Sortiranje podataka

Postupci sortiranja

- Uređivanje slijeda podataka prema nekom pravilu
- Tijekom razvoja računarstva razvijeni su brojni algoritmi za sortiranje:
  - Selection sort (sortiranje odabirom)
  - Bubble sort
  - Insertion sort (sortiranje umetanjem)
  - Shell sort
  - Merge sort (sortiranje spajanjem)
  - Quick sort
  - *Tree sort*
  - *Heap sort*

Prikaz različitih algoritama

Selection sort (sortiranje odabirom)

- Najjednostavniji algoritam
- Načelni postupak:
  1. Pronađi najmanji član u nizu [1..N].
  2. Postavi pronađeni najmanji član na početak niza.
  3. Iterativno ponovi algoritam od koraka 1 za podniz [2…N]
- Složenost izvođenja algoritma je $O(n^2)$

Selection sort (sortiranje odabirom)

- Ostvarenje:

```
for (i=0;i<(N-1);++i)
  for (j=i+1;j<N;++j)
    if (A[i]>A[j])
    {
      temp=A[i];
      A[i]=A[j];
      A[j]=temp;
    }
```

- Tijekom rada algoritma, elementi s indeksom manjim od `i` već su sortirani (nalaze se na svojim mjestima)
- Što je potrebno promijeniti da bi se niz uredio padajućim redoslijedom?

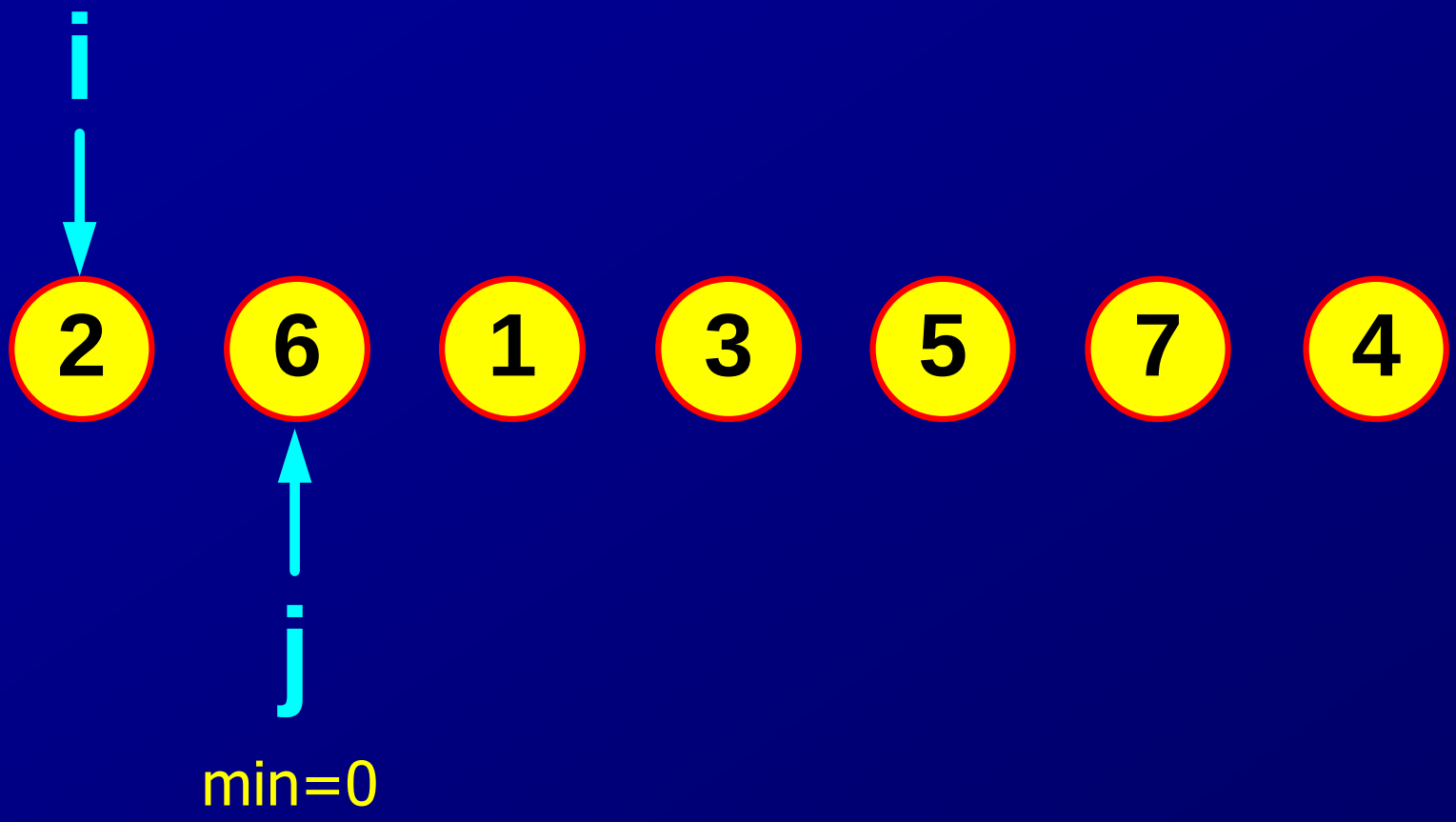
Selection sort (sortiranje odabirom)

- "Optimirano" ostvarenje:

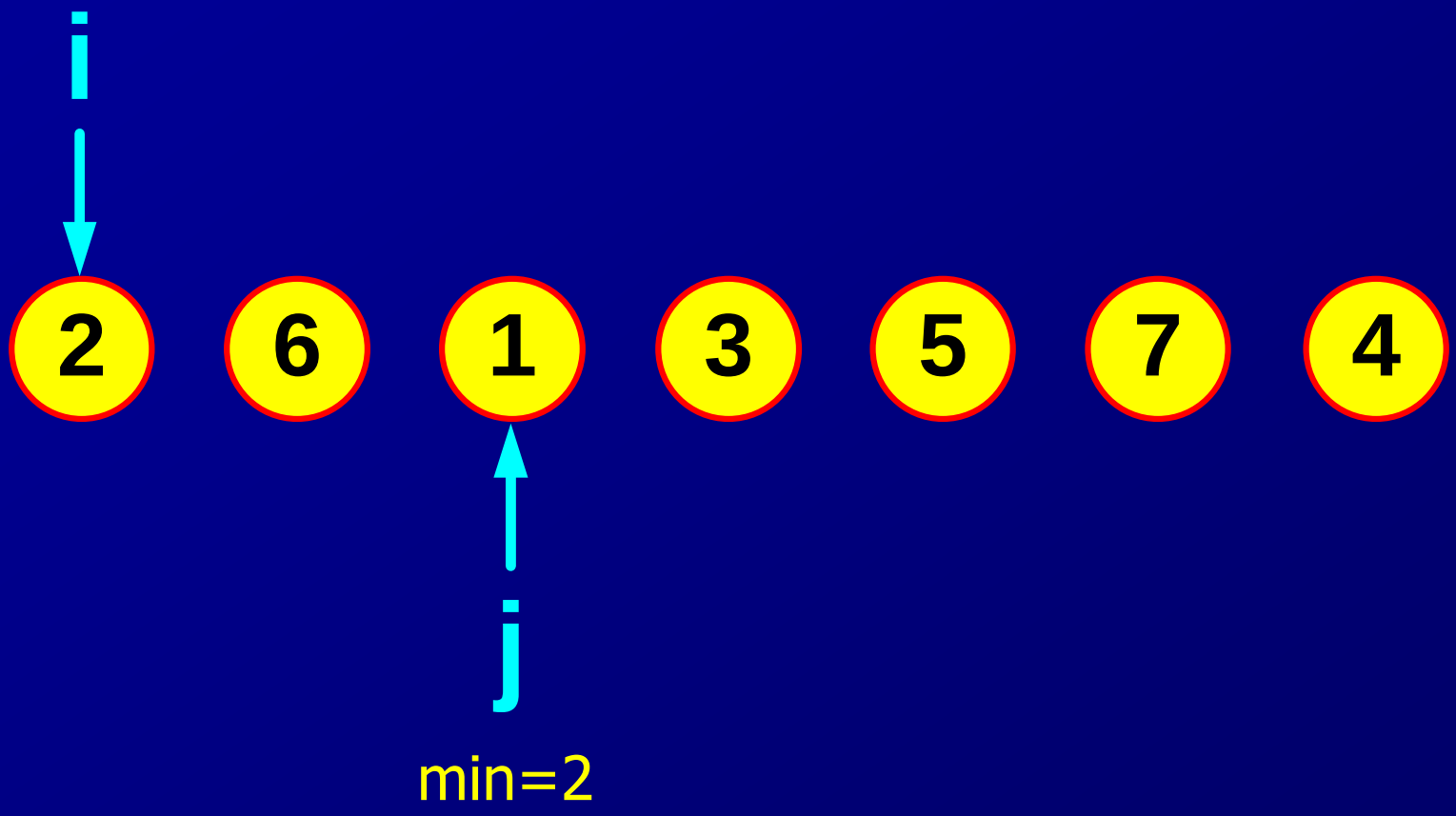
```
for (i=0;i<(N-1);++i) {
  min=i;
  for (j=i+1;j<N;++j)
    if (A[j]<A[min])
      min=j;
  if (min!=i) {
    temp=A[i];
    A[i]=A[min];
    A[min]=temp;
  }
}
```

Sortovi

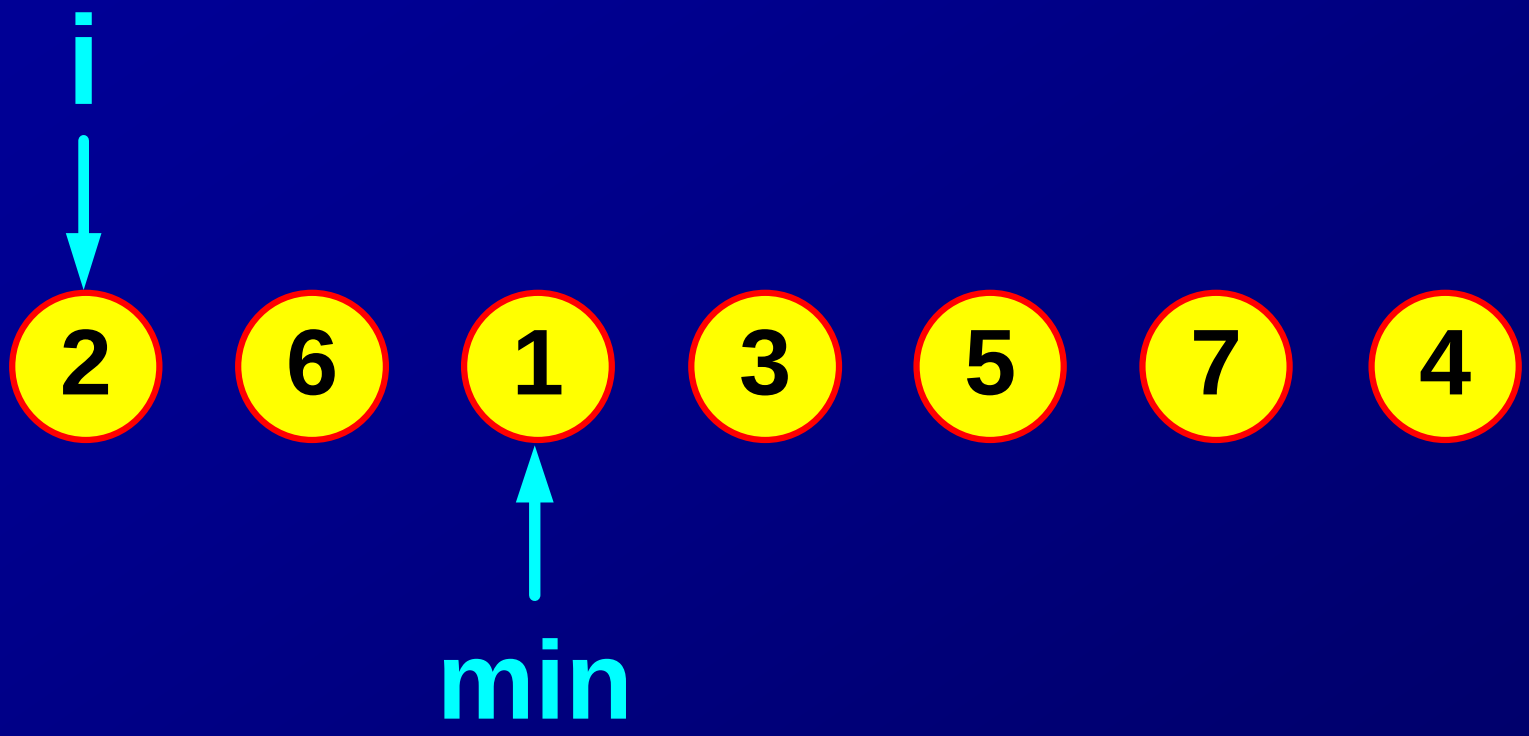
Selection sort (sortiranje odabirom)



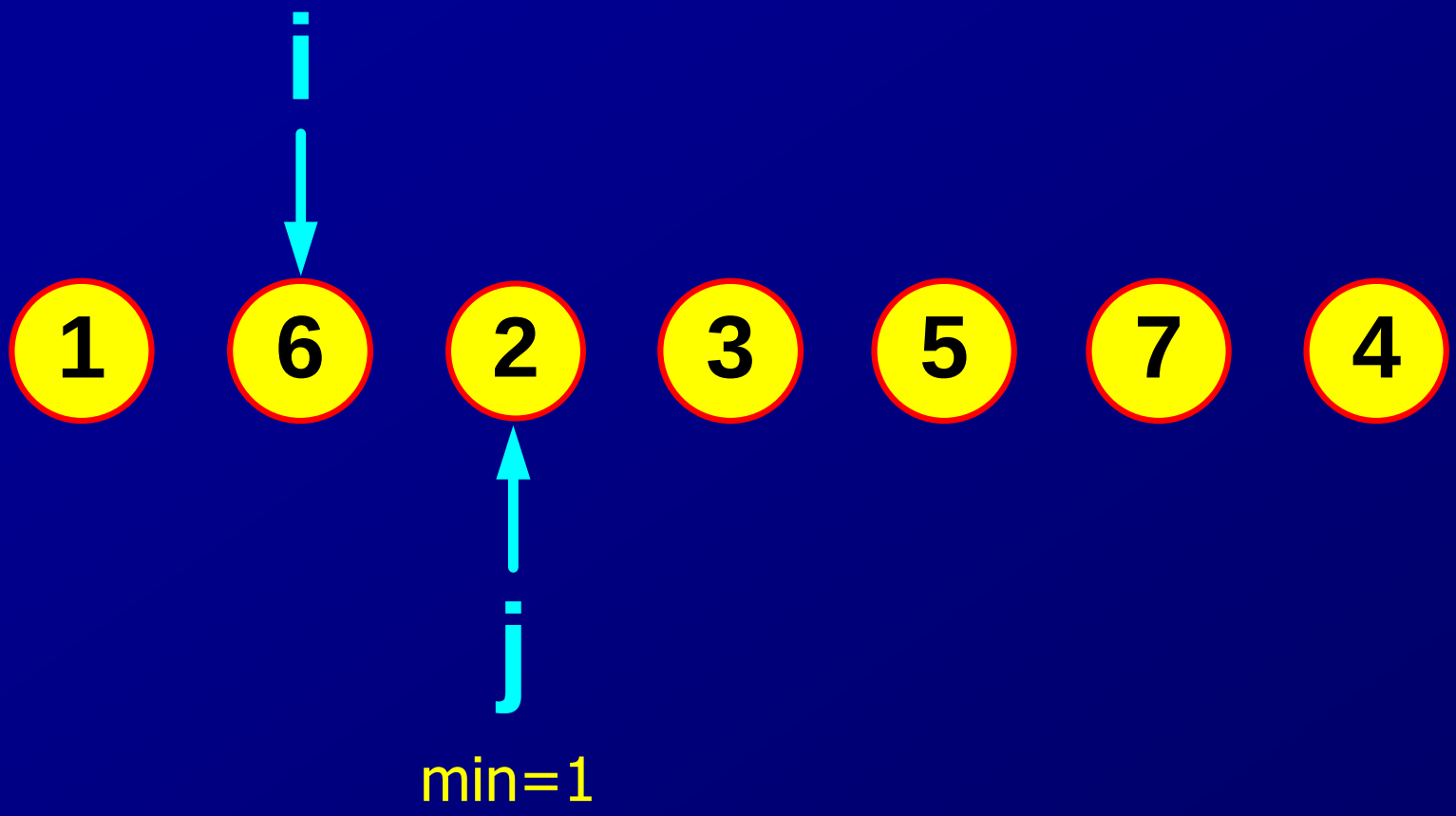
Selection sort (sortiranje odabirom)



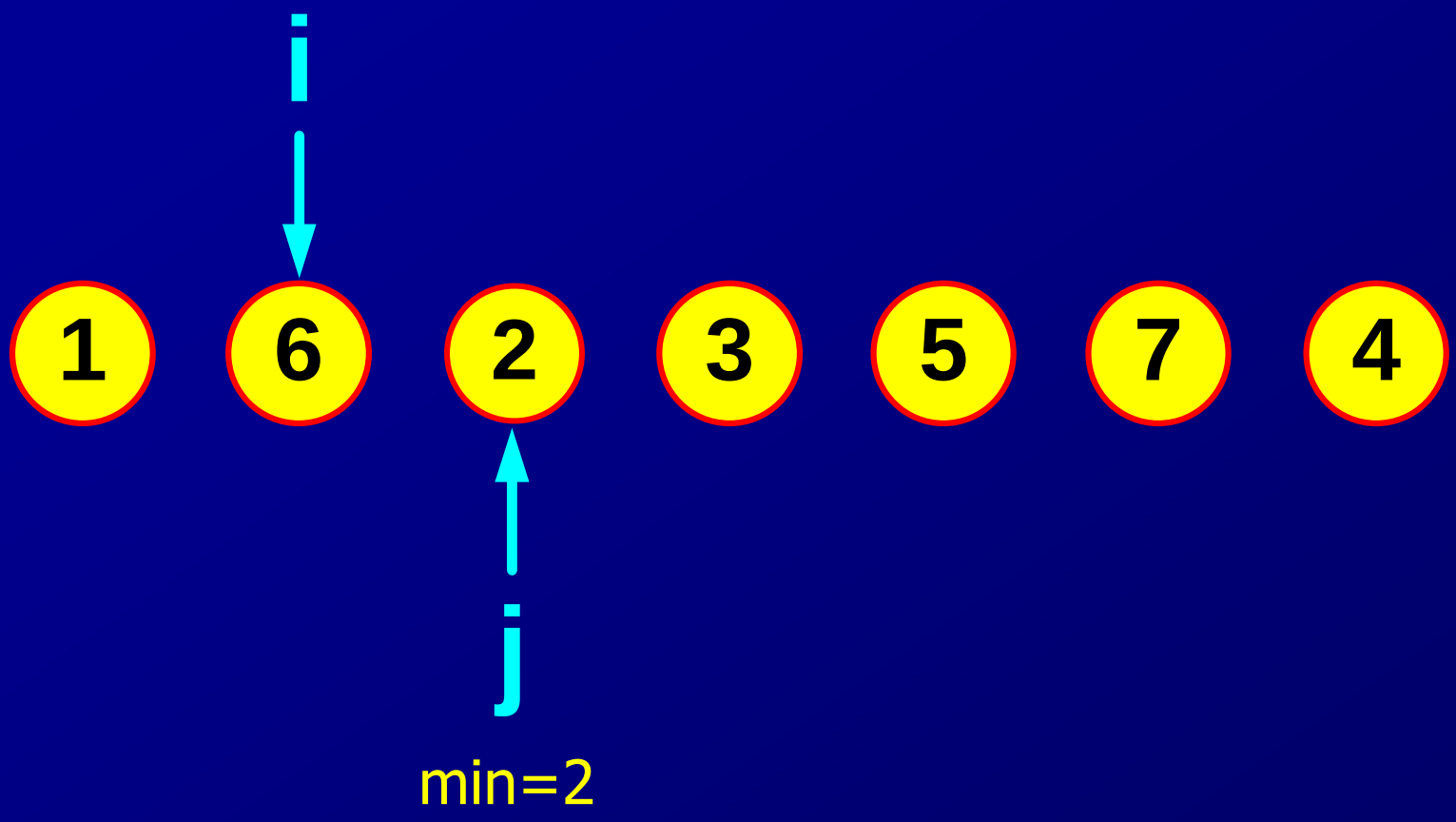
Selection sort (sortiranje odabirom)



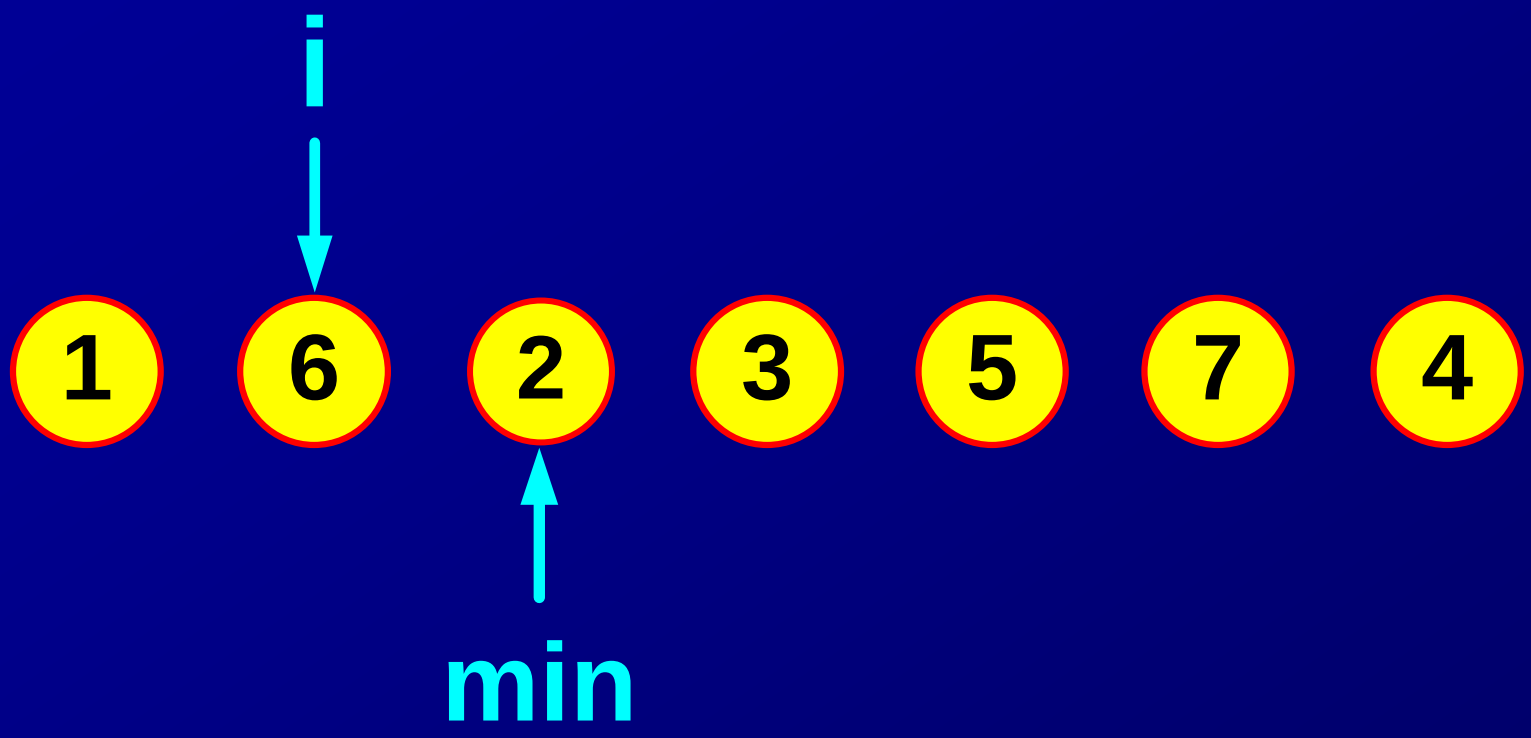
Selection sort (sortiranje odabirom)



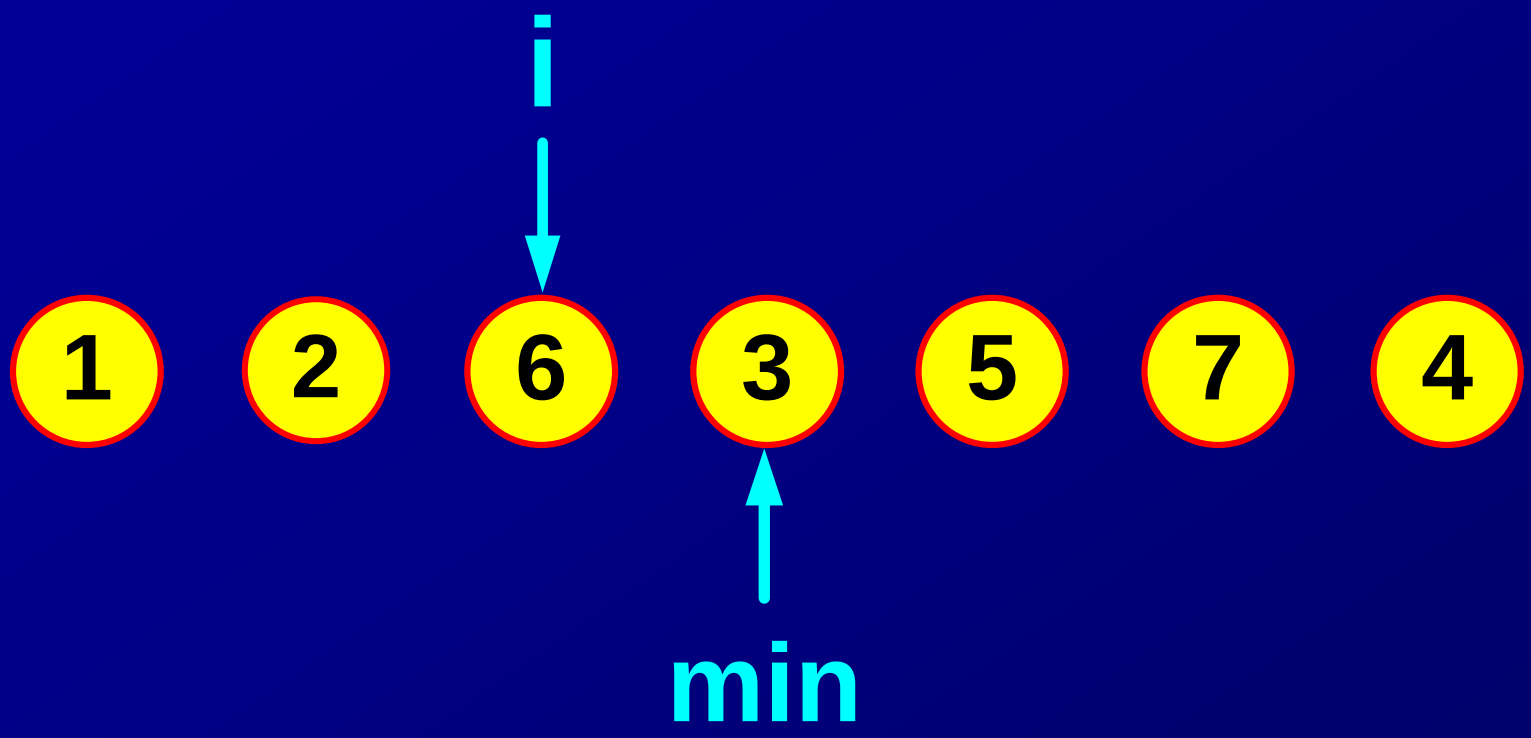
Selection sort (sortiranje odabirom)



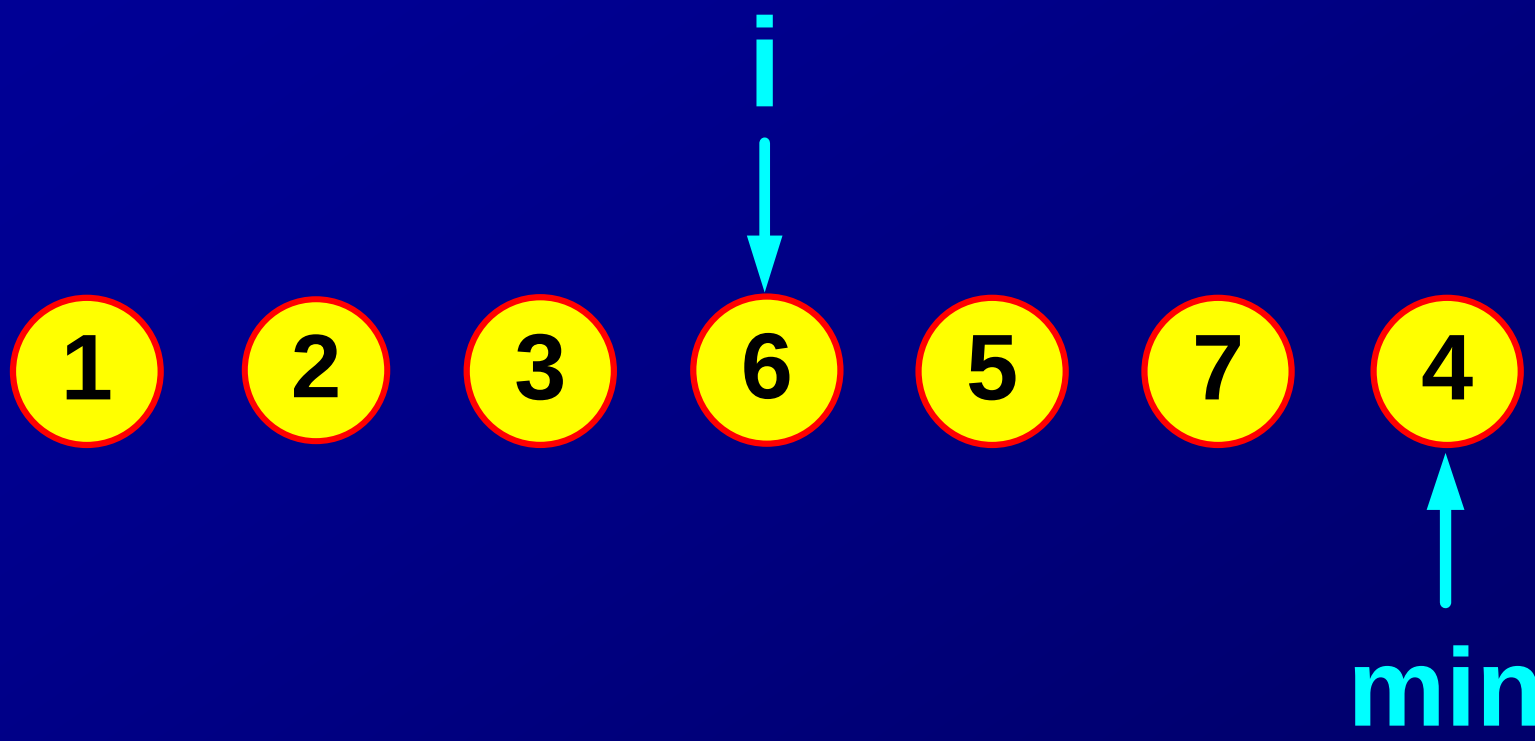
Selection sort (sortiranje odabirom)



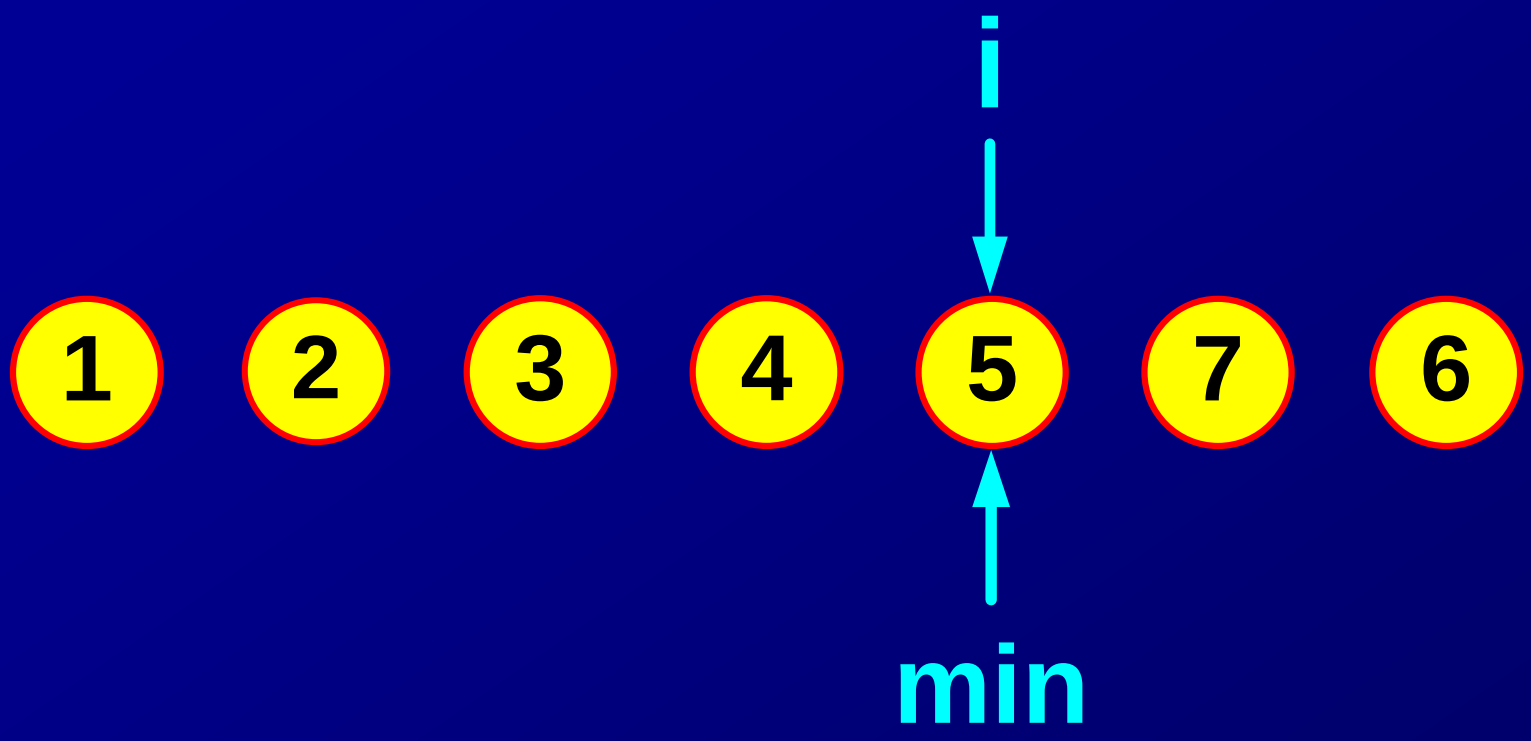
Selection sort (sortiranje odabirom)



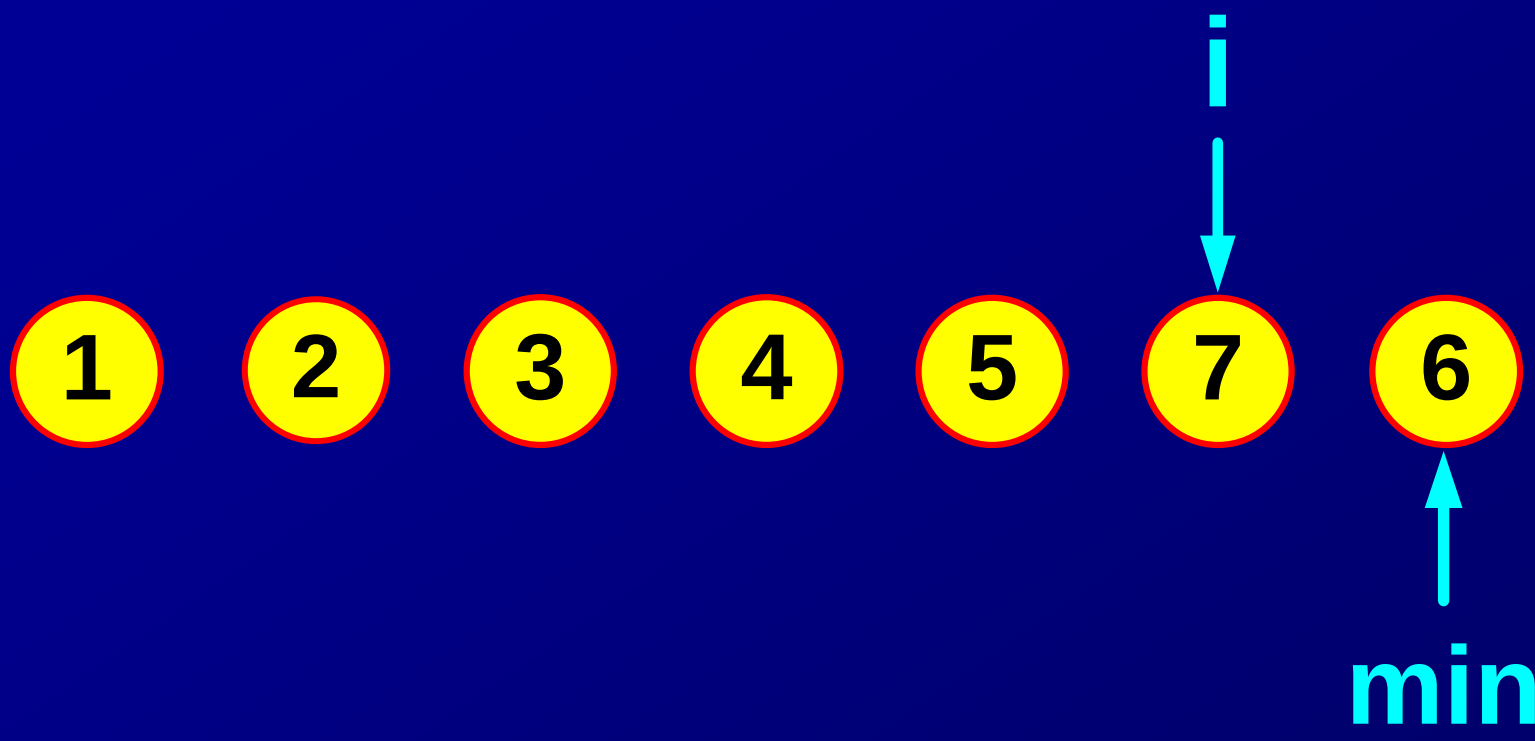
Selection sort (sortiranje odabirom)



Selection sort (sortiranje odabirom)



Selection sort (sortiranje odabirom)



Bubble sort

- Sličan selection sortu
- Određuje se položaj jednog po jednog člana počevši od kraja niza
- Razlika u odnosu na selection sort je da se uspoređuju i zamjenjuju samo susjedni elementi
- Podsjeća na mjehuriće u vodi
  - Počevši od dna, mjehurići koji su lakši od okoline putuju prema površini
    - Mjehurić može biti najmanji ili najveći podatak, ovisno o načinu uređivanja niza
  - Na površinu prvi ispliva najlakši mjehurić (najveći)
- Složenost izvođenja algoritma je $O(n^2)$.

Bubble sort

- Ostvarenje:

```
for (i=0;i<N;++i)
  for (j=0;j<(N-1-i);++j)
    if (A[j]>A[j+1])
    {
      temp=A[j];
      A[j]=A[j+1];
      A[j+1]=temp;
    }
```

- Tijekom rada algoritma, elementi s indeksom većim od `N-1-i` već su sortirani (nalaze se na svojim mjestima)
- Što treba promijeniti da bi se niz uredio padajućim redoslijedom?

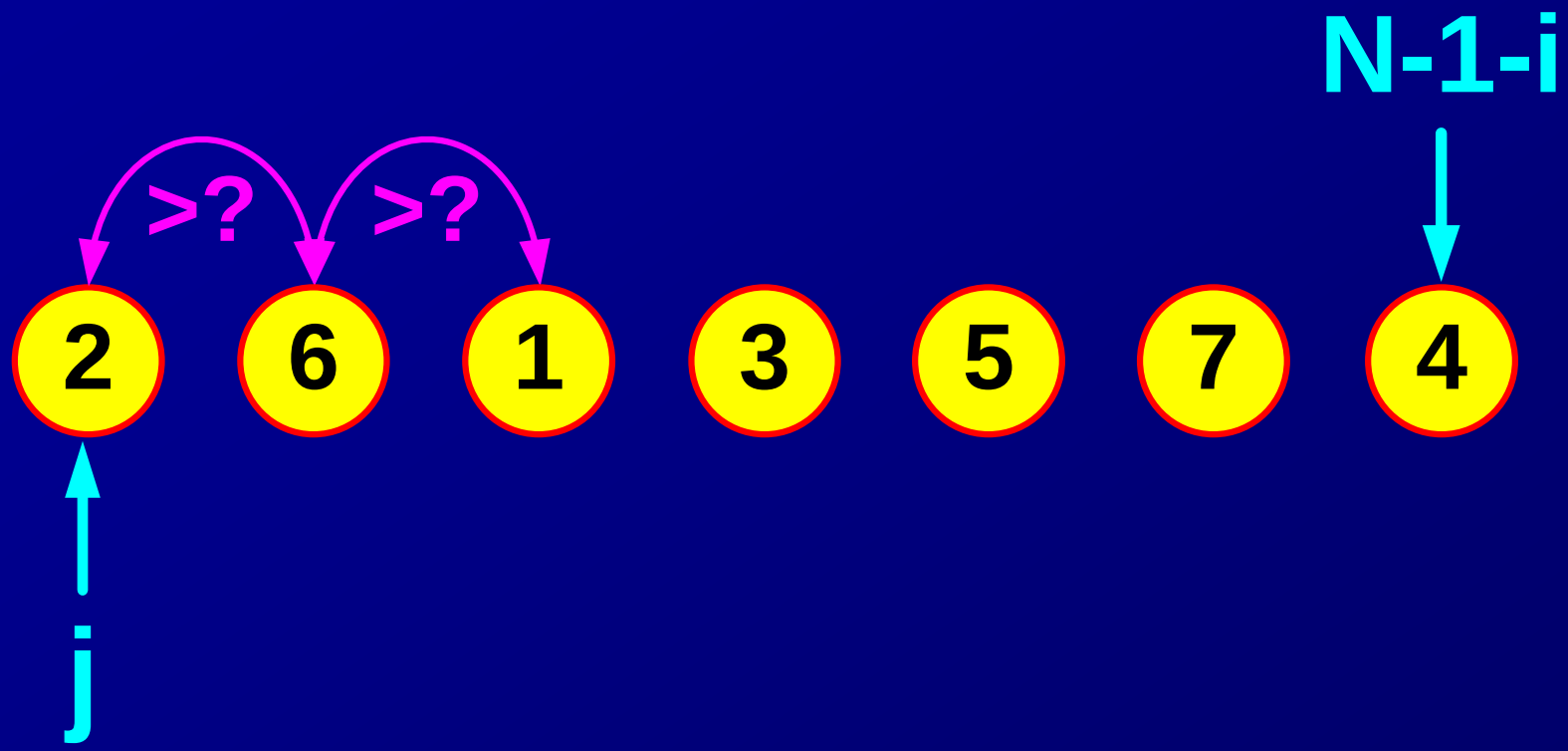
Bubble sort

- "Optimirano" ostvarenje:

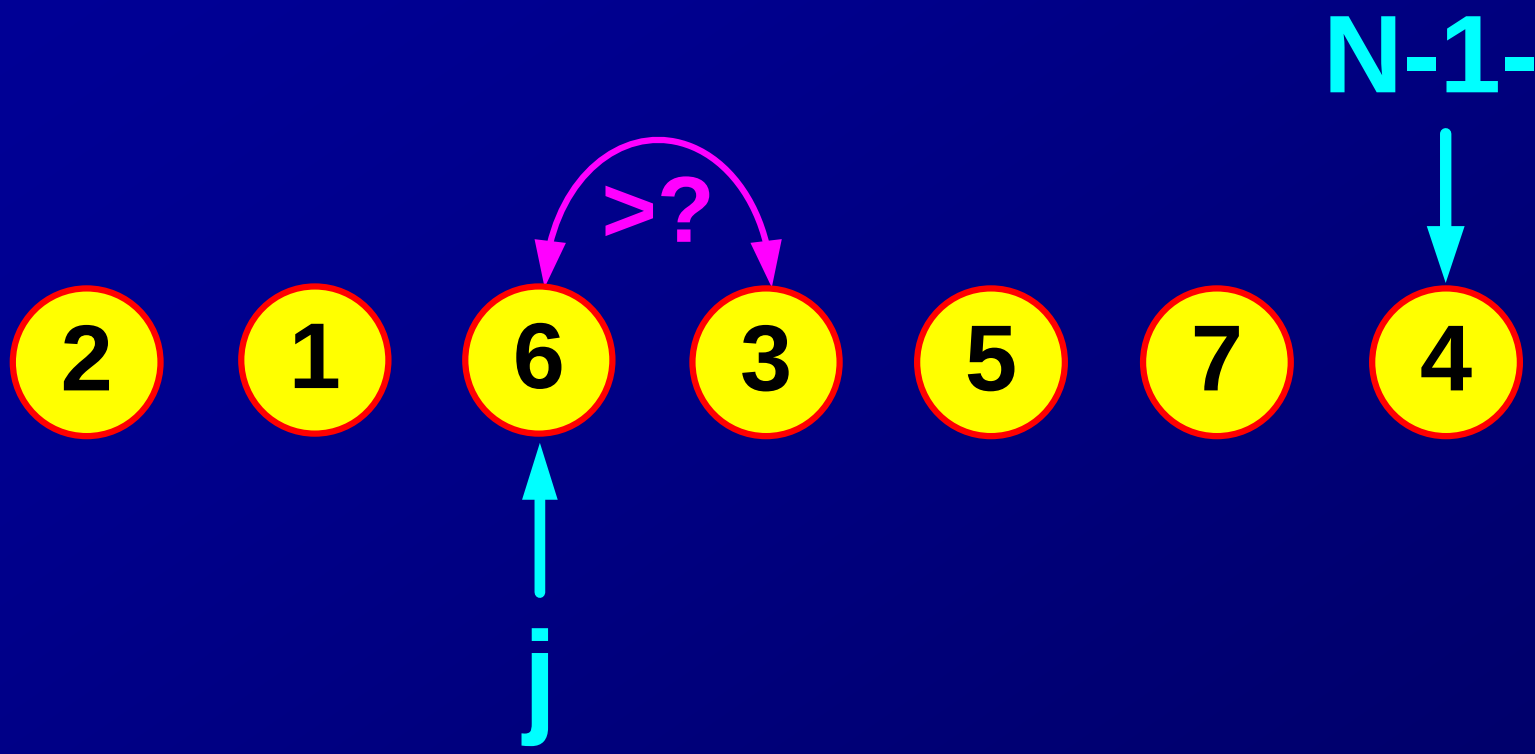
```
for (i=0,bZamjena=1;bZamjena;++i) {
  bZamjena=0;
  for (j=0;j<(N-1-i);++j)
    if (A[j]>A[j+1]) {
      temp=A[i];
      A[i]=A[min];
      A[min]=temp;
      bZamjena=1;
    }
}
```

Sortovi

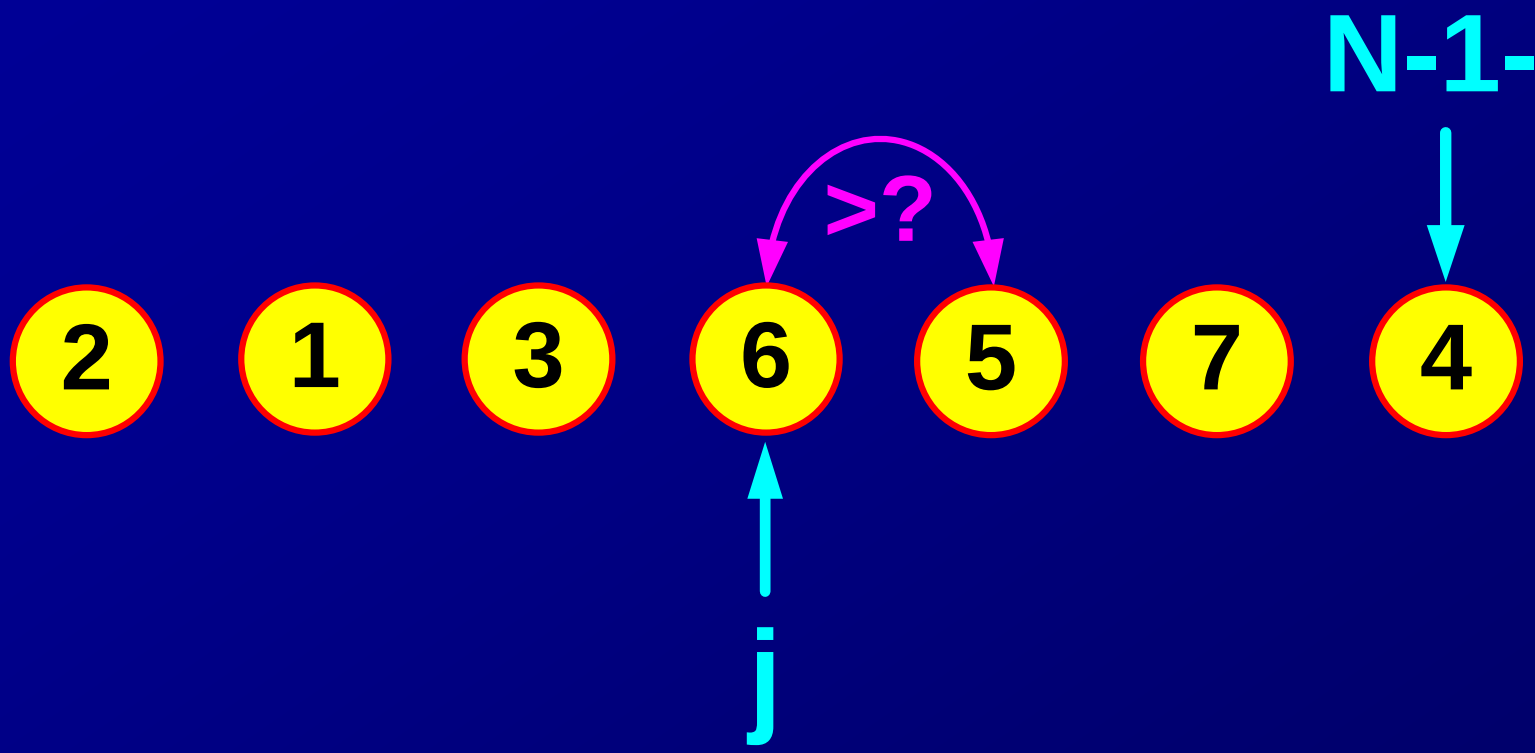
Bubble sort



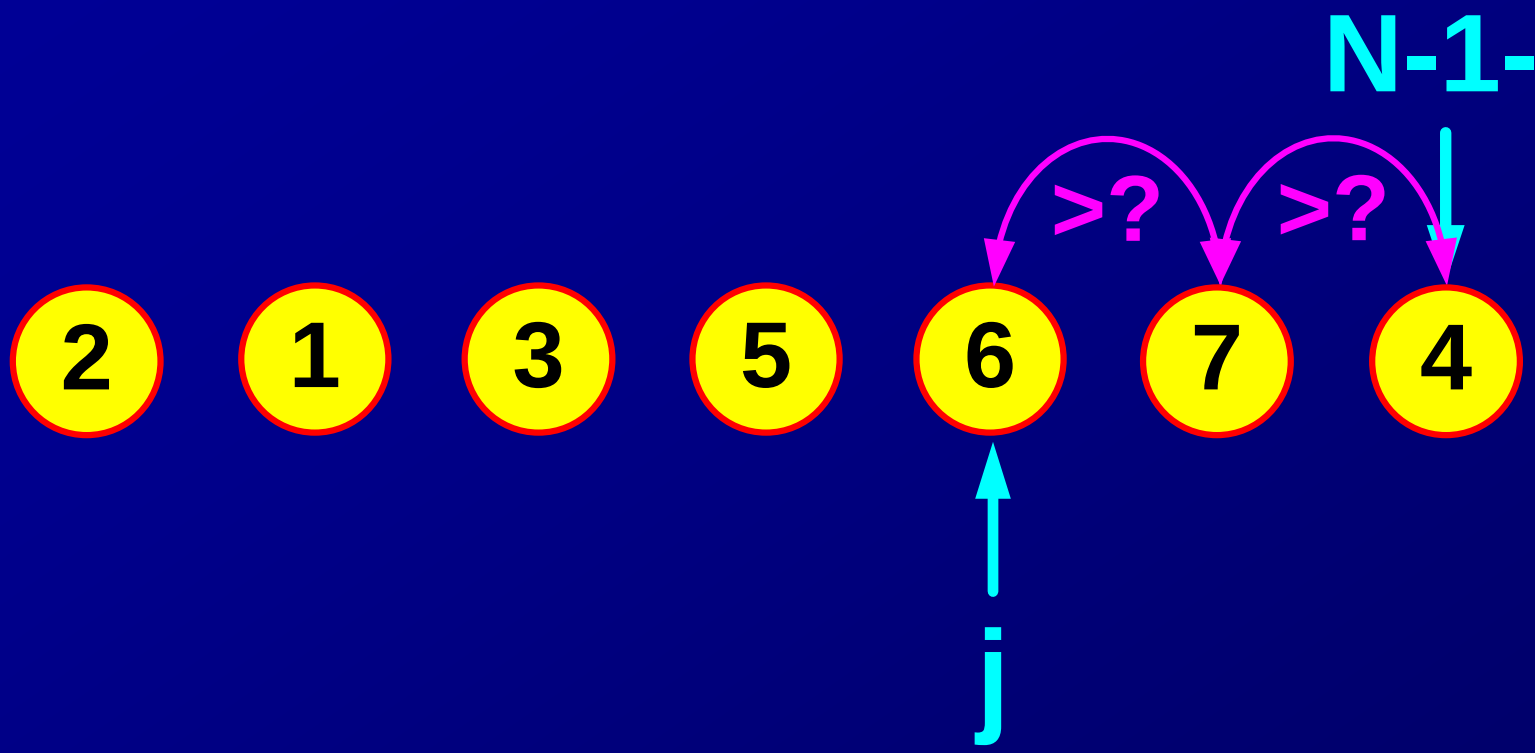
Bubble sort



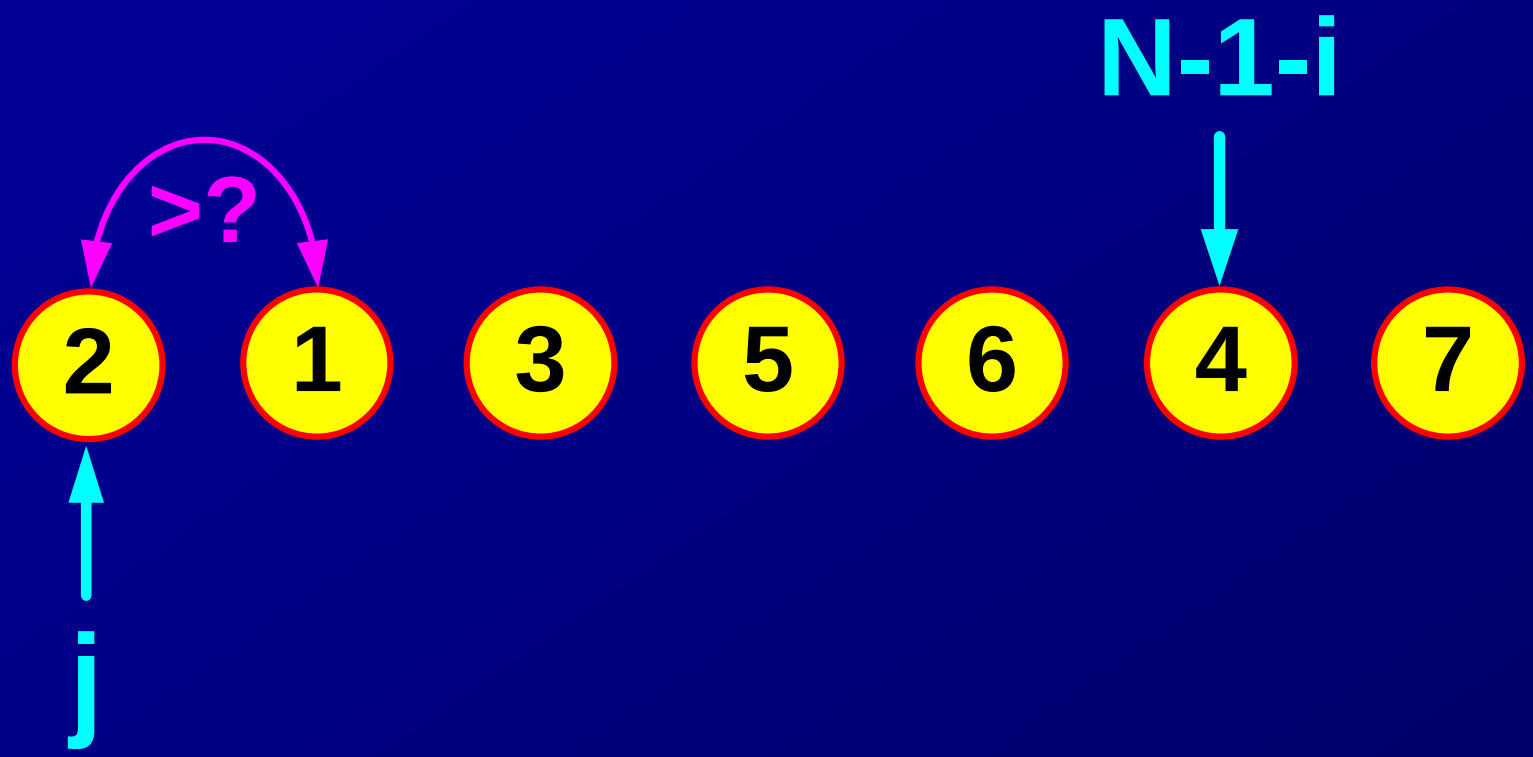
Bubble sort



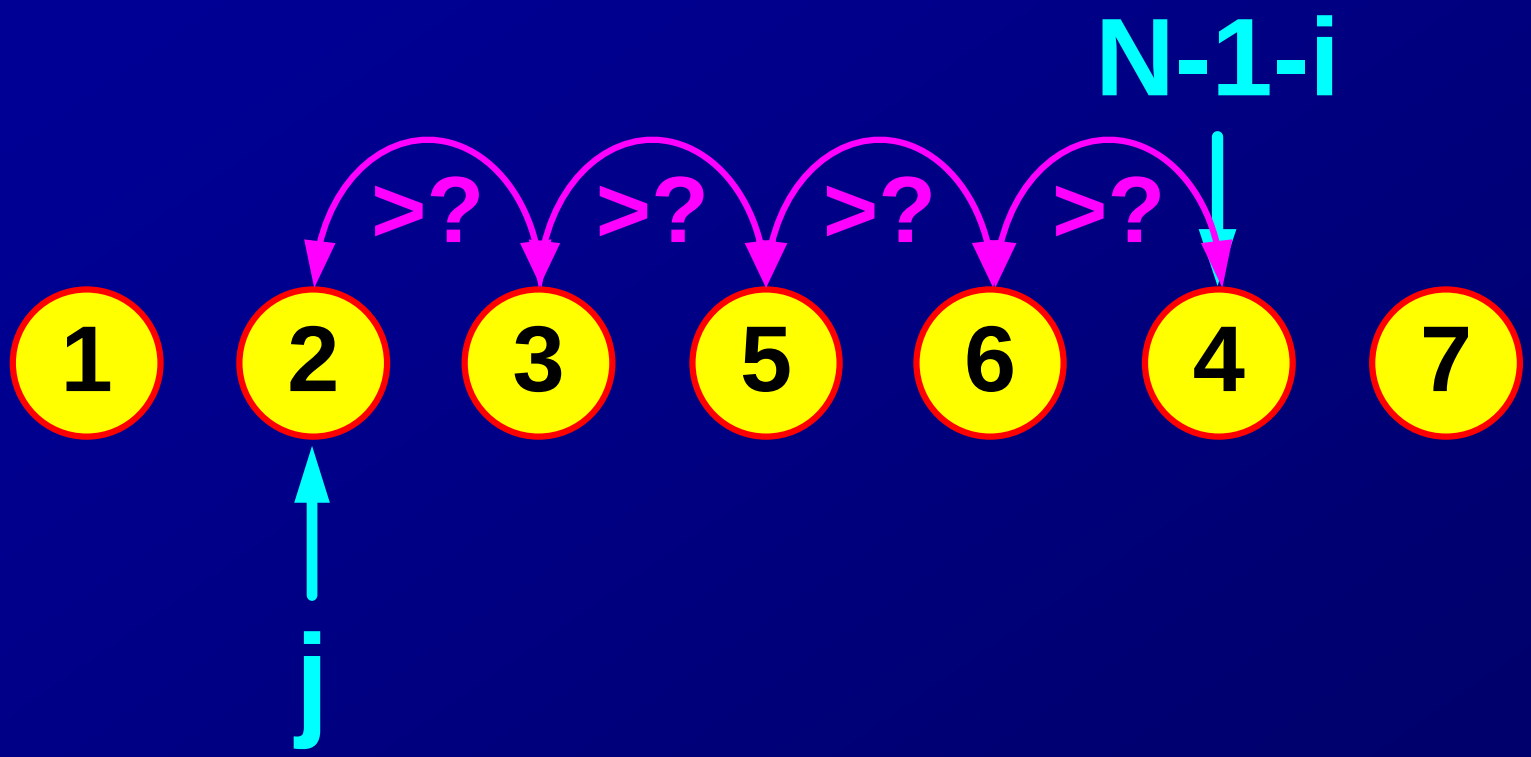
Bubble sort



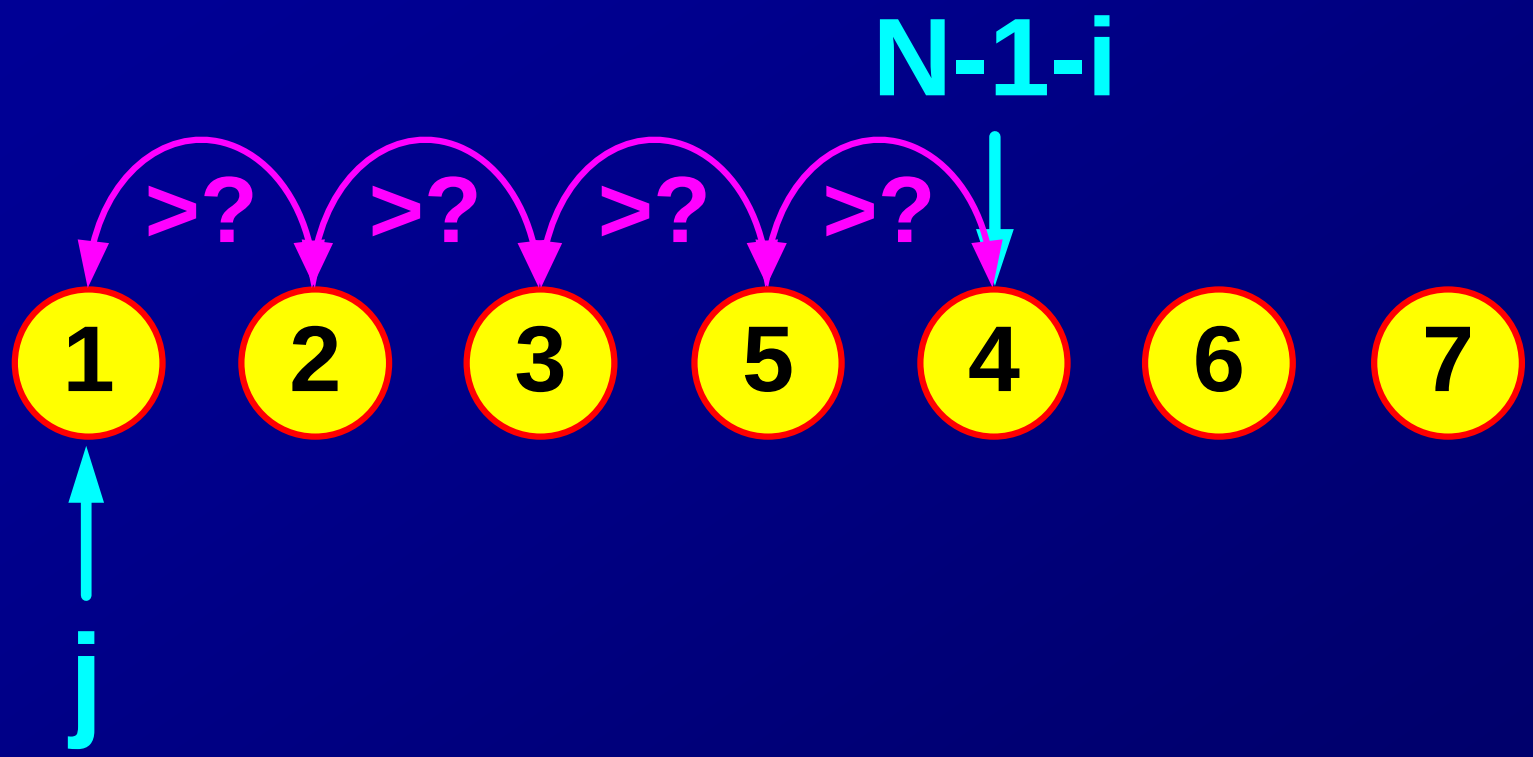
Bubble sort



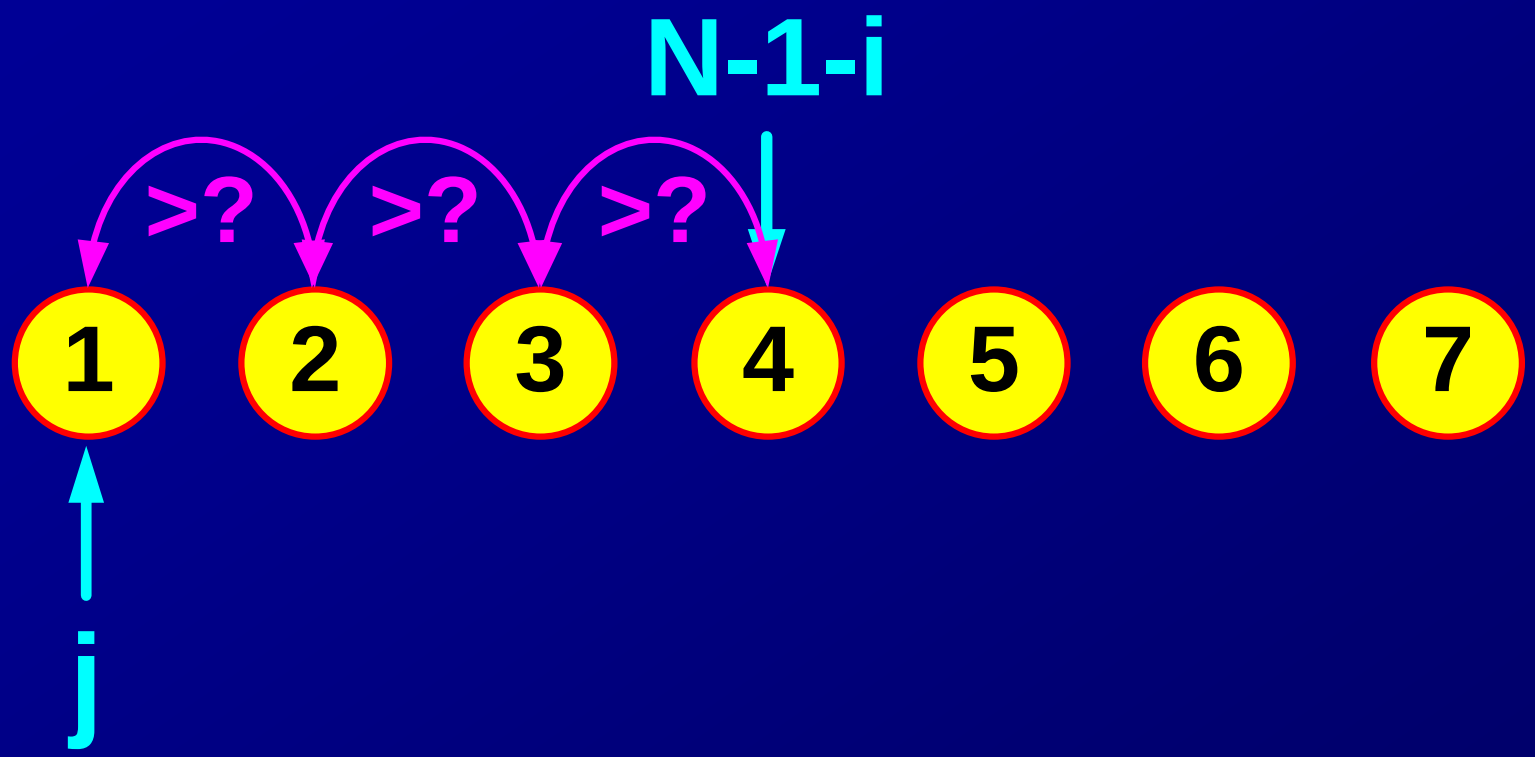
Bubble sort



Bubble sort



Bubble sort



- Algoritam završava jer je prošao "nesortirani" dio bez izvođenja ijedne zamjene

Insertion sort

- Sortiranje umetanjem
- Načelni rad algoritma:
  1. Uzmi postojeći sortirani niz i novi element
  2. Pronađi poziciju na koju treba postaviti novi element
  3. Stvori prazno mjesto na nađenoj poziciji i umetni novi element
  4. Ponovi postupak od koraka 1 za sljedeći novi element
- Sortiranje započinje s nizom duljine 1
- Složenost izvođenja algoritma je $O(n^2)$.

Insertion sort

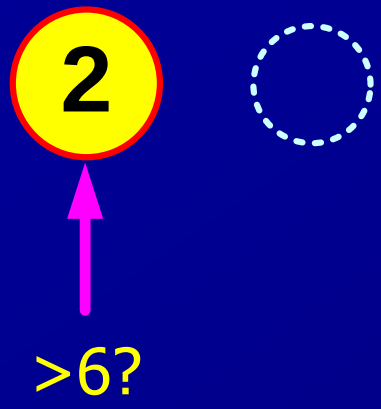
- Ostvarenje:

```
for (i=1;i<N;++i)
{
  temp=A[i];
  for (j=i;j>0;--j)
    if (temp<A[j-1])
      A[j]=A[j-1];
    else
      break;
  A[j]=temp;
}
```

Sortovi

Insertion sort

- Sortirani niz:

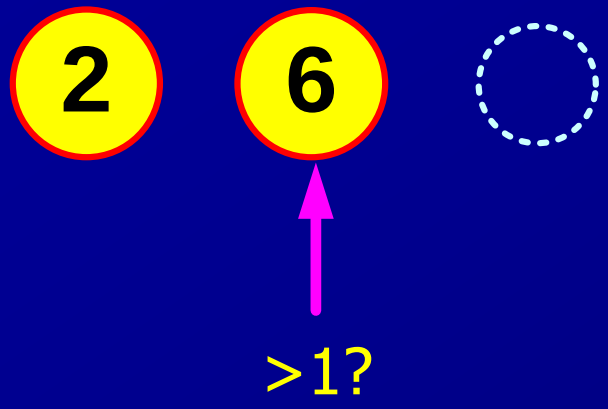


- Novi element:

- Nesortirani podaci:

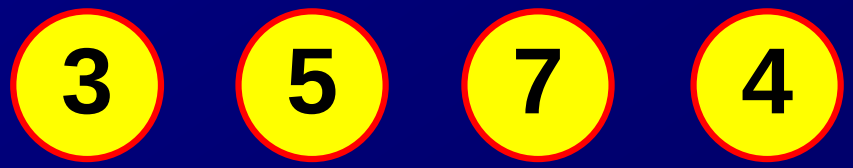
Insertion sort

- Sortirani niz:



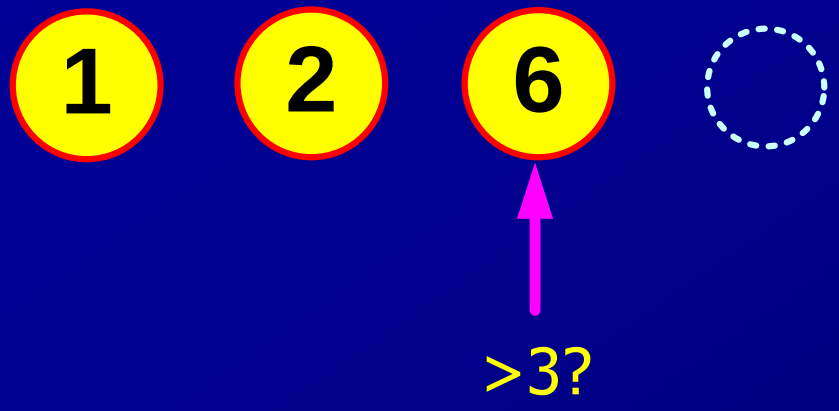
- Novi element:

- Nesortirani podaci:



Insertion sort

- Sortirani niz:

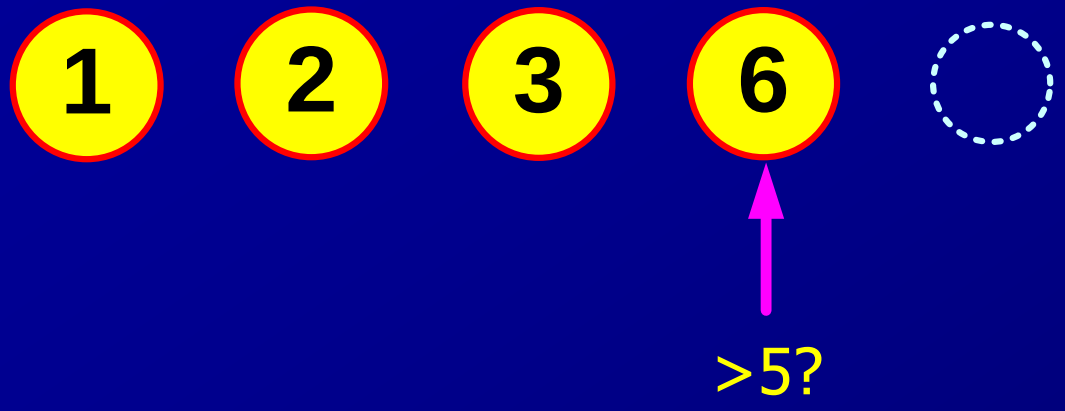


- Novi element:

- Nesortirani podaci:

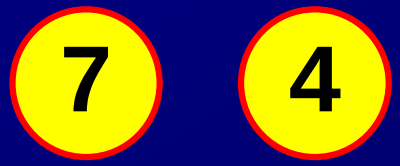
Insertion sort

- Sortirani niz:



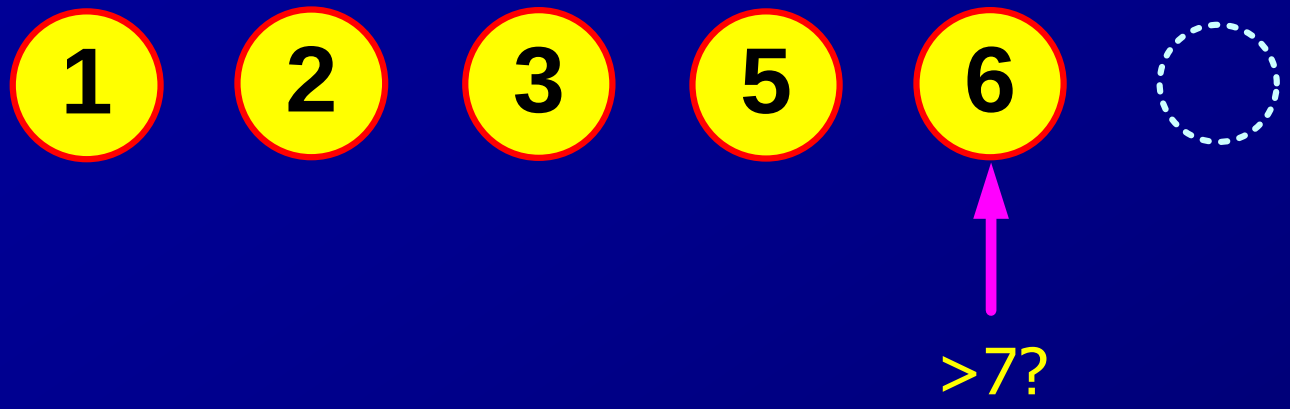
- Novi element:

- Nesortirani podaci:



Insertion sort

- Sortirani niz:

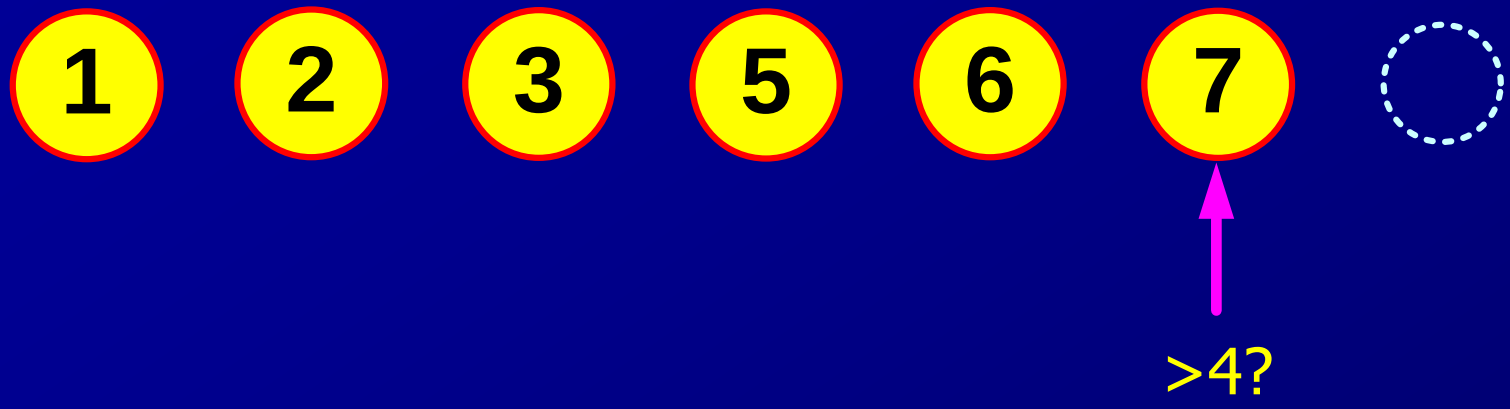


- Novi element:

- Nesortirani podaci:

Insertion sort

- Sortirani niz:



- Novi element:

- Nesortirani podaci:

Shell sort

- Autor algoritma je Donald Shell
- Načelni rad algoritma:
  1. Odredi rastući slijed prirodnih brojeva $h_0=1,\ h_1,\ h_2,\ \ldots\ ,\ h_m$
  2. $k=m$
  3. Sortiraj polje tako da je $\mathbf{A[i] \le A[i+h_k],\ \forall i \in [0, N-1-h_k]}$
     - Za dobiveno polje kaže se da je h_k sortirano
     - Ako se polje sortirano s h_k ponovo sortira, ali s h_{k-1} ono i dalje ostaje h_k sortirano
  4. Smanji k za jedan i ponovi korak 3 postupka za sve preostale h_k
- Nastao je analizom nedostataka *insertion sort* i *bubble sort* algoritma
  - Obrnuto sortirani niz uzrokuje veliki broj nepotrebnih prijenosa vrijednosti elemenata.
- Složenost izvođenja algoritma je $O(n^2)$.
  - Najgori zamislivi slučaj.
  - Analiza složenosti još uvijek je otvoreno pitanje!

Shell sort

- Ostvarenje:

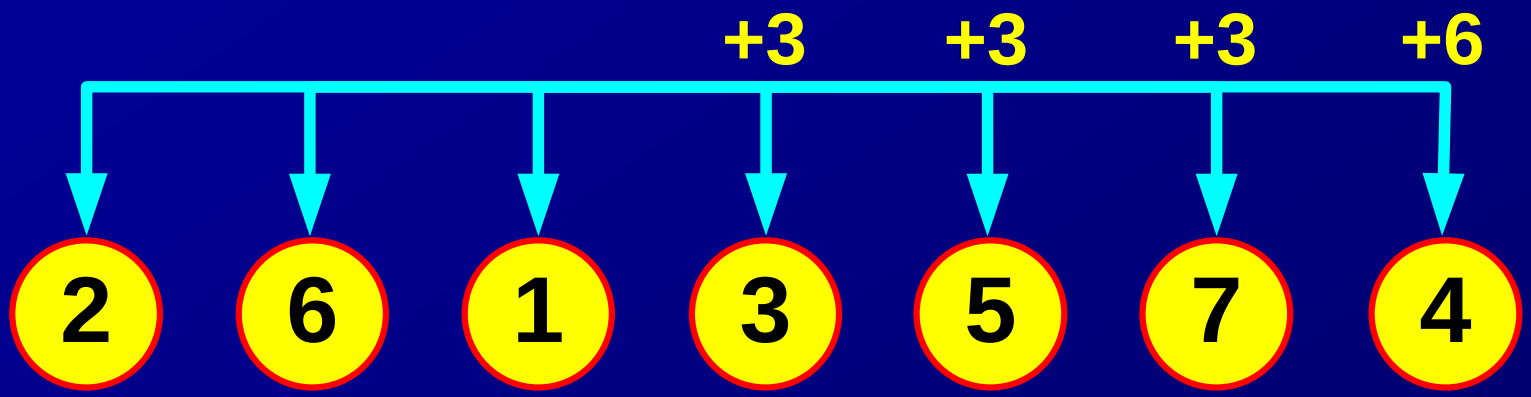
```
shellsort(itemType a[], int N) {
  int i, j, k, h;
  itemType v;
  int incs[16] = { 1391376, 463792, 198768, 86961, 33936,
    13776, 4592, 1968, 861, 336, 112, 48, 21, 7, 3, 1 };

  for ( k = 0; k < 16; ++k)
    for (h = incs[k], i = h; i < N; ++i) {
        v = a[i];
       for (j = i; (j > h) && (a[j-h] > v) ; j -= h)
         a[j] = a[j-h];
        a[j] = v;
      }
}
```

Sortovi

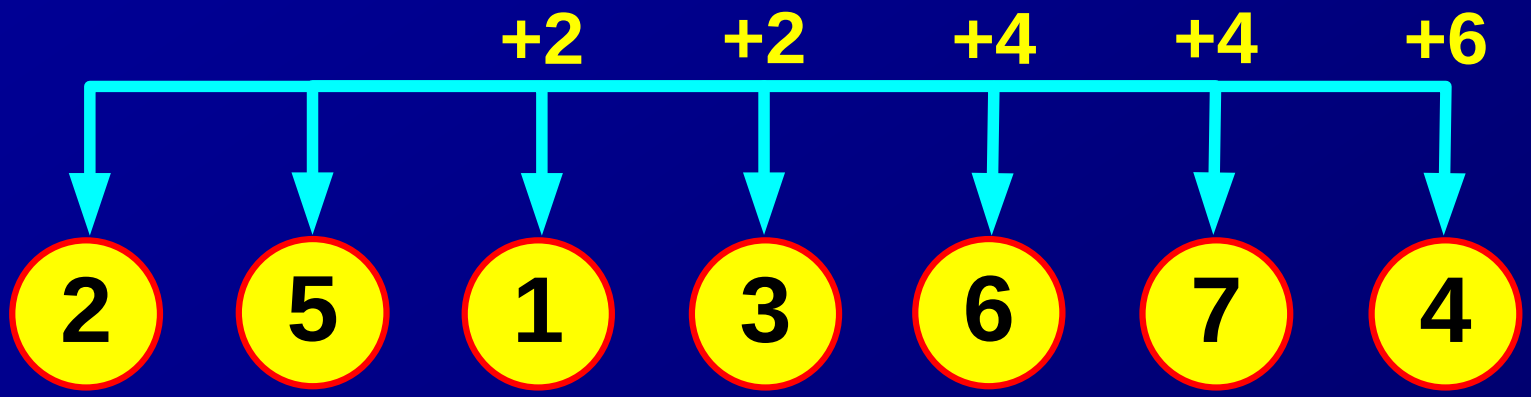
Shell sort

Prvi korak: $h_2=N/2=3$



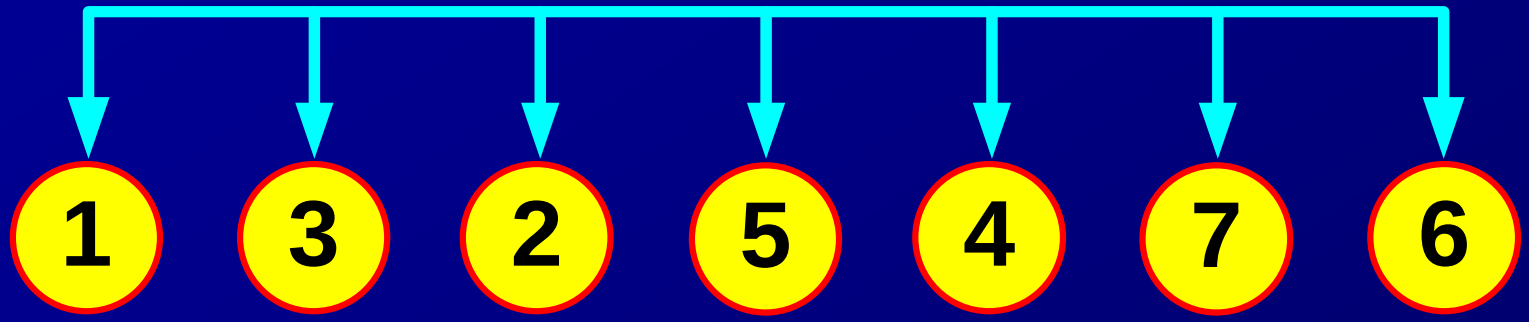
Shell sort

Drugi korak: $h_1=h_2-1=2$



Shell sort

Treći korak: $h_0=h_1-1=1$



Shell sort

- Analiza složenosti Shell sorta:
  - Složenost ovisi o odabiru slijeda h_1, h_2, ... , h_k
  - Najgori slučaj ima složenost $O(n^2)$
  - Prosječno vrijeme izvođenja već je dugo otvoreni (neriješeni) problem.
  - Hibbardov slijed: {1, 3, 7, …, 2^k -1}
    - Za najlošiji slučaj daje $O(n^{3/2})$
    - Prosječno je $O(n^{5/4})$
      - utvrđeno simulacijom, još nije dokazano matematički
  - Sedgewickov slijed: {1, 8, 23, 77, 281, ..., $4^{i+1} + 3 \cdot 2^i + 1$}
    - Za najgori slučaj daje $O(n^{4/3})$
    - Prosječno je $O(n^{7/6})$
  - Ne zna se da li postoji bolji slijed
  - Relativno jednostavan algoritam, a krajnje složena analiza složenosti.
  - Algoritam je dobar za umjerenu količinu ulaznih podataka (desetci tisuća).

Merge sort

- Sortiranje spajanjem (sortirajućim uparivanjem)
  - Na temelju dva sortirana polja (A i B) puni se treće (C).
  - Najučinkovitije za sortirano spajanje dva već sortirana niza.
    - Složenost je O(n).
- Moguća uporaba i za sortiranje jednog (nesortiranog) niza.
  - Koristi se strategija "podijeli pa vladaj" uz rekurziju.
  - Složenost algoritma je $O(n \cdot \log_2 n)$
    - Podjelom problema u dva dijela nastaje $\log_2 n$ razina.
    - Na svakoj razini obavlja se O(n) posla.
- Rijetko se koristi za sortiranje u središnjoj memoriji zbog zahtjeva za dodatnom memorijom i mnogo kopiranja.
  - Ključni algoritam za sortiranje na vanjskoj memoriji.

Merge sort

- Ostvarenje:

```
void mergesort(itemType A[], int N) {
  int sredina;

  if (N>1)
  {
    sredina=N/2;
    mergesort(&A[0],sredina);
    mergesort(&A[sredina],N-sredina);
    spoji(&A[0],sredina,N,pomPolje);
  }
}
```

Sortovi

Merge sort

- Ostvarenje:

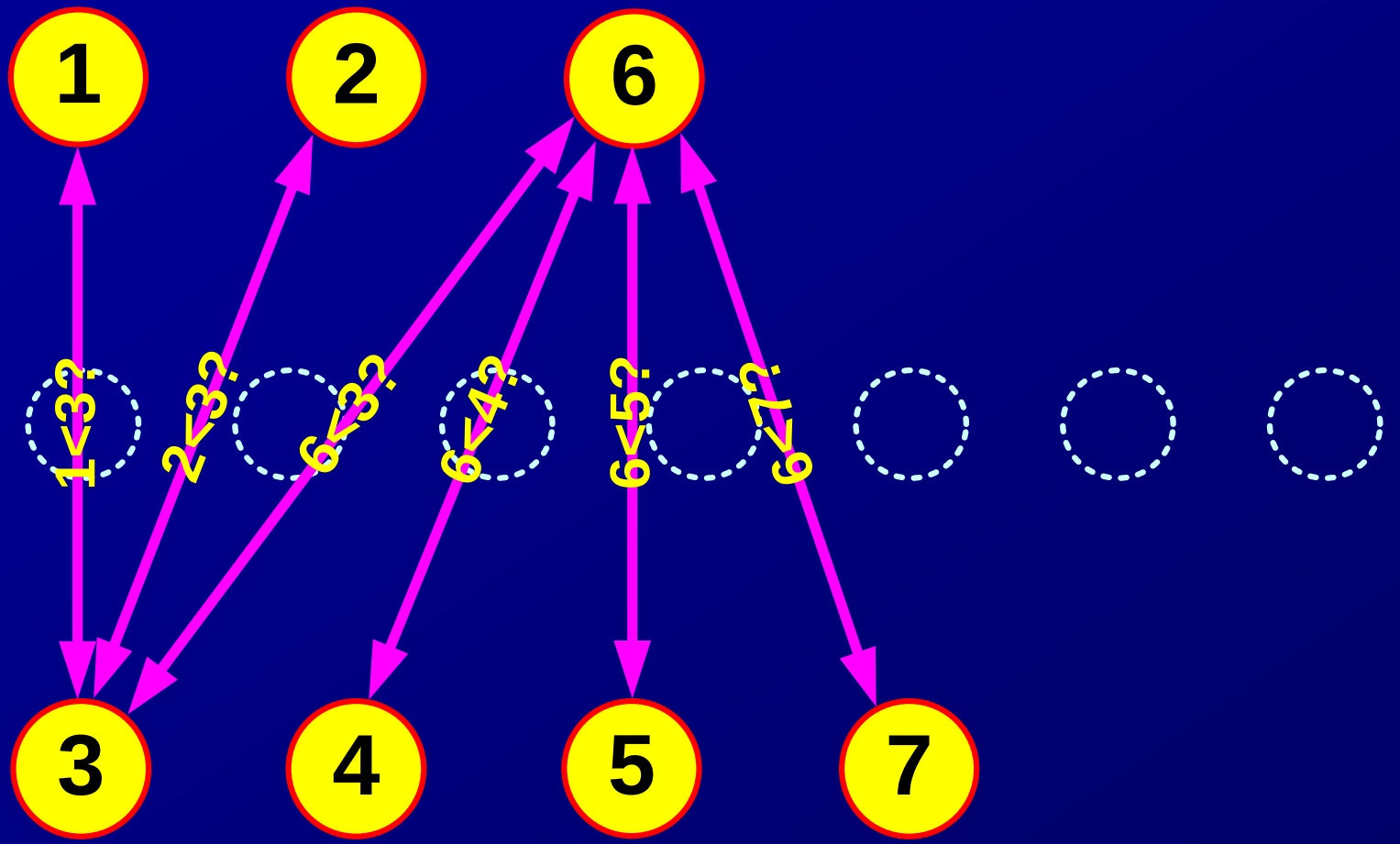
```
void spoji(itemType A[],int N_L,int N,itemType pomPolje[]) {
  int i=0,j=0,k=0;

  i=k=0,j=N_L;
  while ((i<N_L) && (j<N))
    if (A[i]<A[j])
      pomPolje[k++]=A[i++];
    else
      pomPolje[k++]=A[j++];
  while (i<N_L)
    pomPolje[k++]=A[i++];
  while (j<N)
    pomPolje[k++]=A[j++];
  for (k=0;k<N;++k)
    A[k]=pomPolje[k];
}
```

Sortovi

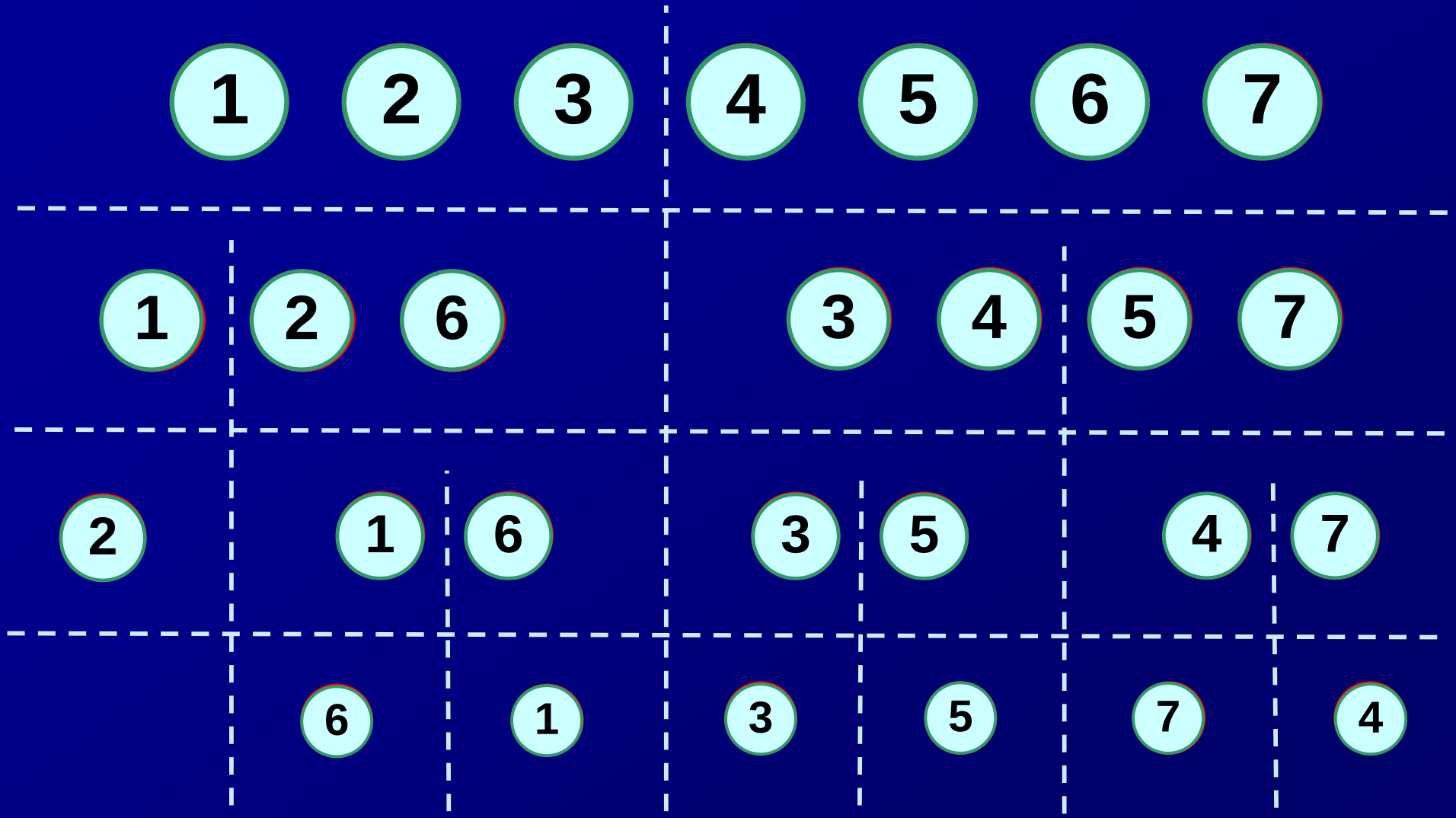
Merge sort

- Spajanje dva sortirana niza



Merge sort

- Rekurzivno sortiranje



Quick sort

- Algoritam je rekurzivan i zasnovan na strategiji "podijeli pa vladaj".
  - Može se lako shvatiti i dokazati da je korektan.
- Najbrži do danas poznati algoritam za sortiranje.
  - Prosječno vrijeme izvođenja je $O(n \cdot \log_2 n)$.
  - Postupak je vrlo brz, uglavnom zbog dobro optimirane unutarnje petlje.
  - Najgori slučaj je $O(n^2)$, ali moguće je postići da vjerojatnost da on nastupi eksponencijalno pada.
- Osnovni algoritam za sortiranje polja `A` sastoji se od četiri koraka:
  1. Ako je broj članova polja `A` manji od 2, vrati se u pozivajući potprogram.
  2. Odaberi bilo koji član `v` u polju `A` kao stožer.
  3. Podijeli preostale članove polja `A` tj. skup `A\{v}` u dva podskupa:
     - $A_1 = \{ x \in A \setminus \{v\} \mid x \leq v \}$ – svi elementi manji od v
     - $A_2 = \{ x \in A \setminus \{v\} \mid x > v \}$ – svi elementi veći od v
  4. Vrati `[quicksort(`A_1`),v,quicksort(`A_2`)]`

Quick sort

- Učinkovitost algoritma ovisi o izboru stožera te o odluci što se radi s članovima niza koji su jednaki stožeru.
- Budući da izbor stožera i postupanje s članovima niza jednakim stožeru nisu jednoznačno određeni u definiciji algoritma, navedeni elementi su pitanje programskog ostvarenja algoritma.
- Učinkovito rješavanje ovih pitanja izravno određuje učinkovitost čitavog algoritma.
- Problem učinkovitosti *quicksort* algoritma detaljno je opisan u Weiss: "Data Structures and Algorithm Analysis in C".

Quick sort

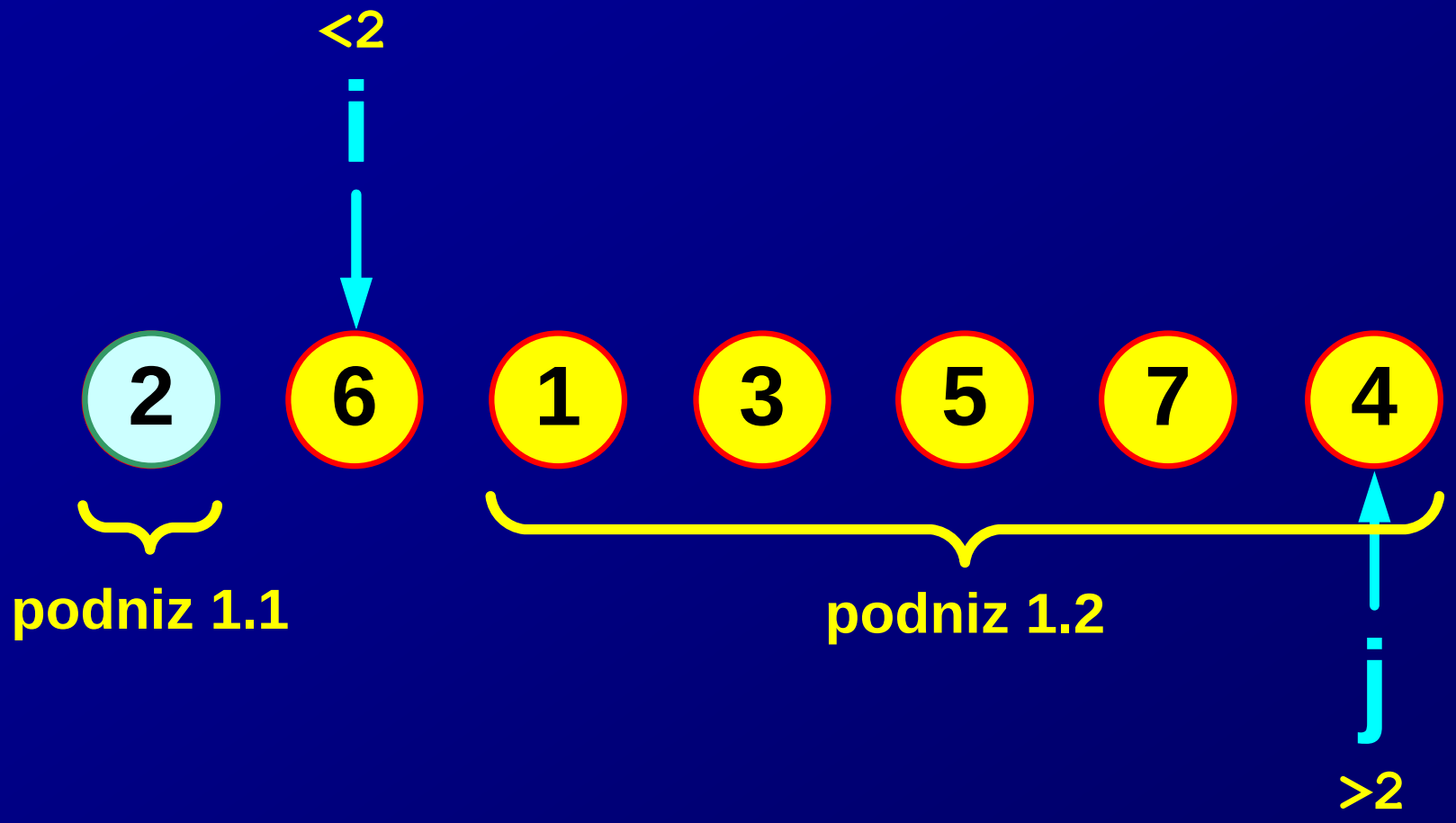
- Ostvarenje:

```
void QSort(tip A[],int l,int d) {
  int i,j; tip temp;
  if (l>=d) return;
  i=l+1; j=d;
  while (i<j) {
    while ((i<j) && (A[i]<A[l])) ++i;
    while ((j>i) && (A[j]>=A[l])) --j;
    if (i<j) { temp=A[i]; A[i]=A[j]; A[j]=temp; }
  }
  if (A[i]>A[l]) --i;
  if (i!=l) { temp=A[i]; A[i]=A[l]; A[l]=temp; }
  QSort(A,l,i-1);
  QSort(A,i+1,d);
}
```

Sortovi

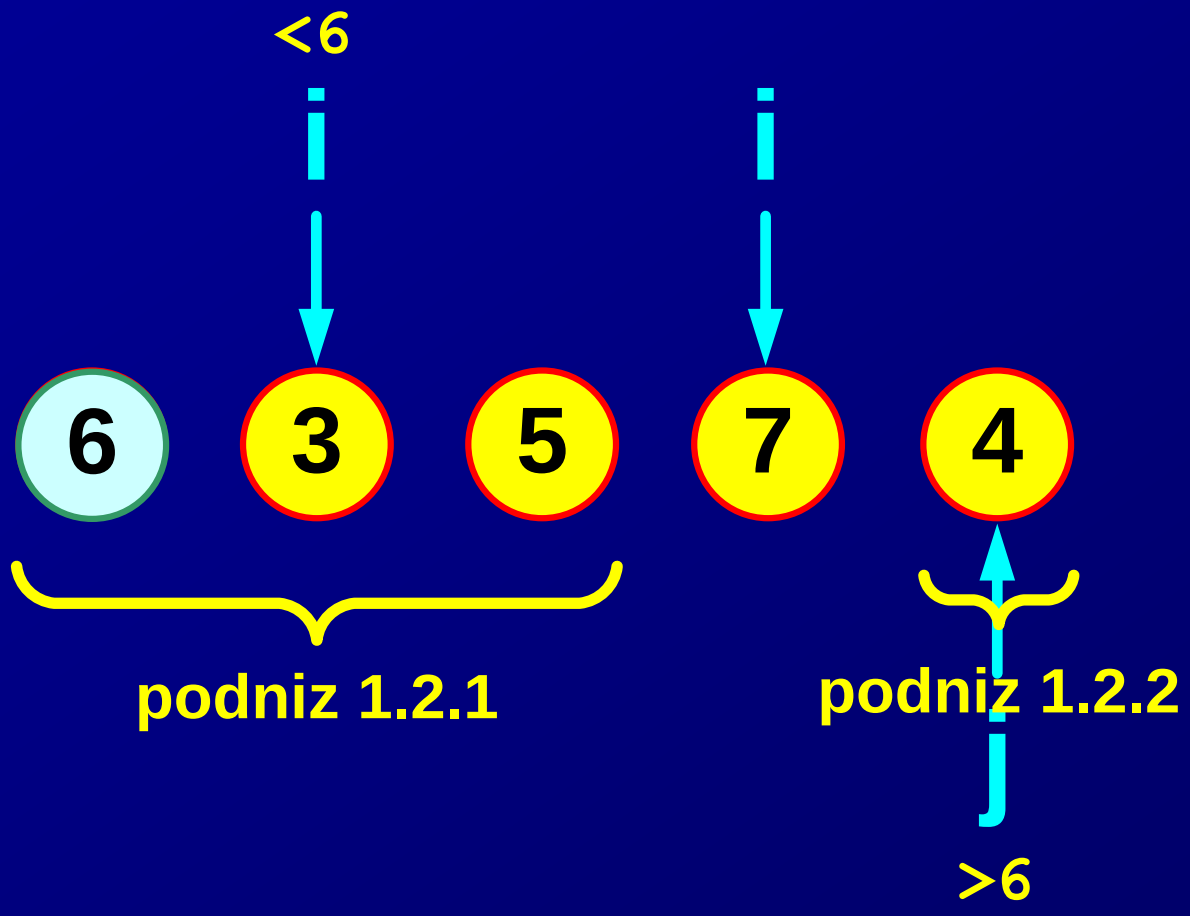
Quick sort

- Stožer je nulti element



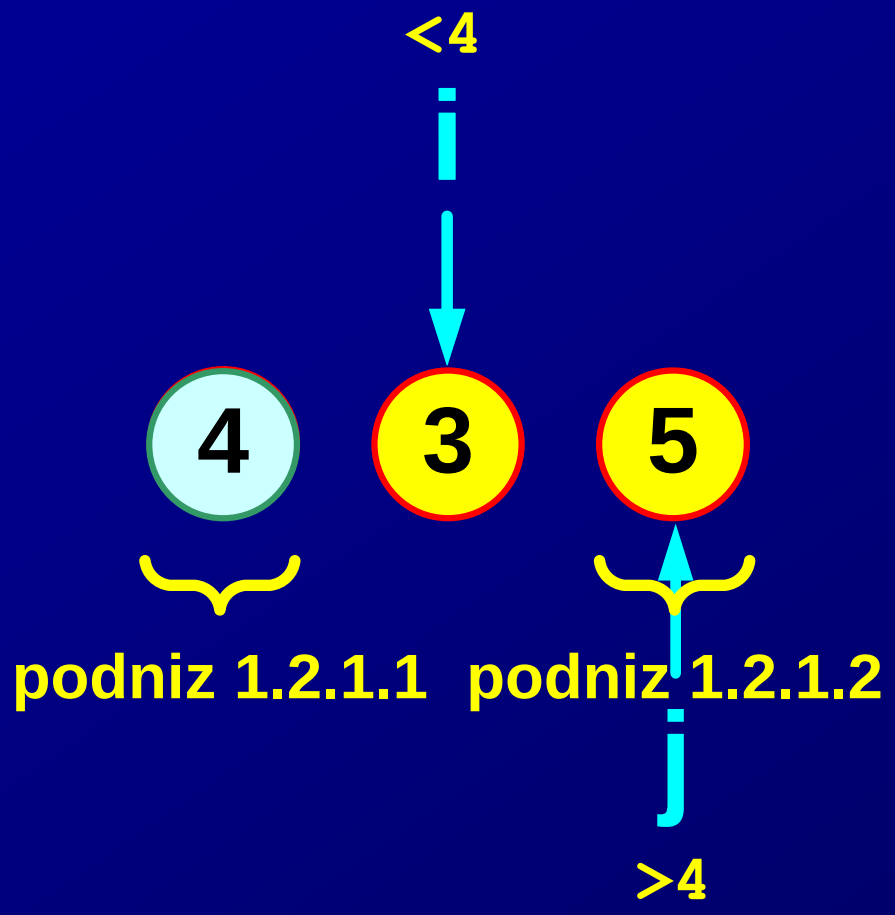
Quick sort

- Sortiranje podniza 1.2



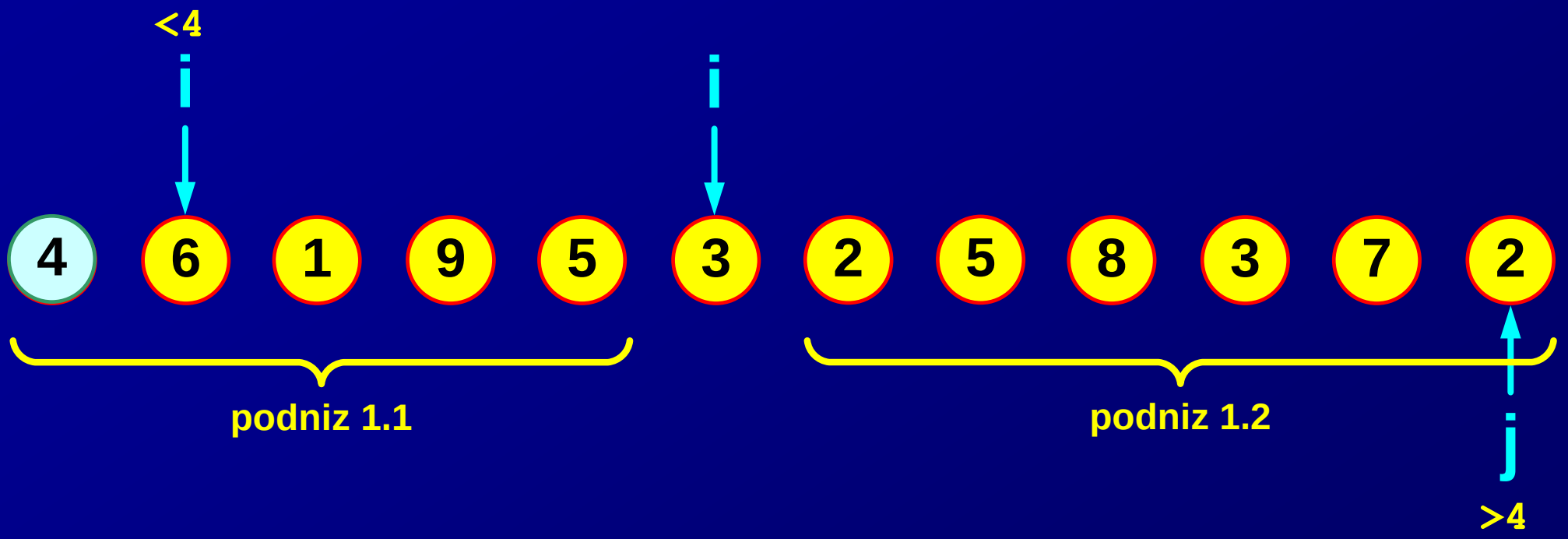
Quick sort

- Sortiranje podniza 1.2.1



Quick sort

- Rad unutarnje petlje



Postupci sortiranja

- Sortiranje velikog broja podataka (milijun i više zapisa) u praksi nije rijetko.
  - Ako bi jedno obavljanje programske petlje trajalo 1 μs
    - trajanje sortiranja odabirom bilo bi reda veličine 10^6 s, odnosno više od 11 dana
    - trajanje *quick sorta* bilo bi reda veličine 20 s
  - Stoga rješenje nije uvijek potrebno tražiti u kupnji bržih i skupljih računala, nego je isplativa i investicija u razvitak i primjenu boljih algoritama.

- Neizravno (indirektno) sortiranje
  - Kada je potrebno sortirati velike strukture, npr. matični broj studenta, prezime, ime, adresa, upisani predmeti i ocjene itd.
  - Nema smisla obavljati mnogo zamjena velikog broja podataka.
  - Ako se podaci sortiraju npr. po matičnom broju (ili po nekom drugom ključu), onda se u posebno polje izdvoje matični brojevi s pripadnim pokazivačima na ostale podatke.
  - Sortira se (bilo kojim od postupaka) samo takvo izdvojeno polje, a pokazivači održavaju konzistentnost podataka.

Zadaci za vježbu (1/2)

- Napisati program koji će u cjelobrojnom polju od `n` članova pronaći `k`-ti najveći član polja.
  a) Učitano polje sortirati po padajućim vrijednostima i ispisati član s indeksom `k-1`.
  b) Učitati `k` članova polja, sortirati ih po padajućim vrijednostima. Učitavati preostale članove polja. Ako je novi član manji od onoga s indeksom `k-1`, onda se ignorira. Ako je novi član veći, onda se umetne na pravo mjesto, a izbaci se posljednji član polja koji bi nakon umetanja imao indeks `k`.
- Za oba navedena postupka odrediti apriorna vremena trajanja i izmjeriti aposteriorna vremena.
- Ponoviti sve za različite postupke sortiranja.

Zadaci za vježbu (2/2)

- Napisati program koji od dvije formatirane sortirane slijedne datoteke napravi novu slijednu datoteku u kojoj su pravilno sortirani podaci iz obje datoteke.

UpariDatoteke