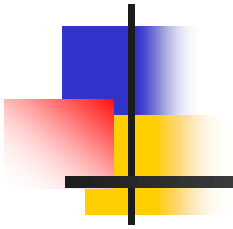
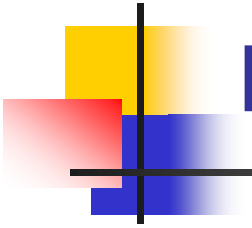


PROGRAMIRANJE II

P-05: Osnovi pretraživanja

prof. dr **Dražen Brđanin**
2023/24



P-05: Osnovi pretraživanja

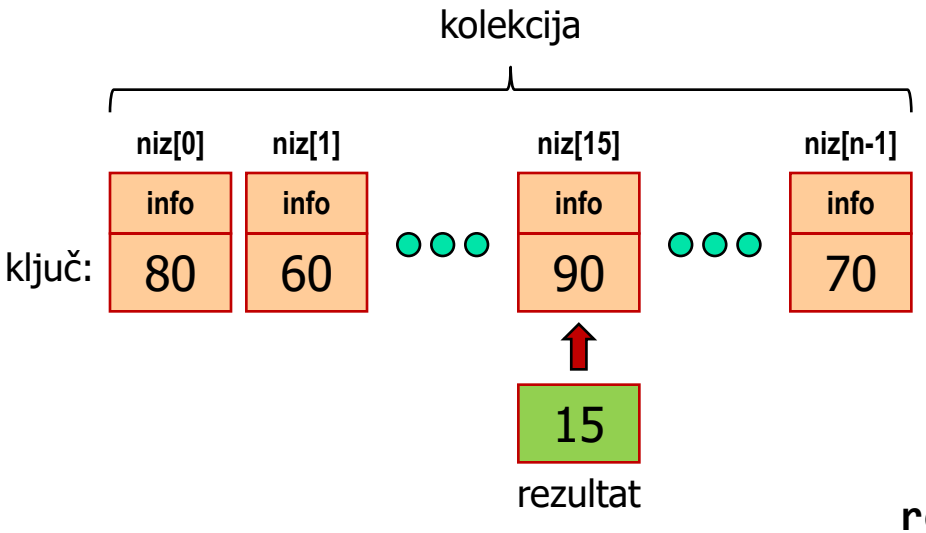
- **Sadržaj predavanja**
  - Osnovni pojmovi o pretraživanju
  - Osnovni algoritmi za pretraživanje
    - sekvencijalno pretraživanje (sa varijantama)
    - binarno pretraživanje (sa varijantama)
  - Osnovi spoljašnjeg pretraživanja

Osnovni pojmovi o pretraživanju

- **PRETRAŽIVANJE = lociranje (pronalaženje) elementa u kolekciji prema nekom kriterijumu (vrijednosti ključa)**
- **Pretraživanje je veoma česta aktivnost (utiče na složenost)**
- **Kolekcija može biti: uređena** ili **neuređena**
- **Klasifikacija pretraživanja prema mjestu:**
  - **unutrašnje**: pretraživanje kolekcije koja se nalazi u operativnoj memoriji
  - **spoljašnje**: pretraživanje kolekcije koja se nalazi u sekundarnoj memoriji (datoteka)
- **Rezultat pretraživanja: uspješno** ili **neuspješno**
- **Ključ pretraživanja:**
  - **primarni ključ**:
    - jedinstveno identifikuje zapis – ne postoje dva zapisa sa istom vrijednošću ključa (npr. ne postoje dvije osobe sa istim JMBG)
  - **sekundarni ključ**:
    - više zapisa u kolekciji može da ima istu vrijednost ključa (npr. svi studenti iz Banjaluke imaju isti broj pošte)

Sekvencijalno pretraživanje

- **Alternativni nazivi: redno ili linearno pretraživanje**
- **Osnovne karakteristike** sekvencijalnog pretraživanja:
  - **najjednostavnija tehnika pretraživanja**
  - pretraživanje **element po element (zapis po zapis)**
  - **jedina moguća tehnika u slučaju neuređenih kolekcija** (i lista)



ključ pretraživanja:

90

Poziv funkcije za pretraživanje:

```
rezultat = SeqSearch(niz, KLJUC, n)
```

Sekvencijalno pretraživanje

- **Implementacija**:

```
int SeqSearch (<tip> niz[], <tip> kljuc, int n)
{
   for (int i=0; i<n; i++)
      if (niz[i] == kljuc) return i;
   return -1;
}
```

```
/* pozivajući kod */
if (SeqSearch(niz,kljuc,n)>=0)
   // printf(”Uspjesno”);
else
   printf(”Neuspjesno”);
```

- **Analiza složenosti**:
  - **najgori slučaj** (traženi podatak nalazi se na kraju kolekcije)

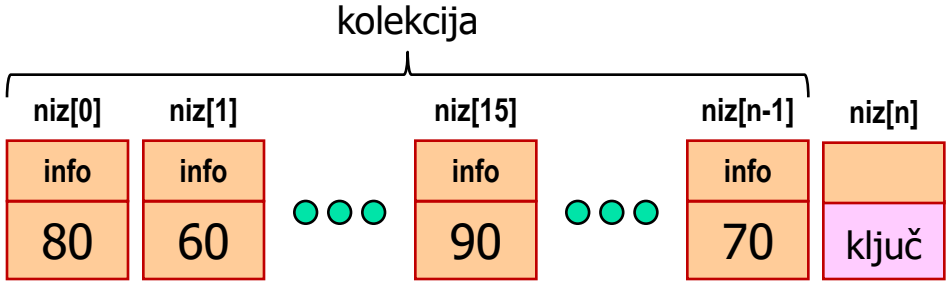
$$T(n)=O(n)$$

  - **najbolji slučaj** (traženi podatak nalazi se na početku kolekcije)

$$T(n)=\Omega(1)$$

Sekvencijalno pretraživanje

- **Unaprijeđena implementacija (sa stražom)**:




```
int SeqSearchStraza (<tip> niz[], <tip> kljuc)
{
   int i=0;
   while (niz[i] != kljuc) i++;
   return i;
}
```

```
/* pozivajući kod */
niz[n] = kljuc;
if (SeqSearchStraza(niz,kljuc) == n)
   printf("Pretrazivanje neuspjesno");
else
   printf("Pretrazivanje uspjesno");
```

Složenost:

Najgori slučaj (podatak na kraju):

$$T(n)=O(n)$$

Najbolji slučaj (podatak na početku):

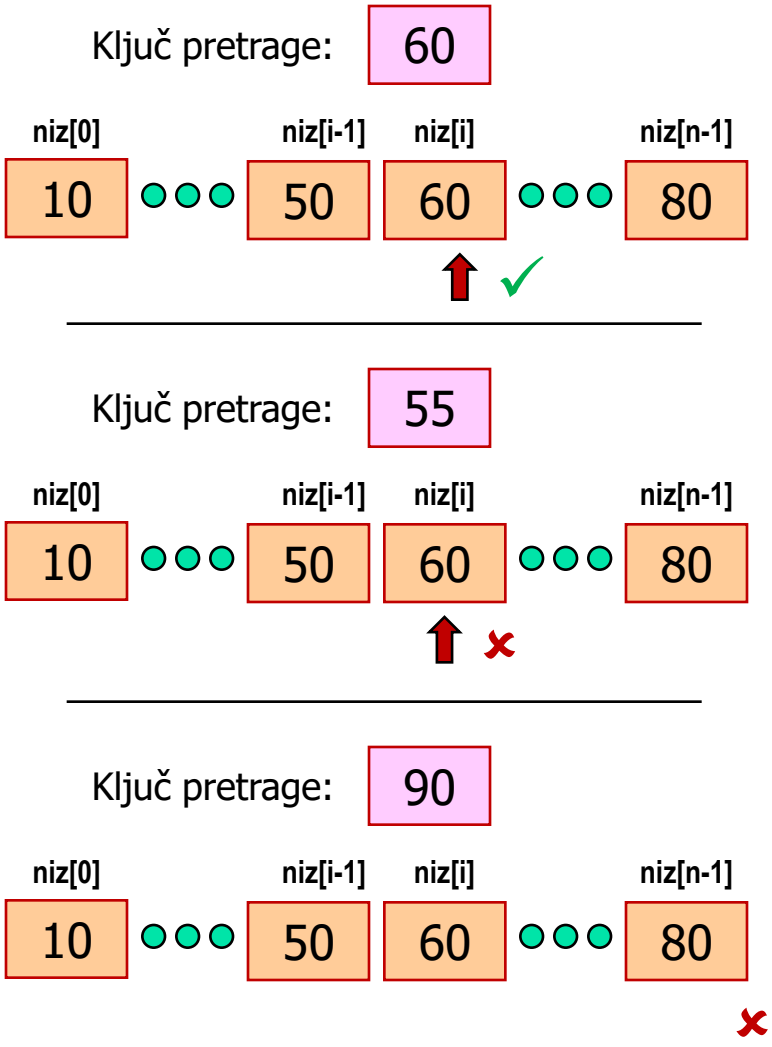
$$T(n)=\Omega(1)$$

Složenost je ista (kao i u prethodnoj implementaciji), ali je izvršavanje malo brže!

Sekvencijalno pretraživanje

- **Unaprijeđena implementacija (ako je kolekcija sortirana)**:
  - Ako je kolekcija sortirana (ključevi su u rastućem poretku), pretraživanje se vrši sve dok je ključ pretrage veći od ključeva u kolekciji ili dok se ne dođe do kraja
  - Element je pronađen ako nismo došli do kraja i ako je ključ pretrage jednak ključu elementa na kojem je pretraga prekinuta

```
int SeqSorted(<tip> niz[], <tip> kljuc, int n)
{
   int i=0;
   while (niz[i]<kljuc && i<n) i++;
   return (i<n && niz[i]==kljuc);
}
```



Najgori slučaj (podatak na kraju): $T(n)=O(n)$

Najbolji slučaj (podatak na početku): $T(n)=\Omega(1)$

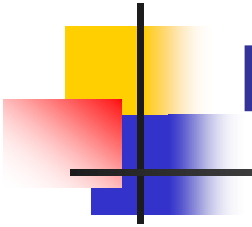
Složenost je ista (kao i u prethodnim implementacijama), ali nema bespotrebnog poređenja.

Sekvencijalno pretraživanje

- **Samoorganizujuće pretraživanje** (***eng. self-organizing search***):
  - **move to front**
    - ako se pretpostavlja da će neki elementi kolekcije češće da se traže, tada se pronađeni element premješta na početak, a svi ostali (koji su se nalazili ispred) jedno mjesto prema kraju
  - **move to back**
    - ako se pretpostavlja da se elementi kolekcije nakon pretraživanja neće dugo ponovo tražiti onda se element nakon pronalaženja premješta na kraj, a svi ostali (koji su se nalazili iza) jedno mjesto prema početku

```
int MoveToFront(<tip> niz[], <tip> kljuc, int n)
{
   for (int i=0; i<n; i++)
      if (niz[i] == kljuc)
      {
         <tip> pom = niz[i];
         for ( ; i>0; i--) niz[i]=niz[i-1];
         niz[0] = pom;
         return 0;
      }
   return -1;
}
```

```
int MoveToBack(<tip> niz[], <tip> kljuc, int n)
{
   for (int i=0; i<n; i++)
      if (niz[i] == kljuc)
      {
         <tip> pom = niz[i];
         for ( ; i<n-1; i++) niz[i]=niz[i+1];
         niz[n-1] = pom;
         return n-1;
      }
   return -1;
}
```

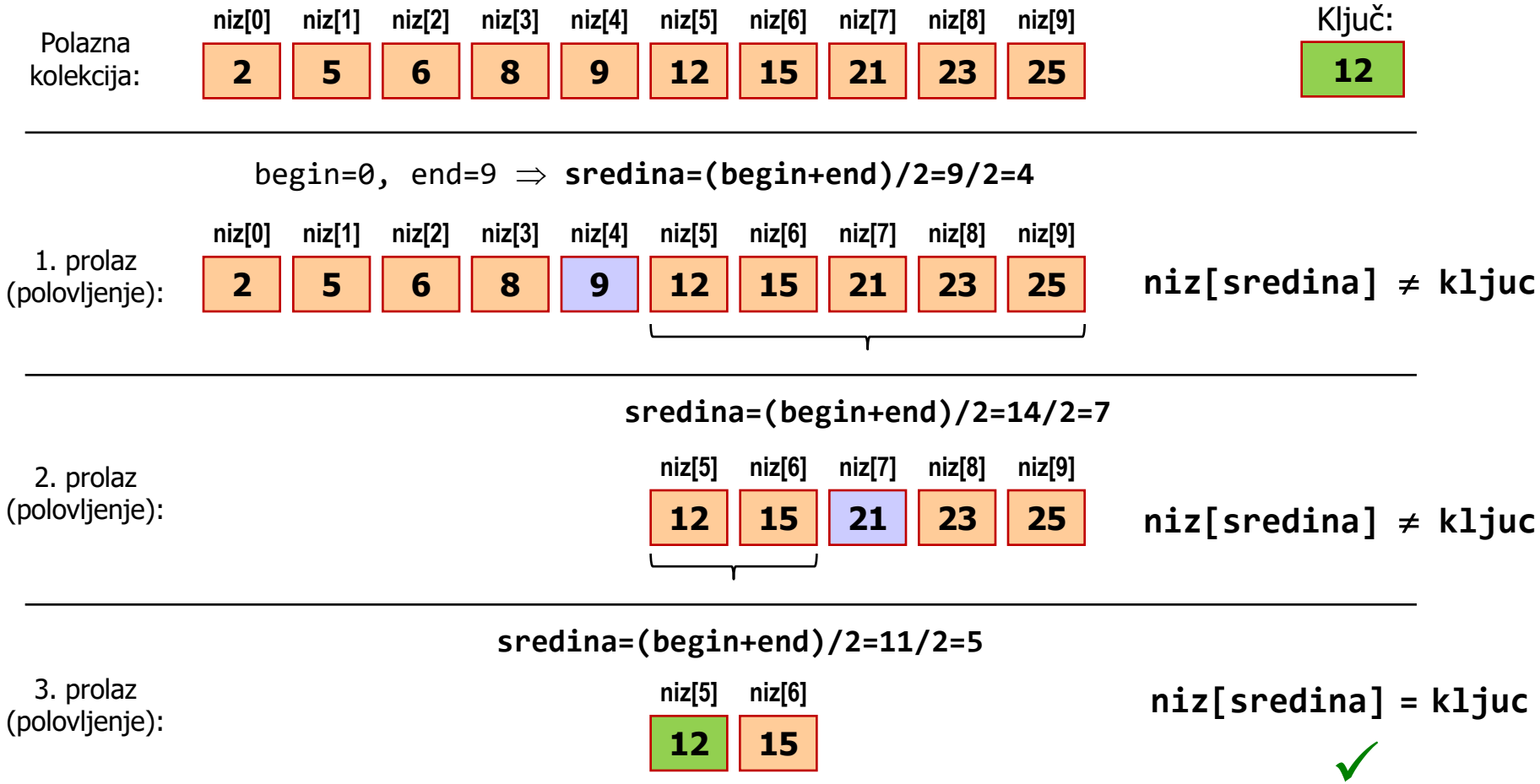


Binarno pretraživanje

- **Alternativni naziv: pretraživanje polovljenjem intervala**
- **Osnovne karakteristike** binarnog pretraživanja:
  - primjer "*divide and conquer*" algoritma
  - ne mora nužno da se odnosi na pretraživanje konačne kolekcije (niz), već može da se primjenjuje i za pretraživanje beskonačne kolekcije (npr. interval na kojem se traži realno rješenje neke jednačine)
  - **pretpostavlja se da je kolekcija u uređenom redoslijedu** (npr. sortiran niz, realni ili cjelobrojni interval itd.)
- **Osnovni princip**:
  - sve dok se ne pronađe tražena vrijednost, polazna kolekcija se dijeli na dva dijela i u svakom koraku odbacuje ona polovina u kojoj se ne nalazi tražena vrijednost, a pretraga nastavlja u dijelu u kojem se sigurno nalazi tražena vrijednost
- **Ako je kolekcija diskretna** (tj. ako sadrži konačan broj elemenata *n*):
  - prvo se određuje srednji element – ako je on jednak ključu traženja pretraga je završena
  - ako je ključ pretrage manji od srednjeg elementa, pretraga se nastavlja u polovini u kojoj se nalaze elementi koji su manji od srednjeg elementa
  - ako je ključ pretrage veći od srednjeg elementa, pretraga se nastavlja u polovini u kojoj se nalaze elementi koji su veći od srednjeg elementa

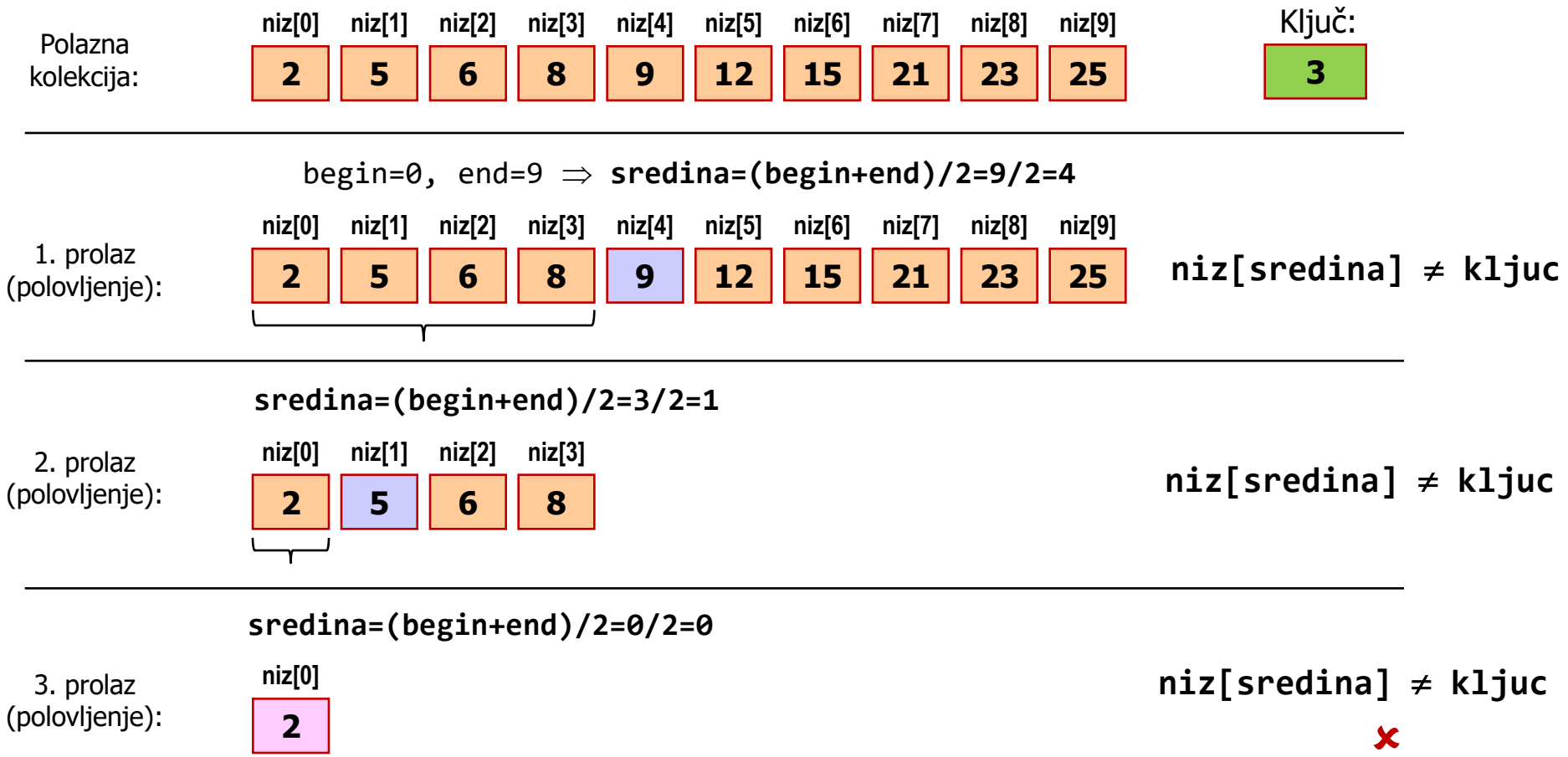
Binarno pretraživanje

- **Binarno pretraživanje niza - ilustracija uspješnog pretraživanja**



Binarno pretraživanje

- **Binarno pretraživanje niza - ilustracija neuspješnog pretraživanja**



Binarno pretraživanje

- **Implementacija (iterativna/nerekurzivna)**:

```
int BinSearch (<tip> niz[], int n, <tip> kljuc)
{
   int begin=0, end=n-1, sredina;
   do
   {
      sredina = (begin+end) / 2;
      if (niz[sredina] == kljuc) return sredina;
      if (kljuc < niz[sredina])
         end = sredina-1;
      else
         begin = sredina+1;
   }
   while (begin <= end);
   return -1;
}
```

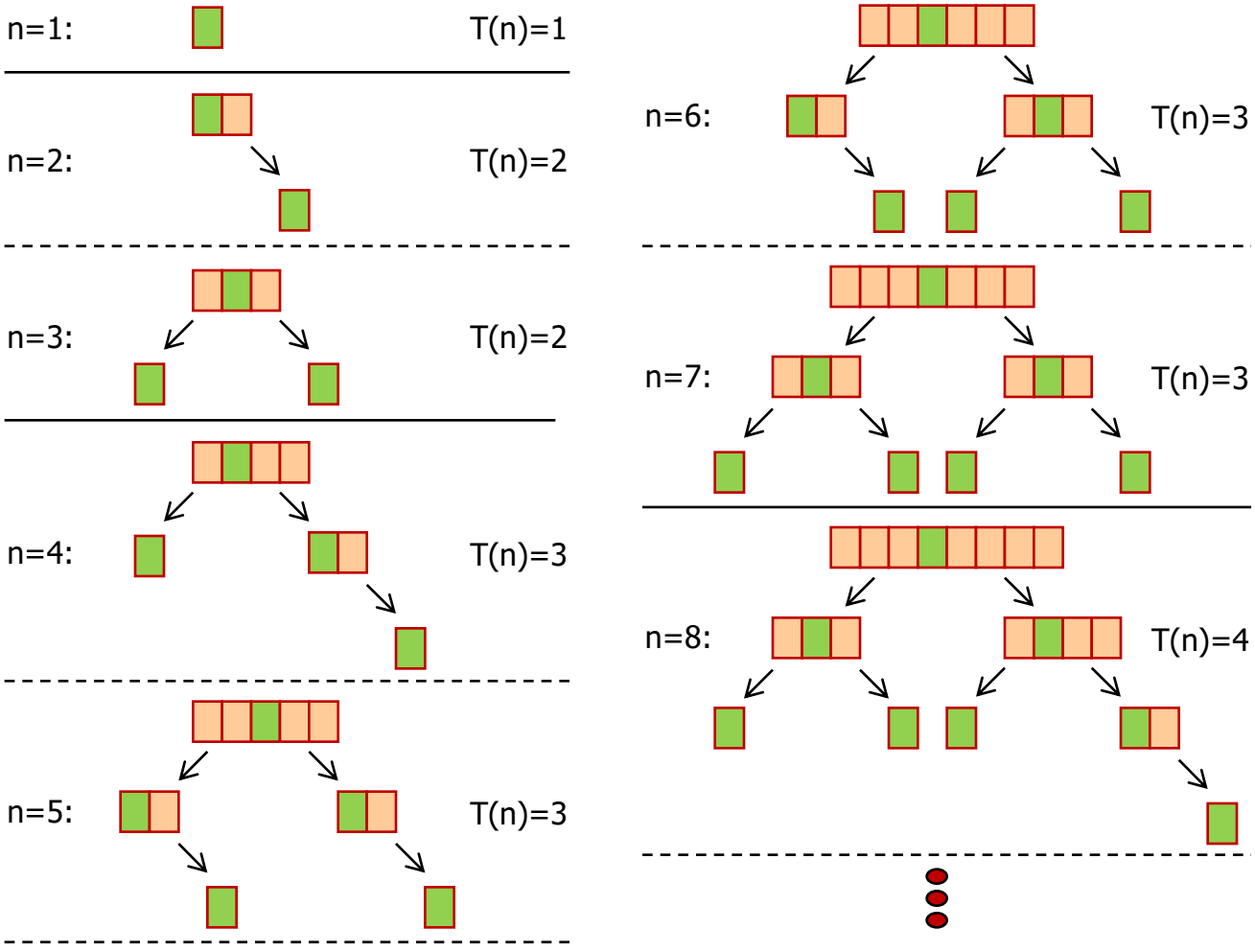
Binarno pretraživanje

- **Implementacija (rekurzivna)**:

```
int BinSearchRek (<tip> niz[], <tip> kljuc, int begin, int end)
{
   int sredina = (begin+end) / 2;
   if (begin > end) return -1;
   if (niz[sredina] == kljuc)
      return sredina;
   else
      if (kljuc < niz[sredina])
         return BinSearchRek(niz,kljuc,begin,sredina-1);
      else
         return BinSearchRek(niz,kljuc,sredina+1,end);
}
```

Binarno pretraživanje

- **Analiza izvršavanja (najgori slučaj)**:



$$2^{T(n)-1} \le n \quad / \log_2$$

$$\log_2 2^{T(n)-1} \le \log_2 n$$

$$(T(n)-1)\cdot \log_2 2 \le \log_2 n$$

$$T(n)-1 \le \log_2 n$$

$$T(n) \le \log_2 n + 1$$

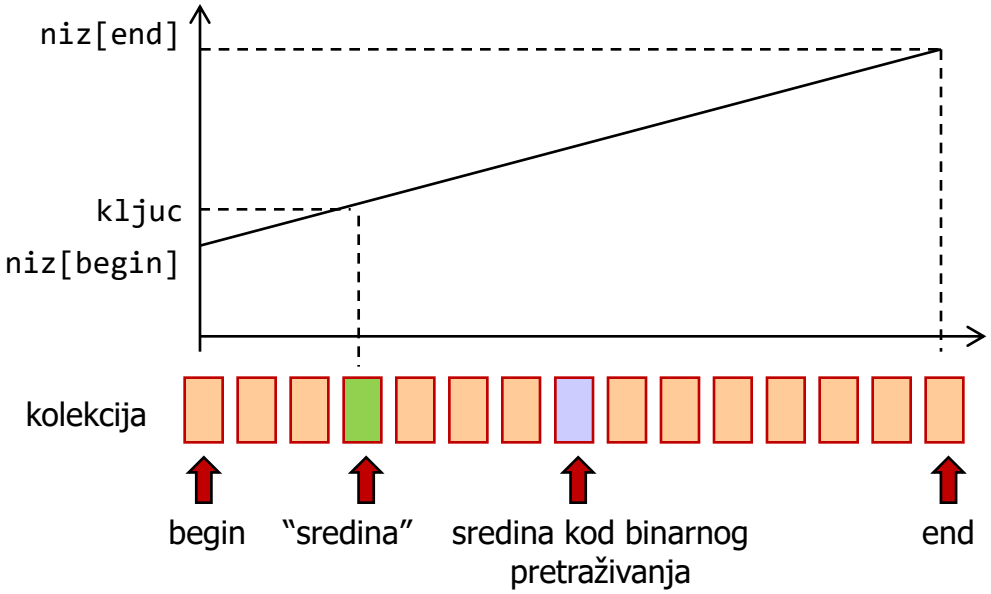
$$T(n) \approx \log_2 n$$

$$\boxed{T(n) = O(\log_2 n)}$$

Varijante binarnog pretraživanja

INTERPOLACIONO PRETRAŽIVANJE

- **Osnovne karakteristike:**
  - Osnovni algoritam binarnog pretraživanja pretražuje kolekciju dijeljenjem na dva podjednaka dijela
  - Ako je ključ traženja blizak donjem ili gornjem kraju, sporo se pronalazi traženi element kolekcije
  - **Brže pretraživanje može da se postigne ako se "sredina" interpolira – procijeni se pozicija što bliže traženom podatku** (**dobri rezultati postižu se u slučaju ravnomjerne raspodjele vrijednosti ključeva u kolekciji**)



Iz sličnosti trouglova slijedi:

$$\frac{sredina - begin}{kljuc - niz[begin]} = \frac{end - begin}{niz[end] - niz[begin]}$$

$$sredina = begin + \frac{(kljuc - niz[begin]) \cdot (end - begin)}{niz[end] - niz[begin]}$$

Sličan pristup primjenjujemo kod ručnog pretraživanja rječnika (npr. ako tražimo neku riječ na B, otvaramo rječnik bliže početku).

Varijante binarnog pretraživanja

- **Implementacija interpolacionog pretraživanja:**

```
int InterSearch (<tip> niz[], int n, <tip> kljuc)
{
  int begin=0, end=n-1, sredina, brojliac, imenilac;
  do
  {
    brojilac = (kljuc-niz[begin]) * (end-begin);
    imenilac = niz[end] – niz[begin];
    sredina = begin + brojilac/imenilac;
    if (niz[sredina] == kljuc) return sredina;
    if (kljuc < niz[sredina])
      end = sredina-1;
    else
      begin = sredina+1;
  }
  while (begin <= end);
  return -1;
}
```

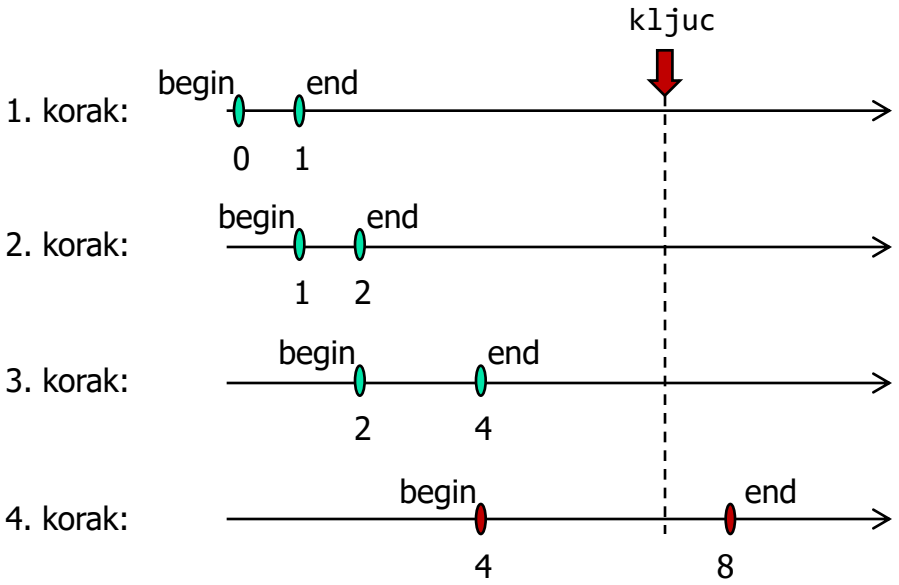
Ako su ključevi uniformno raspoređeni:

$$T(n) = O(\log_2 \log_2 n)$$

Varijante binarnog pretraživanja

EKSPONENCIJALNO PRETRAŽIVANJE (DUPLIRANJE INTERVALA)

- **Osnovne karakteristike:**
  - Binarno/interpolaciono pretraživanje može da se primjenjuje na kolekciju sa poznatim granicama
  - **Ako granice nisu inicijalno poznate** (**sortirana i neograničena kolekcija**), prvo **treba procijeniti donju i gornju granicu**, a zatim primijeniti binarno/interpolaciono pretraživanje
  - **Procjena granica kreće od neke inicijalne vrijednosti** (npr. 0 i 1), **a zatim se u svakom koraku granice ažuriraju** (npr. begin=end i end=end*2), sve dok se ne procijeni interval u kojem se nalazi ključ/željeni podatak




```
int ExpSearch (int kljuc)
{
  int begin=0, end=1;
  while (end < kljuc)
  {
    begin = end;
    end = end * 2;
  }
  return BinSearch(begin,end,kljuc);
}
```

Varijante binarnog pretraživanja

- Primjer eksponencijalnog i binarnog pretraživanja (pogađanje broja):

```
#include <stdio.h>
int main()
{
   int begin=0, end=1, sredina, odg;
   printf("Zamislite prirodan broj!\n");
   while (1)
   {
       printf("Da li je broj veci od %d? (1/0) ", end);
       scanf("%d", &odg);
       if (!odg) break;
       begin = end+1; end *= 2;
   }
   while (begin<end)
   {
       sredina = (begin+end) / 2;
       printf("Da li je broj veci od %d? (1/0)", sredina);
       scanf("%d", &odg);
       if (odg) begin=sredina+1;
       else end=sredina;
   }
   while (begin<end);
   printf("Zamislili ste broj: %d", end);
   return 0;
}
```

Primjer izvršavanja:

```
Zamislite prirodan broj!
Da li je broj veci od 1? (1/0) 1
Da li je broj veci od 2? (1/0) 1
Da li je broj veci od 4? (1/0) 1
Da li je broj veci od 8? (1/0) 0
Da li je broj veci od 6? (1/0) 1
Da li je broj veci od 7? (1/0) 0
Zamislili ste broj: 7
```

Primjer izvršavanja:

```
Zamislite prirodan broj!
Da li je broj veci od 1? (1/0) 1
Da li je broj veci od 2? (1/0) 1
Da li je broj veci od 4? (1/0) 1
Da li je broj veci od 8? (1/0) 1
Da li je broj veci od 16? (1/0) 0
Da li je broj veci od 12? (1/0) 0
Da li je broj veci od 10? (1/0) 1
Da li je broj veci od 11? (1/0) 0
Zamislili ste broj: 11
```

Varijante binarnog pretraživanja

- Primjer eksponencijalnog i binarnog pretraživanja (traženje realne nule jednačine):

```
#include <stdio.h>
#include <math.h>

double rjesenje(double (*f)(double), double eps)
{
    double begin=0, end=0.1, sredina;

    while ((*f)(begin) * (*f)(end) >= 0)
       end = (begin = end) * 2;

    while (end-begin>eps)
    {
        sredina = (begin+end)/2.0;
        if ((*f)(begin) * (*f)(sredina) > 0)
            begin = sredina;
        else
            end = sredina;
    }
    return sredina;
}

int main()
{
   double rez = rjesenje(cos, 0.01);
   printf("cos(%.5f)=%.5f\n", rez, cos(rez));
   return 0;
}
```

Rezultat izvršavanja:

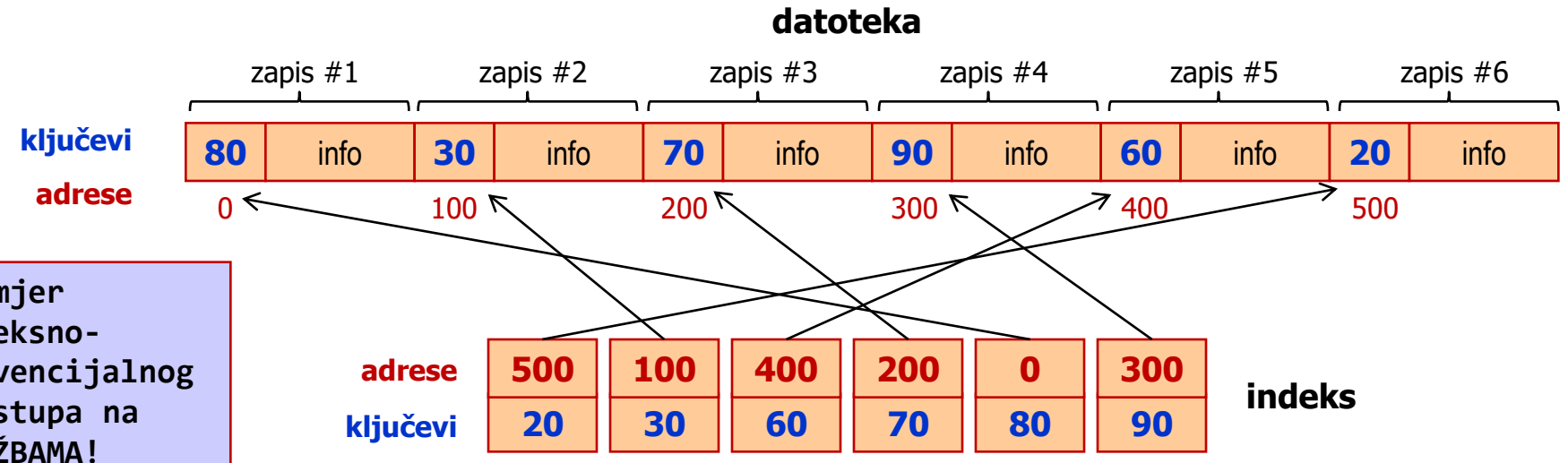
```
cos(1.56875)=0.00205
```

Osnovi spoljašnjeg pretraživanja

- **Spoljašnje (eksterno) pretraživanje = pretraživanje kolekcije u sekundardnoj memoriji (datoteka)**
- **Osnovne karakteristike** datoteka i pristupa datotekama:
  - **datoteke su sekvencijalno organizovane** (zapisi se upisuju redom, jedan iza drugog)
  - **pristup datoteci** može biti:
    - **sekvencijalni** (čitanje zapis po zapis)
    - **direktni** (pozicioniranje i čitanje željenog zapisa, ako se znaju veličine zapisa, odnosno adrese na kojima počinju zapisi)
  - **pristup datoteci je spora (skupa) operacija** (mnogo sporija od pristupa operativnoj memoriji)
  - **sekvencijalni pristup je veoma skup – složenost $O(n)$**
    - u najgorem slučaju treba pristupiti svakom zapisu u datoteci da bi se pronašao odgovarajući zapis ili da bi se konstatovalo da u datoteci ne postoji zapis sa traženom vrijednošću ključa
  - **alternativa: indeksno-sekvencijalni pristup**
    - **zapisi su sekvencijalno upisani u datoteku**
    - **indeks** je pomoćna sortirana struktura koja omogućava indirektan pristup datoteci

Indeksno-sekvencijalni pristup

- **Indeks**
  - indeks je pomoćna sortirana struktura (*look-up* tabela) koja omogućava indirektan pristup datoteci
  - indeks je mnogo manji od sekvencijalne datoteke i mnogo brže se pretražuje (npr. binarno)
  - indeks se "**održava**" pri promjeni daoteke:
    - ključevi u indeksu su uvijek poređani u odgovarajućem redoslijedu
    - svako dodavanje novog zapisa u sekvencijalnu datoteku rezultuje i umetanjem odgovarajućeg para `<ključ,adresa>` u indeks
    - svako brisanje ili izmjena postojećeg zapisa u sekvencijalnoj datoteci rezultuje modifikacijom indeksa



Primjer indeksno-sekvencijalnog pristupa na VJEŽBAMA!