

Programiranje (C)

2. predavanje

Saša Singer

singer@math.hr
web.math.hr/~singer

PMF – Matematički odjel, Zagreb

Obavijesti

- Konzultacije (fiksno): petak, 12–14 sati.

Mislim (nadam se) da nemate ništa u to vrijeme.

- Rješenja problema za “problematične” ili “ugrožene” skupine studenata,

zato što ove godine kolegij ide po novom nastavnom planu, pa je prvi dio programskog jezika C napravljen je na UuR.

Položili UuR ranijih godina

Studenti koji su

- UuR položili prije ove godine (a pali su C).

Za njih organiziramo

- “ubrzano ponavljanje” onog dijela C-a koji je napravljen na UuR,
- u trajanju od 8 sati, u dva bloka od po 4 sata.

Termini za ovu “nadoknadu”:

- subote 17. i 24. ožujka, 11–15 sati, ovdje u (003).

Sutra držim ja (“predavanja”), a sljedeće subote V. Šego (“vježbe”).

Položili UuR ranijih godina — nastavak

Dolazak je, naravno,

- “dobrovoljan”, ali se “toplo preporučuje”!

Napomena: bilo bi **vrlo pohvalno** da prije toga

- pogledate (i pročitate) materijale iz UuR-a,
- posebno: predavanja 7–15 i pripadne vježbe.

To se nalazi na **službenom** webu

<http://web.math.hr/nastava/uur.>

Predavanja (s malo više teksta) su i na **mom** webu

<http://www.math.hr/~singer.>

Profesori kojima je C izborni

Studenti profesorskog smjera/studija koji su

- upisali C kao izborni kolegij,
(bez da su prethodno slušali UuR).

Za njih se organizira

- zasebna nastava cijelog kolegija,
- tako da C ide “od početka”, slično nekadašnjem nastavnom planu.

Predavanja drži doc. G. Nogo, a vježbe asist. V. Šego.

Termini za ovu nastavu (zasad, koliko znam):

- start je u srijedu 21. ožujka, 8–10 sati, u (004).

Pratite obavijesti (bilo gdje: oglasna ploča, web).

Profesori kojima je C izborni — nastavak

Napomena:

- Ova promjena, u načelu, vrijedi **za sve** takve studente!

Ako netko od njih **dobrovoljno želi** ostati na “ovom” C-u,

- a to znači **nastavu** i **polaganje po pravilima** ovog kolegija, molim da se najkasnije **sljedeći tjedan javi** svom nastavniku (**meni** ili kolegici **Nogo**), da ga uredno dodamo na popis.

Ne zaboravite!

Službena web stranica kolegija je:

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

Tamo su:

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

Ako mislite da bi tamo trebalo biti još nešto, slobodno predložite! Ideja je da tamo bude sve što vam može pomoći.

Završetak ponavljanja gradiva UuR-a

Sadržaj predavanja

- Ponavljanje onog dijela C-a koji je napravljen na UuR:
 - Polja.
 - Pokazivači i polja.
 - Polje kao argument funkcije.
 - Rekurzivne funkcije.

Polja

Primjer. Što će ispisati sljedeći program?

```
#include <stdio.h>
    int array[] = {4, 5, -8, 9, 8, 0, -2, 1, 9, 3};
    int index;

int main(void){
    index = 0;
    while (array[index] != 0)
        ++index;
    printf("Broj elemenata polja prije nule: %d\n",
           index);
    return (0); }
```

Polja — rješenje

Primjer. Što će ispisati sljedeći program?

```
#include <stdio.h>
    int array[] = {4, 5, -8, 9, 8, 0, -2, 1, 9, 3};
    int index;

int main(void){
    index = 0;
    while (array[index] != 0)
        ++index;
    printf("Broj elemenata polja prije nule: %d\n",
           index);
    return (0); }
```

Broj elemenata polja prije nule: 5

Pokazivači i polja

Zapamtiti: Ime polja je sinonim za

- konstantni pokazivač koji sadrži adresu prvog elementa polja

Polje može biti formalni (i stvarni) argument funkcije. U tom slučaju:

- ne prenosi se cijelo polje po vrijednosti (kopija polja!),
- već funkcija dobiva (po vrijednosti) pokazivač na prvi element polja.

Unutar funkcije elementi polja mogu se

- dohvatiti i promijeniti, korištenjem indeksa polja.

Razlog: aritmetika pokazivača (v. sljedeću stranicu).

Pokazivači i polja — nastavak

Primjer. Krenimo od deklaracija

```
int a[10], *pa;
```

Tada je: **a = &a[0]**. Ne samo to, pokazivaču mogu dodati i oduzeti “indeks” (tzv. “**aritmetika pokazivača**”).

Općenito vrijedi: **a + i = &a[i]**, gdje je **i** neki cijeli broj.

```
*(a + 1) = 10; /* ekviv. s a[1] = 10; */  
...  
pa = a; /* ekviv. s pa = &a[0]; */  
pa = pa + 2; /* &a[2] */  
pa++; /* &a[3] */  
*(pa + 3) = 20; /* ekviv. s a[6] = 20; */
```

Pokazivači i polja (1)

Primjer. Što će ispisati sljedeći program?

```
#include <stdio.h>
    int array[] = {4, 5, -8, 9, 8, 0, -2, 1, 9, 3};
    int *array_ptr;

int main(void){
    array_ptr = array;
    while ((*array_ptr) != 0)
        ++array_ptr;
    printf("Broj elemenata polja prije nule: %d\n",
           array_ptr - array);
    return (0); }
```

Pokazivači i polja (1) — rješenje

Primjer. Što će ispisati sljedeći program?

```
#include <stdio.h>
    int array[] = {4, 5, -8, 9, 8, 0, -2, 1, 9, 3};
    int *array_ptr;

int main(void){
    array_ptr = array;
    while ((*array_ptr) != 0)
        ++array_ptr;
    printf("Broj elemenata polja prije nule: %d\n",
           array_ptr - array);
    return (0); }
```

Broj elemenata polja prije nule: 5

Pokazivači i polja (2)

Primjer. Što će ispisati sljedeći program?

```
#include <stdio.h>
#define MAX 10

void main(){
    int a[MAX];
    int i, *p;
    p = a;
    for (i = 0; i < MAX; i++)
        a[i] = i;
    printf("%d\n", *p);
    return; }
```

Pokazivači i polja (2) — rješenje

Primjer. Što će ispisati sljedeći program?

```
#include <stdio.h>
#define MAX 10

void main(){
    int a[MAX];
    int i, *p;
    p = a;
    for (i = 0; i < MAX; i++)
        a[i] = i;
    printf("%d\n", *p);
    return; }
```

0 (neki kompjajleri očekuju int main(...))

Polje kao argument funkcije

Primjer. Napišite funkcije **unos** i **ispis** te glavni program koji upisuje i ispisuje polje s maksimalno 100 elemenata.

```
#include <stdio.h>
#define MAX 100

void unos(int a[], int n) {
    int i;
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]); }

void ispis(int *a, int n) {
    int i;
    for (i = 0; i < n; i++)
        printf("%d\n", *a++); }
```

Polje kao argument funkcije (nastavak)

```
int main(void){  
    int n, polje[MAX];  
    /* Koliko ce se bajtova rezervirati? */  
  
    scanf("%d", &n);  
  
    unos(polje, n);  
    ispis(polje, n);  
    return (0);  
}
```

Primjedba: Pri upisu podataka oni se “upisuju na slijepo” (ne zna se što se upisuje) – loš stil programiranja.

Rekurzivne funkcije

Primjer. Što će ispisati sljedeći program?

```
int f(int n){  
    if (n == 0)  
        return 2;  
    else  
        return f(--n); }
```

```
int main(void){  
    printf("%d\n", f(4));  
    return (0); }
```

Rekurzivne funkcije — rješenje

Primjer. Što će ispisati sljedeći program?

```
int f(int n){  
    if (n == 0)  
        return 2;  
    else  
        return f(--n); }
```

```
int main(void){  
    printf("%d\n", f(4));  
    return (0); }
```

2

Pitanje: što se ispiše ako napišemo **f(n--)**? Oprez! Zašto?

Funkcije

Sadržaj predavanja

- Funkcije:
 - Prijenos argumenata po vrijednosti i adresi.
 - Rekurzivne funkcije
 - Fibonaccijevi brojevi.
 - QuickSort algoritam.

Definicija funkcije — ponavljanje

Funkcija je programska cjelina koja

- uzima neke ulazne podatke,
- izvršava određeni niz naredbi,
- i vraća rezultat svog izvršavanja pozivnom programu.

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 (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.

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 variable** (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, kopiraj).

Ako funkcijom želimo promijeniti vrijednost nekog podatka, pripadni argument treba biti pokazivač na taj podatak (njegova adresa).

- Tada se adresa prenosi po vrijednosti (tj. kopira u funkciju),
- ali smijemo promijeniti sadržaj na toj adresi (operatorom 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 argumenta (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 bit će:

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**.

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 bit će:

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!

Rekurzivne funkcije

C dozvoljava da se funkcije koriste **rekurzivno**, tj.

- funkcija poziva samu sebe.

U pravilu:

- rekurzivni programi su kraći,
- ali izvođenje traje dulje.

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

Napomena: Svaki rekurzivni algoritam mora imati “nerekurzivni” dio, koji omogućava “prekidanje” rekurzije. Najčešće je to **inicijalizacija** rekurzije.

Fibonaccijevi brojevi

Primjer. Drugi standardni primjer **rekurzivne** funkcije (osim faktorijela) su Fibonaccijevi brojevi, definirani **rekurzijom**

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

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);
}
```

Ali, ovo **nemojte** raditi. **Zabranujem!**

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!

Fibonaccijevi brojevi petljom

I ovo ide puno brže običnom petljom:

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

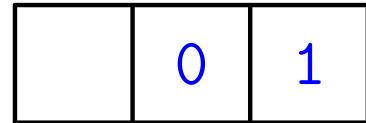
Za realizaciju tog algoritma trebamo “prozor” od samo 3 člana niza:

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

Fibonaccijevi brojevi

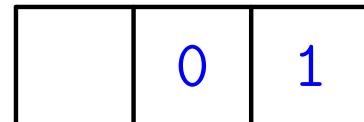
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:



fp fn

Što se stvarno zbiva s prozorom:

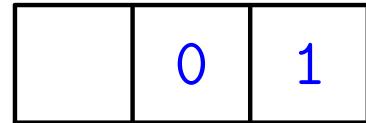


fp fn

Fibonaccijevi brojevi

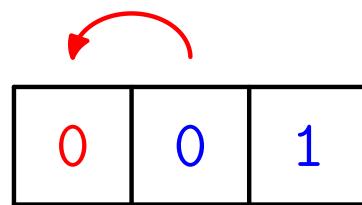
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:



fp fn

Što se stvarno zbiva s prozorom:



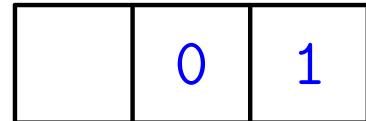
fpp fp fn

fpp = fp

Fibonaccijevi brojevi

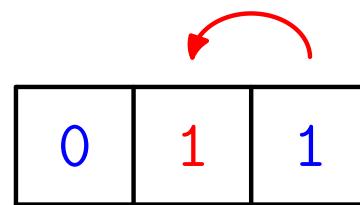
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:



fp fn

Što se stvarno zbiva s prozorom:



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 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:

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

$$fn = fp + fpp$$

fpp fp fn

Fibonaccijevi brojevi

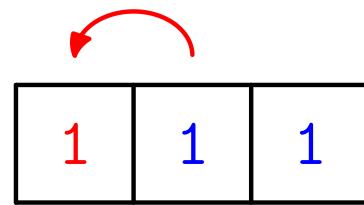
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:



fpp = fp

fpp fp fn

Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:

1	1	1
---	---	---

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 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:

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

$$fn = fp + fpp$$

fpp fp fn

Fibonaccijevi brojevi

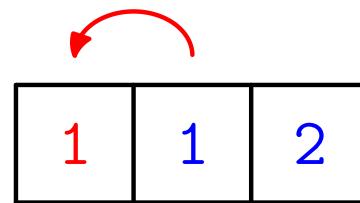
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:



fpp = fp

fpp fp fn

Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:

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

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 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:

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

$$fn = fp + fpp$$

fpp fp fn

Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:

2	2	3
---	---	---

fpp = fp

fpp fp fn

Fibonaccijevi brojevi

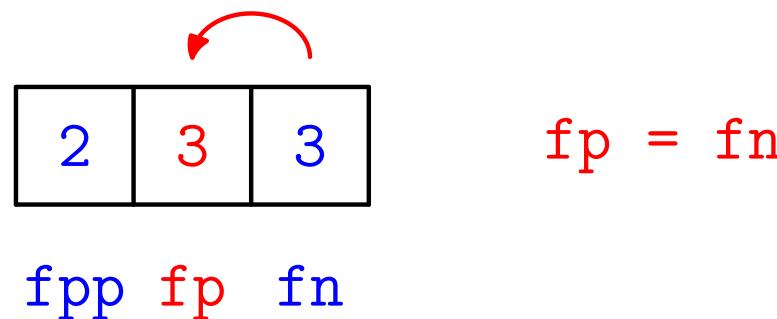
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:



Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:

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

fn = fp + fpp

fpp fp fn

Fibonaccijevi brojevi

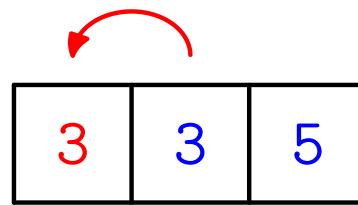
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:



fpp = fp

fpp fp fn

Fibonaccijevi brojevi

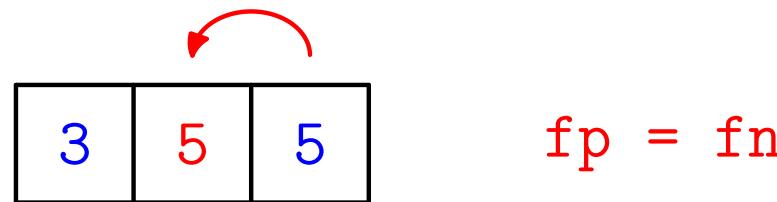
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:

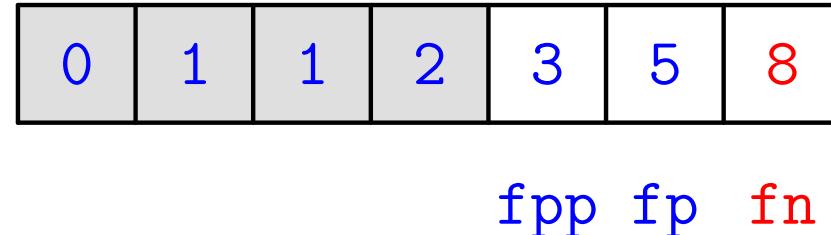


fpp fp fn

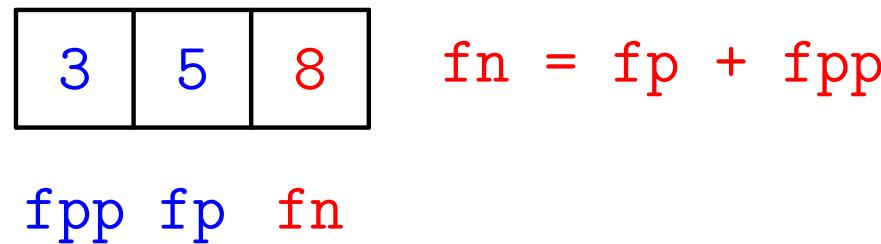
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:



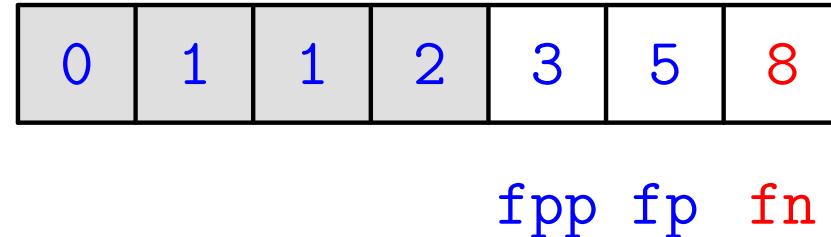
Što se stvarno zbiva s prozorom:



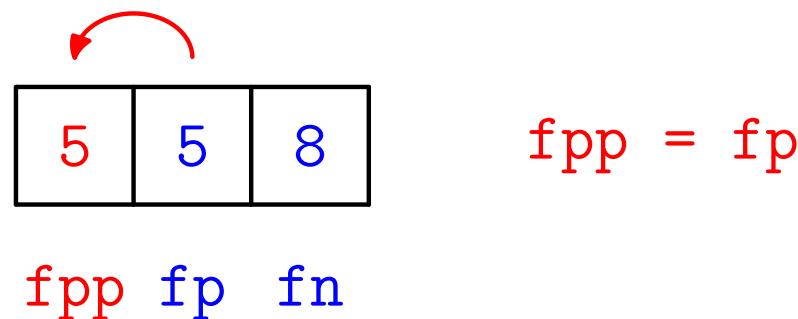
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:



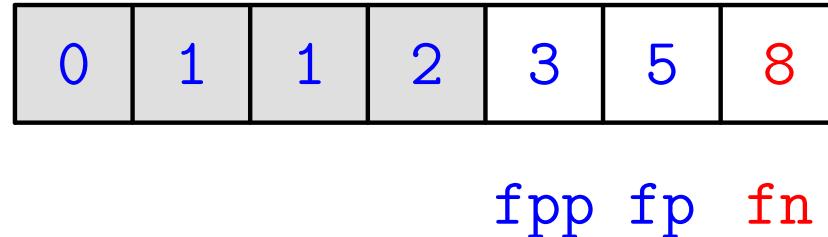
Što se stvarno zbiva s prozorom:



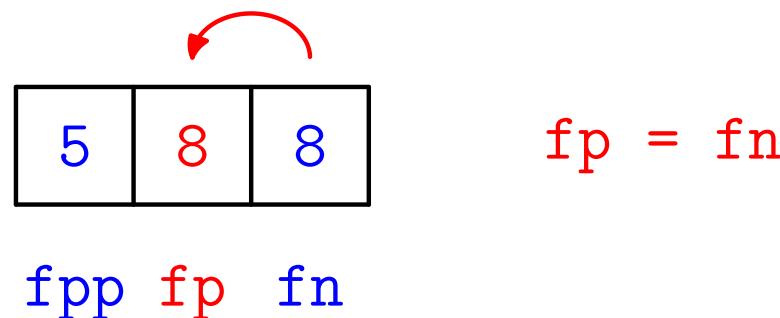
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:



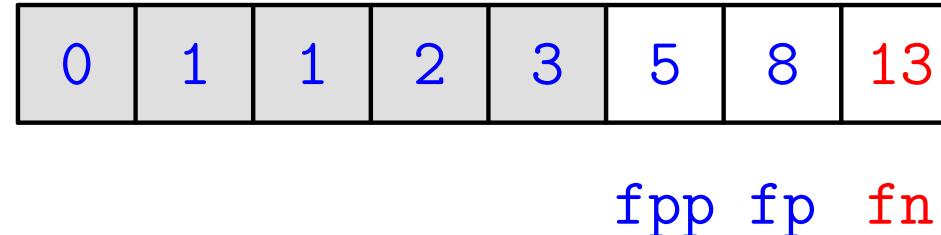
Što se stvarno zbiva s prozorom:



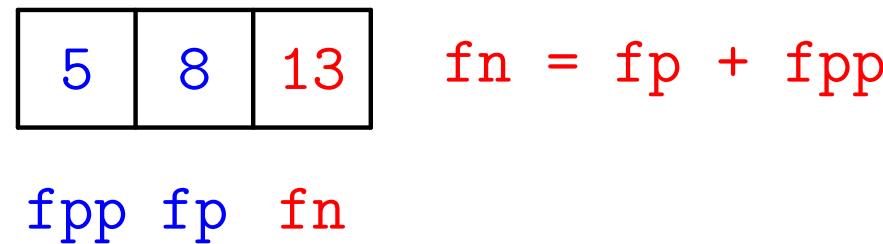
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:



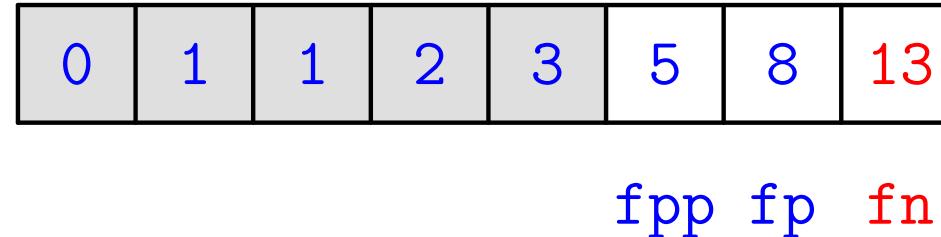
Što se stvarno zbiva s prozorom:



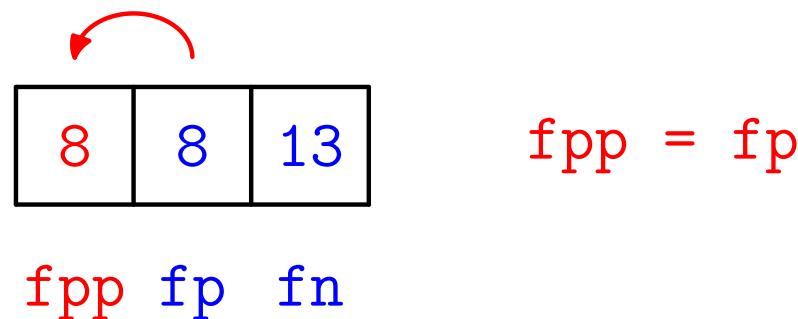
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:



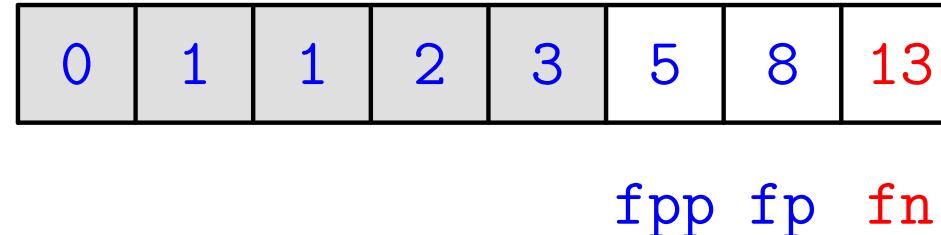
Što se stvarno zbiva s prozorom:



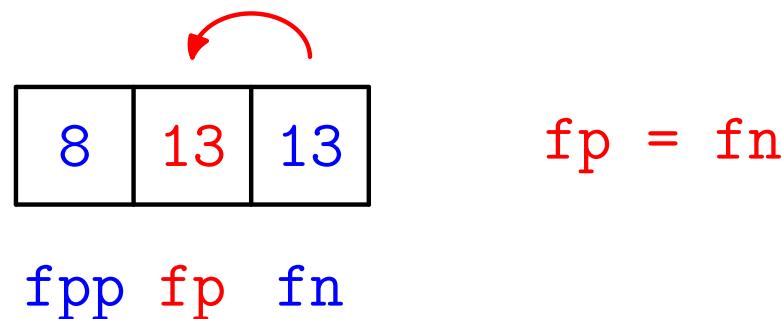
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:



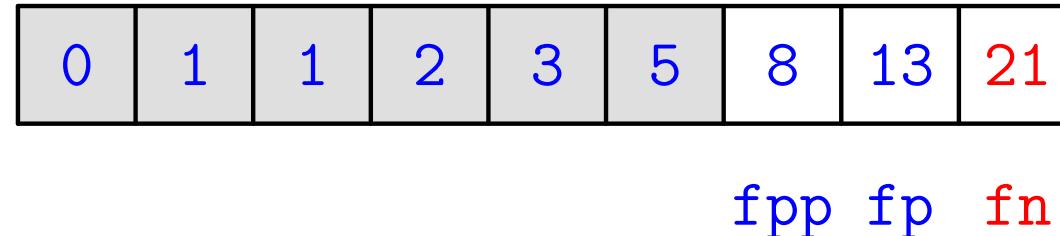
Što se stvarno zbiva s prozorom:



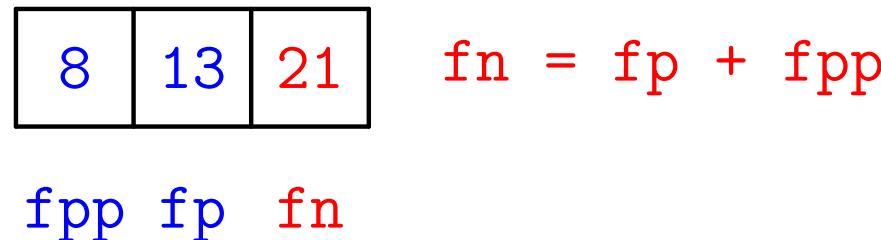
Fibonaccijevi brojevi

Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:



Što se stvarno zbiva s prozorom:



Fibonaccijevi 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). */
    f_pp = 0;
    f_p = 1;
```

Fibonaccijevi brojevi petljom (nastavak)

```
f_p = 0; /* Prosli 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;  
}
```

Fibonaccijevi 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 $F_{46} = 1836311903$.

QuickSort — uvod i skica algoritma

QuickSort se temelji na principu **podijeli pa vladaj**.

- Uzmemo jedan element x_k iz niza i dovedemo ga na njegovo **pravo** mjesto.
- 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). Možemo i tu dozvoliti jednakost.
- Ako smo **dobro** izabrali, tj. ako je mjesto x_k blizu sredine, onda ćemo morati sortirati dva polja **polovične duljine**.
- U **najgorem** slučaju, ako smo izabrali “**krivi**” x_k , morat ćemo sortirati polje duljine $n - 1$.

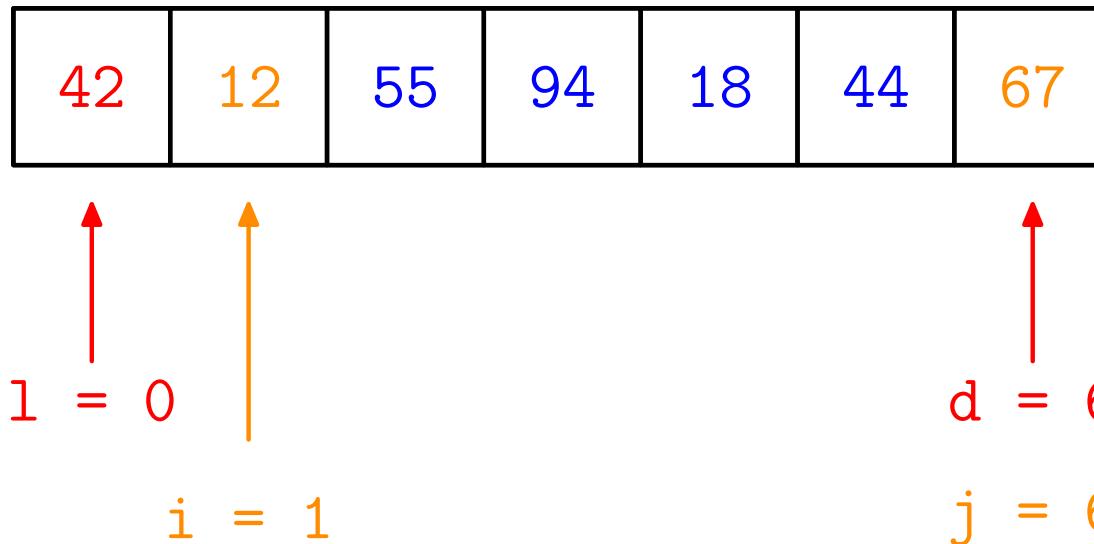
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

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

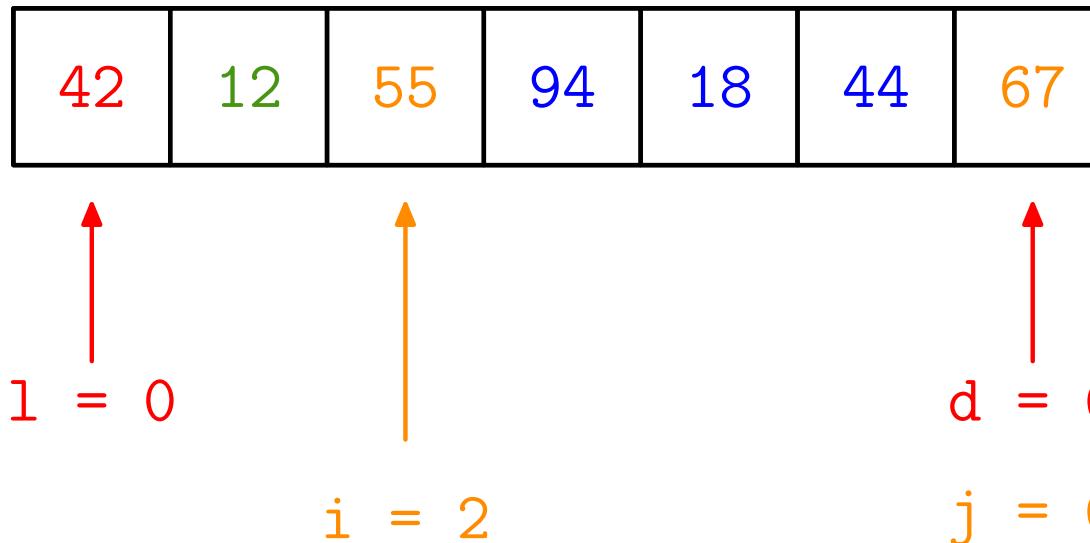
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



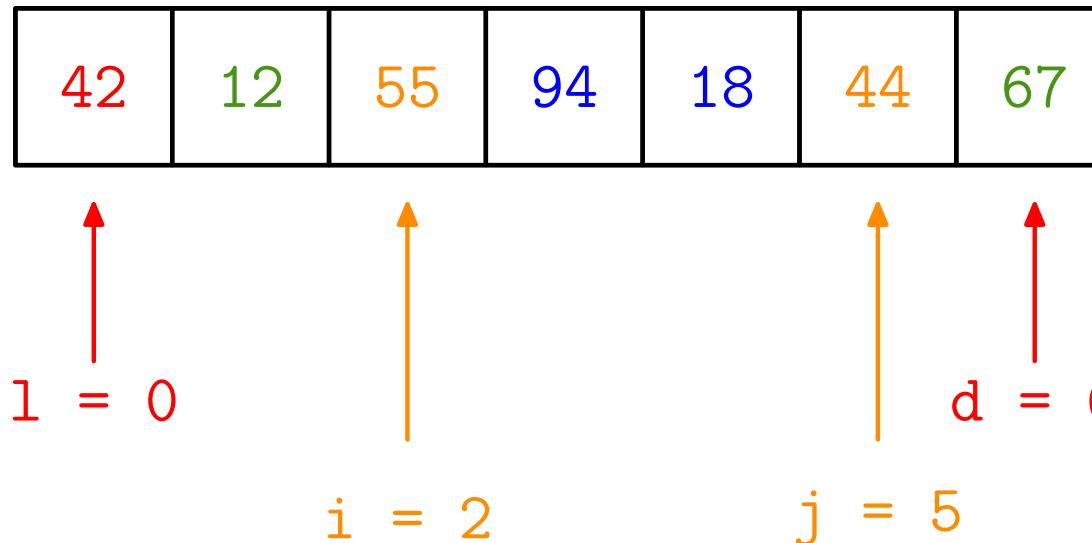
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



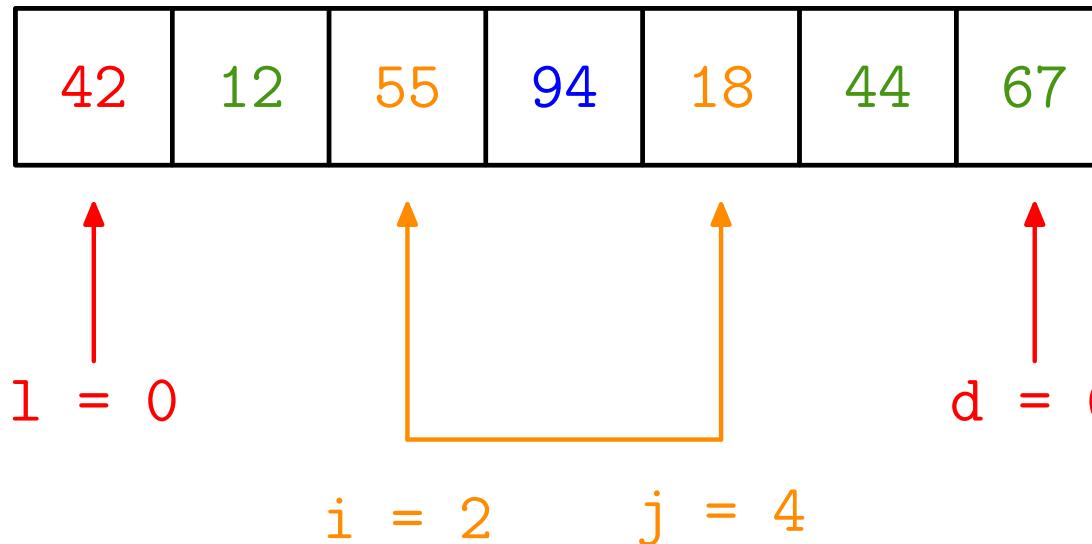
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



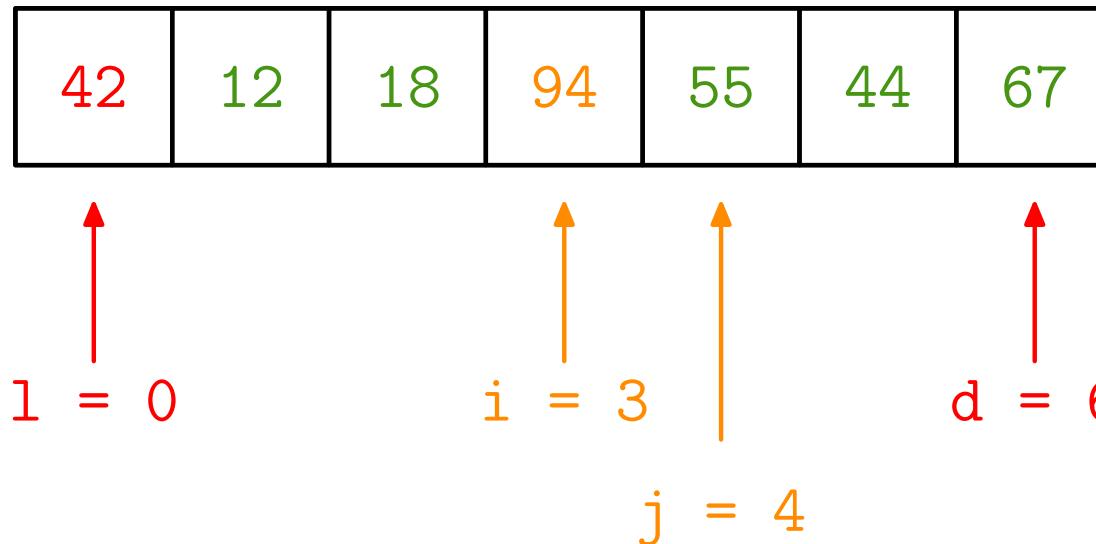
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



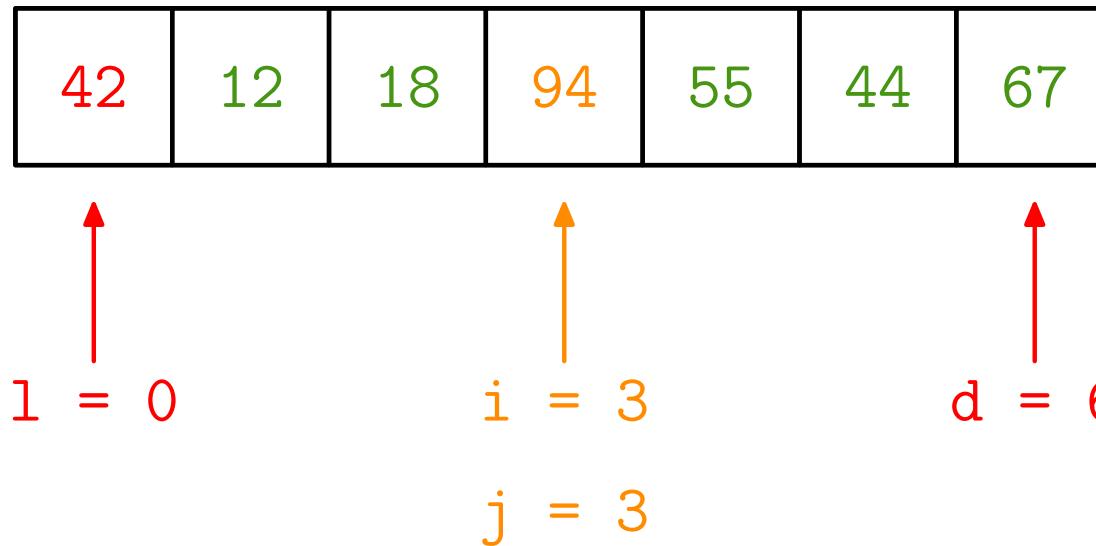
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



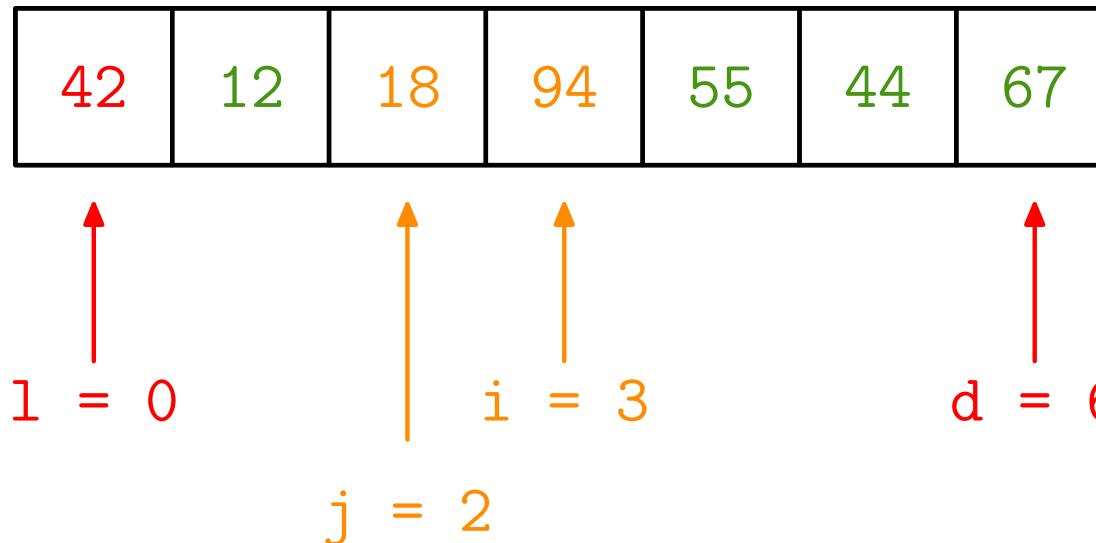
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



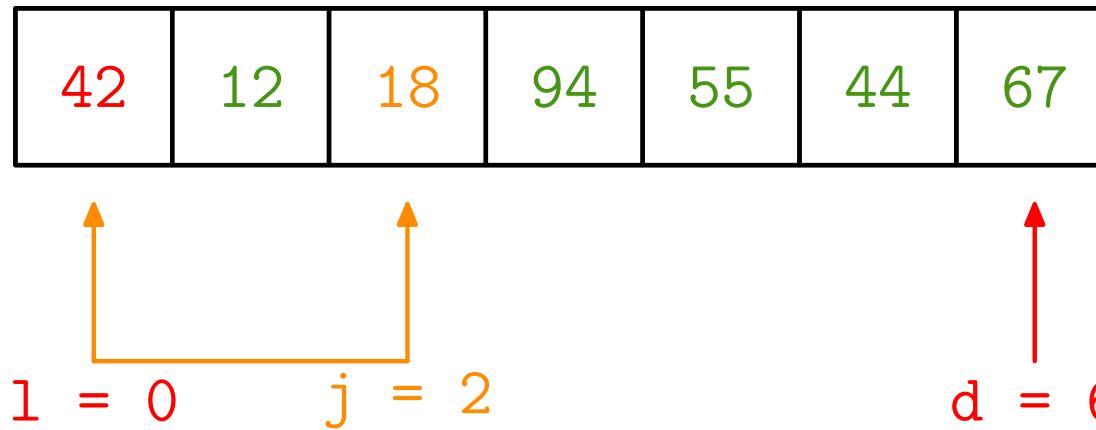
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



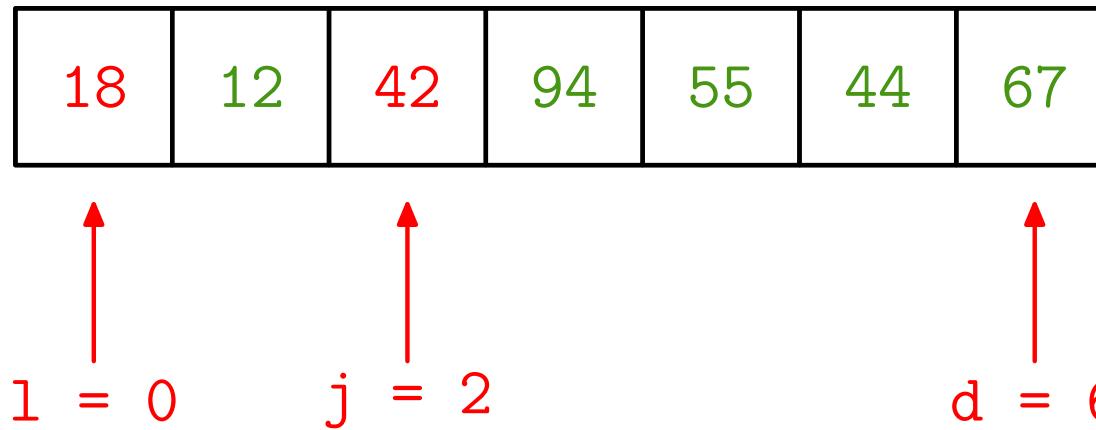
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

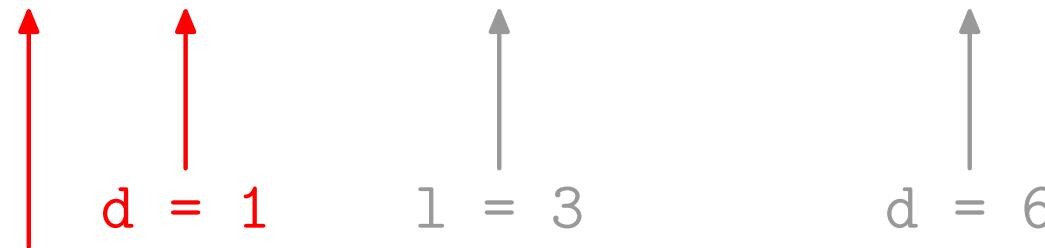
Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

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

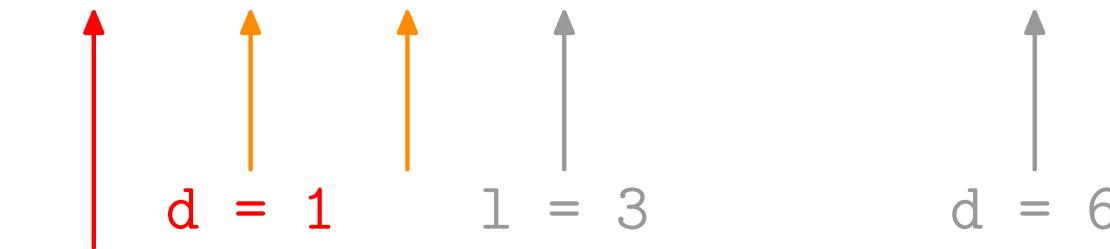


$$\begin{aligned} l &= 0 \\ i &= 1 \\ j &= 1 \end{aligned}$$

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

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



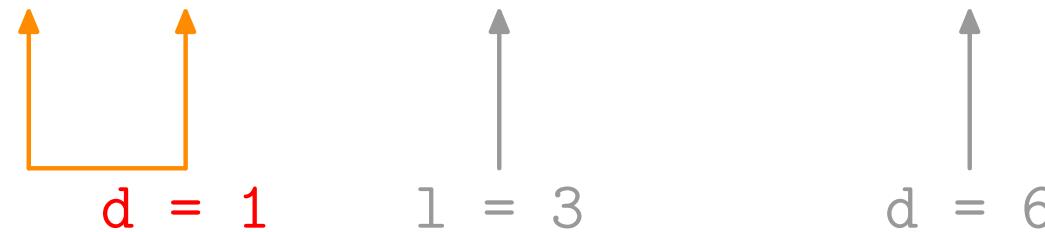
$$l = 0 \quad i = 2$$

$$j = 1$$

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

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

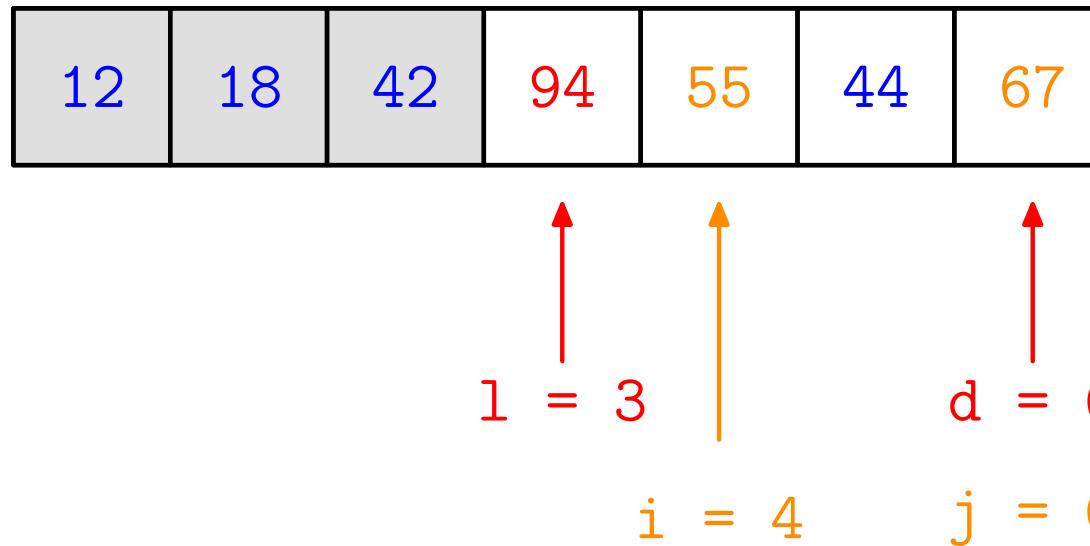


$l = 0$

$j = 1$

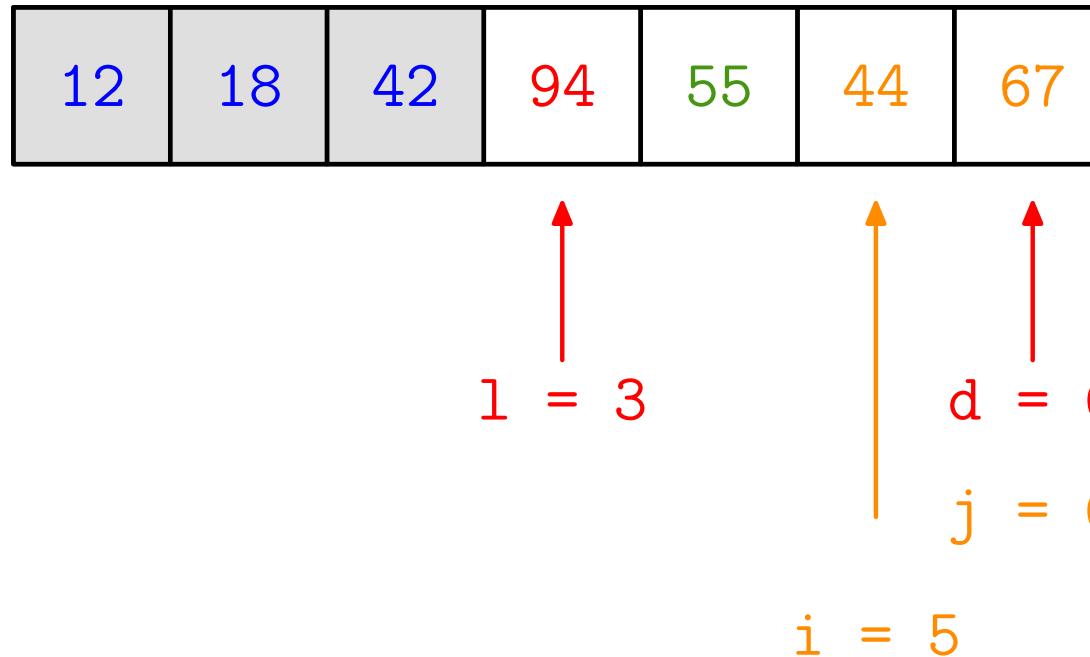
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

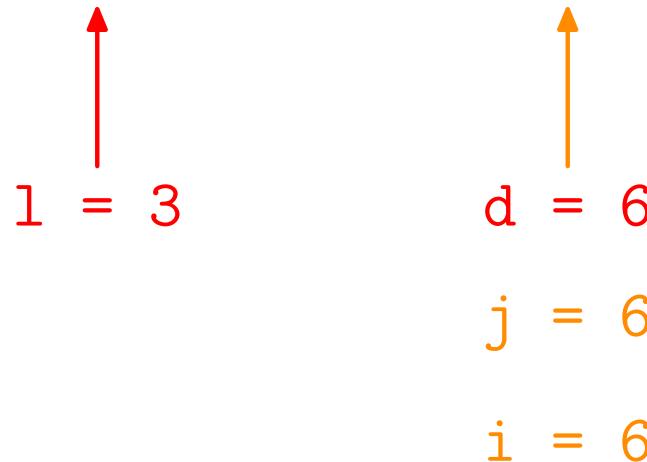
Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

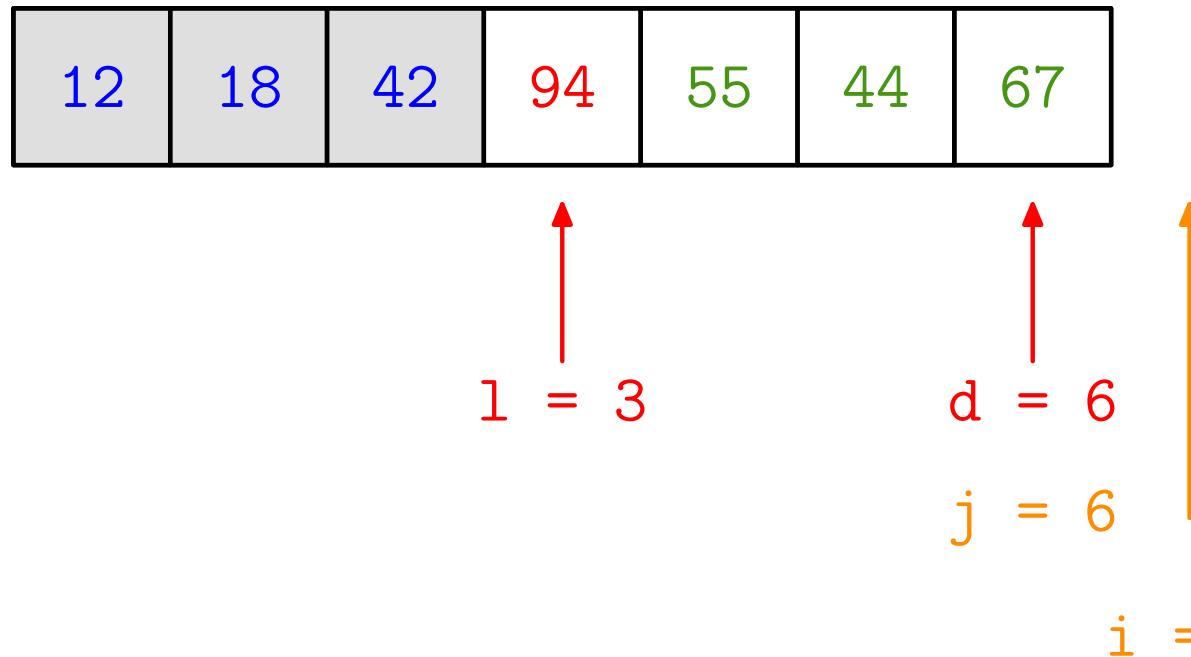
Primjer. Sortirajte korištenjem quicksorta zadano polje.

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



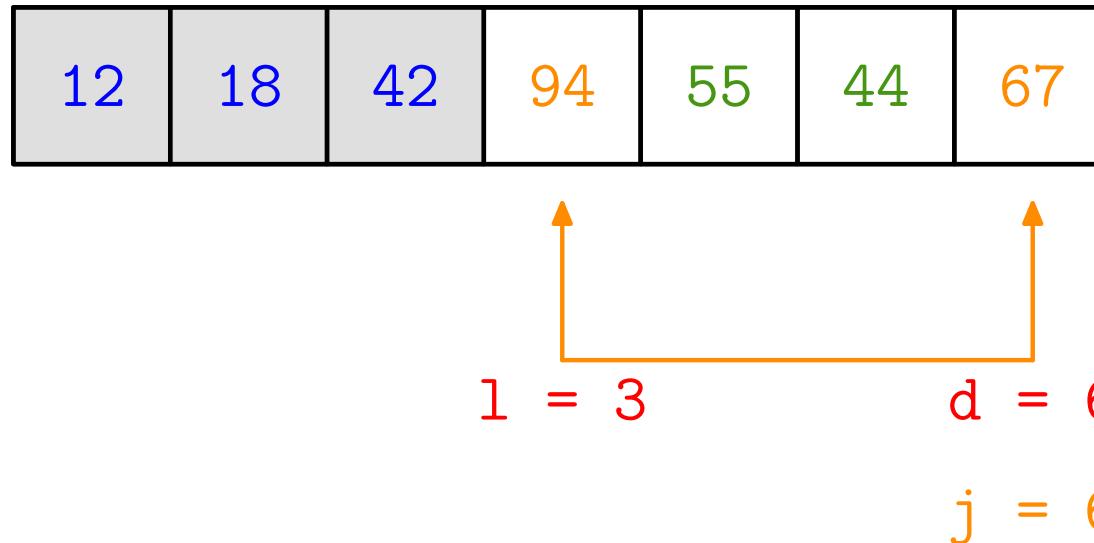
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



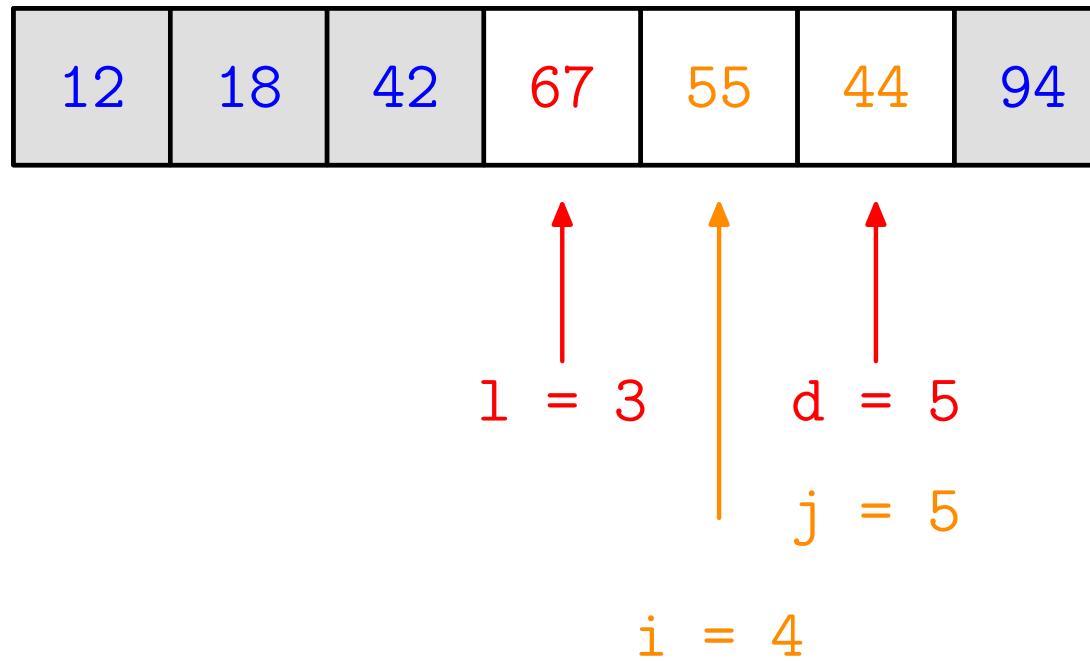
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

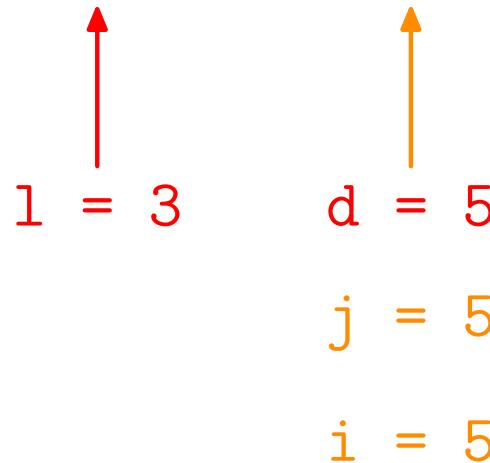
Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

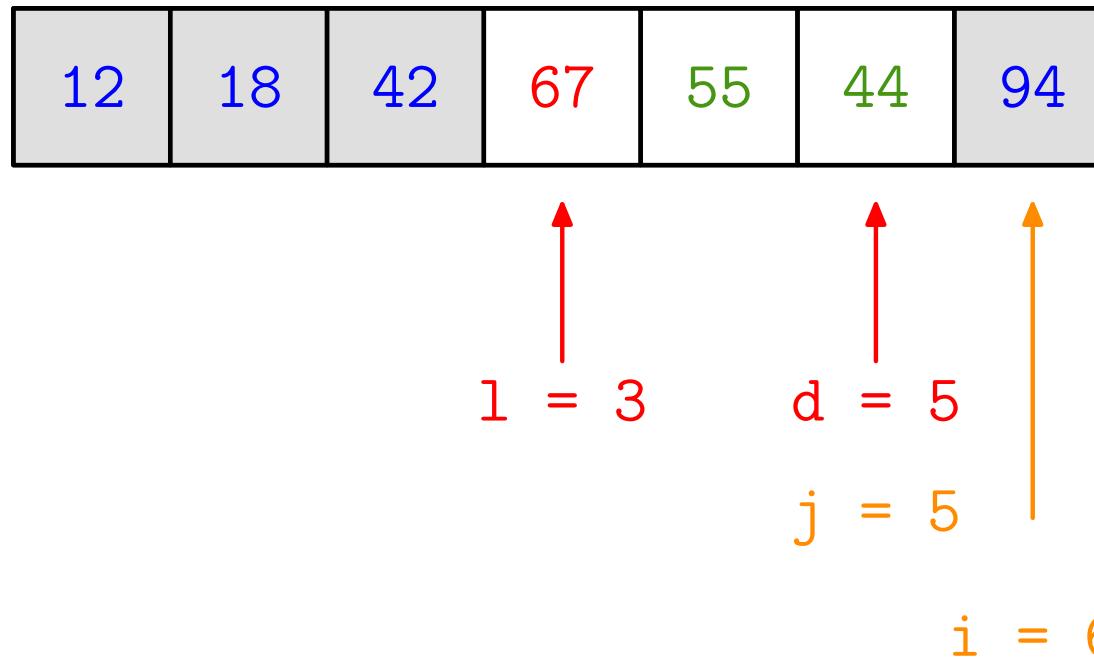
Primjer. Sortirajte korištenjem quicksorta zadano polje.

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



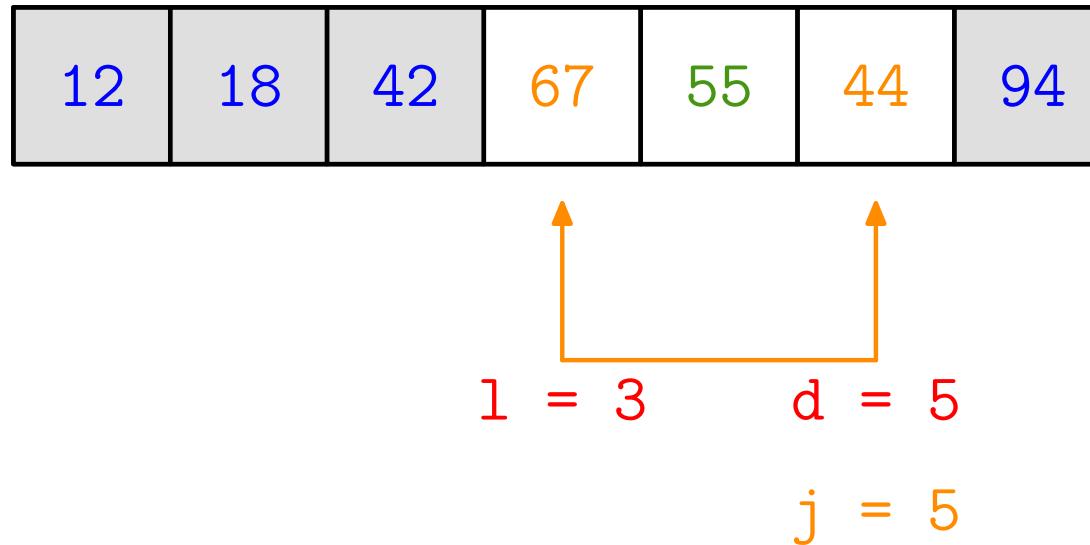
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

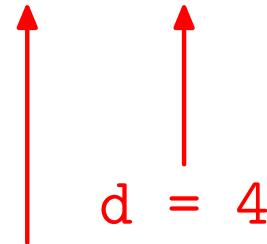
Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

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

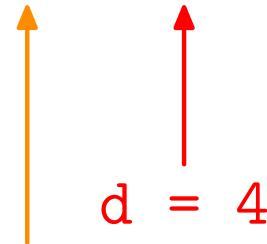


$$\begin{aligned}l &= 3 \\i &= 4 \\j &= 4\end{aligned}$$

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

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



$$\begin{array}{l} l = 3 \\ j = 3 \end{array} \quad i = 4$$

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

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

QuickSort — složenost

Za **složenost** vrijedi:

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

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

QuickSort — funkcija swap

```
#include <stdio.h>

/* Sortiranje niza QuickSort algoritmom.
   x[l] je kljucni element - 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 — cijeli program