

Programiranje 2


1. predavanje

Saša Singer

`singer@math.hr`

`web.math.pmf.unizg.hr/~singer`

PMF – Matematički odsjek, Zagreb



Dobar dan, dobro došli

Sadržaj predavanja (početak)

- Uvod u kolegij:
 - Tko sam, što sam i kako do mene.
 - Pravila lijepog ponašanja.
 - Računarski kolegiji na preddiplomskom studiju.
 - Cilj kolegija “**Programiranje 2**”.
 - Pregled sadržaja kolegija.
 - Ostale važne informacije o kolegiju. Posebno:
 - “**Pravila igre**” ili način polaganja ispita.
 - Literatura.
 - Korisni linkovi — službena web stranica kolegija.

Sadržaj predavanja (nastavak)

- **Funkcije** (ponavljanje):
 - Načini prijenosa argumenata:
 - “po vrijednosti”, “po adresi”.
 - Prijenos argumenata po vrijednosti u C-u.
 - Prijenos adresa — “varijabilni” argumenti.
- **Rekurzivne funkcije.**
 - Fibonaccijevi brojevi — **NE TAKO** i kako treba.
 - QuickSort algoritam.

Informacije — vježbe

Promjena u rasporedu za vježbe, od sljedećeg tjedna (13. 3.):

- Grupa PDM1 (M–P) — asist. Ciganović
 - srijeda, 10–12, u (A101).

Informacije — najava odrade

U petak, 5. 4., **nema predavanja** u redovitom terminu.

- Održava se **Dan i noć na PMF-u**, pa **nema nastave**.

Računajte na to da ćemo

- to **5. predavanje** morati **odraditi** u nekom terminu.

Prijedlog je da to napravimo **što prije** — za **10** dana, u:

- **ponedjeljak, 18. 3.**, od **16–18** sati u **(003)**.

Potvrđim još sljedeći tjedan (kad se raspored ustabili).

Informacije — web stranica

Moja web-stranica za Programiranje 2 je

<https://web.math.pmf.unizg.hr/~singer/prog2/>

ili, skraćeno,

<https://web.math.hr/~singer/prog2/>

Tamo su:

- kompletna predavanja iz prošlih godina, a stizat će i nova (kako nastaju),
- zip arhiva svih primjera i programa s predavanja, (zajedno s exe verzijama, Intel C).

Kopija je na adresi

<http://degiorgi.math.hr/~singer/prog2/>

Informacije — kolokviji

Programiranje 2 je u kolokvijskom razredu **C3**.

Službeni termini svih kolokvija su:

- Prvi kolokvij: petak, 26. 4. 2019., u 15 sati.
- Drugi kolokvij: petak, 21. 6. 2019., u 15 sati.
- Popravni kolokvij: utorak, 3. 9. 2019., u 15 sati.

Uputa: “**izbjegnite**” popravni — obavite to **ranije!**

Informacije — demonstratori

Prog2 ima istih 6 demonstratora kao i Prog1. To su:

- Al Depope
- Božidar Grgur Drmić
- Daniela Ivanković
- Sanjin Jurić Fot
- Ena Škopelja
- Patrik Vasung

Najavite se demosima koji dan ranije, na pr. e-mailom.

- Njihove termine i e-mail adrese nađete na službenoj web-stranici kolegija (čim dogovore termine).
- Postavite miša na ime, u prozoru kliknite na E-mail.

Informacije — ritam nastave

Tek toliko da ste **svjesni**:

- do 1. kolokvija imamo “**standardnih**” 7 tjedana nastave,
- a **ne** samo 6 ili punih 8.

Ovaj semestar, također, imamo

- **samo** 13 tjedana nastave, umjesto nekadašnjih 14.

Zato predavanja idu **istim** “ritmom” kao ranijih godina.

- **Preskačemo** ponavljanje gradiva iz **Prog1** — visi na webu kao **nulto** predavanje.

Stvarno, **Programiranje 2** ima ukupno 12 tjedana nastave,

- zato jer **jedan** obično **propadne** zbog raznih blagdana.

Ako ne propadne — zadnji tjedan služi kao rezerva :-)

Uvod u kolegij

Sadržaj

- Uvod u kolegij:
 - Tko sam, što sam i kako do mene.
 - Pravila lijepog ponašanja.
 - Računarski kolegiji na preddiplomskom studiju.
 - Cilj kolegija “Programiranje 2”.
 - Pregled sadržaja kolegija.
 - Ostale važne informacije o kolegiju. Posebno:
 - “Pravila igre” ili način polaganja ispita.
 - Literatura.
 - Korisni linkovi — službena web stranica kolegija.

Na samom početku

- **Moja malenkost** (u punom “sjaju”):

izv. prof. dr. sc. **Saša Singer**

- **Službeni osobni podaci:**

- ured (soba, kabinet): **227**, drugi kat,

- e-mail: **singer@math.hr**

- web stranica: **<http://web.math.hr/~singer/>**

- odn. **<http://web.math.pmf.unizg.hr/~singer/>**

- **Konzultacije:**

- službeno: **petak, 12–14 sati,**

- ili — po dogovoru.

Osnovna pravila “lijepog” ponašanja

Imam nekoliko lijepih **zamolbi** u rubrici “**kultura**”.

● Prva i osnovna je

razumna tišina,

tj. da pričanjem **ne ometate** izvođenje nastave.

● Zatim, **ne kasnite** na predavanje.

● Održavajte **razuman red** u predavaonici.

● **Mobilne telefone**, molim, **utišajte**.

Ukratko o kolegijima iz računarstva

Programiranje 2 — skraćeno = Prog2, je drugi od (barem) 4 računarska kolegija na preddiplomskom studiju Matematika:

- Programiranje 1 (Prog1), prije toga Uvod u računarstvo,
- Programiranje 2 (Prog2), prije toga Programiranje (C),
- Strukture podataka i algoritmi (SPA),
- Računarski praktikum I (RP1).

Napomena: Raniji kolegiji su preduvjet za kasnije (navedenim redom, od 1. do 4. semestra).

Prog2 je drugi osnovni kolegij iz računarstva. Ne šaljite se ...

- Tko ima problema s Prog2, vrlo će teško “preživjeti” ostatak.

Cilj kolegija Programiranje 2

Ovaj kolegij,

- kao nastavak na Prog1 i preduvjet za SPA,

ima 2 osnovna cilja:

- savladavanje osnovnih tehnika programiranja, tj. realizacija osnovnih algoritama,
- učenje konkretnog programskog jezika — C, koji je sredstvo za realizaciju tih algoritama.

Cilj kolegija Programiranje 2 (nastavak)

Očekivana znanja i vještine — koje Vi trebate steći:

- razumijevanje konceptata i praktični rad s
 - funkcijama,
 - pokazivačima,
 - složenim strukturama podataka (polja, strukture, vezane liste),
 - datotekama,
- razumijevanje sintakse i semantike naredbi programskog jezika C,
- sposobnost pisanja osnovnih algoritama u programskom jeziku C.

Pregled sadržaja kolegija

Teme — posložene kao elementi programskog jezika C:

- Funkcije (ponavljanje) i rekurzivne funkcije.
- Struktura programa.
- Dvodimenzionalna i višedimenzionalna polja.
- Pokazivači. Pokazivači i polja. Pokazivači i funkcije.
- Strukture.
- Dinamičke strukture podataka. Vezane liste.
- Datoteke.
- Pretprocesorske naredbe.
- Standardna C biblioteka.

Kako položiti Programiranje 2?

Ocjena se formira na temelju zbroja bodova iz 2 dijela:

- 1. kolokvij — ima (najmanje) 40 bodova,
- 2. kolokvij — ima (najmanje) 60 bodova,

Nije greška — zaista se može osvojiti preko 100 bodova.

Za prolaz je potrebno:

- zaraditi ukupno barem 45 bodova iz kolokvija (prvi i drugi zajedno, ili popravni),
- s tim da na barem jednom zadatku (na nekom kolokviju) treba zaraditi najmanje 80% mogućih bodova.

Za razliku od Prog1, na Prog2 — svi zadaci su programski.

Polaganje ispita — popravni (*Ne koristiti!*)

Popravni kolokvij je “zadnji vlak za spas” i

- obuhvaća gradivo **cijelog** kolegija (uključivo i **Prog1**).

Uvjeti za **prolaz** su **isti** kao i prije!

Na **popravni** možete **samo** ako ste:

- zaradili barem **30 bodova** na **redovitim** kolokvijima,
- s tim da na **barem jednom zadatku** (na **nekom** kolokviju) imate **najmanje 50%** mogućih bodova.

Okruglo, ti preduvjeti su oko **2/3 prolaza** “redovitim” putem.

Izgleda vrlo “**oštro**”, ali iskustvo i statistika kažu da

- s **manje** od toga — **nemate** nikakve šanse za **prolaz**.

Polaganje ispita — tablica ocjena

Na kraju, evo kako se tako zarađeni bodovi pretvaraju u tzv.

● prvu ponuđenu ocjenu (može, ali ne mora biti konačna.)

Tablica bodovi \mapsto ocjene:

Bodovi	Ocjena
0 – 44	1
45 – 59	2
60 – 74	3
75 – 89	4
90 i više	5

To vrijedi za zbroj bodova — onih koji se “zbrajaju”.

Polaganje ispita (nastavak)

U načelu, **usmenog** ispita (“završne provjere znanja”) **NEMA**, tj. završna provjera znanja = **upis ocjene**. Mogući **izuzeci** su:

- 🔴 po **želji** — ako **položite**, a **niste zadovoljni** ocjenom,
- 🔴 po **kazni** — nastavnik vas **IMA PRAVO** pozvati na usmeni ispit (na pr., zbog **prepisivanja** na kolokviju).

Napomena: usmeni je **praktični** (za računalom).

Tako zarađena **konačna** ocjena može biti

- 🔴 i **manja** od one **prvo ponuđene**, uključivo i **pad kolegija**.

Pravila polaganja ispita su na službenoj **web stranici** kolegija.

Ovdje ide priča da “**nema šale**”.

Kako položiti ispit — najvažnije + upozorenje!

“Nema šale” \iff programiranje se uči prvenstveno

- samostalnim pisanjem programa na računalu.

Nema zamjene za to iskustvo!

- Ne može ga netko steći za vas, umjesto vas.

Upozorenje: C nije jednostavan jezik i

- nije izmišljen za učenje programiranja.

Svakako,

- isprobajte programe s predavanja i vježbi.

Sve je dostupno na webu

- službenom i/ili mojem — v. malo dalje.

Literatura za Prog2

Osnovna literatura su, naravno,

- predavanja i vježbe,

s popratnim materijalima (na pr. programi na webu).

Dodatna literatura — ukratko (više riječi je bilo na Prog1):

- **Brian W. Kernighan** i **Dennis M. Ritchie**,
The C Programming Language (second edition),
Prentice Hall, Upper Saddle River, New Jersey, 1988.
- **B. S. Gottfried**, **Theory and Problems of Programming with C** (second edition), Schaum's outline series,
McGraw-Hill, New York, 1996.
(Uputa: tražite najnovije izdanje.)

Programska podrška za C

Za praktično programiranje u C-u, možete koristiti

- Code::Blocks (umjesto DevC++), MS Visual Studio, ..., na Windowsima,
- Code::Blocks, cc, gcc na Unix/Linux platformi.

Ponavljam:

- isprobajte programe s predavanja i vježbi.

Osim toga, (is)koristite demonstratore.

Korisni linkovi

Službena web stranica kolegija je:

<http://degiorgi.math.hr/prog2/>

Tamo su:

- predavanja prof. Nogo i link na moja predavanja (moja predavanja su na mom webu, da ne bude “kaos”),
- vježbe,
- službeni podsjetnik (“šalabahter”),
- sve bitne obavijesti,
- svašta drugo — pogledajte!

Korisni linkovi (nastavak)

Isplati se relativno često svratiti, jer se

- sve važne stvari prvo pojave na webu.

Na primjer, rezultati kolokvija!

Ako mislite da bi na službenom webu trebalo biti još nešto, slobodno predložite!

- Ideja je da tamo bude sve što vam može pomoći.

Molba: Ako nešto ne radi, odmah javite nastavnicima ili asistentima. Najbolje,

- meni (na singer), a ja ću “proslijediti dalje”.

Korisni linkovi — forum

Na kraju, postoji i “društveno mjesto” na webu Matematičkog odsjeka — tzv. **forum**:

<http://degiorgi.math.hr/forum/>

Tamo ima i podforum za **Programiranje 1 i 2** na kojem se svašta nađe.

Stvarno, web i forum još uvijek (na daljinu) održava

- **Vedran Šego** — još uvijek vrlo “živ” kao **vsego**,
- dugogodišnji asistent iz **Prog** i autor **skripte za vježbe**,
- a pred nekoliko godina je bio i nastavnik (i “natjerao” me da podosta sredim ova predavanja).

Veliko HVALA!

Molba — za predavanja i sve moje materijale

Lijepo molim, ako uočite neku **grešku** i sl., bez **ustručavanja**,

👉 **javite mi** (najlakše mailom) — bit će popravljena :-)

Nakon silnih godina,

👉 ja čitam “ono što **hoću**”, a **ne** ono što **zaista** piše.

Isto vrijedi i za sve **programe** na mom webu!

Naravno, ako nešto nije jasno, izgleda “čudno”, ...

👉 **pitajte me** – ne grizem!

(bar **ne prije** kolokvija i ispita).

Ima li pitanja?

Slušam ...

Funkcije

Sadržaj

- **Funkcije** (ponavljanje):
 - Načini prijenosa argumenata:
 - “po vrijednosti”, “po adresi”.
 - Prijenos argumenata po vrijednosti u C-u.
 - Prijenos adresa — “varijabilni” argumenti.
- **Rekurzivne funkcije.**
 - Fibonaccijevi brojevi — **NE TAKO** i kako treba.
 - QuickSort algoritam.

Definicija funkcije — ponavljanje

Funkcija je programska cjelina koja

- uzima neke ulazne podatke,
- izvršava određeni niz naredbi,
- i vraća rezultat svog izvršavanja na mjesto poziva.

Definicija funkcije ima oblik:

```
tip_podatka ime_funkcije(tip_1 arg_1,  
                        ..., tip_n arg_n)  
{  
    tijelo funkcije  
}
```

Načini prijenosa argumenata

Formalni i stvarni argumenti (ili parametri):

- Argumenti deklarirani u definiciji funkcije nazivaju se formalni argumenti.
- Izrazi koji se pri pozivu funkcije nalaze na mjestima formalnih argumenata nazivaju se stvarni argumenti.

Veza između formalnih i stvarnih argumenata uspostavlja se

- prijenosom argumenata, prilikom poziva funkcije.

Sasvim općenito, postoje dva načina prijenosa (ili predavanja) argumenata, prilikom poziva funkcije:

- prijenos vrijednosti argumenata — engl. “call by value”,
- prijenos adresa argumenata — engl. “call by reference”.

Prijenos argumenata po vrijednosti

Kod prijenosa **vrijednosti** argumenata,

- funkcija prima **kopije** vrijednosti **stvarnih** argumenata, što znači da

- funkcija **ne može izmijeniti stvarne** argumente.

Stvarni argumenti **mogu** biti **izrazi**. Prilikom poziva funkcije,

- **prvo** se izračuna **vrijednost** tog izraza,

- a **zatim** se ta **vrijednost prenosi** u funkciju,

- i **kopira** u odgovarajući **formalni** argument.

Prijenos argumenata po adresi

Kod prijenosa **adresa** argumenata,

- funkcija prima **adrese stvarnih** argumenata,

što znači da

- funkcija **može izmijeniti stvarne** argumente, tj. **sadržaje** na tim **adresama**.

Stvarni argumenti, u principu, **ne mogu** biti **izrazi**,

- već **samo varijable**,
- odnosno, **objekti** koji **imaju adresu**.

Prijenos argumenata u C-u

U C-u postoji **samo** prijenos argumenata **po vrijednosti**.

- Svaki **formalni** argument ujedno je i **lokalna** varijabla u toj funkciji.
- **Stvarni** argumenti u **pozivu** funkcije su **izrazi** (izračunaj vrijednost, kopiraj ju u **formalni** argument).

Ako funkcijom želimo **promijeniti** vrijednost nekog **podatka** (tzv. “**varijabilni argument**”), pripadni argument

- **treba** biti **pokazivač na taj podatak**, tj. njegova **adresa!**

Tada se **adresa** prenosi **po vrijednosti** — **kopira** u funkciju (promjena te kopije **ne** mijenja **stvarnu** adresu),

- ali smijemo **promijeniti sadržaj** na toj **adresi**, koristeći operator dereferenciranja *****.

Prijenos vrijednosti argumenata

Primjer. Prijenos vrijednosti argumenata (`kvad_1.c`).

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

```
void kvadrat(int x, int y)
{
    y = x*x;
    printf("Unutar funkcije: x = %d, y = %d.\n",
           x, y);
    return;
}
```

Kvadrat od `x` sprema se u lokalnoj varijabli `y`, pa nema traga izvan funkcije `kvadrat`.

Prijenos vrijednosti argumenata (nastavak)

```
int main(void) {
    int x = 3, y = 5;

    printf("Prije poziva: x = %d, y = %d.\n", x, y);
    kvadrat(x, y);
    printf("Nakon poziva: x = %d, y = %d.\n", x, y);
    return 0;
}
```

Rezultat izvršavanja programa je:

Prije poziva: x = 3, y = 5.

Unutar funkcije: x = 3, y = 9.

Nakon poziva: x = 3, y = 5.

Prijenos adresa argumenata

Primjer. Prijenos **adresa** argumenata (**kvad_2.c**).

```
#include <stdio.h>

void kvadrat(int *x, int *y)
{
    *y = *x**x; /* = (*x) * (*x). */
    printf("Unutar funkcije: x = %d, y = %d.\n",
           *x, *y);
    return;
}
```

Kvadriramo **sadržaj** od **x** i spremamo ga u **sadržaj** od **y**, pa **ostaje** trag **izvan** funkcije **kvadrat** — **mijenja** se ***y**.

Prijenos adresa argumenata (nastavak)

```
int main(void) {
    int x = 3, y = 5;

    printf("Prije poziva: x = %d, y = %d.\n", x, y);
    kvadrat(&x, &y);
    printf("Nakon poziva: x = %d, y = %d.\n", x, y);
    return 0;
}
```

Rezultat izvršavanja programa je:

Prije poziva: x = 3, y = 5.

Unutar funkcije: x = 3, y = 9.

Nakon poziva: x = 3, y = 9.

Napomene uz primjer

U **prvom** primjeru

```
void kvadrat(int x, int y)
```

`x` i `y` su lokalne varijable tipa `int`.

U **drugom** primjeru

```
void kvadrat(int *x, int *y)
```

`x` i `y` su lokalne varijable tipa `int *`, tj. **pokazivači** na `int`.

Nije lijepo da se **razne** stvari **isto** zovu! Recimo, `px` i `py` bi bilo bolje u drugom primjeru.

“**Prava**” realizacija bi bila

```
void kvadrat(int x, int *py)
```

jer `x` ne mijenjamo!

Korektni prijenos argumenata

Primjer. Korektni prijenos argumenata — `y` je “**varijabilni**” argument, pa prenosimo **adresu** `py` (`kvad_3.c`).

```
#include <stdio.h>

void kvadrat(int x, int *py)
{
    *py = x*x;
    printf("Unutar funkcije: x = %d, y = %d.\n",
           x, *py);
    return;
}
```

Kvadrat od `x` spremamo u **sadržaj** od `py`, pa **ostaje** trag **izvan** funkcije `kvadrat` — **mijenja** se `*py`.

Korektni prijenos argumenata (nastavak)

```
int main(void) {
    int x = 3, y = 5;

    printf("Prije poziva: x = %d, y = %d.\n", x, y);
    kvadrat(x, &y);
    printf("Nakon poziva: x = %d, y = %d.\n", x, y);
    return 0;
}
```

Rezultat izvršavanja programa je:

Prije poziva: x = 3, y = 5.

Unutar funkcije: x = 3, y = 9.

Nakon poziva: x = 3, y = 9.

Korektni prijenos argumenata (nastavak)

Potpuni pregled stanja stvari dobivamo ispisom adresa, vrijednosti i sadržaja na adresama (`kvad_p3.c`):

U glavnom programu (funkcija `main`):

adresa od `x` (`&x`) = 0130C000

adresa od `y` (`&y`) = 0130C004

Prije poziva funkcije:

vrijednost od `x` (`x`) = 3

vrijednost od `y` (`y`) = 5

Unutar funkcije `kvadrat`:

adresa od `x` (`&x`) = 0019FDE8

adresa od `py` (`&py`) = 0019FDF0

vrijednost od `x` (`x`) = 3

vrijednost od `py` (`py`) = 0130C004

sadržaj od `py` (`*py`) = 9

Korektni prijenos argumenata (nastavak)

Nakon poziva funkcije:

vrijednost od `x` (`x`) = 3

vrijednost od `y` (`y`) = 9

Komentar. Prethodni primjeri služe **samo** za **ilustraciju**.

Naravno, jedina **razumna** realizacija funkcije za kvadrat je

- 🔴 funkcija koja prima **jedan** argument i **vraća** kvadrat tog argumenta.
-

```
int kvadrat(int x)
{
    return x*x;
}
```

Rekurzivne funkcije

Rekurzivne funkcije

Programski jezik C dozvoljava tzv. rekurzivne funkcije, tj.

- da funkcija poziva samu sebe.

U pravilu,

- rekurzivni algoritmi su kraći,
- ali izvođenje, u načelu, traje dulje.

Katkad — puno dulje, ako puno puta računamo istu stvar.
Zato oprez!

Napomena. Svaki rekurzivni algoritam mora imati

- “nerekurzivni” dio, koji omogućava prekidanje rekurzije.

Najčešće je to neki if u inicijalizaciji rekurzije.

Fibonaccijski brojevi

Fibonaccijski brojevi

Primjer. Osim faktorijela, drugi standardni primjer rekurzivne funkcije su Fibonaccijski brojevi, definirani rekurzijom

$$F_n = F_{n-1} + F_{n-2}, \quad n \geq 2, \quad \text{uz} \quad F_0 = 0, \quad F_1 = 1.$$

Po definiciji, možemo napisati rekurzivnu funkciju:

```
long int fib(int n)
{
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib(n - 1) + fib(n - 2);
}
```

Može i `long unsigned`. Ali, nemojte to raditi. **Zabranjujem!**

Fibonaccijevi brojevi (nastavak)

Ovdje je broj rekurzivnih poziva **ogroman** i **veći** od samog broja F_n .

Ne vjerujete? Dodajmo funkciji **globalni** brojač poziva `broj_poziva` (`fib_r.c`).

```
long int fib(int n)
{
    ++broj_poziva;    /* Globalni brojac poziva. */
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib(n - 1) + fib(n - 2);
}
```

Za $n = 20$ rezultat je $F_{20} = 6765$, a za računanje treba **21891** poziv funkcije!

Fibonaccijski brojevi petljom

Zadatak. Dokažite da je broj **rekurzivnih** poziva funkcije **fib** za računanje F_n , uz $n \geq 2$, jednak $(F_1 + \dots + F_n) + F_{n-1}$.

Uputa: **fib(k)** se poziva F_{n+1-k} puta, za $k = 1, \dots, n$, a **fib(0)** se poziva F_{n-1} puta! ■

Ovo, kao i faktorijske, ide **puno brže** običnom **petljom**:

● **novi** član je **zbroj** prethodna dva, uz “**pomak**” članova.

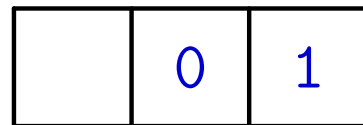
Za realizaciju tog algoritma trebamo “**prozor**” od samo **3** **susjedna** člana niza:

- **fn** = **novi** član,
- **fp** = **prošli** član,
- **fpp** = **pretprošli** član.

Fibonaccijski brojevi

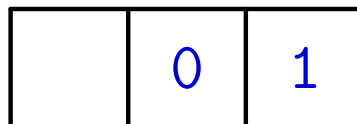
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_1):



fp fn

Što se stvarno zbiva s prozorom: $fp = F_0$, $fn = F_1$

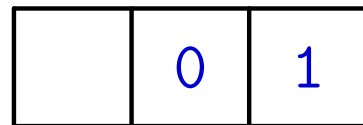


fp fn

Fibonaccijski brojevi

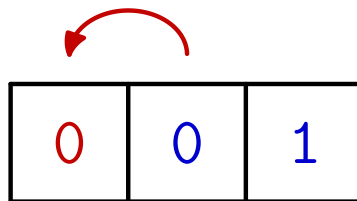
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_1):



fp fn

Što se stvarno zbiva s prozorom: $f_{pp} = F_0$



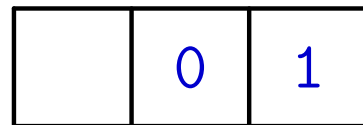
fpp fp fn

$f_{pp} = fp$

Fibonaccijski brojevi

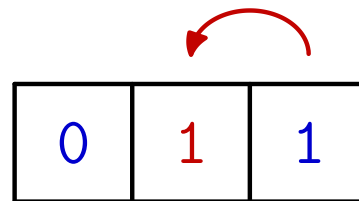
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_1):



fp fn

Što se stvarno zbiva s prozorom: fp = F_1



fp = fn

fpp fp fn

Fibonaccijski brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_2):

0	1	1
---	---	---

fpp fp fn

Što se stvarno zbiva s prozorom: $fn = F_0 + F_1 = F_2$

0	1	1
---	---	---

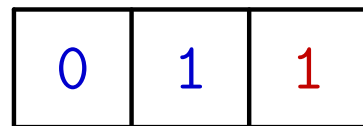
fn = fp + fpp

fpp fp fn

Fibonaccijski brojevi

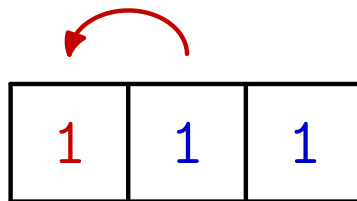
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_2):



fpp fp fn

Što se stvarno zbiva s prozorom: fpp = F_1



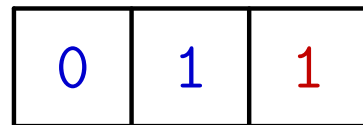
fpp = fp

fpp fp fn

Fibonaccijski brojevi

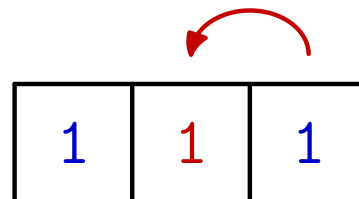
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_2):



fpp fp fn

Što se stvarno zbiva s prozorom: fp = F_2



fp = fn

fpp fp fn

Fibonaccijski brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_3):

0	1	1	2
---	---	---	---

fpp fp fn

Što se stvarno zbiva s prozorom: $fn = F_1 + F_2 = F_3$

1	1	2
---	---	---

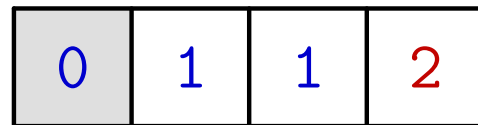
fn = fp + fpp

fpp fp fn

Fibonaccijski brojevi

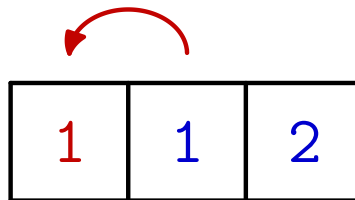
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_3):



fpp fp fn

Što se stvarno zbiva s prozorom: $fpp = F_2$



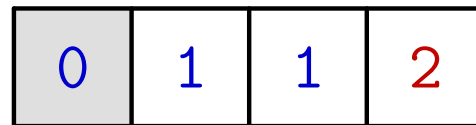
fpp = fp

fpp fp fn

Fibonaccijski brojevi

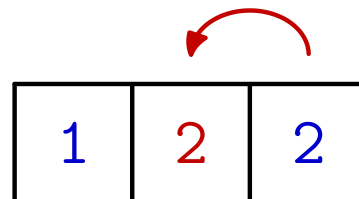
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_3):



fpp fp fn

Što se stvarno zbiva s prozorom: fp = F_3



fp = fn

fpp fp fn

Fibonaccijski brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_4):

0	1	1	2	3
---	---	---	---	---

fpp fp fn

Što se stvarno zbiva s prozorom: $fn = F_2 + F_3 = F_4$

1	2	3
---	---	---

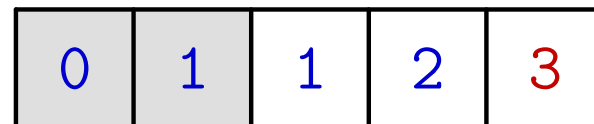
fn = fp + fpp

fpp fp fn

Fibonaccijski brojevi

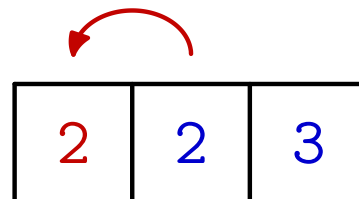
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_4):



fpp fp fn

Što se stvarno zbiva s prozorom: $fpp = F_3$



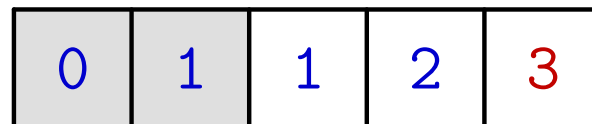
fpp fp fn

fpp = fp

Fibonaccijski brojevi

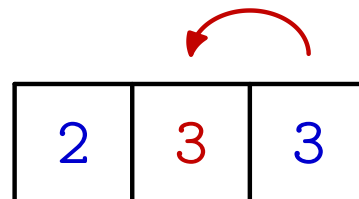
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_4):



fpp fp fn

Što se stvarno zbiva s prozorom: $fp = F_4$



fp = fn

fpp fp fn

Fibonaccijski brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_5):

0	1	1	2	3	5
---	---	---	---	---	---

fpp fp fn

Što se stvarno zbiva s prozorom: $fn = F_3 + F_4 = F_5$

2	3	5
---	---	---

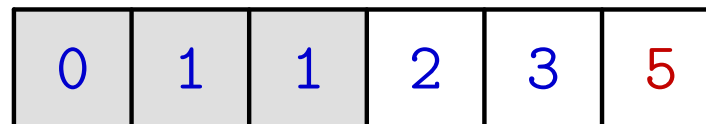
fn = fp + fpp

fpp fp fn

Fibonaccijski brojevi

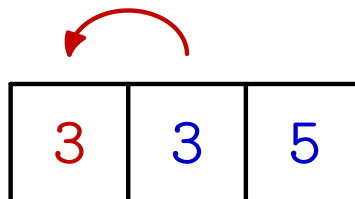
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_5):



fpp fp fn

Što se stvarno zbiva s prozorom: fpp = F_4



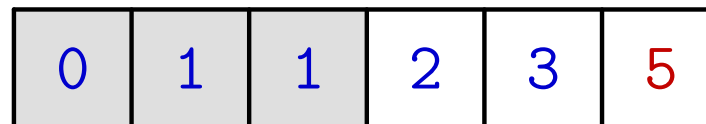
fpp = fp

fpp fp fn

Fibonaccijski brojevi

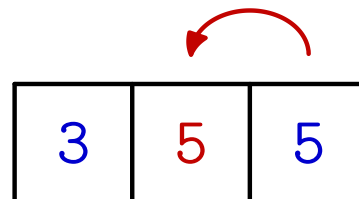
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_5):



fpp fp fn

Što se stvarno zbiva s prozorom: $fp = F_5$



fp = fn

fpp fp fn

Fibonaccijski brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_6):

0	1	1	2	3	5	8
---	---	---	---	---	---	---

fpp fp fn

Što se stvarno zbiva s prozorom: $fn = F_4 + F_5 = F_6$

3	5	8
---	---	---

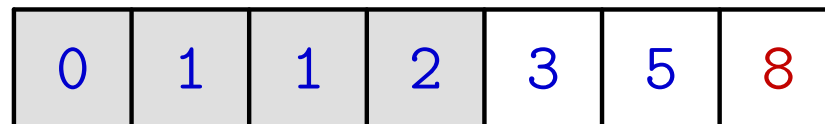
fn = fp + fpp

fpp fp fn

Fibonaccijski brojevi

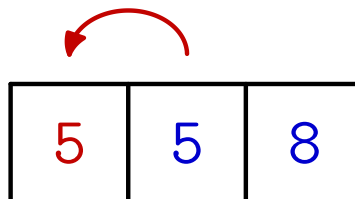
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_6):



fpp fp fn

Što se stvarno zbiva s prozorom: $fpp = F_5$



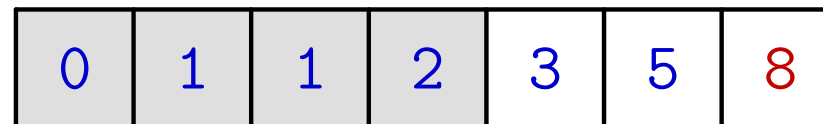
fpp = fp

fpp fp fn

Fibonaccijski brojevi

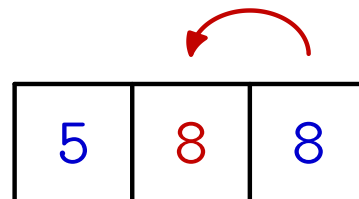
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_6):



fpp fp fn

Što se stvarno zbiva s prozorom: $fp = F_6$



fp = fn

fpp fp fn

Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_7):

0	1	1	2	3	5	8	13
---	---	---	---	---	---	---	----

fpp fp fn

Što se stvarno zbiva s prozorom: $fn = F_5 + F_6 = F_7$

5	8	13
---	---	----

fn = fp + fpp

fpp fp fn

Fibonaccijski brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_7):

0	1	1	2	3	5	8	13
---	---	---	---	---	---	---	----

fpp fp fn

Što se stvarno zbiva s prozorom: $fpp = F_6$

8	8	13
---	---	----

fpp = fp

fpp fp fn

Fibonaccijski brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_7):

0	1	1	2	3	5	8	13
---	---	---	---	---	---	---	----

fpp fp fn

Što se stvarno zbiva s prozorom: $fp = F_7$

8	13	13
---	----	----

fp = fn

fpp fp fn

Fibonaccijski brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $F_0 = 0$, $F_1 = 1$.

Prozor širine 3 susjeda “putuje” nizom (zadnji je F_8):

0	1	1	2	3	5	8	13	21
---	---	---	---	---	---	---	----	----

fpp fp fn

Što se stvarno zbiva s prozorom: $fn = F_6 + F_7 = F_8$

8	13	21
---	----	----

fn = fp + fpp

fpp fp fn

Fibonaccijski brojevi petljom (nastavak)

Iterativna (nerekurzivna) verzija funkcije za Fibonaccijeve brojeve (`fib_a.c`).

```
long int fibonacci(int n)
{
    long int f_n, f_p, f_pp; /* Namjerno NE inic.*/
    int i;

    if (n == 0) return 0; /* F[0] */
    if (n == 1) return 1; /* F[1] */

    /* Sad inicijaliziramo prva dva.
       Inicijalizacija odgovara
       stanju za n = 1 (a ne 2). */
```

Fibonaccijski brojevi petljom (nastavak)

```
f_p = 0;  /* Prošli F[0] */
f_n = 1;  /* Ovaj    F[1] */

for (i = 2; i <= n; ++i) {
    f_pp = f_p;          /* F[i - 2] */
    f_p  = f_n;          /* F[i - 1] */
    f_n  = f_p + f_pp;   /* F[i]    */
}

return f_n;
}
```

Fibonaccijski brojevi (kraj)

Ima još **puno brži** algoritam za računanje F_n ,

- složenost mu je $O(\log n)$, a ne $O(n)$,

ali se **ne isplati** za **male** n .

Naime, **najveći** prikazivi Fibonaccijev broj na 32 bita

- u tipu **int** (i u tipu **long int**) je $F_{46} = 1\,836\,311\,903$,

- a u tipu **unsigned** (može i **long**) je $F_{47} = 2\,971\,215\,073$.

Dakle, **korektne** rezultate dobivamo samo za $n \leq 46$ (ili 47), a tad je **dovoljno brz** i obični **aditivni** algoritam.

Usput, **najveći** prikazivi Fibonaccijev broj na 64 bita

- u tipu **long int** je $F_{92} = 7\,540\,113\,804\,746\,346\,429$,

- u **long unsigned** je $F_{93} = 12\,200\,160\,415\,121\,876\,738$.

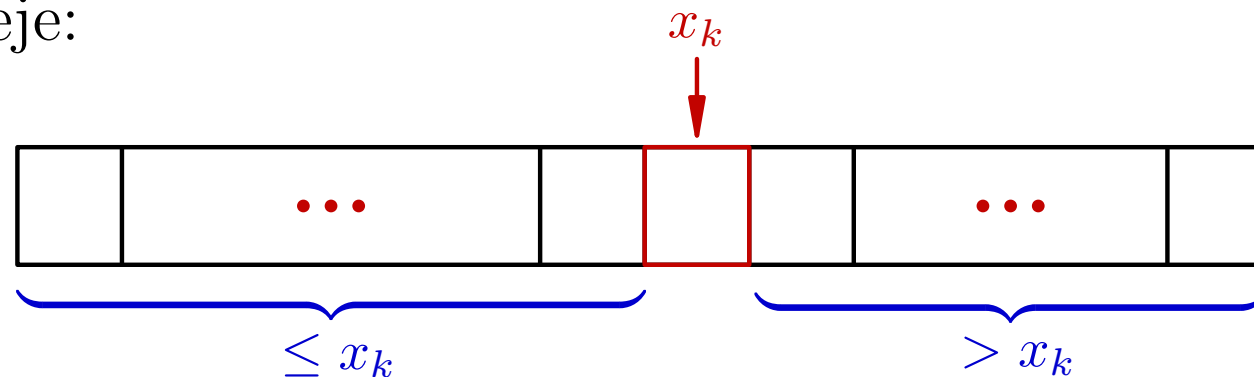
QuickSort algoritam

QuickSort — uvod i skica algoritma

QuickSort se temelji na principu “**podijeli, pa vladaj**”.

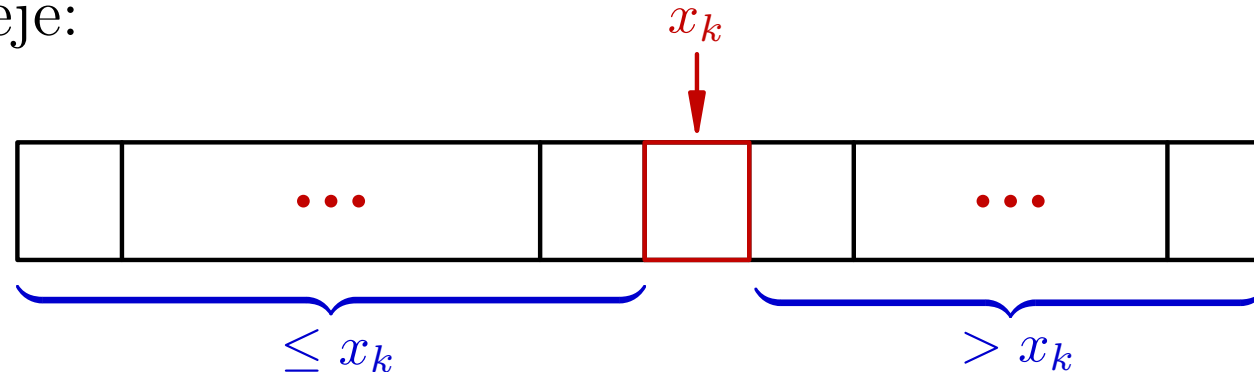
- Uzmemo **jedan** element x_k iz niza (tzv. **ključni** element) i dovedemo ga na njegovo **pravo** mjesto u nizu.
- Lijevo** od njega ostavimo elemente koji su **manji** ili **jednaki** njemu (u bilo kojem poretku).
- Desno** od njega ostavimo elemente koji su **veći** od njega (u bilo kojem poretku).

Skica ideje:



QuickSort — uvod i skica algoritma (nastavak)

Skica ideje:



“Podijeli, pa vladaj” = **sortiraj** lijevi i desni podniz (bez x_k).

- Ako smo **dobro** izabrali, tj. ako je **pravo** mjesto x_k blizu **sredine** niza, onda ćemo morati sortirati (rekurzivno)
 - **dva** manja niza, približno **polovične** duljine.
- U **najgorem** slučaju, ako smo izabrali “**krivi**” x_k — dođe na “**rub**”, morat ćemo sortirati **jedan** niz duljine $n - 1$.

QuickSort — razrada algoritma

U danom trenutku, rekurzivna funkcija za QuickSort treba sortirati nesređeni dio niza

- između “lijevog” indeksa l i “desnog” indeksa d .

Ta dva indeksa (i polje) su argumenti funkcije.

Posla ima ako i samo ako

- taj dio niza ima barem 2 elementa, tj. ako je $l < d$.

Za tzv. ključni element, najčešće se uzima $k = l$, tj.

- “prvi” element x_l treba dovesti na njegovo pravo mjesto u tom komadu niza.

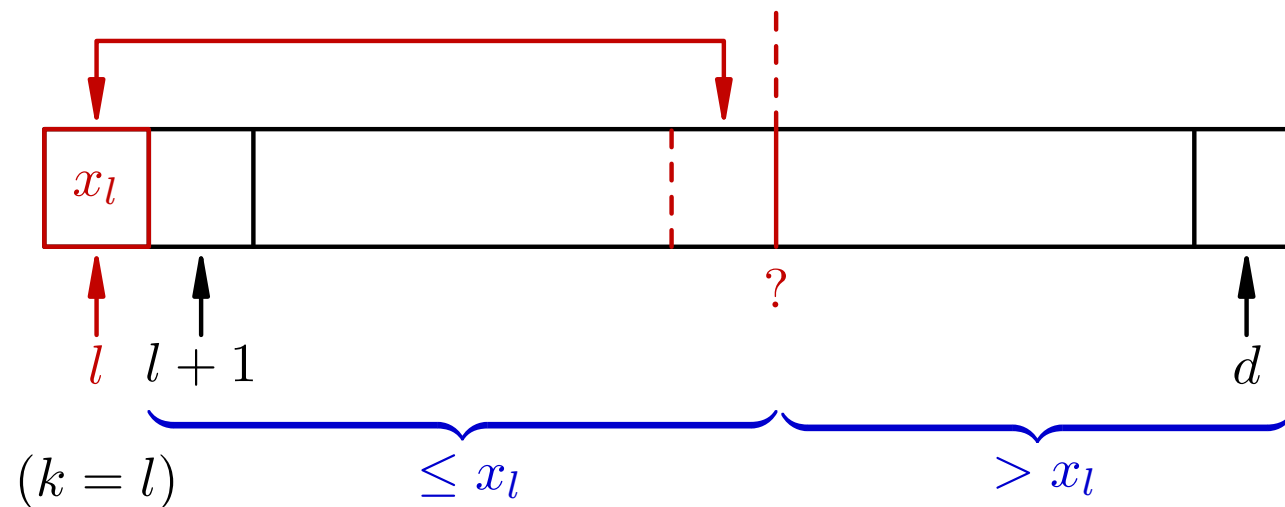
Razlog: element x_l služi kao “branik” na lijevom rubu niza.

QuickSort — razrada algoritma (nastavak)

Dogovor:

- lijevo u nizu (ispred njegove prave pozicije) stavljamo elemente koji su **manji ili jednaki** x_l ,
- desno u nizu (iza njegove prave pozicije) stavljamo elemente koji su **strogo veći** od x_l .

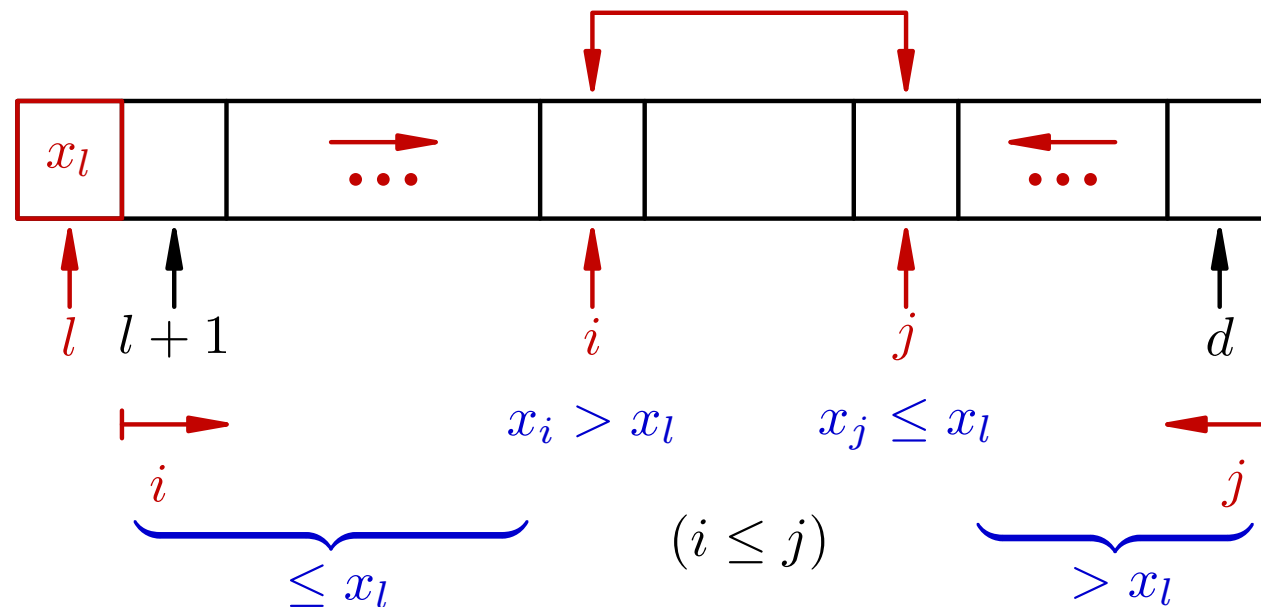
Tada će pravo mjesto elementa x_l biti **zadnje** u **lijevom** dijelu.



QuickSort — razrada algoritma (nastavak)

Kako se traži “pravo” mjesto elementa x_l ?

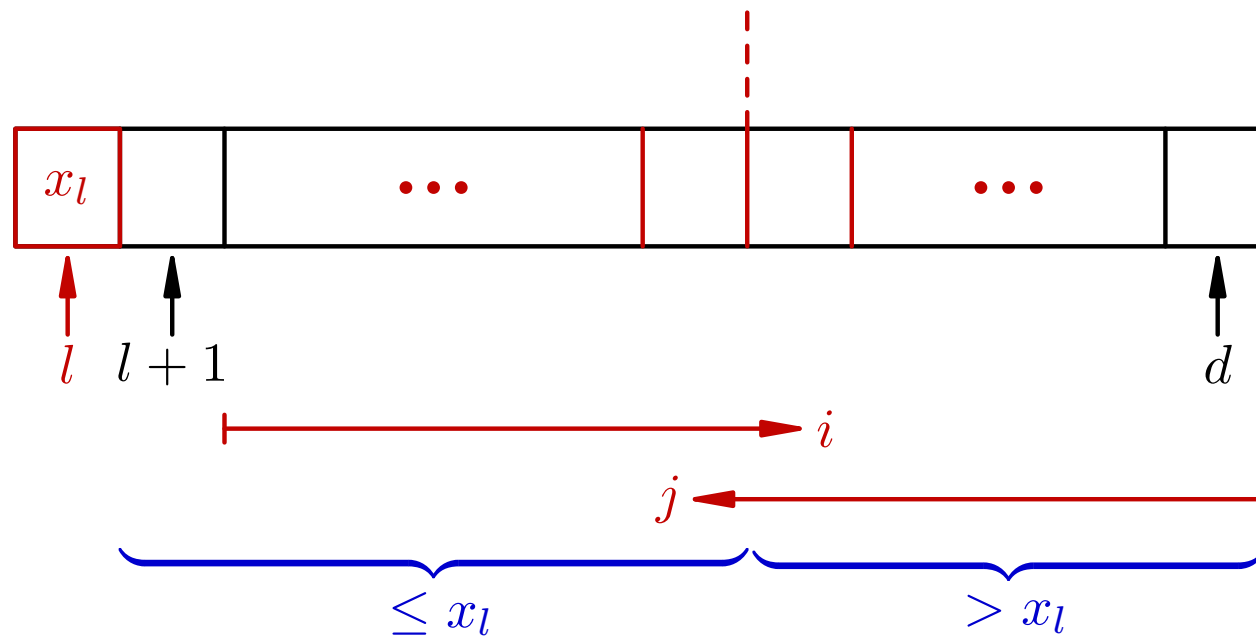
- Dvostranim pretraživanjem po ostatku niza.
- Sa svake strane (lijeve i desne) tražimo prvi sljedeći element koji “ne spada” na tu stranu niza.
- Ako nađemo takav par — zamijenimo im mjesta!



QuickSort — razrada algoritma (nastavak)

Kraj dvostrane pretrage — kad smo gotovi?

- Indeksi i i j moraju se “**preklopiti**” — stići u **obratni** poredak $j < i$.
- **Pravo** mjesto elementa x_l je na **indeksu** j , pa napravimo zamjenu (ako treba).



QuickSort — razrada algoritma (nastavak)

Algoritam za dvostrano pretraživanje:

```
if (l < d) {
    i = l + 1;
    j = d;

    /* Prolaz mora i za i == j */
    while (i <= j) {
        while (i <= d && x[i] <= x[l]) ++i;
        while (x[j] > x[l]) --j;
        if (i < j) swap(&x[i], &x[j]);
    }
```

Uočiti: S desne strane (po j) **ne treba** provjera $j > l$, jer x_l služi kao “**branik**” — sigurno prekida petlju za $j = l$.

QuickSort — razrada algoritma (nastavak)

Iza toga treba još:

- dovesti element x_l na njegovo pravo mjesto — indeks tog mjesta je j ,
- rekurzivno sortirati lijevi i desni podniz, bez x_j .

```
    if (l < j) swap(&x[j], &x[l]);
    quick_sort(x, l, j - 1);
    quick_sort(x, j + 1, d);
}    /* Kraj if (l < d). */
```

QuickSort — primjer

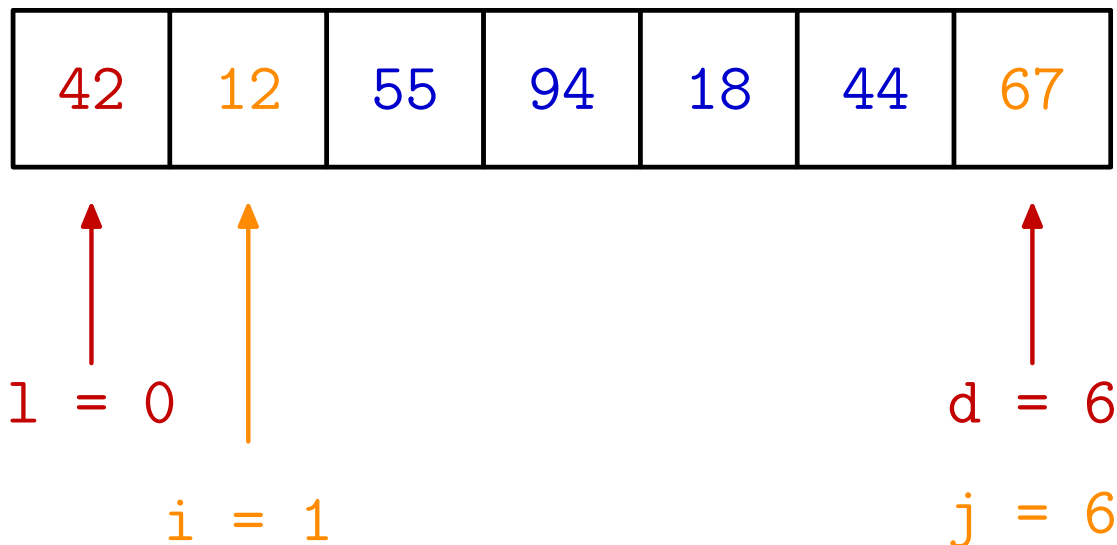
Primjer. Quicksort algoritmom sortirajte zadano polje.

42	12	55	94	18	44	67
----	----	----	----	----	----	----

Sortiramo cijeli niz $[x_0, x_1, x_2, x_3, x_4, x_5, x_6]$.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.



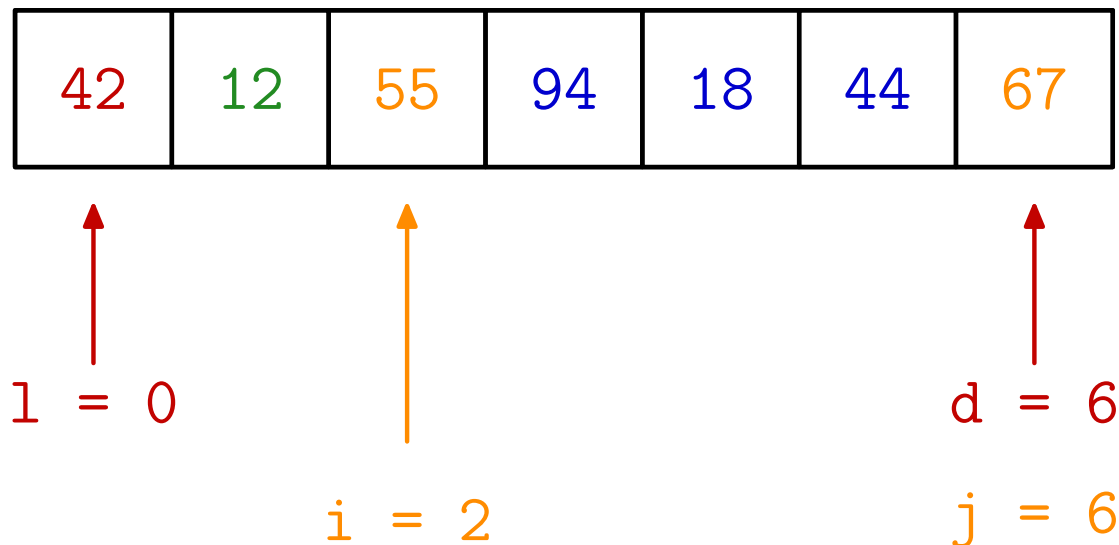
Prvi element 42 je ključni element.

Početak dvostrane pretrage. Krećemo s lijeve strane.

12 je na dobroj strani (≤ 42) — idemo dalje.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.



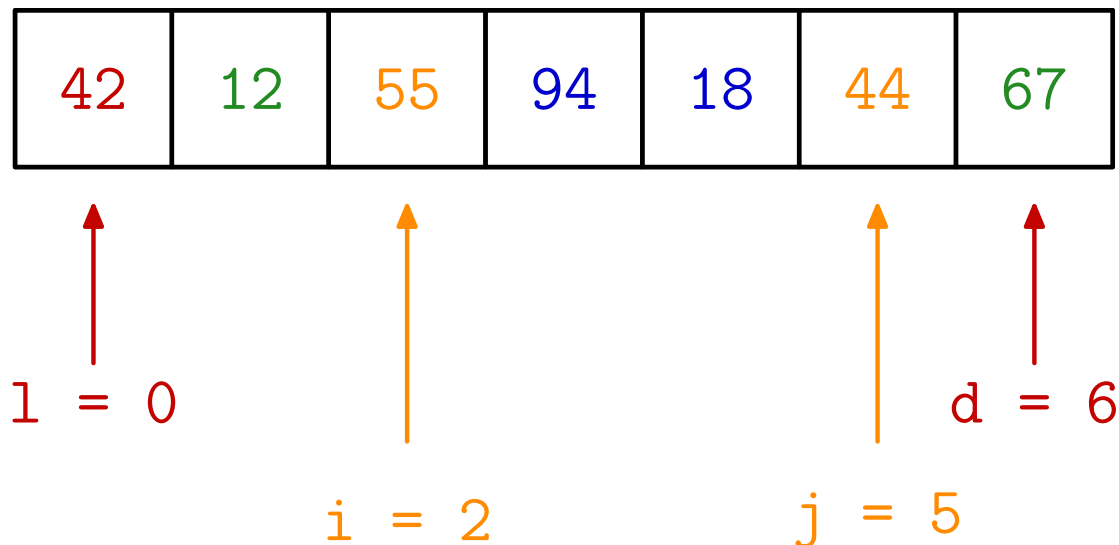
55 je na **krivoj** strani (> 42) — stop s lijeve strane.

Krećemo s **desne** strane.

67 je na **dobroj** strani (> 42) — idemo dalje.

QuickSort — primjer

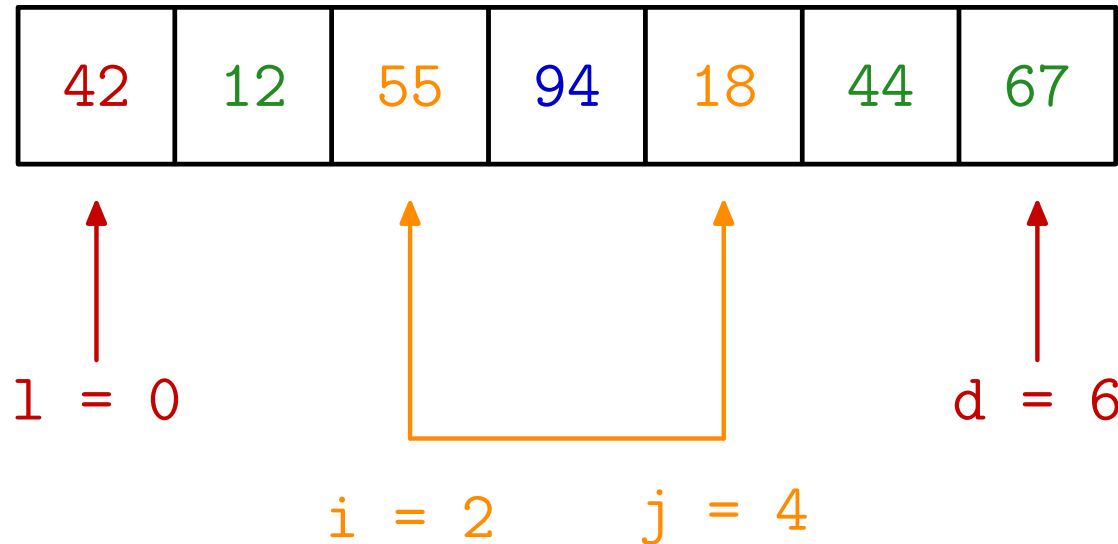
Primjer. Quicksort algoritmom sortirajte zadano polje.



44 je na dobroj strani (> 42) — idemo dalje.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.

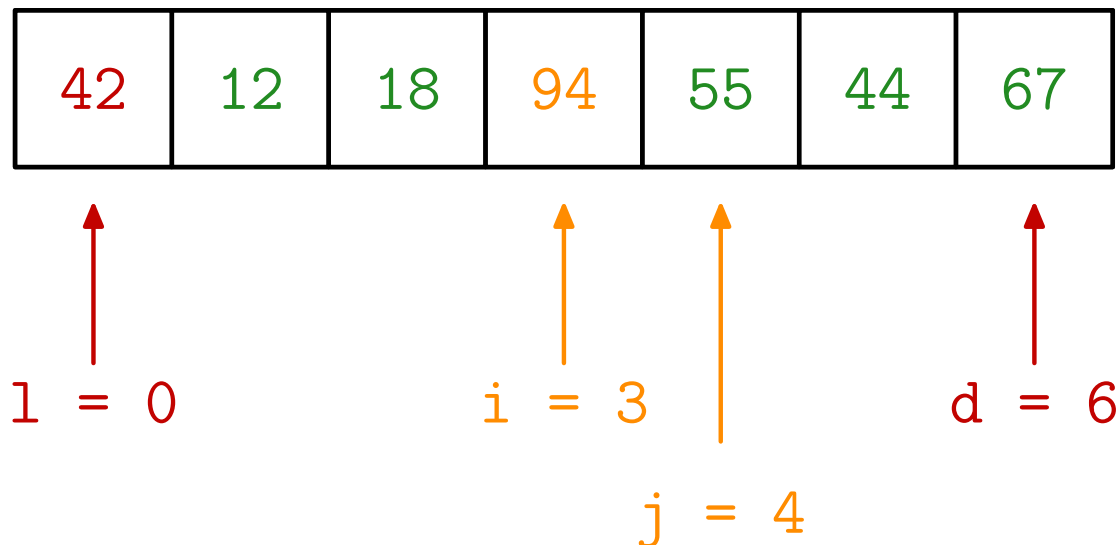


18 je na **krivoj** strani (≤ 42) — stop s desne strane.

$i < j \implies$ zamjena para elemenata na krivim stranama.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.

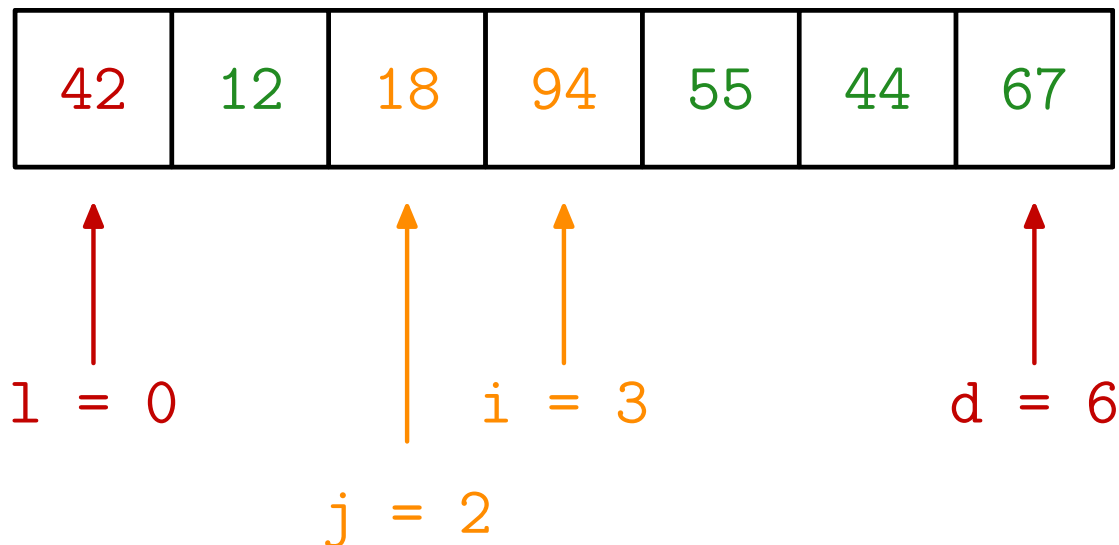


Nastavak dvostrane pretrage s lijeve strane.

94 je na **krivoj** strani (> 42) — stop s lijeve strane.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.

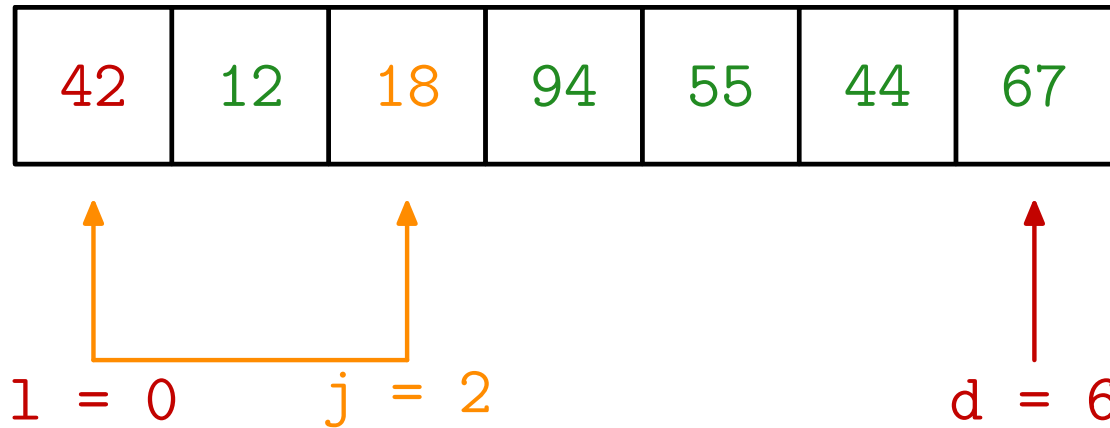


18 je na **krivoj** strani (≤ 42) — stop s desne strane.

$j < i \implies$ nema zamjene, kraj dvostrane pretrage.

QuickSort — primjer

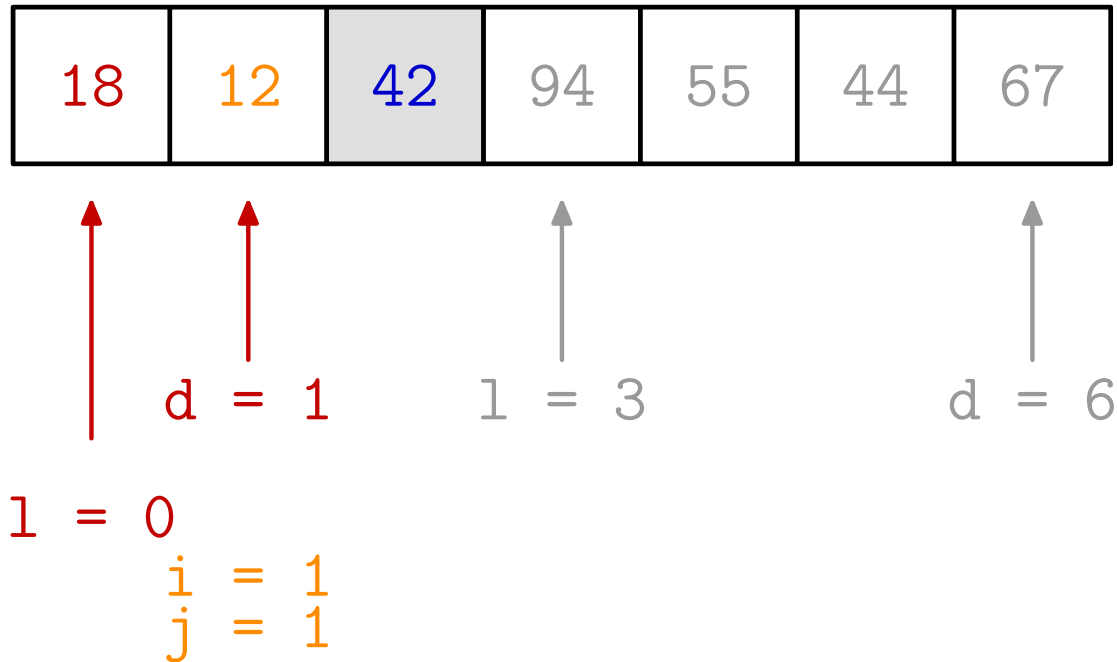
Primjer. Quicksort algoritmom sortirajte zadano polje.



$l < j \implies$ zamjena x_l i x_j .

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.



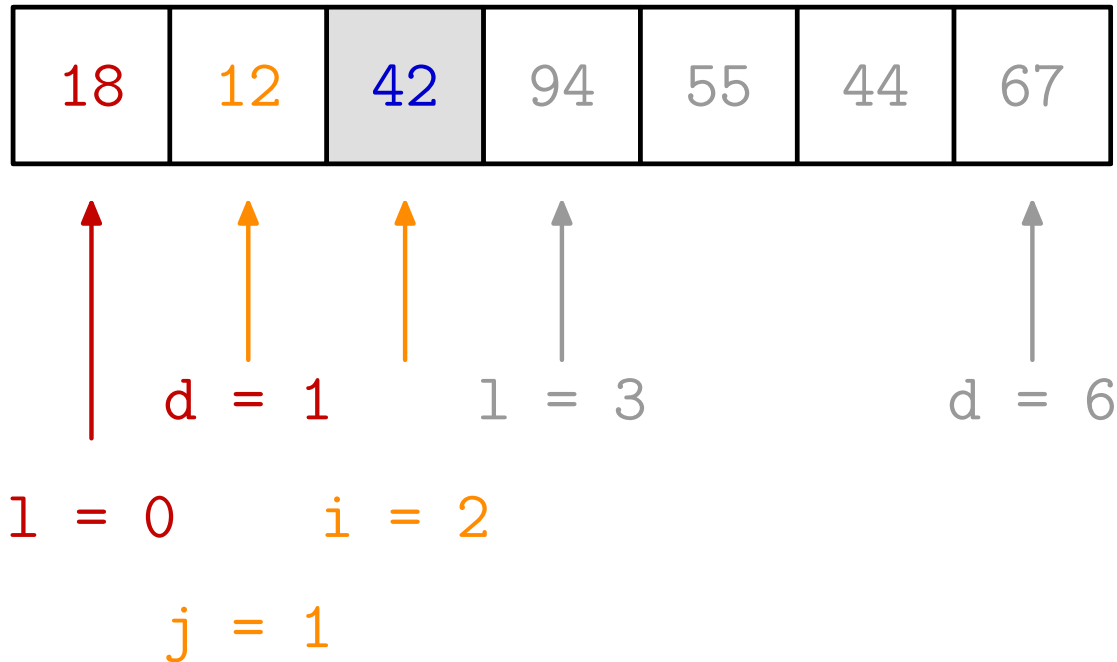
Sortiramo $[x_0, x_1]$. 18 je ključni element.

Početak dvostrane pretrage. Krećemo s lijeve strane.

12 je na dobroj strani (≤ 18) — idemo dalje.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.



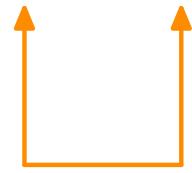
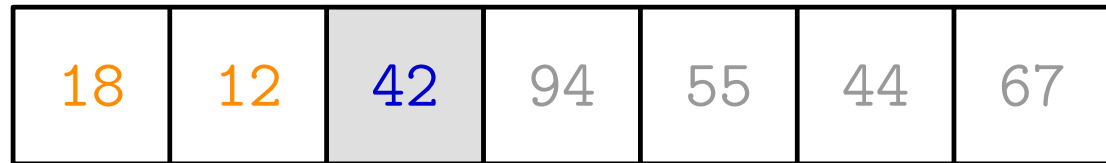
$i > d$ — stop s lijeve strane. Krećemo s **desne** strane.

12 je na **krivoj** strani (≤ 18) — stop s desne strane.

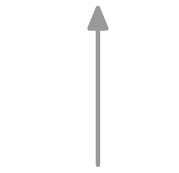
$j < i \implies$ nema zamjene, kraj dvostrane pretrage.

QuickSort — primjer

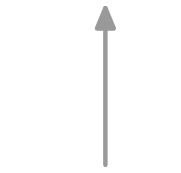
Primjer. Quicksort algoritmom sortirajte zadano polje.



$$d = 1$$



$$l = 3$$



$$d = 6$$

$$l = 0$$

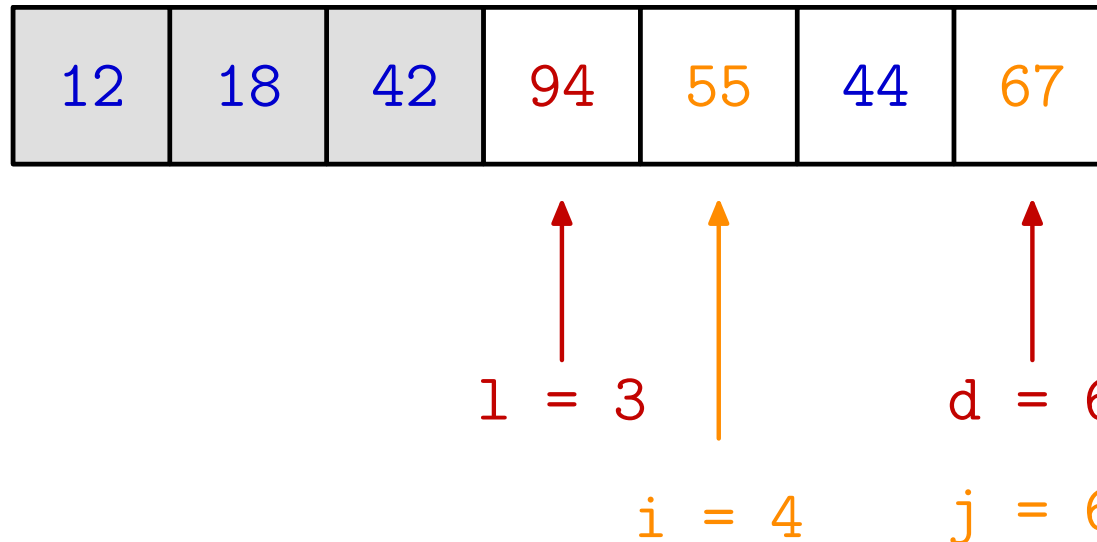
$$j = 1$$

$l < j \implies$ zamjena x_l i x_j . Pravo mjesto za 18 je x_1 .

Lijevi podniz je $[x_0]$, a desni je prazan — oba rekurzivna poziva se odmah vrata, jer nema posla. Gotovi s $[x_0, x_1]$.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.



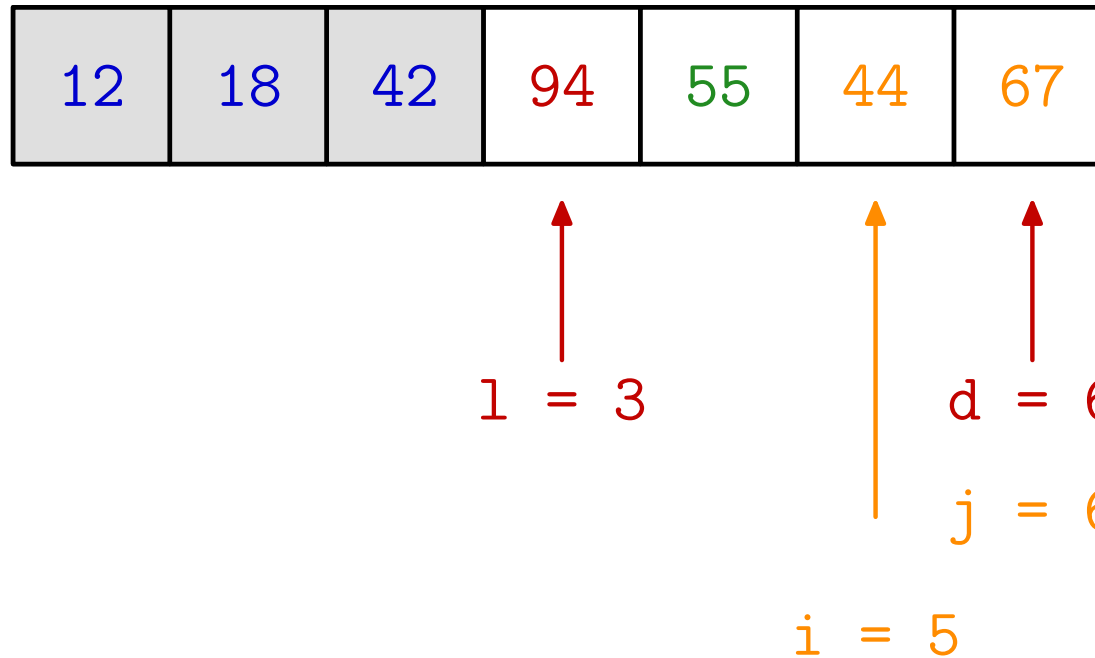
Sortiramo $[x_3, x_4, x_5, x_6]$. 94 je ključni element.

Početak dvostrane pretrage. Krećemo s lijeve strane.

55 je na dobroj strani (≤ 94) — idemo dalje.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.



44 je na dobroj strani (≤ 94) — idemo dalje.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.

12	18	42	94	55	44	67
----	----	----	----	----	----	----

↑
l = 3

↑
d = 6
j = 6
i = 6

67 je na dobroj strani (≤ 94) — idemo dalje.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.

12	18	42	94	55	44	67
----	----	----	----	----	----	----

↑
 $l = 3$

↑
 $d = 6$

$j = 6$

↑
 $i = 7$

$i > d$ — stop s lijeve strane. Krećemo s **desne** strane.

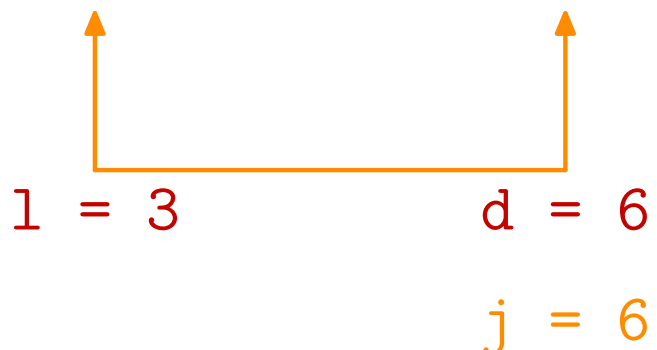
67 je na **krivoj** strani (≤ 94) — stop s desne strane.

$j < i \implies$ nema zamjene, kraj dvostrane pretrage.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.

12	18	42	94	55	44	67
----	----	----	----	----	----	----



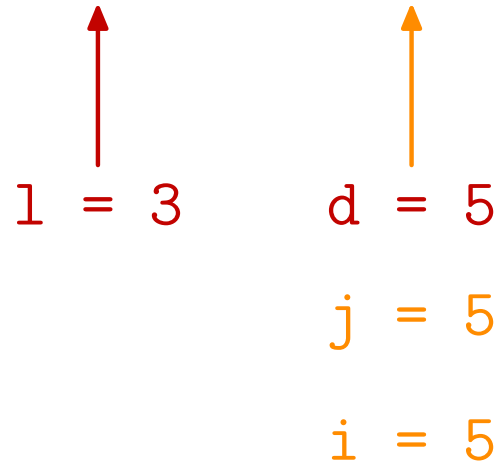
$l < j \implies$ zamjena x_l i x_j . Pravo mjesto za 94 je x_6 .

Lijevi podniz je $[x_3, x_4, x_5]$, a desni je prazan.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.

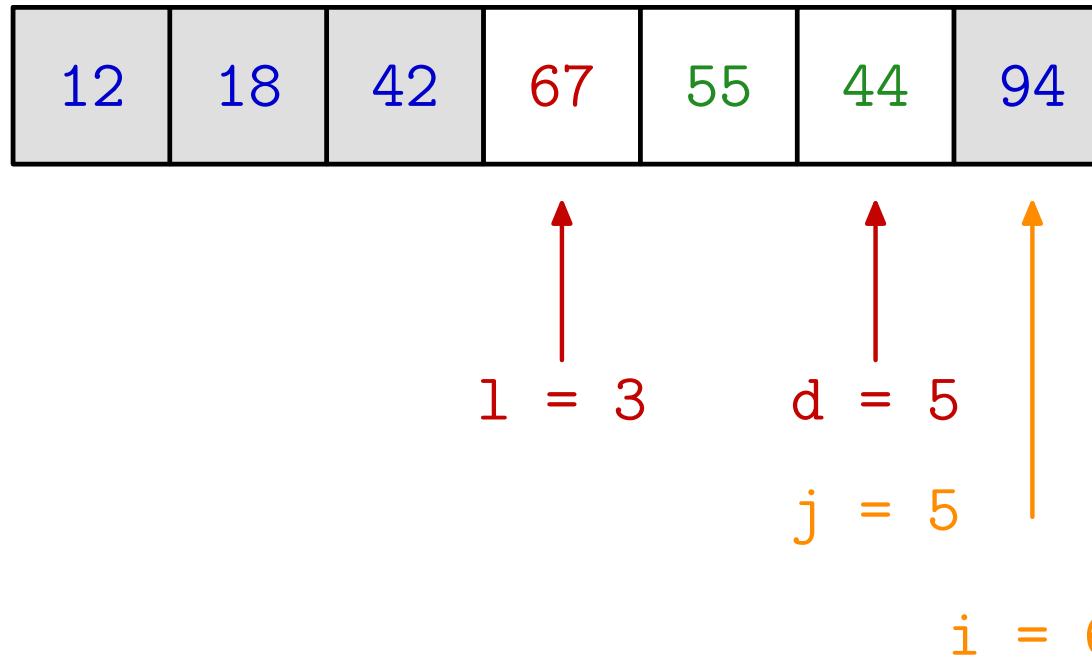
12	18	42	67	55	44	94
----	----	----	----	----	----	----



44 je na dobroj strani (≤ 67) — idemo dalje.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.



$i > d$ — stop s lijeve strane. Krećemo s **desne** strane.

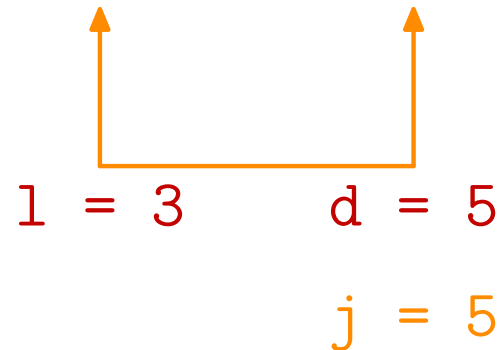
44 je na **krivoj** strani (≤ 67) — stop s desne strane.

$j < i \implies$ nema zamjene, kraj dvostrane pretrage.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.

12	18	42	67	55	44	94
----	----	----	----	----	----	----



$l < j \implies$ zamjena x_l i x_j . Pravo mjesto za 67 je x_5 .

Lijevi podniz je $[x_3, x_4]$, a desni je prazan.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.

12	18	42	44	55	67	94
----	----	----	----	----	----	----

$$\begin{array}{c} \uparrow \qquad \uparrow \\ d = 4 \\ \\ l = 3 \\ \begin{array}{l} i = 4 \\ j = 4 \end{array} \end{array}$$

Sortiramo $[x_3, x_4]$. 44 je ključni element.

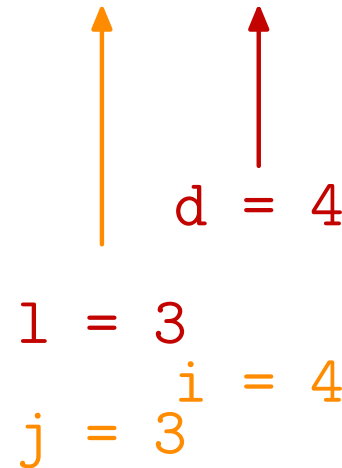
Slijeva: 55 je na **krivoj** strani (> 44) — stop s lijeve strane.

Zdesna: 55 je na **dobroj** strani (> 44) — idemo dalje.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.

12	18	42	44	55	67	94
----	----	----	----	----	----	----



44 je na **krivoj** strani (≤ 44) — stop s desne strane (**branik**).

$j < i \implies$ nema zamjene, kraj dvostrane pretrage.

$l = j \implies$ nema zamjene x_l i x_j . Pravo mjesto za 44 je x_3 .

Lijevi podniz je prazan, a desni je $[x_4]$ — nema posla, **gotovo**.

QuickSort — primjer

Primjer. Quicksort algoritmom sortirajte zadano polje.

12	18	42	44	55	67	94
----	----	----	----	----	----	----

QuickSort — složenost

Za složenost vrijedi:

- prosječna složenost je $O(n \log_2 n)$, za slučajne dobro razbacane nizove,
- složenost u najgorem slučaju je $O(n^2)$, za već sortirani i naopako sortirani niz.

Autor QuickSort-a je C. A. R. Hoare, 1962. godine.

U nastavku je dan cijeli program (`qsort_1.c`).

QuickSort — funkcija swap

```
#include <stdio.h>

/* Sortiranje QuickSort algoritmom.
   Prvi element x[l] je ključni element
   i dovodimo ga na pravo mjesto u polju. */

void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
    return;
}
```

QuickSort — funkcija quick_sort

```
void quick_sort(int x[], int l, int d)
{
    int i, j;

    if (l < d) {
        i = l + 1;
        j = d;

        /* Prolaz mora i za i == j */
        while (i <= j) {
            while (i <= d && x[i] <= x[l]) ++i;
            while (x[j] > x[l]) --j;
            if (i < j) swap(&x[i], &x[j]);
        }
    }
}
```

QuickSort — funkcija quick_sort (nastavak)

```
        if (l < j) swap(&x[j], &x[l]);
        quick_sort(x, l, j - 1);
        quick_sort(x, j + 1, d);
    }

    return;
}
```

QuickSort — glavni program

```
int main(void) {
    int i, n;
    int x[] = {42, 12, 55, 94, 18, 44, 67};

    n = 7;
    quick_sort(x, 0, n - 1);

    printf("\n Sortirano polje x:\n");
    for (i = 0; i < n; ++i) {
        printf(" x[%d] = %d\n", i, x[i]);
    }
    return 0;
}
```

QuickSort — poboljšanja

Poboljšanja “našeg” jednostavnog algoritma:

- Za $n = 2, 3$ — sort izravno, provjerom zamjena.

Ako je duljina polja $n > 3$, onda za ključni element

- uzmi “srednjeg” od neka 3 elementa (ubrzanje oko 30%).

Kontrola “dubine” rekurzije:

- Odmah obradi kraće od preostala dva polja,
- a dulje polje ide na tzv. programski stog (engl. stack).

Ima još raznih “trikova”, pa se nemojte čuditi da je tako “ispeglani” QuickSort iz neke programske biblioteke

- puno brži od “našeg” algoritma!

Sortiranje i pretraživanje u standardnoj biblioteci

U standardnoj C biblioteci — datoteka zaglavlja `<stdlib.h>`, postoje i sljedeće dvije funkcije:

- `qsort` — QuickSort algoritam za općenito sortiranje niza podataka,
- `bsearch` — Binarno traženje zadanog podatka u sortiranom nizu.

U ovim funkcijama moramo sami zadati

- funkciju za uspoređivanje podataka u nizu.

O njima će biti više riječi na zadnjem predavanju, kad naučimo još neke potrebne stvari o pokazivačima. Na primjer,

- kako se jedna funkcija šalje drugoj funkciji kao argument.

Funkcije `qsort` i `bsearch`

Prototip funkcije `qsort` za **sortiranje** niza:

```
void qsort(void *base, size_t n, size_t size,  
           int (*comp) (const void *, const void *));
```

Prototip funkcije `bsearch` za **binarno traženje** zadanog podatka u **sortiranom** nizu:

```
void *bsearch(const void *key, const void *base,  
              size_t n, size_t size,  
              int (*comp) (const void *, const void *));
```

Vraća **pokazivač** na **nađeni** podatak (ako ga ima), ili `NULL`.

Zadnji argument u obje funkcije je **pokazivač** na **funkciju** za **uspoređivanje** članova niza.

Usporedba algoritama sortiranja (*Intel C*)

Vrijeme (u s) za sortiranje polja s $n = 10^5$ (10^6) elemenata:

Algoritam	Slučajno	Uzlazno	Silazno
min_1	1.763	1.766	1.765
min_2	1.758	1.755	1.861
max_1	1.422	1.425	1.423
max_2	1.424	1.425	1.428
bubble_1	12.064	2.154	5.367
bubble_2	11.952	0.000	5.361
ins_1	1.212	0.000	2.431
ins_1a	1.191	0.000	2.383
→ qs_p2_1	0.101	2.528	2.295
→ qs_std	0.174	0.005	0.005