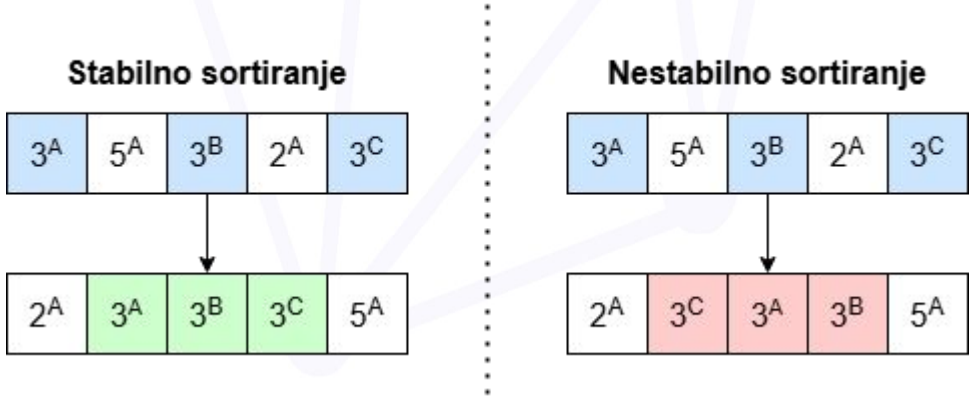
Teorija Algoritama

Algoritmi za sortiranje

Algoritmi za sortiranje

- **Problem:** Sortirati (urediti) niz u nekom jasno definisanom redosledu (najčešće rastući ili opadajući)
- Mogući pristupi:
  - Sortiranje zasnovano na poređenju elemenata;
  - Sortiranje zasnovano na pretpostavci o elementima.
- **Stabilno sortiranje:**
  - Stabilnost prilikom sortiranja garantuje da elementima sa istom vrednošću za sortiranje neće biti promenjen redosled tokom sortiranja.



Algoritmi za sortiranje zasnovani na poređenju

- Selection sort
- Bubble sort
- Insertion sort
- Shell sort
- Quicksort
- Merge sort

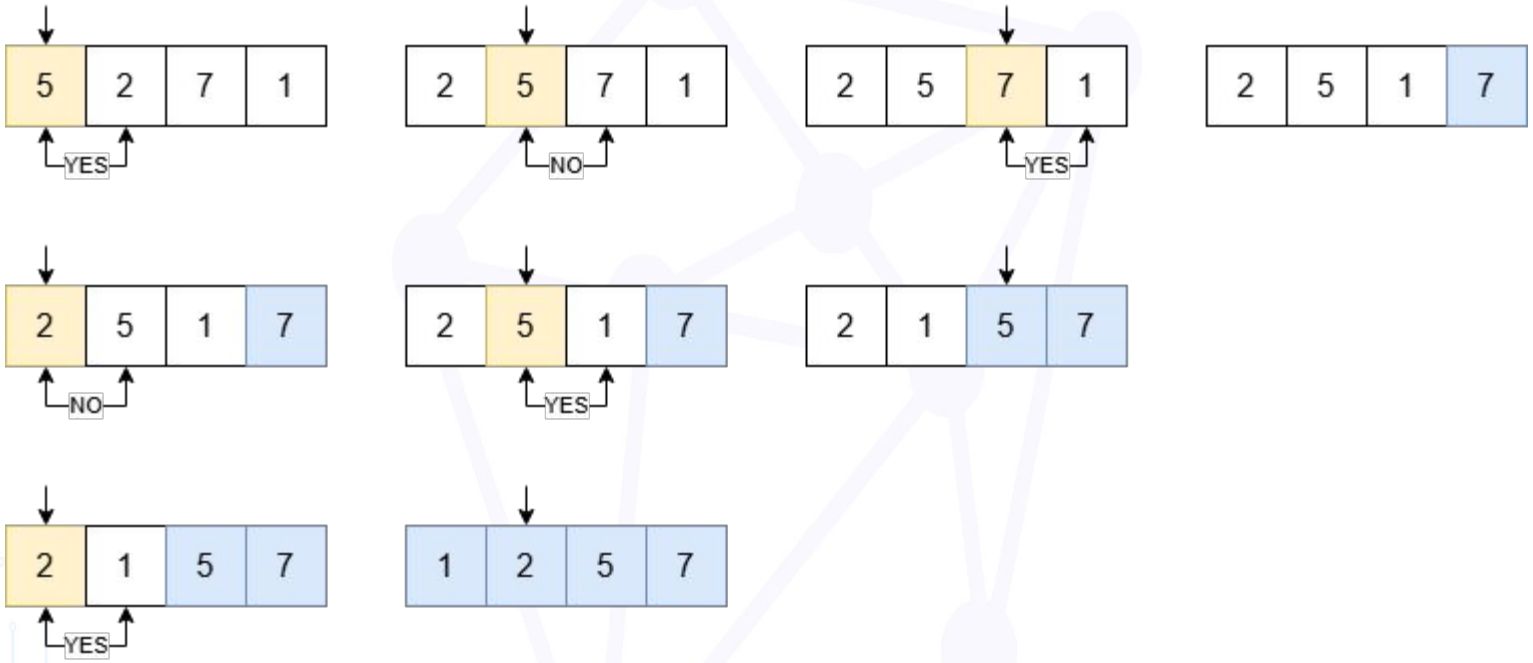
Selection sort

- Jedan od najintuitivnijih algoritama za sortiranje.
- Koraci:
  - Pronađi najmanji/najveći element u nizu;
  - Postavi ga na početak;
  - Ponavljaj prva dva koraka nad podnizom `[k-1 … n-1]`, gde je **k** trenutna iteracija.
- Vremenska složenost: $O\left(n^2\right), \Theta\left(n^2\right), \Omega\left(n^2\right)$
- Prostorna složenost: $O(1)$ (u iterativnoj implementaciji)
- Rekurzivna formula: $T(n) = T(n-1) + O(n)$
- Stabilnost: *Nije stabilan*
- Primer: *`selection.c`*

Bubble sort

- Još jedan od najintuitivnijih algoritama za sortiranje.
- Radi po principu da u svakoj iteraciji *izgurava* najveći/najmanji element na kraj.
- Koraci:
  - Počni od prvog elementa u nizu i ako je veći/manji od narednog elementa, zameni ih;
  - Ukoliko se desila zamena, nastavi sa elementom koji si zamenio, u suprotnom uzmi naredni;
  - Kada najveći/najmanji element u nizu ispliva na kraj niza, ponovi prethodne korake za podniz `[0 … n-k-1]`, gde je **k** trenutna iteracija.
- Vremenska složenost: $O\left(n^2\right), \Theta\left(n^2\right), \Omega\left(n^2\right)$ ($\Omega(n)$ u optimizovanoj verziji)
- Prostorna složenost: $O(1)$ (u iterativnoj implementaciji)
- Rekurzivna formula: $T(n) = T(n-1) + O(n)$
- Stabilnost: *Stabilan* (obratiti pažnju na uslov poređenja)
- Primer: *`bubble.c`*

Bubble sort - primer

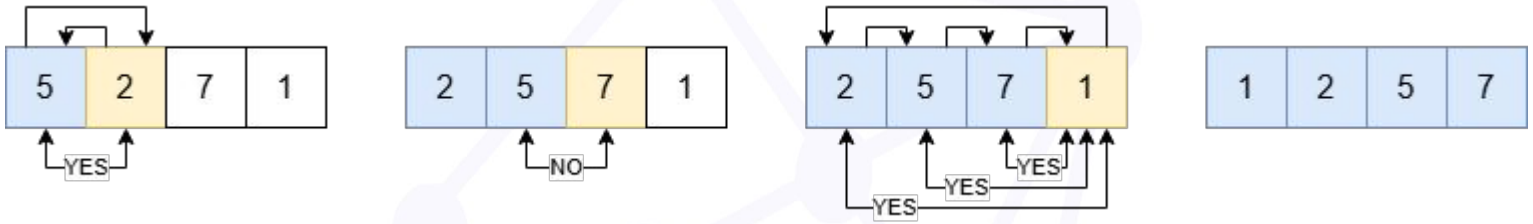


Insertion sort

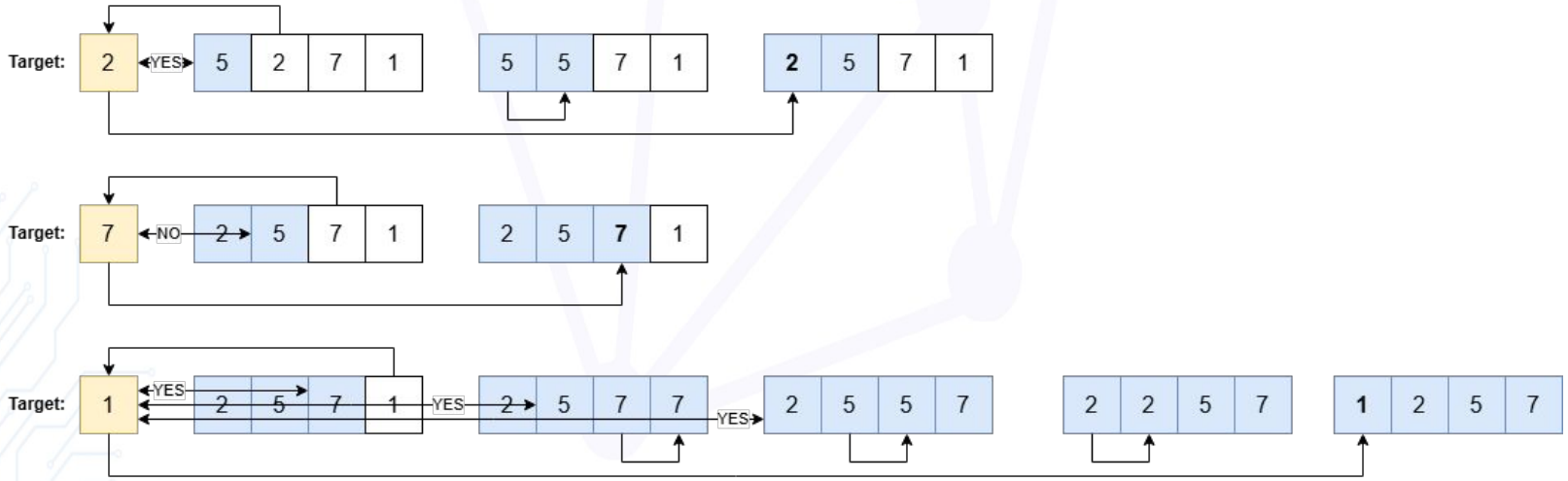
- Insertion sort, takođe jedan od intuitivnijih algoritama za sortiranje, podseća na način kako ljudi sortiraju karte za igranje u ruci.
- Ideja je da se na početku niza formira sortirani podniz, a da se u svakoj iteraciji prvi element u nesortiranom delu niza postavlja na svoju pravu poziciju u sortiranom delu niza.
- Ovaj algoritam je veoma efikasan za male nizove koji su inicijalno već delimično sortirani.
- Vremenska složenost: $O\left(n^2\right), \Theta\left(n^2\right), \Omega(n)$
- Prostorna složenost: $O(1)$(u iterativnoj implementaciji)
- Rekurzivna formula: $T(n) = T(n-1) + O(n)$
- Stabilnost: *Stabilan*
- Primer: *insertion.c*

Insertion sort - primer

- Teorijski:



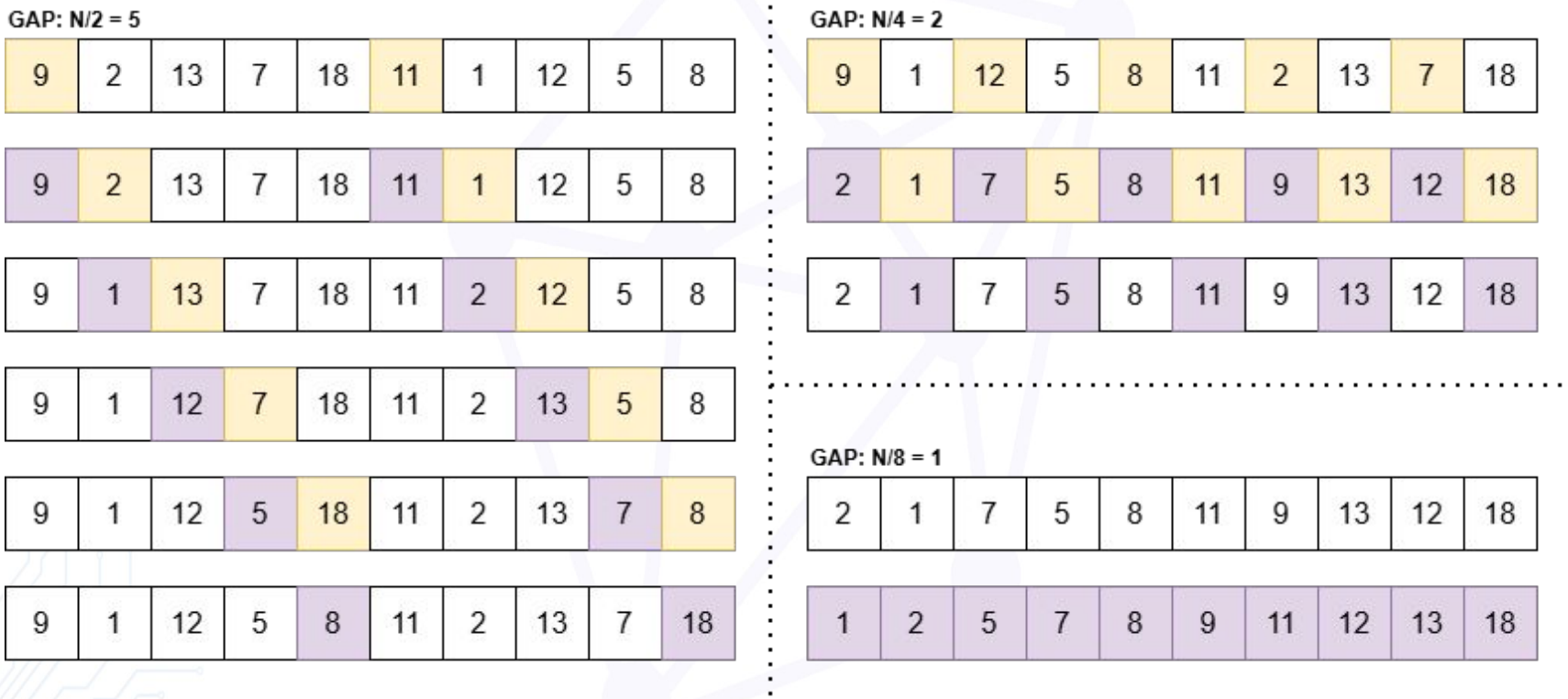
- U implementaciji:



Shell sort

- Algoritam je osmislio Donald Shell 1959. Godine.
- Zasniva se na formiranju niza raskoraka po kojima se vrši sortiranje.
- Originalni niz raskoraka formiran je formulom: $g = \frac{N}{2^k}, k \in \{1, 2, 3, \dots\}$
- Ideja je da sortiranje vrši posmatrajući niz u raskoracima:
  - Počinje se od najvećeg raskoraka;
  - Na primer, ako imamo niz od 100 elemenata, prvi raskorak će biti 50:
    - Sortiramo podnizove sačinjene od sledećih elemenata: `[a[0],a[50]]`, `[a[1],a[51]]`, `[a[2],a[52]]` … `[a[49],a[99]]`;
    - Zatim uzimamo naredni raskorak (25) i sortiramo sledeće podnizove: `[a[0],a[25],a[50],a[75]]`, … `[a[24],a[49],a[74],a[99]]`;
    - I tako za sve raskorake.
    - Za sortiranje se koristi *Insertion sort*.
- Stabilnost: *Nije stabilan*
- Primer: *`shell.c`*

Shell sort - primer



Shell sort

- Vremenska složenost Shell sort algoritma zavisi od niza raskoraka:
  - *Shell*-ov originalan niz:
    - Formula: $g = \frac{N}{2^k}, k \in \{1, 2, 3, \dots\}$ Niz: $\frac{N}{2}, \frac{N}{4}, \frac{N}{8}, \dots 1$
    - Vremenska složenost *(worst-case)*: $O\left(n^2\right)$
  - *Hibbard*-ov niz:
    - Formula: $g = 2^k - 1$ Niz: $1, 3, 7, 15, 31, \dots$
    - Vremenska složenost (*worst-case*): $O\left(n^{1.5}\right)$
  - *Knuth*-ov niz:
    - Formula: $g = \frac{3^k - 1}{2}$ Niz: $1, 4, 13, 40, 121, \dots$
    - Vremenska složenost (*worst-case*): $O\left(n^{1.5}\right)$
  - *Sedgewick*-ov niz (1986):
    - Koristi dve formule istovremeno: $g = 9 \cdot \left(2^k - 2^{\frac{k}{2}}\right) + 1, k \in \{0, 2, 4, \dots\}$
      $g = 8 \cdot 2^k - 6 \cdot 2^{\frac{k+1}{2}} + 1, k \in \{1, 3, 5, \dots\}$
    - Niz: $1, 5, 17, 41, 109, 209, \dots$
    - Vremenska složenost (*worst-case*): $O(n^{1.\overline{33}})$
  - *Ciura*-ov niz (empirijski dobijen):
    - Niz: $1, 4, 10, 23, 57, 132, 301, 701, \dots$
    - Nema poznatu vremensku složenost, ali se u praksi pokazao kao najbolji.
- Prostorna složenost je konstantna, a rekurzivna formula ne postoji.

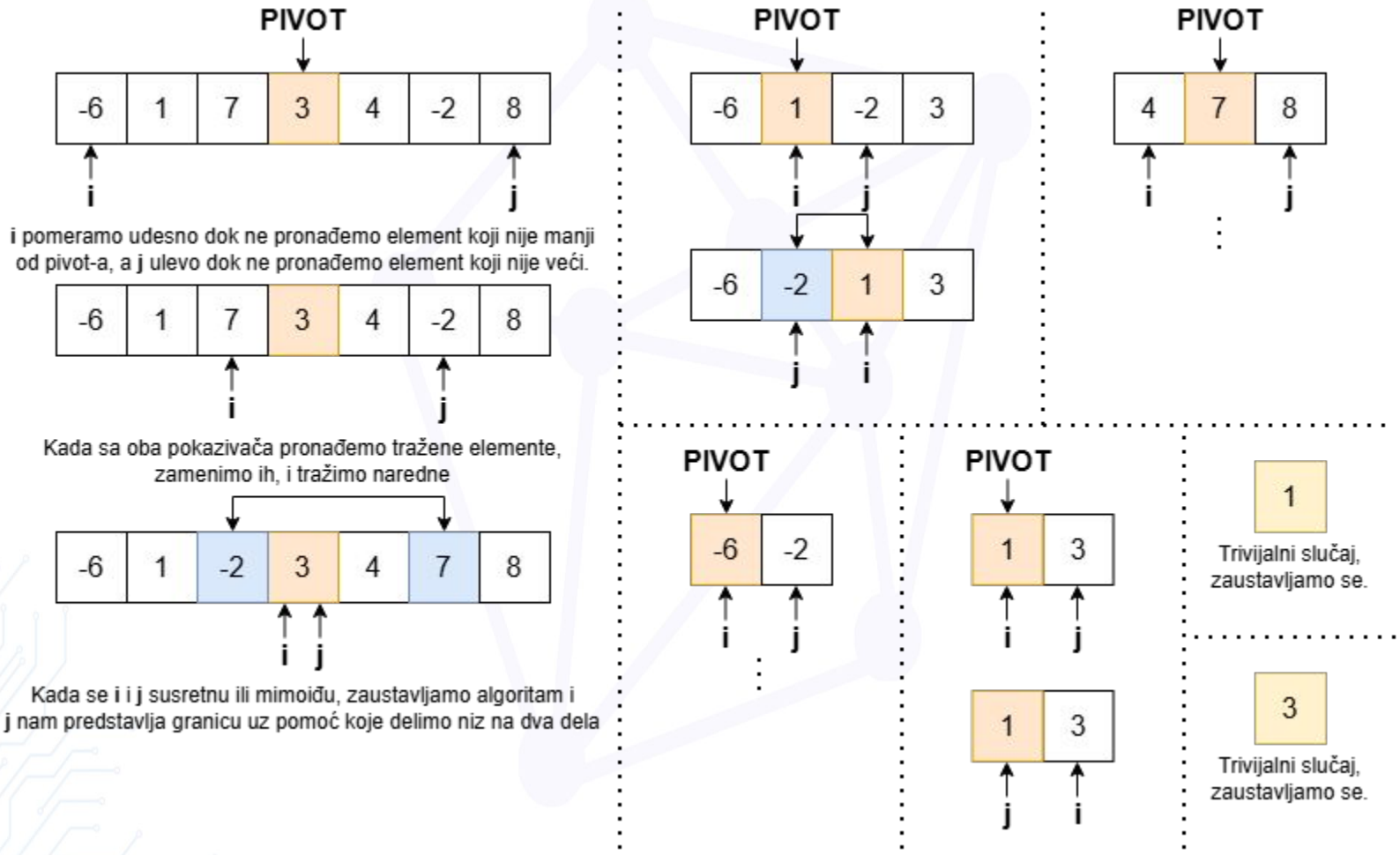
Merge sort

- Algoritam poznat sa prethodnih vežbi.
- Dosta vremenski efikasan i konstantan, ali ima nešto veću prostornu složenost.
- Vremenska složenost: $\Theta(n \cdot \log n)$
- Prostorna složenost: $O(n)$ - pomoćni prostor
  $O(\log n)$ - rekurzivni stek
- Rekurzivna formula: $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n)$
- Stabilnost: *Stabilan*
- Primer: *merge.c*
- Najčešće se koristi kada se zna da na raspolaganju imamo dovoljno memorije.

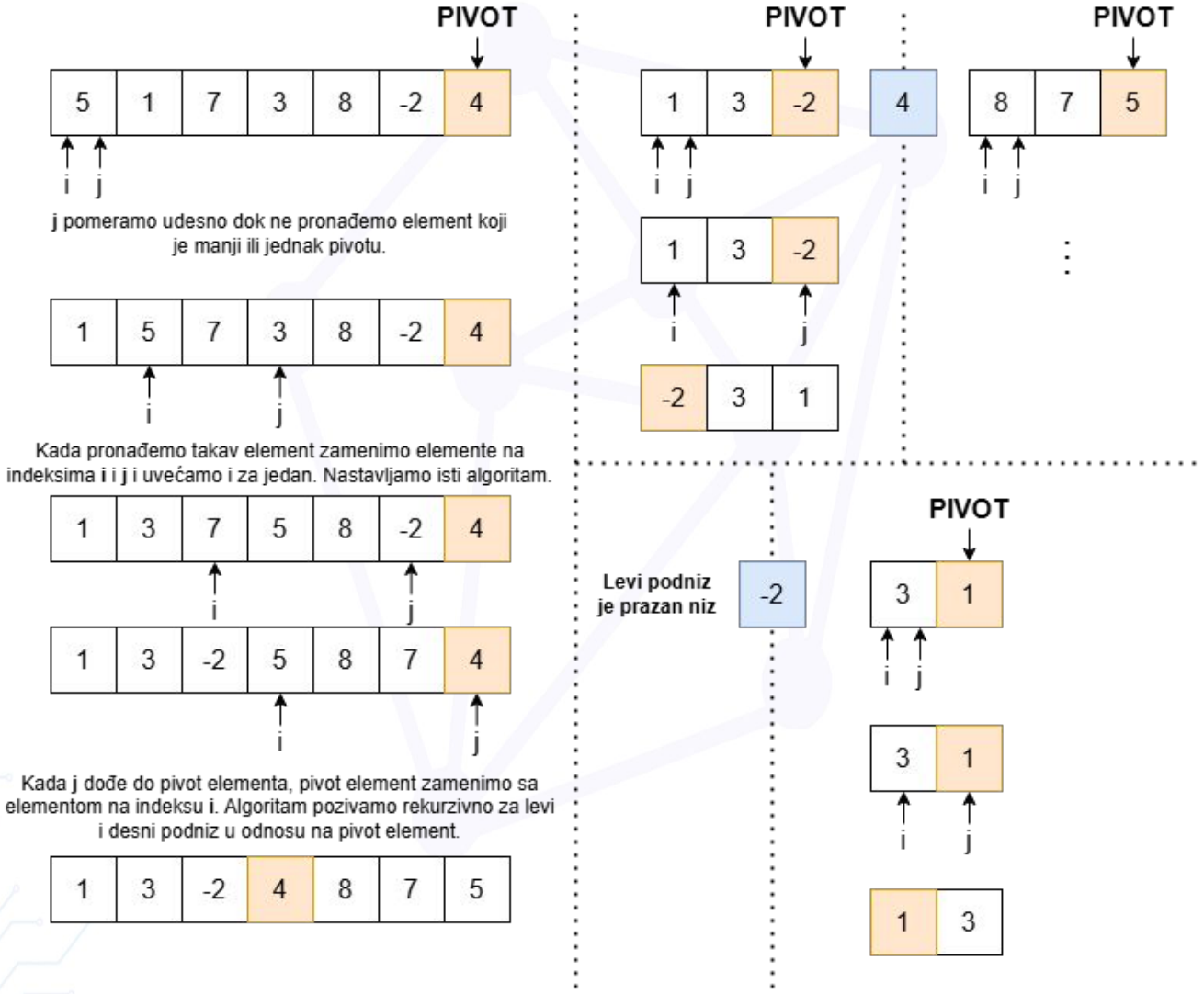
Quicksort

- Algoritam koji se takođe zasniva na strategiji *Zavadi pa vladaj*.
- U prosečnom i najboljem slučaju, vremenski jednako efikasan kao i *Merge sort*.
- Koristi znatno manje memorije od *Merge Sort*-a
- Zasniva se na odabiru *pivot* elementa u odnosu na koji delimo niz na dva dela, tako da su u jednom delu svi elementi manji od pivota, a u drugom svi elementi veći od pivota.
- Algoritam se sastoji iz dva dela:
  - Particionisanje - odabir pivot elementa i podešavanje niza da ispunjava prethodno naveden kriterijum;
  - Rekurzivni poziv - poziva se isti algoritam nad dva podniza.
- Šeme particionisanja:
  - *Hoare*-ova (primer: `quickHoare.c`)
  - *Lomuto*-ova (primer: `quickLomuto.c`)

Quicksort - Hoare primer



Quicksort - Lomuto primer



Quicksort

- Performanse Quicksort-a zavise od odabira pivot elementa.
- Idealan pivot je *median*.
- Da li možemo da dovoljno efikasno pronađemo median?
- **Best case:**
  - Rekurzivna formula: $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n)$
  - Vremenska složenost: $O(n \cdot \log n)$
  - Prostorna složenost: $O(1)$ - pomoćni prostor;
    $O(\log n)$ - rekurzivni stek.
- **Worst case:**
  - Rekurzivna formula: $T(n) = T(n-1) + \Theta(n)$
  - Vremenska složenost: $O\left(n^2\right)$
  - Prostorna složenost: $O(1)$ - pomoćni prostor;
    $O(n)$- rekurzivni stek.
- Nizom dokaza, može se dokazati da je prosečan slučaj približno jednak najboljem slučaju.
- Stabilnost: *Nije stabilan*

Drugi algoritmi za sortiranje

- Counting sort;
- Radix sort;
- Bucket sort;
- I mnogi drugi koje nećemo obuhvatiti gradivom na ovom predmetu.

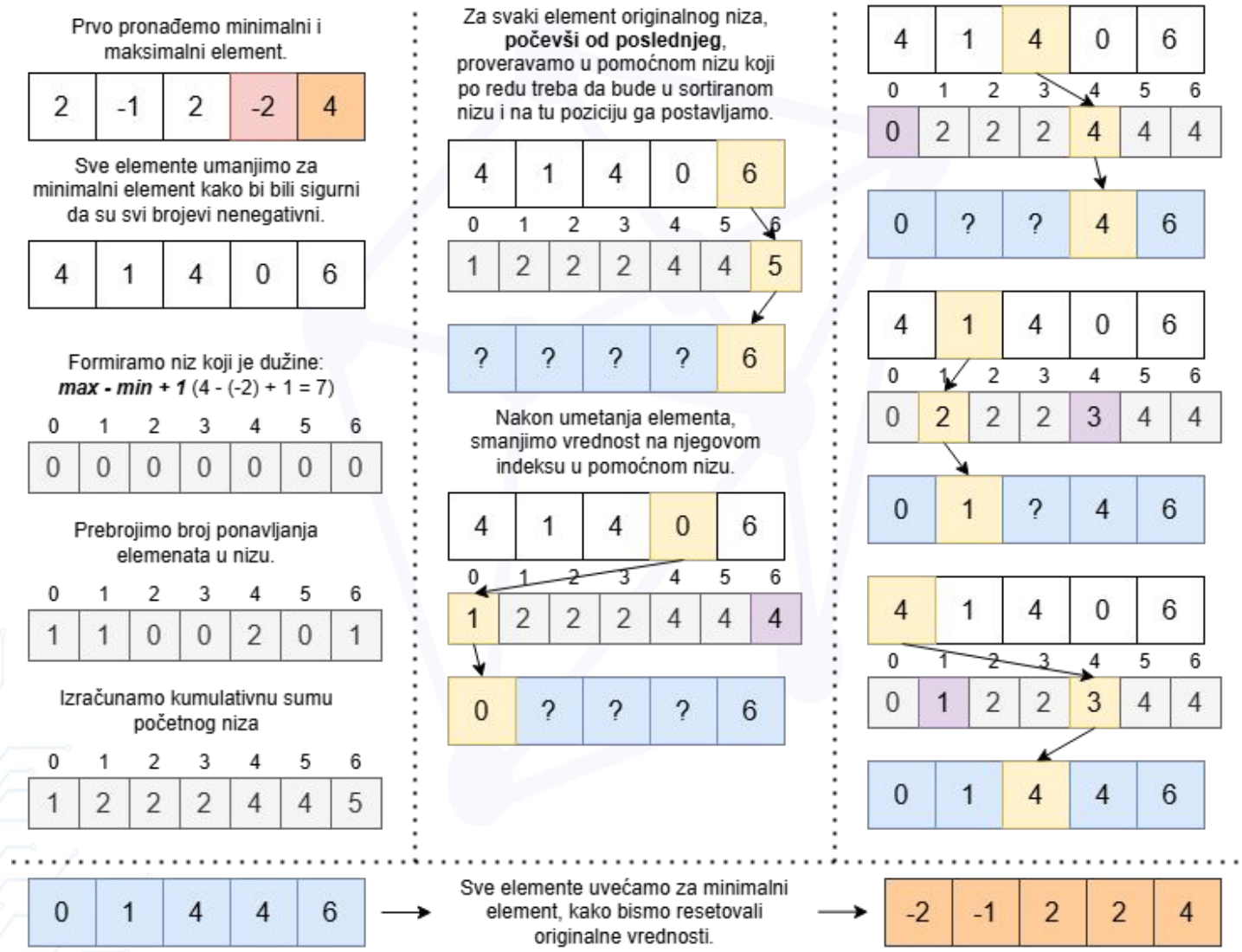
Counting sort

- Algoritam zasnovan na brojanju ponavljanja elemenata u nizu.
- Veoma efikasan za nizove čiji su elementi sličnih vrednosti.
- Ideja je da se broji ponavljanje elemenata u nizu, zatim se kumulativnom sumom može odrediti na kom indeksu ti elementi u sortiranom nizu trebaju da se nalaze.
- Nema rekurzivnu formulu jer nije rekurzivan algoritam.
- Vremenska složenost: $\Theta(n + k)$

  n - broj elemenata;

  k - opseg elemenata.
- Prostorna složenost: $O(n + k)$
- Stabilnost: *Stabilan, ali obratiti pažnju na implementaciju*
- Ograničenja:
  - Indeksi su celi brojevi, pa Counting sort radi samo sa celim brojevima;
  - Indeksi su nenegativni brojevi, pa Counting sort radi samo sa pozitivnim brojevima, ali uz malu adaptaciju može raditi i sa negativnim.
- Primer: *`counting.c`*

Counting sort - primer



Radix sort

- Koncipiran na tome da sortira brojeve uzimajući u obzir samo jednu cifru broja (npr. deseticu ili stotinu), počevši od najmanje značajne.
- Oslanja se na neki **stabilni sort** algoritam koji radi u linearnom vremenu (npr. Counting sort)
- Radix sort u kombinaciji sa Counting sort-om predstavlja modifikovanu verziju istog, tako da ima lošiju vremensku složenost, ali bolju prostornu.
- Radix sort ima ista ograničenja kao i Counting sort.
- Nema rekurzivnu formulu jer nije rekurzivan algoritam.
- Vremenska složenost: $\Theta(d \cdot (n + k))$
  d - broj cifara u najvećem broju, ***n*** - broj elemenata, ***k*** - opseg (uglavnom 10)
- Prostorna složenost: $O(n + k)$
- Stabilnost: ***Stabilan!!!***
- Primer: *radix.c*

Bucket sort

- Koncipiran na pretpostavci da su elementi uniformno raspoređeni.
- Elementi sličnih vrednosti se raspoređuju u manje skupove.
- Sortiranjem manjih skupova i povezivanjem istih, sortira se i početni skup.
- Bucket sort je takođe iterativan, pa nema korisnu rekurzivnu funkciju.
- Vremenska složenost zavisi od algoritma koji se koristi za sortiranje manjih skupova, kao i od uniformnosti elemenata, ali u prosečnom slučaju: $O(n + k)$, gde je ***n*** broj elemenata, a ***k*** broj manjih skupova.
- Prostorna složenost: $O(n + k)$
- Stabilnost: *zavisi od algoritma koji koristi za sortiranje manjih nizova*
- Primer: `bucket.c`

Bucket sort - primer