RI203
Uvod u računarske algoritme

dr.sc. Edin Pjanić

Pregled predavanja

- Pretraživanje
  - sekvencijalno pretraživanje
    - u nesortiranim podacima
    - u sortiranim podacima
  - binarno pretraživanje
  - interpolacijsko pretraživanje

Pretraživanje i sortiranje

- Računari se vrlo često koriste za rješavanje problema tipa:
  - Traženja određenog podatka u skupu sačinjenog od velikog broja podataka
  - Postavljanje podataka u određeni poredak – operacija koju nazivamo sortiranje
- Razvijeni su razni algoritmi pretraživanja i sortiranja (generalizovani i specijalizirani).

Pretraživanje

- **Pretraživanje** (traženje) - lociranje elementa unutar liste ili neke druge strukture podataka (niz, vektor i sl.)
- Razlozi za pretraživanje:
  - utvrđivanje da li je element član liste
  - dobijanje svih podataka o elementu
  - ako je lista posebno organizovana (npr. sortirana), pronalaženje pozicije za umetanje novog elementa
  - pronalaženje lokacije elementa za brisanje
- Dva najpoznatija i najkorištenija metoda pretraživanja:
  - sekvencijalno (linearno) pretraživanje,
  - binarno pretraživanje.

Sekvencijalno pretraživanje – nesortirana lista

- **Ključ** – član elementa liste koji jedinstveno određuje taj element u listi (npr. šifra, JMBG, br. indeksa, reg. broj auta itd.)
- **Sekvencijalno (linearno) pretraživanje** – algoritam za traženje elementa sa zadanim ključem unutar nesortirane liste:

1. Uzmemo prvi element
2. Sve dok je ključ trenutnog elementa različit od traženog ključa i ako nismo došli do kraja liste:
   - uzmi naredni element,
   - ponovi pod 2.

- Kako znamo da nema traženog elementa u listi?

Sekvencijalno pretraživanje - primjer

- Neka lista sadrži ključeve:

  4 22 34 11 2 7 55

- Tražimo vrijednost 7:
  - sekvencijalno provjeravamo 4, 22, 34, 11, 2 i 7
- Tražimo vrijednost 34:
  - provjeravamo 4, 22 i 34
- Tražimo vrijednost 76:
  - provjeravamo 4, 22, 34, 11, 2, 7 i 55

- Kako znamo da nema traženog elementa u listi?

Sekvencijalno pretraživanje – **sortirana** lista

- **Sekvencijalno (linearno) pretraživanje** – algoritam za traženje elementa unutar **sortirane** liste sa zadanim ključem (pretpostavimo da je lista sortirana od "manjeg" ka "većem"):

    1. Uzmemo prvi element
    2. Sve dok je ključ trenutnog elementa manji od traženog ključa i ako nismo došli do kraja liste:
        - uzmi naredni element,
        - ponovi pod 2.

- Kako znamo da nema traženog elementa u listi?
- Ako je lista sortirana: Koja nam je korist od toga?

Sekvencijalno pretraživanje u sortiranoj listi - primjer

- Neka je lista sortirana i sadrži ključeve:

  2 4 7 11 22 34 55

- Tražimo vrijednost 7:
  - sekvencijalno provjeravamo 2, 4 i 7
- Tražimo vrijednost 34:
  - provjeravamo 2, 4, 7, 11, 22 i 34
- Tražimo vrijednost 10:
  - provjeravamo 2, 4, 7, 11
- Tražimo vrijednost 76:
  - provjeravamo 2, 4, 7, 11, 22, 34 i 55

- Kako znamo da nema traženog elementa u listi?

Sortirana ili nesortirana lista

- Uporedimo brzinu pretraživanja na primjeru nesortirane liste telefonskog imenika sa listom koja sadrži iste podatke ali sortirane po abecednom redu!
- Kako bi teklo i koliko bi trajalo traženje imena Abazović Adnan?

- Nesortirana lista:
  - ako je Abazović Adnan u listi?
  - ako Abazović Adnan nije u listi?

- Sortirana lista:
  - ako je Abazović Adnan u listi?
  - ako Abazović Adnan nije u listi?

Sortirana ili nesortirana lista

- Kako bi teklo i koliko bi trajalo traženje imena Žunić Željko?

- Nesortirana lista:
  - ako je Žunić Željko u listi?
  - ako Žunić Željko nije u listi?

- Sortirana lista:
  - ako je Žunić Željko u listi?
  - ako Žunić Željko nije u listi?

- Kako bi teklo traženje imena Lakić Alma?

Sortirana ili nesortirana lista

- Zapažanja:
  - pretraživanje je brže u sortiranoj listi samo ako se tražena vrijednost ne nalazi u listi
- Osim toga, da bismo imali tu malu prednost, podaci se moraju prethodno sortirati

- Zaključak:
  - efikasnost pretraživanja sortirane i nesortirane liste je približno ista

Sekvencijalno pretraživanje - analiza

- Odredimo broj operacija poređenja koje treba izvršiti prilikom pretraživanja. (sortirana/nesortirana lista)
  - tražena vrijednost na 1. mjestu => broj poređenja je 1
  - tražena vrijednost na 2. mjestu => broj poređenja je 2
  - tražena vrijednost na 3. mjestu => broj poređenja je 3
  - tražena vrijednost na n. mjestu => broj poređenja je n
  - tražena vrijednost nije u listi => broj poređenja je n
- Prosječan broj poređenja za ključ koji je u listi:
  - $T(n) = \frac{1 + 2 + 3 + ... + n}{n}$
  - $T(n) = \frac{n(n+1)}{2n} = \frac{n+1}{2}$ => $O(n)$

Sekvencijalno pretraživanje - analiza

- **Prednosti** sekvencijalnog pretraživanja:
  - Jednostavan algoritam
  - Elementi liste mogu biti u bilo kojem poretku
- **Slabosti**:
  - Algoritam je neefikasan (spor).
  - Za listu on N elemenata u prosjeku se mora provjeriti N/2 elemenata
  - Moraju se provjeriti svi elementi u listi (N) da bi se utvrdilo da se tražena vrijednost ne nalazi u listi

Primjer generisanja pseudoslučajnih brojeva

Radi generisanja testnih podataka vrlo često nam treba sekvenca proizvoljnih brojeva.
U tu svrhu se možemo poslužiti C funkcijama `srand` i `rand` koje su deklarisane u zaglavlju `<stdlib.h>`, odnosno `<cstdlib>`.

```
#include <ctime>
#include <cstdlib>
#include <iostream>

int main(void)
{
  srand(time(NULL)); // inicijalizacija sekvence pseudoslucajnih brojeva,
                     // pozvati samo jednom
  std::cout << "Generisanje 10 nenegativnih brojeva manjih od 100\n" << std::endl;
  for(int i=0; i<10; ++i)
  {
    int r = rand() % 100; // rand() vraca slucajni broj izmedju 0 i RAND_MAX (stdlib.c)
    std::cout << i << ". slucajni broj je " << r << std::endl;
  }

  return 0;
}
```

Primjer – sekvencijalno pretraživanje

```
template<typename It, typename T>
It find(It pocetak, It kraj, const T& v)
{
 for(auto i=pocetak; i!=kraj; ++i)
  if(*i == v) return i;
 return kraj;
}
```

```
template<typename It, typename T, typename F>
It find(It pocetak, It kraj, const T& v, const F& f)
{
 for(auto i=pocetak; i!=kraj; ++i)
  if( f(*i, v) ) return i;
 return kraj;
}
```

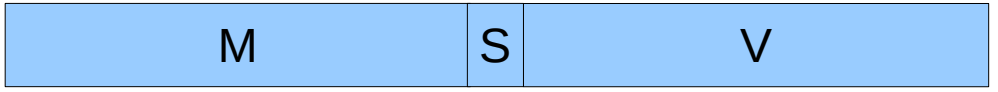
```
...
 std::vector<int> a;
 for(int i=0;i<100; ++i) a.push_back(rnd());
 auto it = find(a.begin(),a.end(), 55);
...
```

Binarno pretraživanje

- Ako je lista sortirana i znamo koliko imamo elemenata, te ako imamo direktan i brz, O(1), pristup svakom elementu (npr. niz), možemo upotrijebiti drugačiju strategiju (nesekvencijalnu).
- Binarno pretraživanje radi slično poznatom principu "podijeli pa savladaj" (divide and conquer).
- Naziv **binarno** potiče od toga što algoritam radi tako da u svakom koraku polovi listu na **dva** dijela dok ne dođe do tražene vrijednosti ili do zaključka da se tražena vrijednost ne nalazi u listi.
- U svakom koraku se eliminiše dio liste u kojem se sa sigurnošću može tvrditi da nema traženog elementa.
- Ne postoji algoritam pretraživanja koji je efikasniji od ovog algoritma.

Binarno pretraživanje - algoritam

1. Podijeliti sortiranu listu (niz) na dva dijela, formalno - tri dijela. Traženi element, ako postoji, je u jednom od ta tri dijela:
    1. srednji element - S
    2. elementi sa lijeve strane srednjeg elementa ("manji" od srednjeg) - M
    3. elementi sa desne strane srednjeg elementa ("veći" od srednjeg) - V

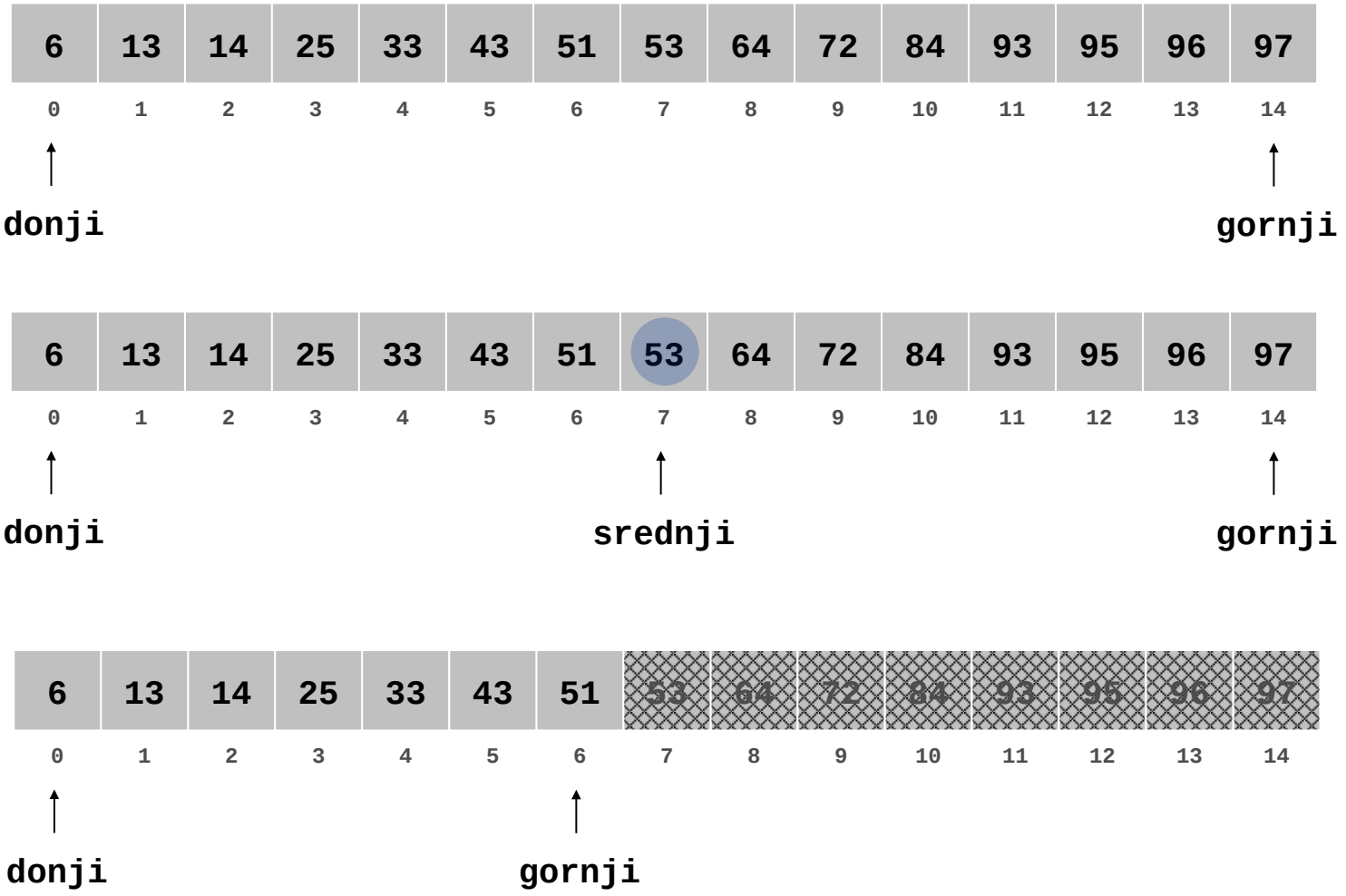


2. Ako je **srednji** element tražena vrijednost – KRAJ. U suprotnom, idi na korak 1 koristeći samo onu polovicu liste iz prethodnog koraka u kojoj se može nalaziti tražena vrijednost.
3. Ponavljaj korake 1 i 2 dok se ne pronađe tražena vrijednost ili dok se ne iscrpe svi elementi koje ima smisla provjeravati.

Primijetite: *Uvijek provjeravamo samo srednji element.*

Binarno pretraživanje - primjer

- Neka je dat sortirani niz u kome tražimo vrijednost 33.



Tražimo između indeksa **donji** i **gornji**

Uvijek krećemo od srednjeg elementa:
srednji=(gornji+donji)/2

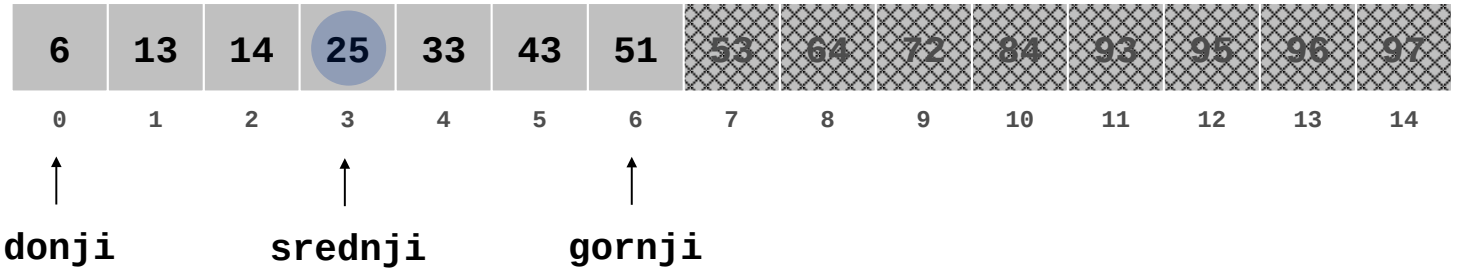
niz[srednji]

33 < 53 pa tražimo samo u lijevom dijelu:
gornji = srednji - 1

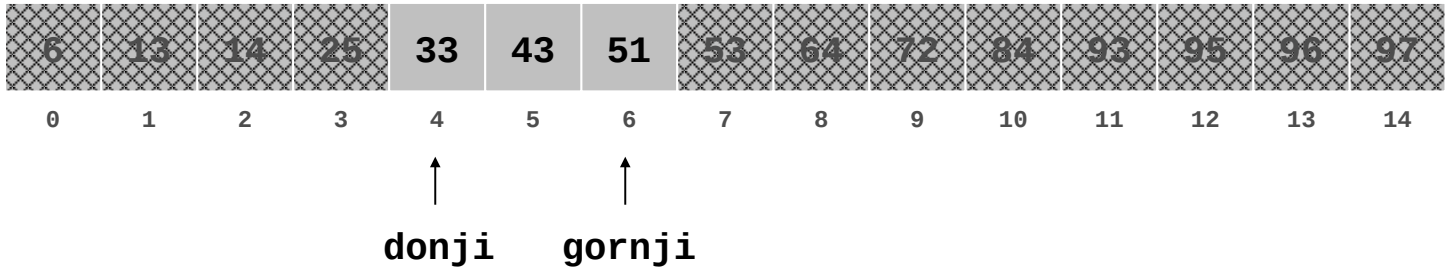
U prvom koraku smo eliminisali potrebu za provjeravanjem desne polovice elemenata.

Binarno pretraživanje - primjer

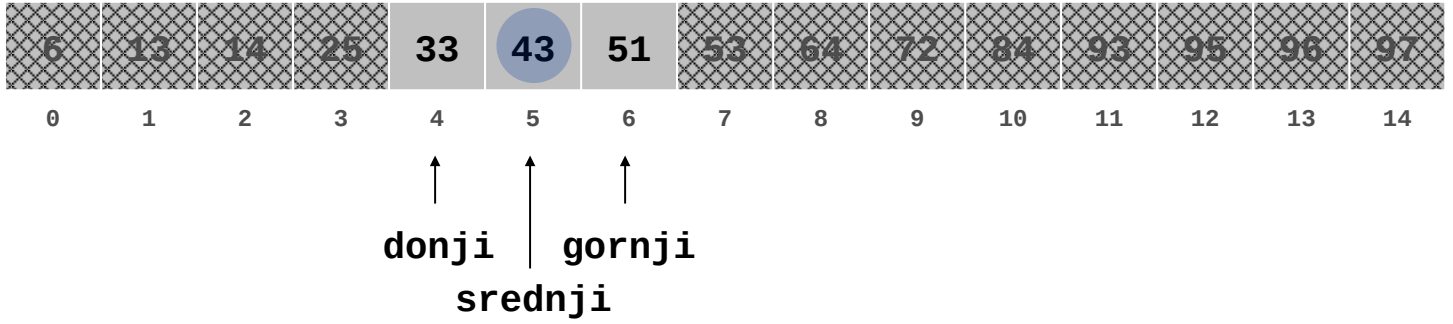
- Neka je dat sortirani niz u kome tražimo vrijednost 33.



Izračunavamo novi indeks:
srednji=(gornji+donji)/2



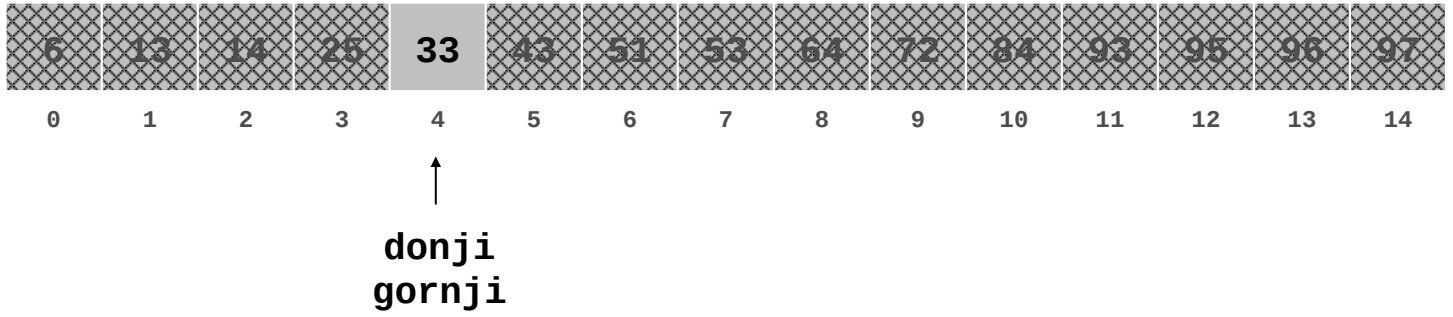
33 > 25 pa tražimo samo u desnom dijelu:
donji = srednji + 1



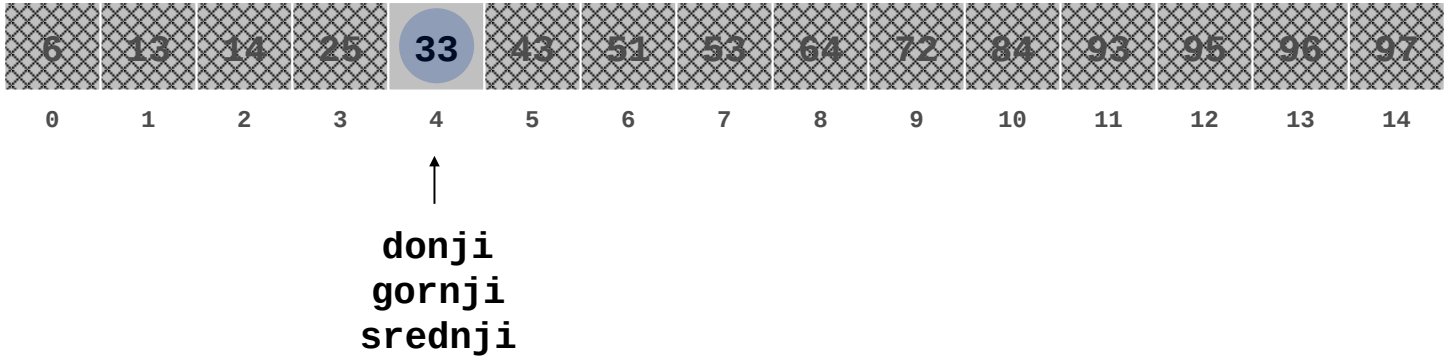
Izračunavamo novi indeks:
srednji=(gornji+donji)/2

Binarno pretraživanje - primjer

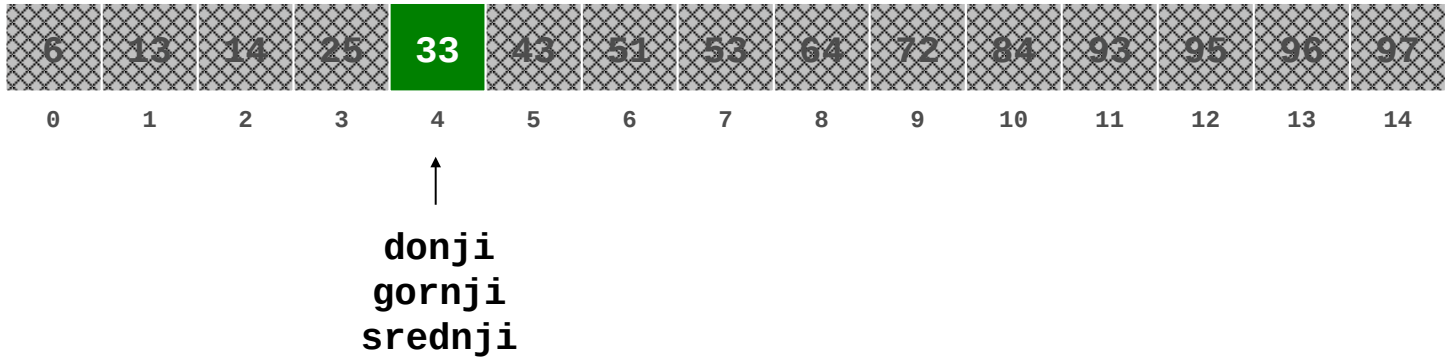
- Neka je dat sortirani niz u kome tražimo vrijednost 33.



33 < 43 pa tražimo samo u lijevom dijelu:
donji = srednji + 1



Izračunavamo novi srednji indeks:
srednji=(gornji+donji)/2



Nađena je tražena vrijednost.

Binarno pretraživanje - analiza

- U prethodnom primjeru smo tražili broj u listi od 15 brojeva i pronašli smo ga u 4 pokušaja.

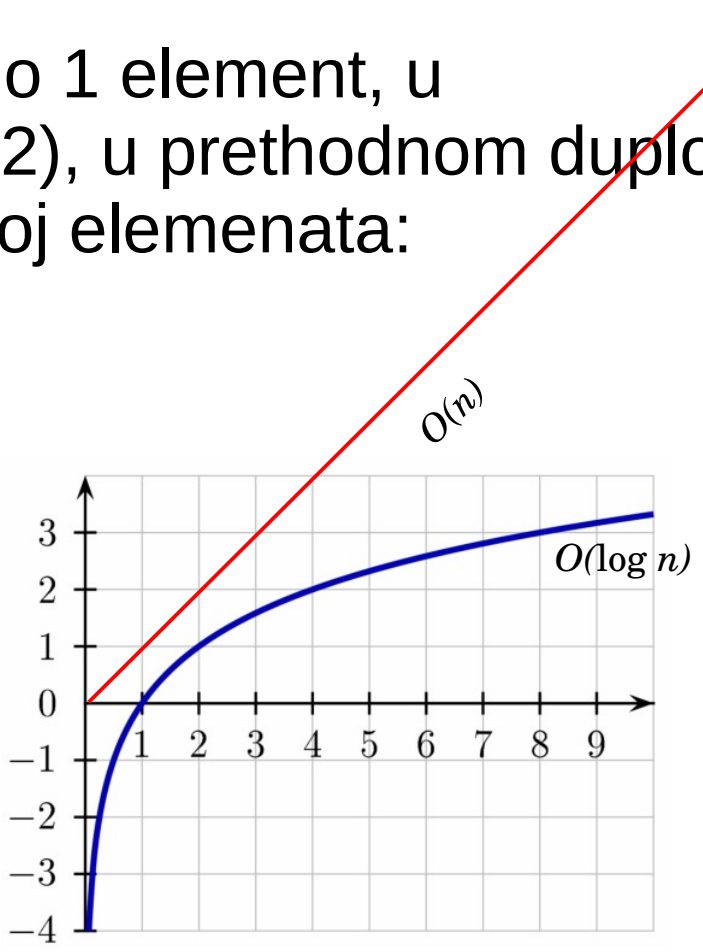
- Šta ako tražimo u listi on 32 elementa?
  - 1. pokušaj – lista se polovi – 16 elemenata
  - 2. pokušaj – 8 elemenata
  - 3. pokušaj – 4 elemenata
  - 4. pokušaj – 2 elemenata
  - 5. pokušaj – 1 element
- Dakle, traženi element ćemo pronaći, ili ćemo utvrditi da ne postoji, u najviše 5 pokušaja.

Binarno pretraživanje - analiza

- Tražimo u listi on 512 elementa:
  - 1. pokušaj – lista se polovi – 256 elemenata
  - 2. pokušaj – 128 elemenata
  - 3. pokušaj – 64 elemenata
  - 4. pokušaj – 32 elemenata
  - 5. pokušaj – 16 element
  - 6. pokušaj – 8 element
  - 7. pokušaj – 4 element
  - 8. pokušaj – 2 element
  - 9. pokušaj – 1 element

Binarno pretraživanje - analiza

- Ako tražimo element u listi od **N** elemenata tako da napravimo najveći mogući broj koraka za takvu listu (označimo taj broj koraka sa **p**), u kojem su odnosu **N** i **p**?
- Krenimo otpozadi!
- Na kraju pretraživanja imamo samo 1 element, u predzadnjem pretraživanju duplo (2), u prethodnom duplo od toga (4) itd. svaki put se duplira broj elemenata:
- $N=1 * 2 * 2 * ... * 2 = 2^p$
- $\Rightarrow p=\log_2 N=\lg N$
- Binarno pretraživanje je $O(\log n)$



Binarno pretraživanje – prednosti/slabosti

- Za listu od 1.048.576 elemenata pri sekvencijalnom pretraživanju bismo imali najviše 1.048.576 a u prosjeku 524.266 iteracija, dok bismo pri binarnom pretraživanju imali najviše 20 iteracija.

- **Prednosti** binarnog pretraživanja:
  - Mnogo efikasnije od sekvencijalnog pretraživanja.
  - Ne postoji algoritam pretraživanja koji ima veću efikasnost.
- **Slabosti** binarnog pretraživanja:
  - Niz mora biti sortiran.
  - Moramo imati direktan pristup elementima (npr. preko indeksa).

Interpolacijsko pretraživanje

- Pretpostavimo da tražite broj neke osobe (npr. Žunić Željko) u telefonskom imeniku od 1000 strana.
- Gdje ćete otvoriti imenik?
- Naravno, pri kraju imenika.
- Ideja interpolacijskog pretraživanja počiva na sličnom principu kao binarno pretraživanje sa razlikom u odabiru "srednjeg" elementa.
- Ovdje se "srednji" element ne bira kao sredina intervala već se izračunava na osnovu vrijednosti koja se traži i vrijednosti krajnjih elemenata u trenutnom segmentu koji se pretražuje tako da se pokuša procijeniti gdje bi u takvom sortiranom segmentu mogao biti traženi podatak.
  - `pos = lo + [ (x-arr[lo])*(hi-lo) / (arr[hi]-arr[lo]) ]`
- U prosjeku, daje bolje rezultate od binarnog pretraživanja u slučajevima uniformne raspodjele vrijednosti unutar niza.

Binarno i interpolacijsko pretraživanje

Računanje indeksa:

- Kod standardnog binarnog pretraživanja:
  - `pos = (lo+hi) / 2, ili`
  - `pos = lo + (hi-lo) / 2`
- Kod interpolacijskog pretraživanja
  - `pos = lo + (hi-lo) * (x-arr[lo])/(arr[hi]-arr[lo])`
- Srednji element je onda:
  - `arr[pos]`