom vremenu. Ovo ne protivureči dokazu prethodne teoreme, jer se ovde koristi činjenica da vrednosti brojeva mogu da se efikasno koriste kao adrese. Takođe, ovaj algoritam nije generalni i ne može se primeniti na proizvoljan niz.