

Programiranje 2

2. predavanje

Saša Singer

singer@math.hr

web.math.hr/~singer

PMF – Matematički odjel, Zagreb

Sadržaj predavanja

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

Informacije

Konzultacije (fiksno):

- petak, 12–14 sati.

Praktični kolokvij (prvi krug) — trenutna namjera je:

- da se to održi 4. i 5. tjedan nastave,
- dakle, 25. 3. — 5. 4.,
- subote su namjerno uključene.

Pratite obavijesti na vježbama i webu!

Završetak ponavljanja gradiva iz kolegija Programiranje 1

Sadržaj

- Ponavljanje onog dijela C-a koji je napravljen na Prog1:
 - Polje kao argument funkcije.
 - Rekurzivne funkcije.

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;
}
```

Za **polje** se rezervira $100 * 4 = 400$ bajtova.

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?

```
#include <stdio.h>
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?

```
#include <stdio.h>
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

- 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 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 **variabile**,
- 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,

- pripadni argument treba biti pokazivač na taj podatak, tj. njegova adresa!
- Tada se adresa prenosi po vrijednosti — kopira u funkciju,
- 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 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 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**.

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!

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.

Fibonaccijevi brojevi

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.

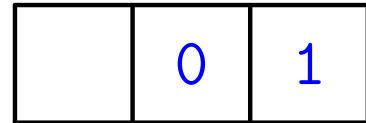
Za realizaciju tog algoritma trebamo “prozor” od samo 3 člana niza:

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

Fibonaccijevi brojevi

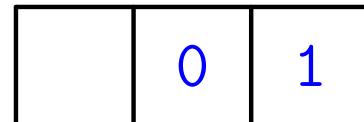
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:



fp fn

Što se stvarno zbiva s prozorom:

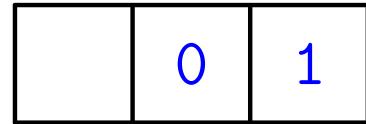


fp fn

Fibonaccijevi brojevi

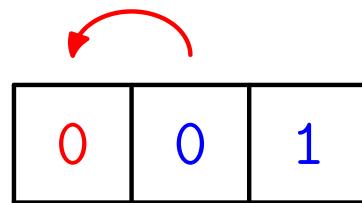
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:



fp fn

Što se stvarno zbiva s prozorom:



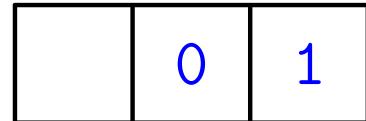
fpp fp fn

fpp = fp

Fibonaccijevi brojevi

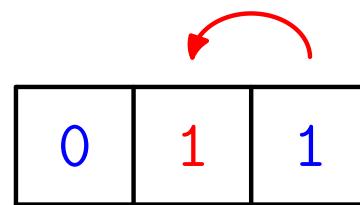
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

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

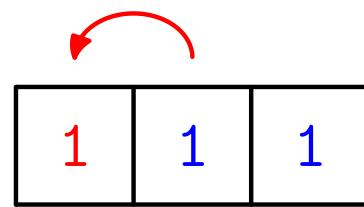
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

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

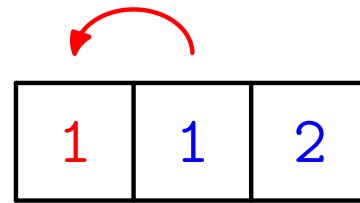
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

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

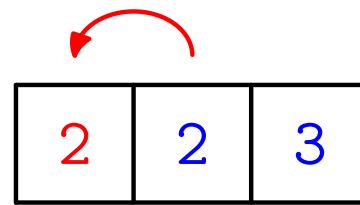
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

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

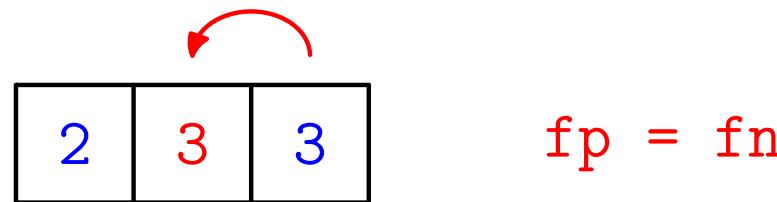
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:



fpp fp fn

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:

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

fn = fp + fpp

fpp fp fn

Fibonaccijevi brojevi

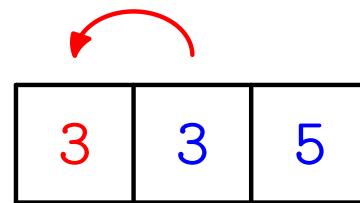
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

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

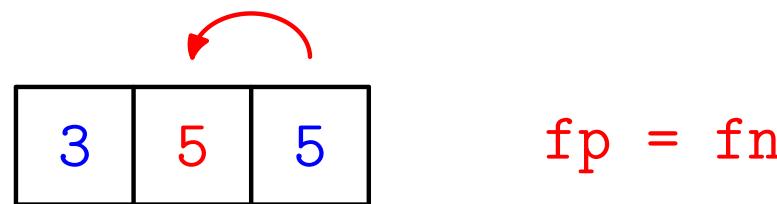
Primjer. Napišite iterativni algoritam koji računa Fibonaccijeve brojeve, počevši od $f_0 = 0$, $f_1 = 1$.

Prozor širine 3 putuje nizom:

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

fpp fp fn

Što se stvarno zbiva s prozorom:



fpp fp fn

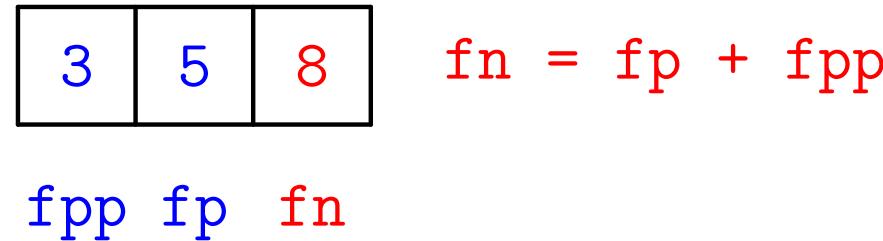
Fibonaccijevi brojevi

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

Prozor širine 3 putuje nizom:



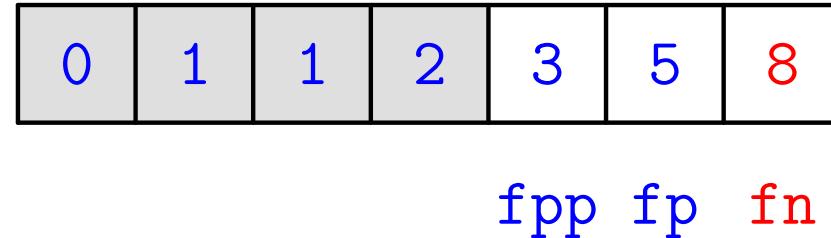
Što se stvarno zbiva s prozorom:



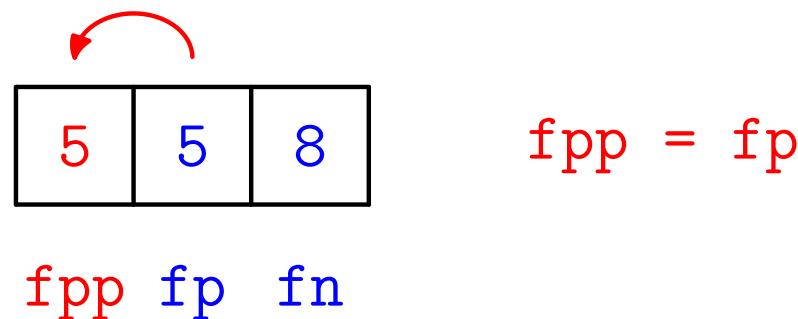
Fibonaccijevi brojevi

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

Prozor širine 3 putuje nizom:



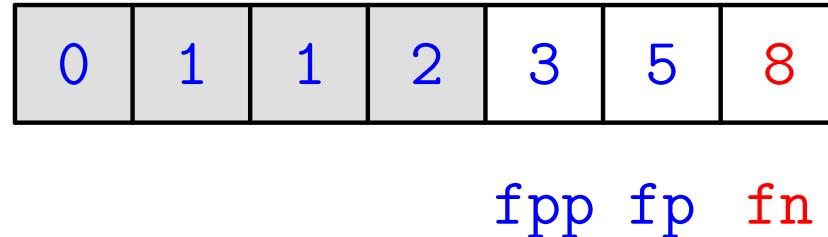
Što se stvarno zbiva s prozorom:



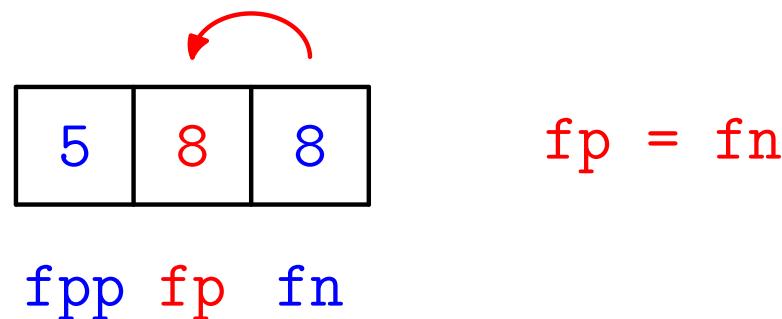
Fibonaccijevi brojevi

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

Prozor širine 3 putuje nizom:



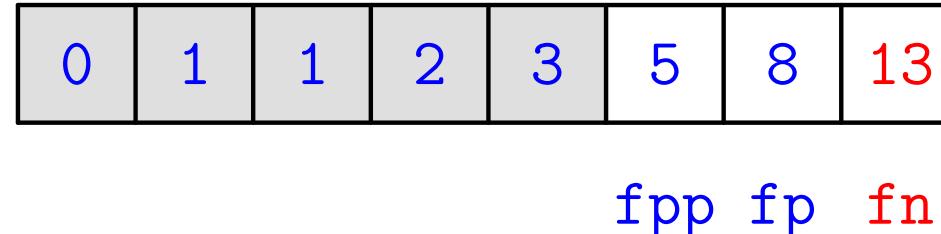
Što se stvarno zbiva s prozorom:



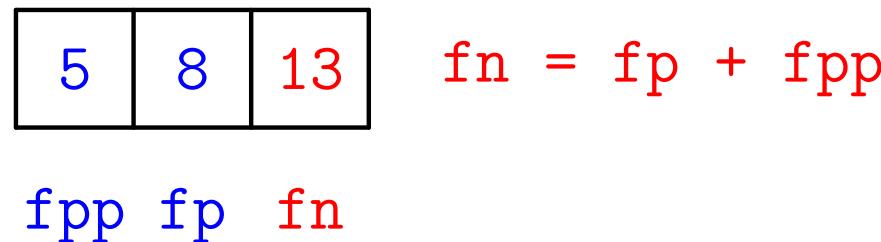
Fibonaccijevi brojevi

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

Prozor širine 3 putuje nizom:



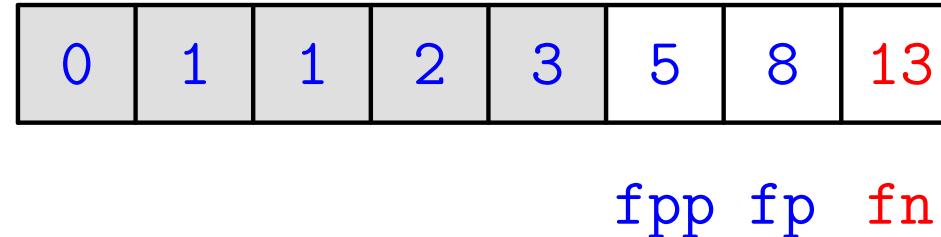
Što se stvarno zbiva s prozorom:



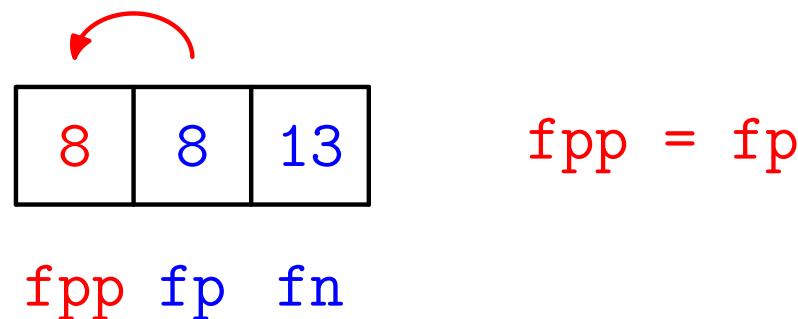
Fibonaccijevi brojevi

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

Prozor širine 3 putuje nizom:



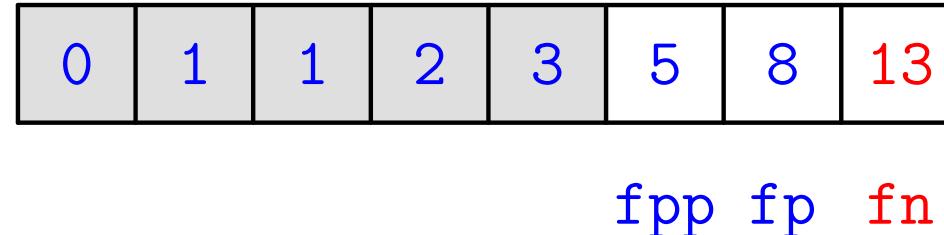
Što se stvarno zbiva s prozorom:



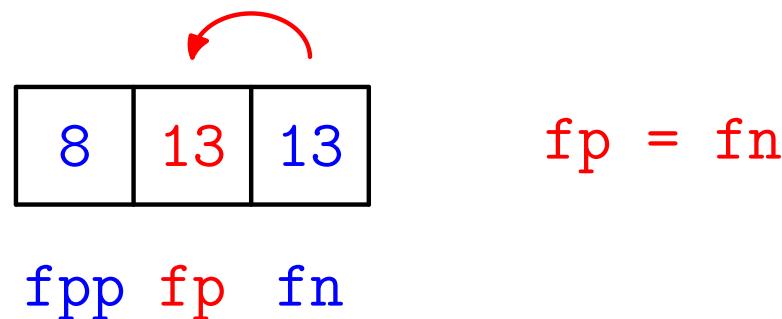
Fibonaccijevi brojevi

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

Prozor širine 3 putuje nizom:



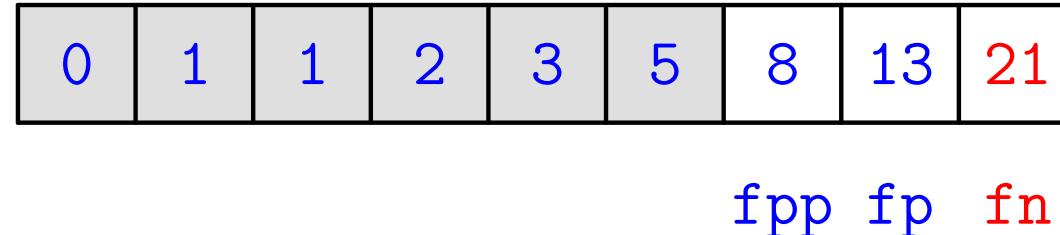
Što se stvarno zbiva s prozorom:



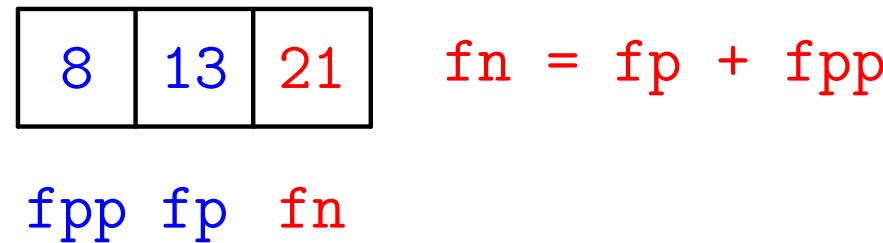
Fibonaccijevi brojevi

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 algoritam

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 — 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.

QuickSort — razrada algoritma (nastavak)

Dogovor:

- lijevo (ili ispred) njega stavljamo elemente koji su manji ili jednaki x_l ,
- desno (ili iza) njega stavljamo elemente koji su strogo veći od x_l .

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

Kako se traži “pravo” mjesto?

- 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)

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]);  
    }  
}
```

QuickSort — razrada algoritma (nastavak)

Iza toga treba još:

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

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

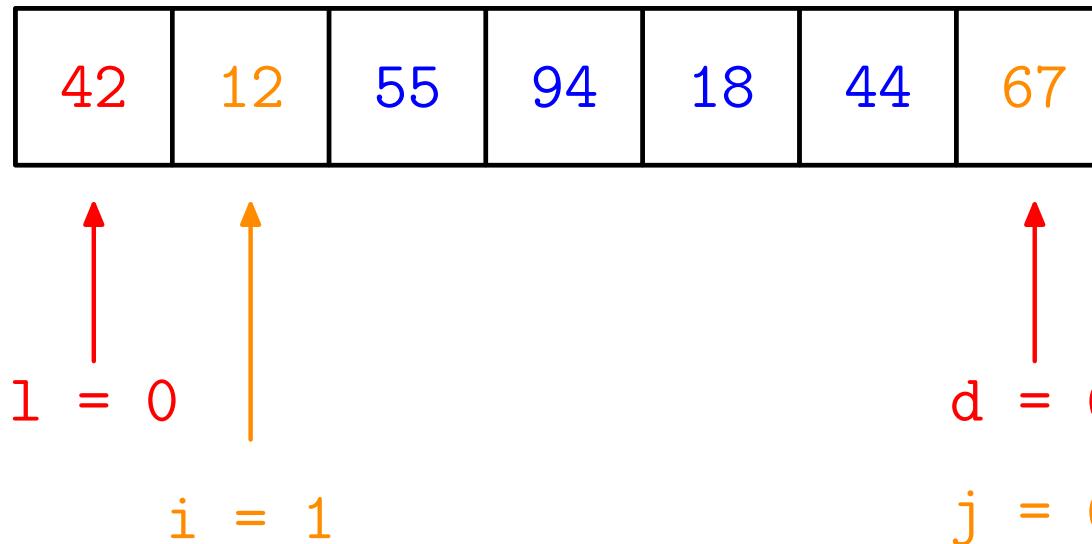
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

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

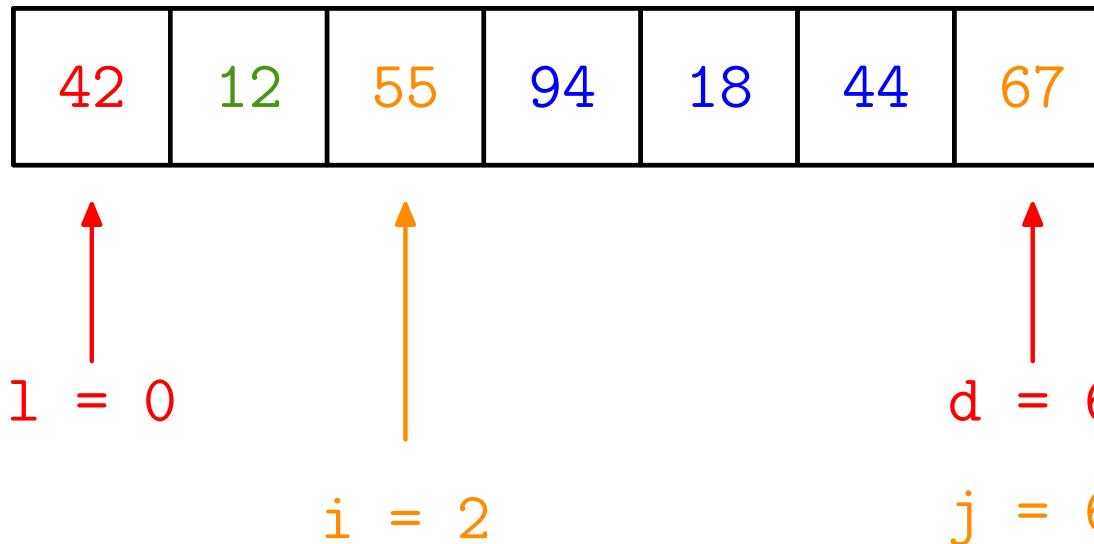
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



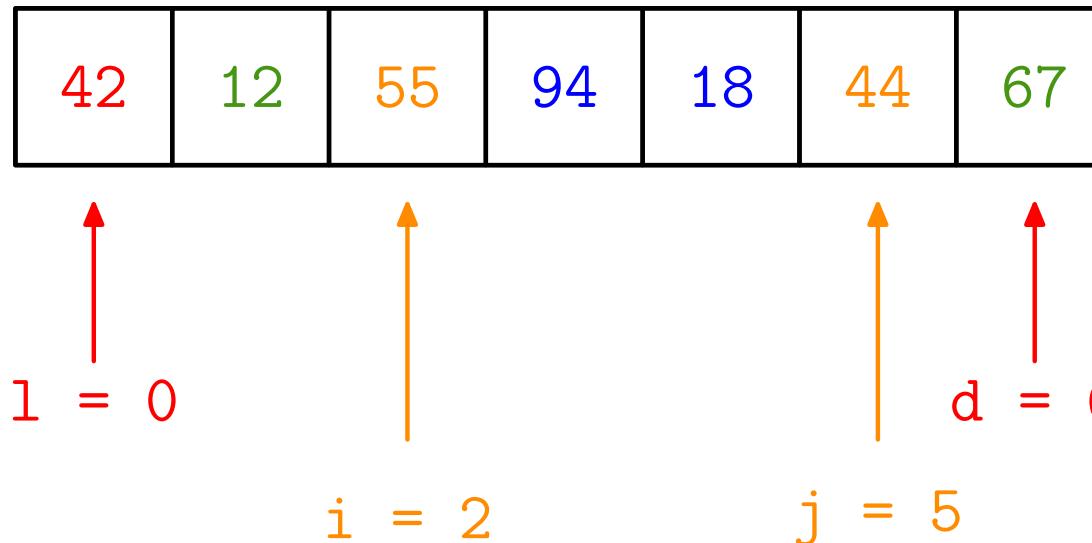
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



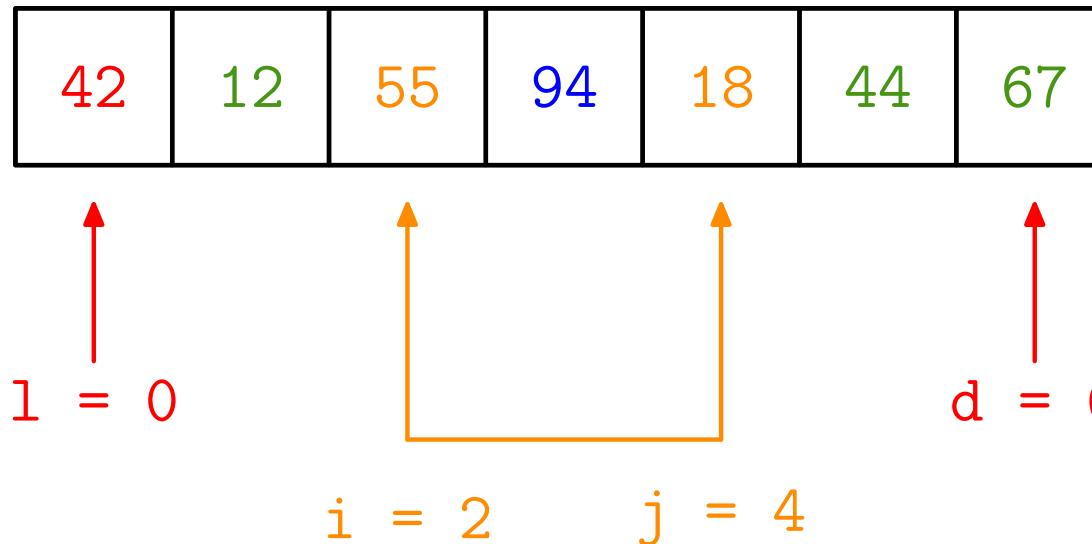
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



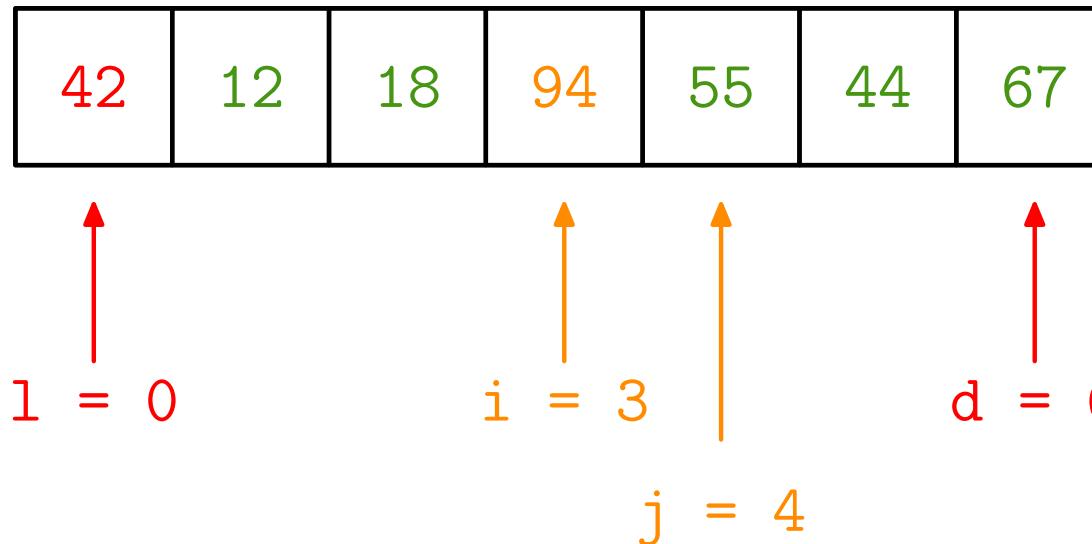
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



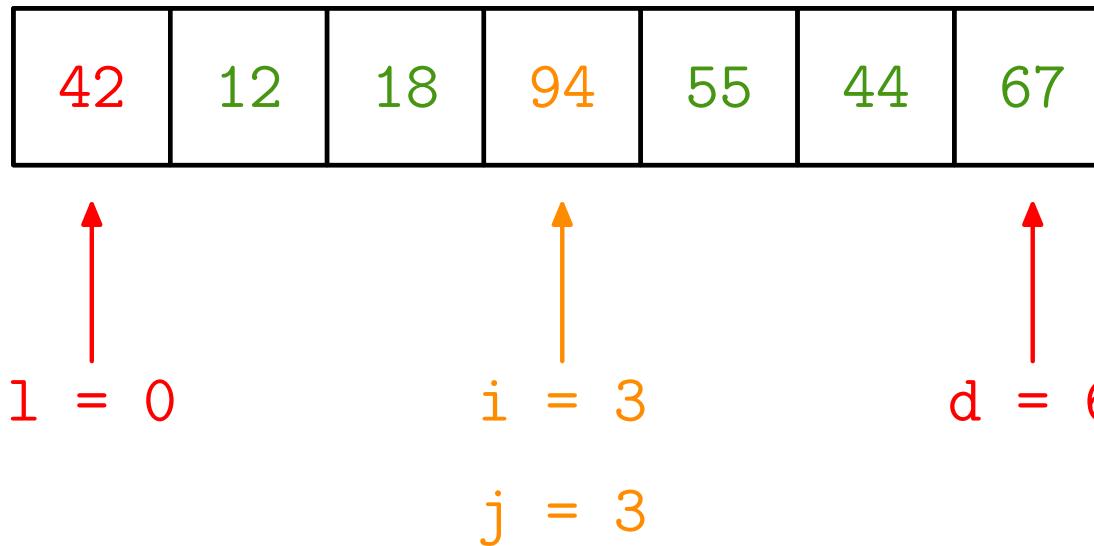
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



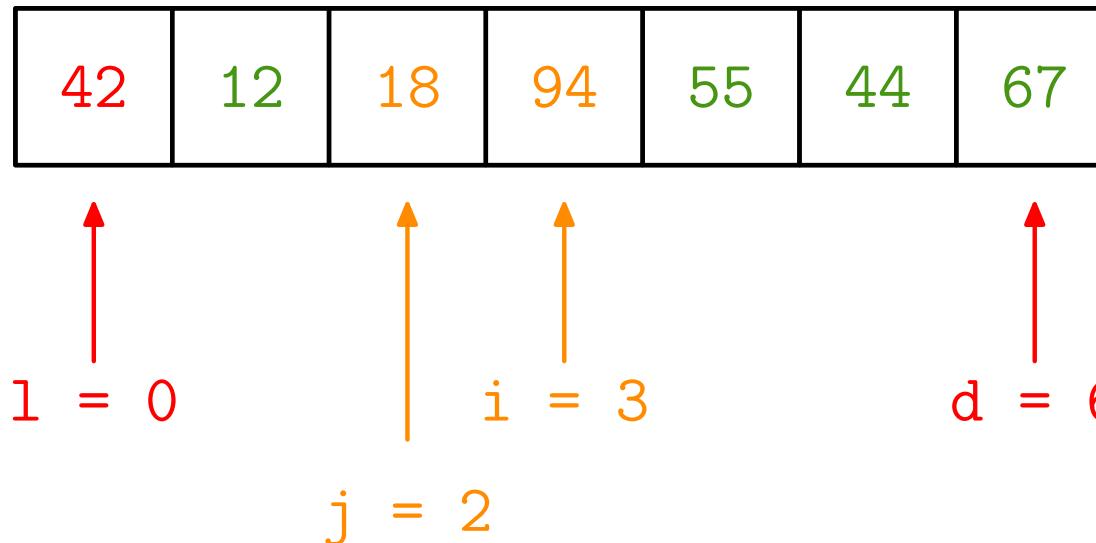
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



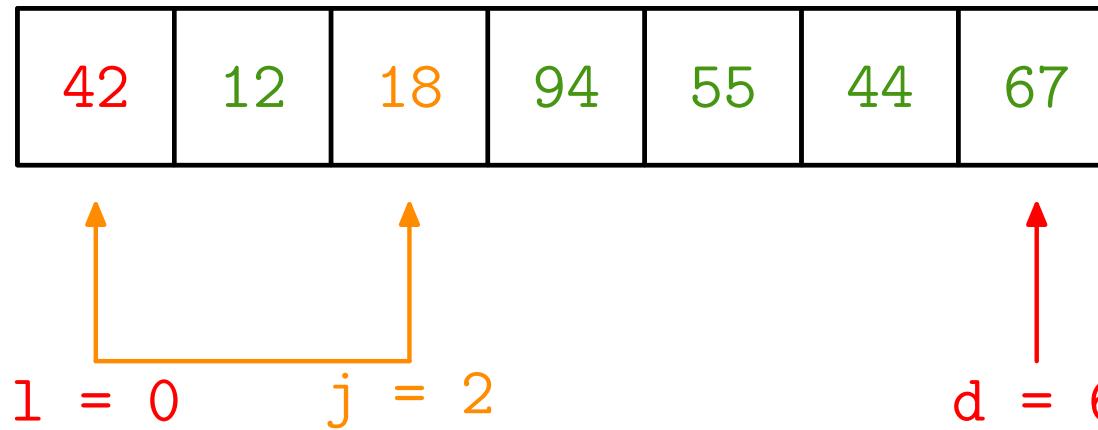
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



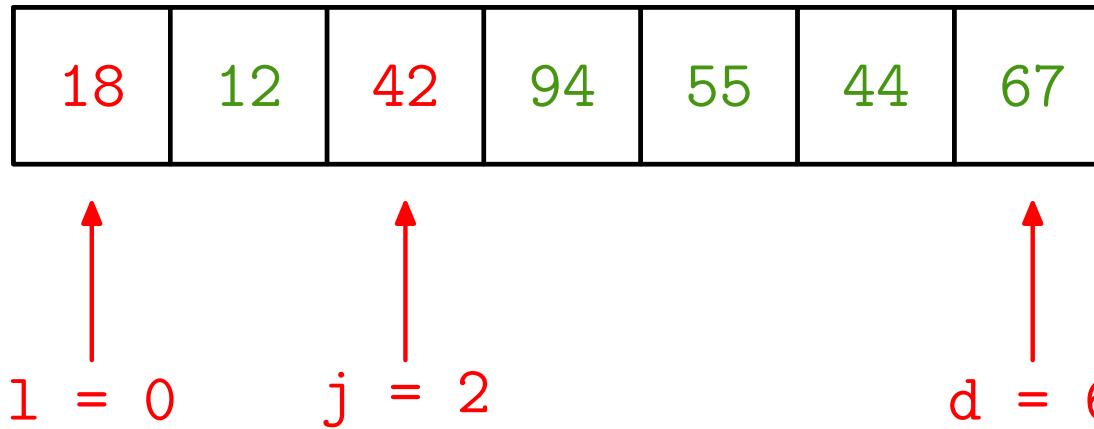
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

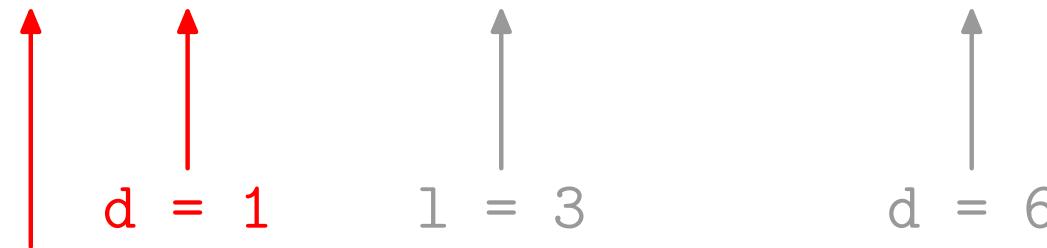
Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

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

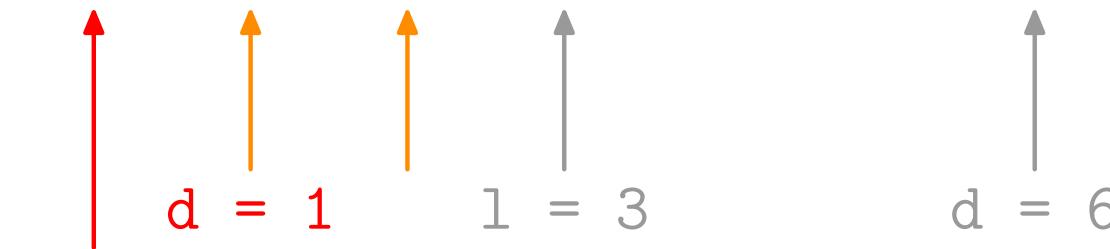


$$\begin{aligned}l &= 0 \\i &= 1 \\j &= 1\end{aligned}$$

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

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



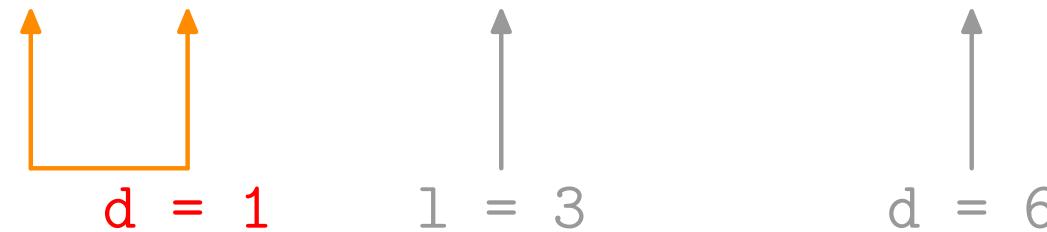
$$l = 0 \quad i = 2$$

$$j = 1$$

QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

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

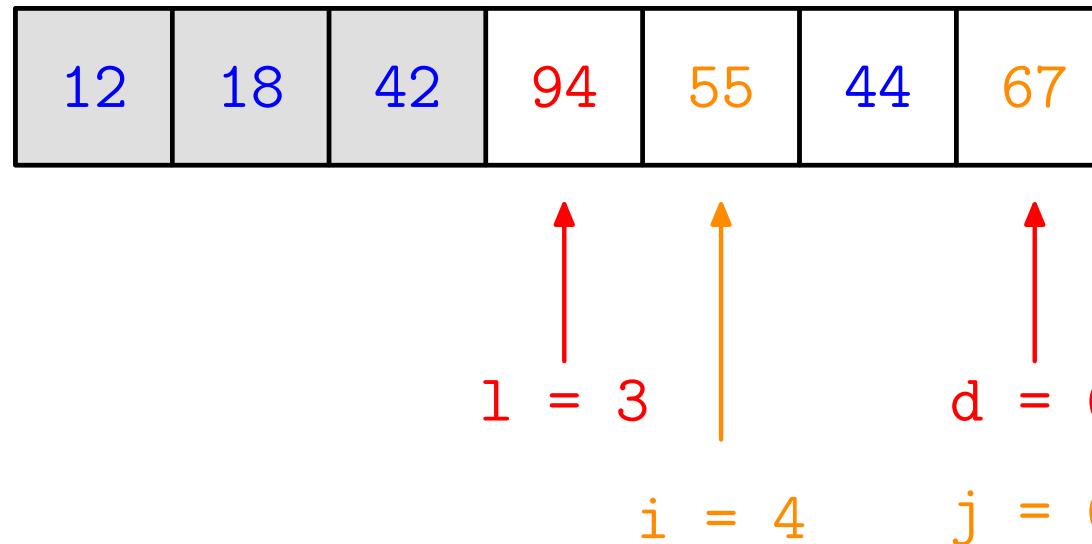


$l = 0$

$j = 1$

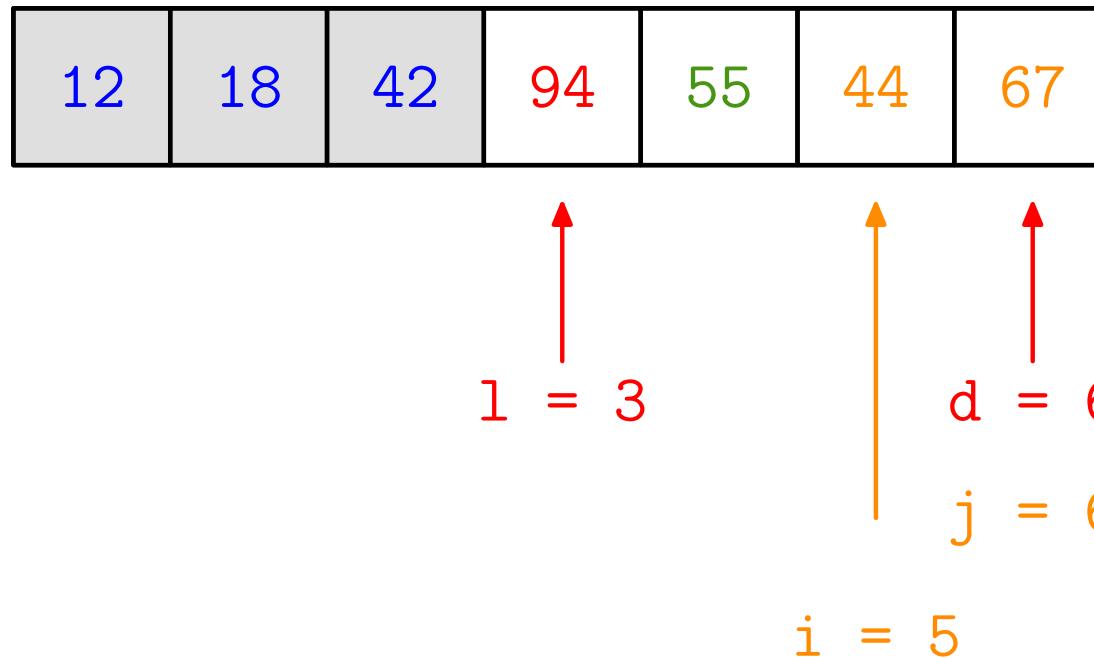
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



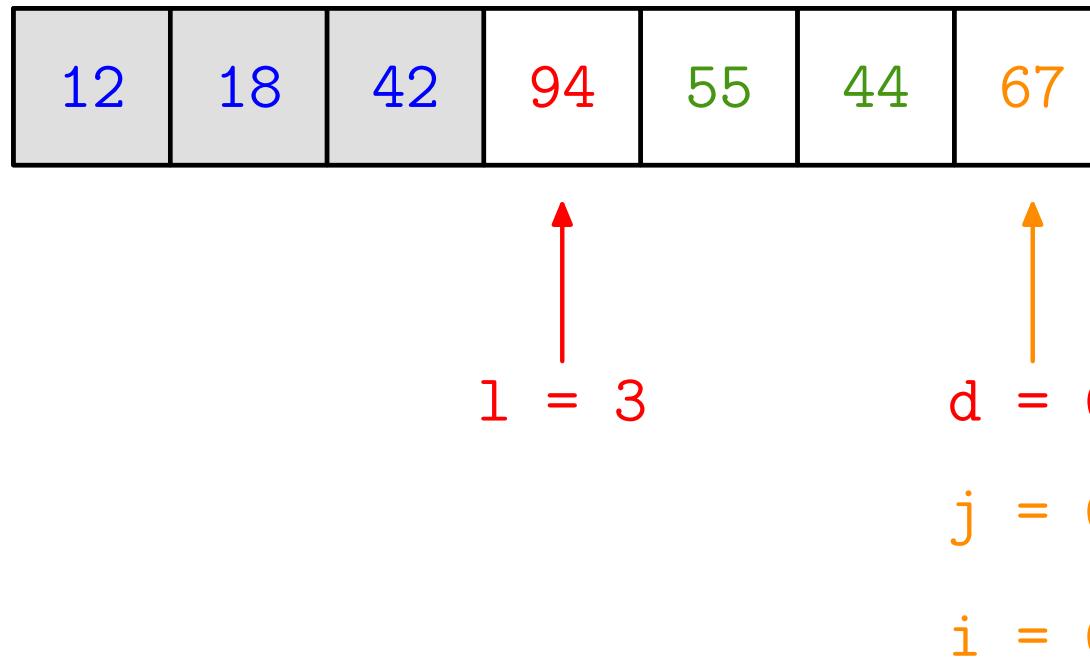
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



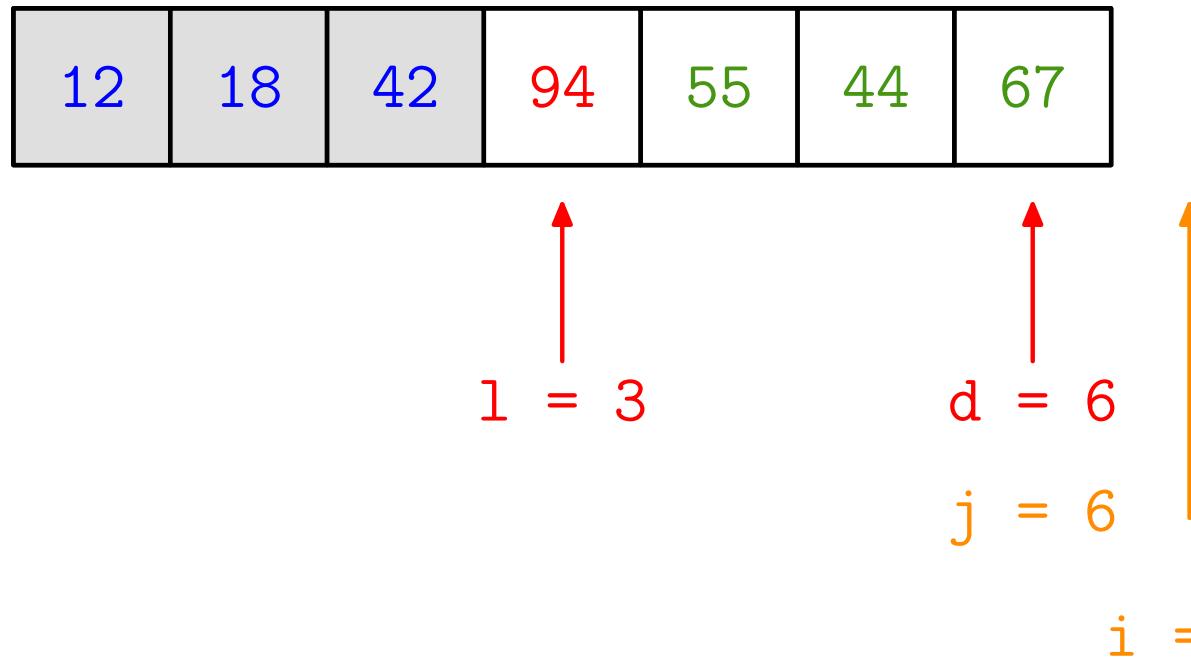
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



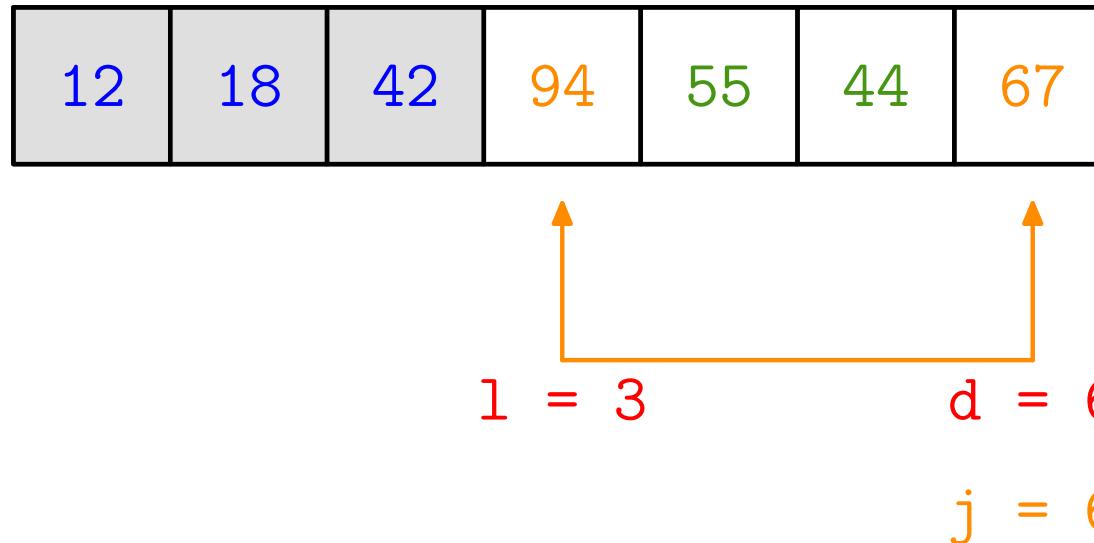
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

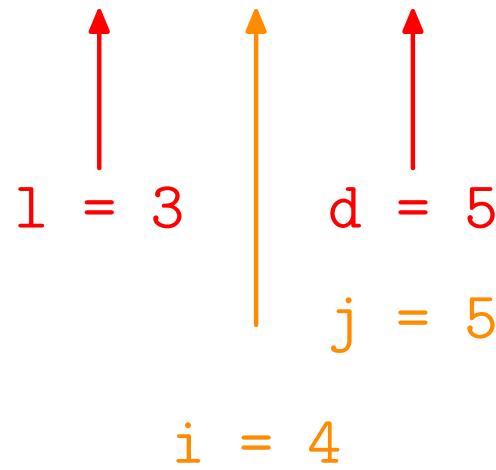
Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

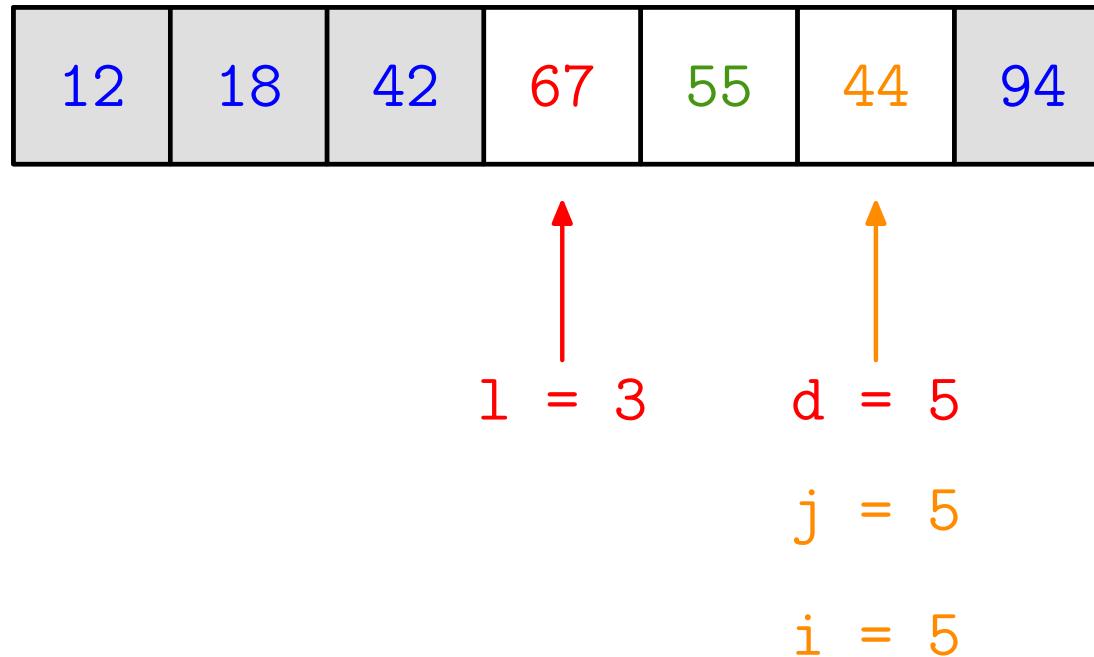
Primjer. Sortirajte korištenjem quicksorta zadano polje.

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



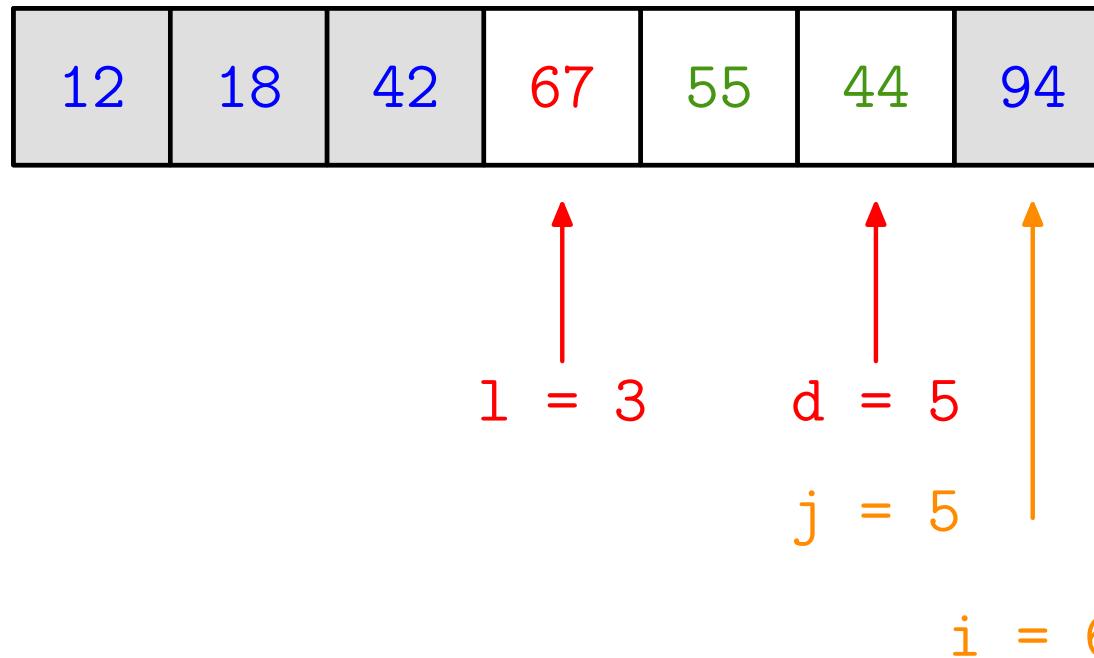
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



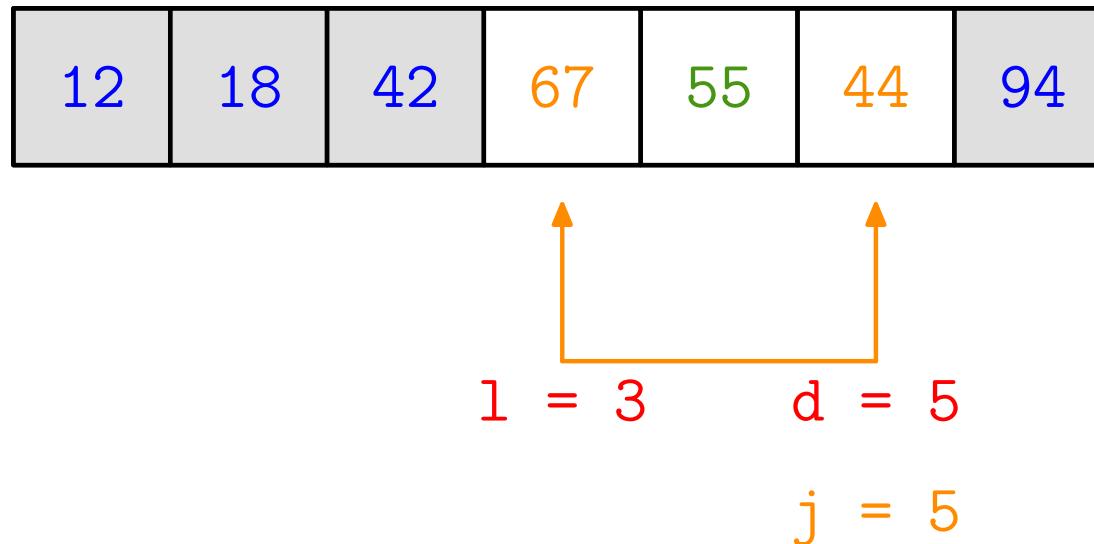
QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

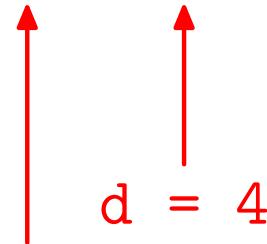
Primjer. Sortirajte korištenjem quicksorta zadano polje.



QuickSort

Primjer. Sortirajte korištenjem quicksorta zadano polje.

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

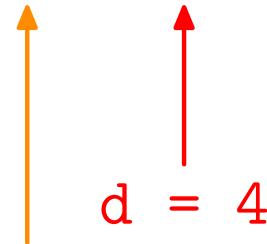


$$\begin{array}{l} l = 3 \\ i = 4 \\ j = 4 \end{array}$$

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;
}
```