Dr Dinu Dragan

- INFORMACIONI INŽENJERING –

TEORIJA ALGORITAMA

ALTERNATIVNI PRISTUPI U DIZAJNU ALGORITAMA

TEHNIKE DIZAJNIRANJA INTERFEJSA

- Postoji širok dijapazon tehnika za dizajn algoritama
  - inkrementalni pristup
  - rekurzivni metod
  - pohlepni metod
  - linearno programiranje
  - randomizacija
  - dinamičko programiranje
- Koji pristup je korišćen u INSERTION sortiranju?
- **Inkrementalni pristup!**
- Kako? Objasnite …
- Išli smo element po element, i u sortirani podniz ***A[1 .. j – 1]*** dodavali smo element ***A[j]*** na odgovarajuće mesto kako bi održali invarijantost petlje i sortiranost niza

ALTERNATIVNI PRISTUP – ZAVADI-PA-VLADAJ

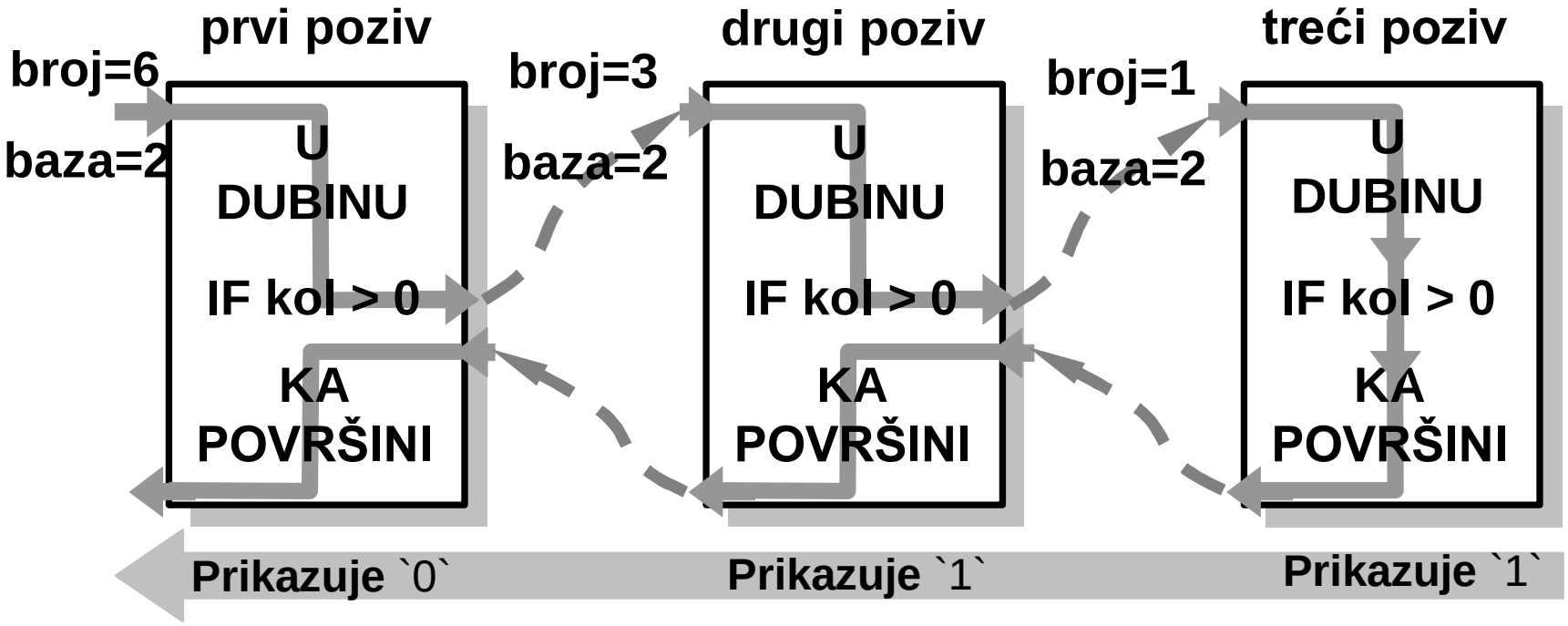
- Zavadi-pa-vladaj (divide-and-conquer) pristup znači da pojednostavimo problem, pa da ga rešavamo parče po parče
- Ovoj tehnici za dizajniranje algoritama ćemo posvetiti čitavo jedno predavanje
- Bazira se na rekurziji i primenjuje se na probleme koje imaju rekurzivnu strukturu ili se mogu svesti/modifikovati tako da dobiju rekurzivnu strukturu
- Problem se razbija na potprobleme koji su slični originalnom, ali jednostavniji po svojoj prirodi
  - npr. da li je lakše sortirati 8 brojeva ili 4 ili 2 broja …
- Algoritam se rekurzivno poziva sve dok se ne dođe do najjednostavnijeg koraka, pa se posle rekurzivno vraća ka prvom pozivu

ZAVADI-PA-VLADAJ – UVOD …

- Zavadi-pa-vladaj pristup ima tri osnovna koraka:
  - Zavadi/podeli (divide) problem na jednostavnije potprobleme istog problema
  - Vladaj/reši (conquer) potproblem rekurzivnim pozivom (razbijanje) na još manje potprobleme, ali ako je potproblem dovoljno jednostavan, reši ga na direktan način
  - Kombinuj (combine) potprobleme u rešenje za originalni problem
- Kada se to rešava potprogramima, onda postoje 3 faze u izvršavanju potprograma:
  - Poziv u dubinu
  - Uslovno izvršavanje
  - Poziv ka površini

… ZAVADI-PA-VLADAJ – UVOD

- Potprogram je istovremeno sebi superordinaran i subordinaran
- Primer konverzije decimalnog broja u binarni

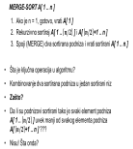


MERGE SORTIRANJE

MERGE SORTIRANJE …

- Algoritam za merge sortiranje radi po principu zavadi-pa-vladaj
- Algoritam radi intuitivno u sledećim fazama
  - **Podeli:** Niz od ***n*** brojeva deli se na dva podniza sa ***n / 2*** elemenata svaki
  - **Reši**: Sortiraj svaki podniz rekurzivno korišćenjem merge sortiranja
  - **Kombinuj**: Kombinuj dva sortirana podniza u sortirani niz (rešenje početnog problema)
- Uslov za završetak rekurzije, kretanje ka vrhu, nastaje onog trenutka kada se u **Reši** koraku kao rezultat deljenja dobija podniz sa samo jednim elementom

… MERGE SORTIRANJE …



MERGE-SORT A[1 .. n]

1. Ako je n = 1, gotovo, vrati ***A[1]***
2. Rekurzivno sortiraj ***A[1 ..*** $\lceil n/2 \rceil$ ***]*** i ***A[*** $\lceil n/2 \rceil$***+1 .. n]***
3. Spoji (MERGE) dva sortirana podniza i vrati sortirani ***A[1 .. n]***

- Šta je ključna operacija u algoritmu?
- Kombinovanje dva sortirana podniza u jedan sortirani niz
- **Zašto?**
- Da li su podnizovi sortirani tako je svaki element podniza ***A[1 ..*** $\lceil n/2 \rceil$ ***]*** uvek manji od svakog elementa podniza ***A[*** $\lceil n/2 \rceil$***+1 .. n]*** ???
- Nisu! Šta onda?

… MERGE SORTIRANJE …

- MERGE operacija onda mora da prilikom spajanja dva niza brine o tome da spajanje nizova rezultuje u sortirani niz
- Kako?
- Kako su oba niza sortirana, onda je to vrlo jednostavno
- Kao kada postoje dve hrpe sortiranih karata:
  1. Sa svake gomile se uzme po jedan karta
  2. Uporede se i najmanja se stavlja u ruku
  3. Zatim se uzme nova karta sa gomile na kojoj se nalazila odabrana karta
  4. Ponavljaju se koraci 2 i 3 sve dok se jedna od gomila ne isprazni
  5. (nema potrebe nastavljati sa preostalom gomilom jer je ona svakako već sortirana)

… MERGE SORTIRANJE …

20 12

13 11

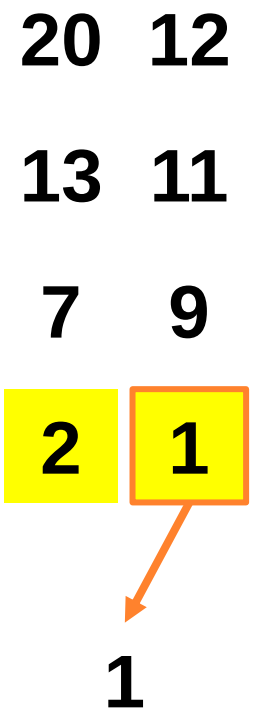
7 9

2 1

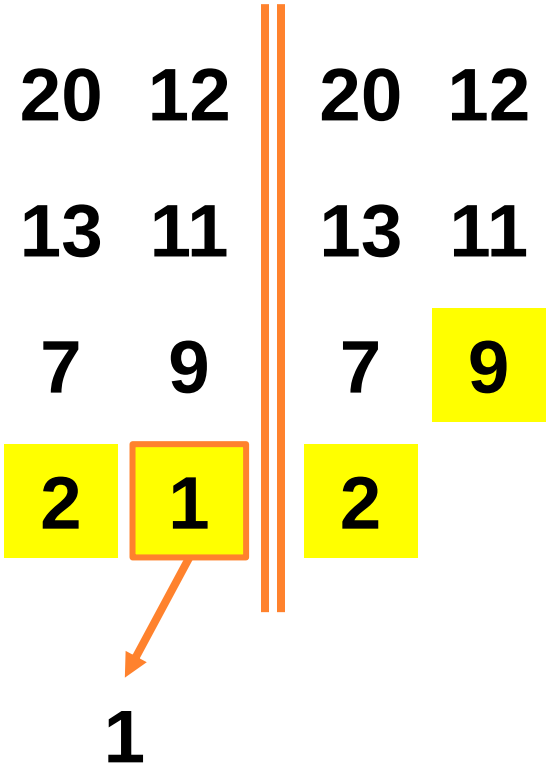
… MERGE SORTIRANJE …

20	12
13	11
7	9
2	1

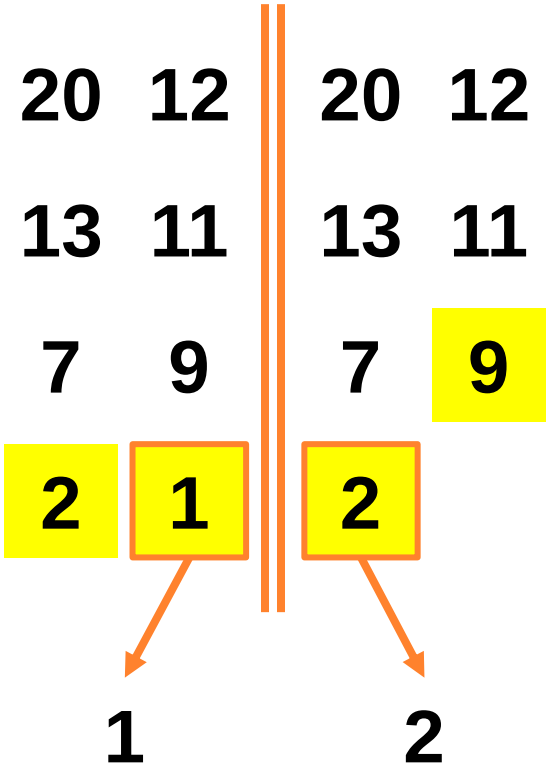
… MERGE SORTIRANJE …



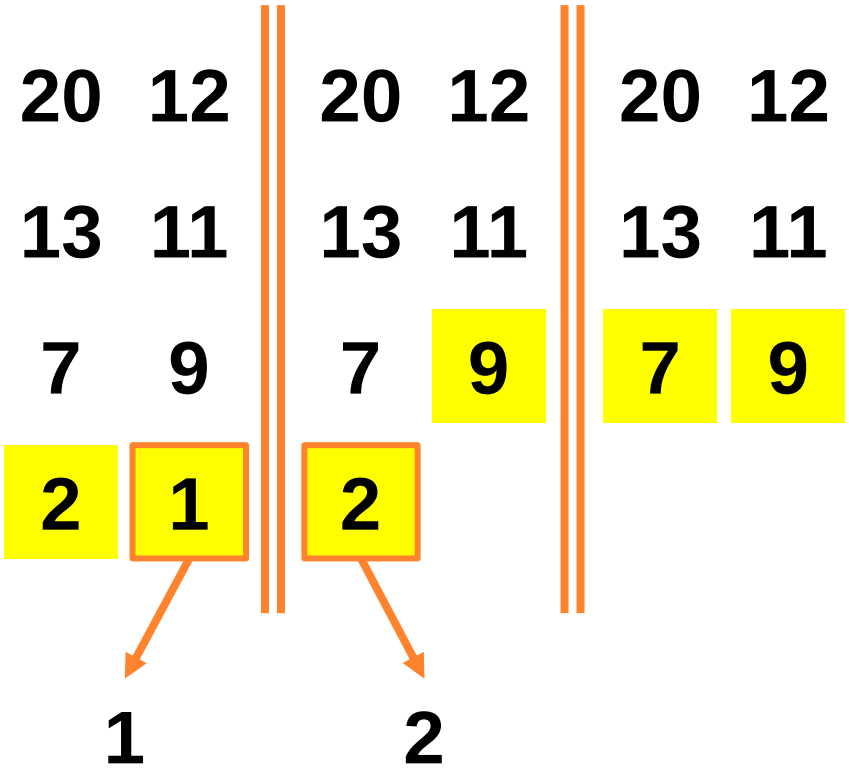
… MERGE SORTIRANJE …



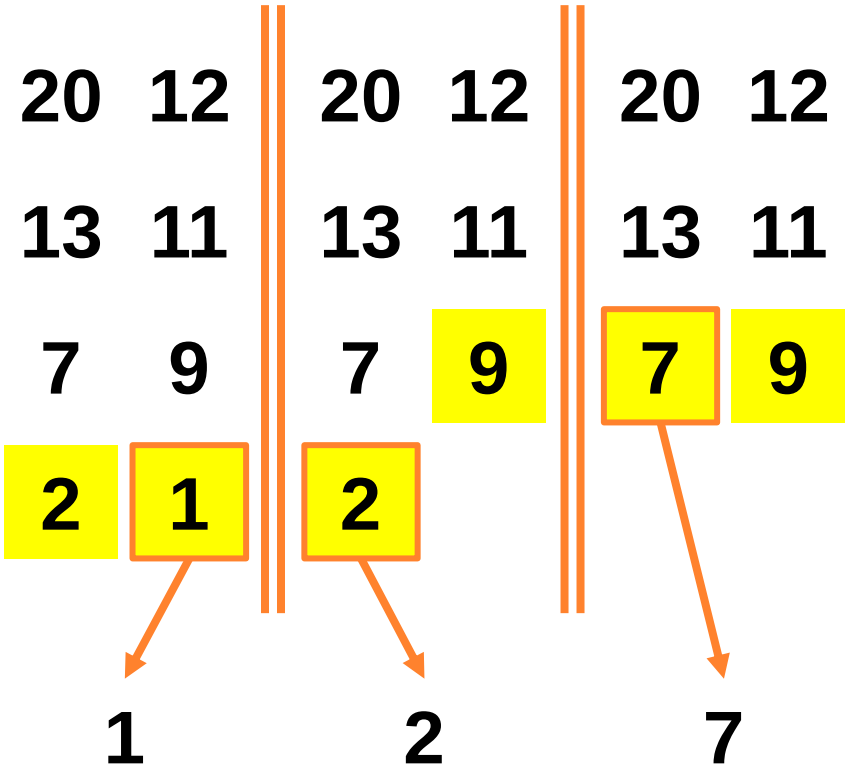
… MERGE SORTIRANJE …



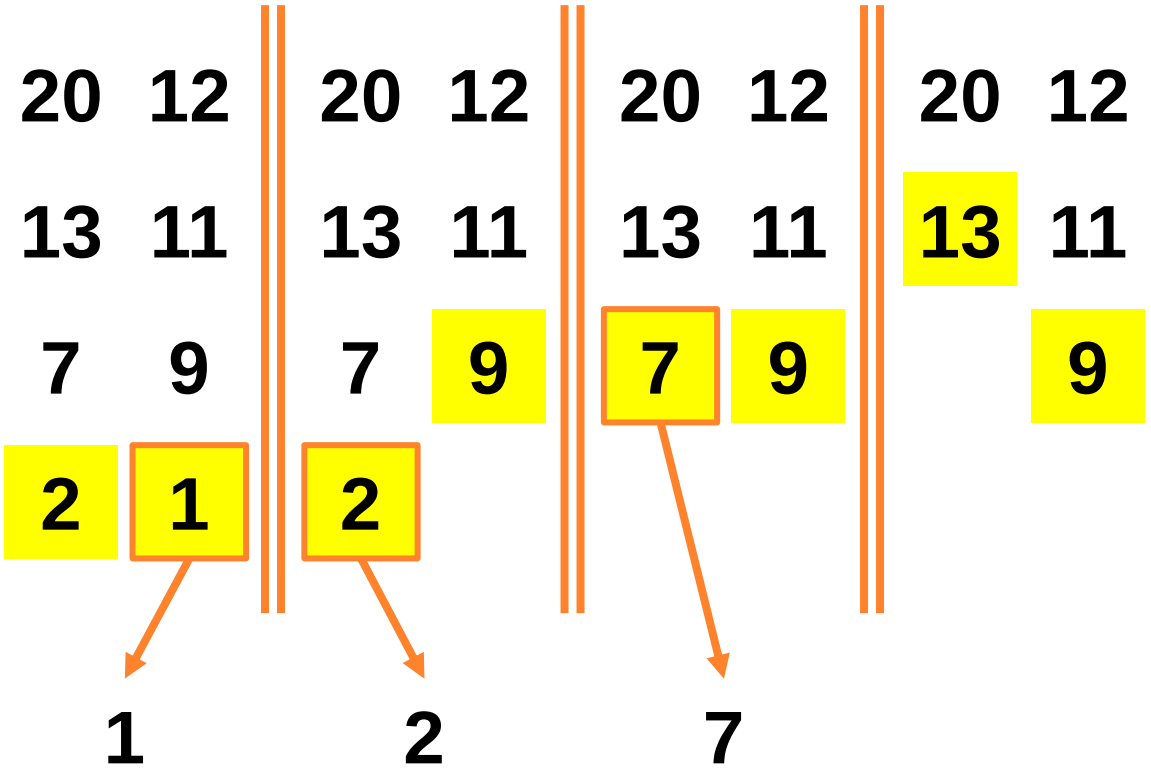
… MERGE SORTIRANJE …



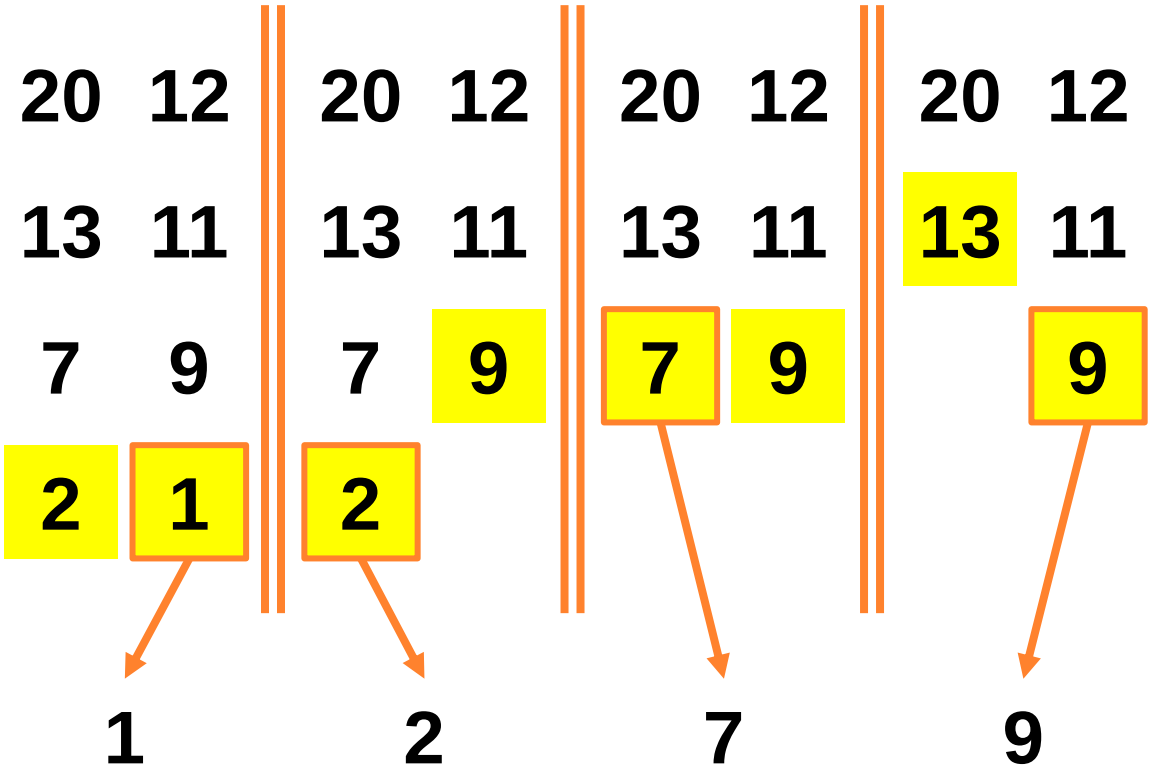
… MERGE SORTIRANJE …



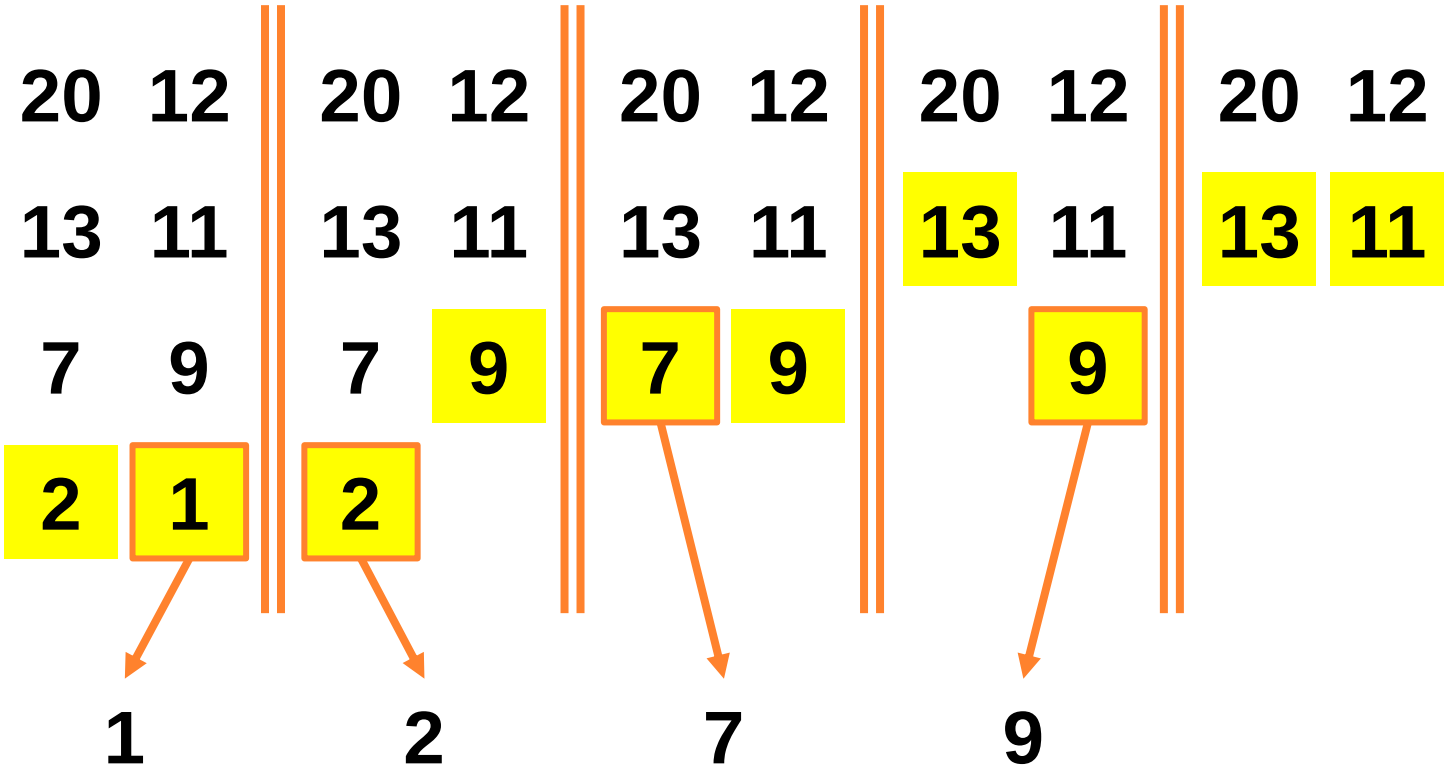
… MERGE SORTIRANJE …



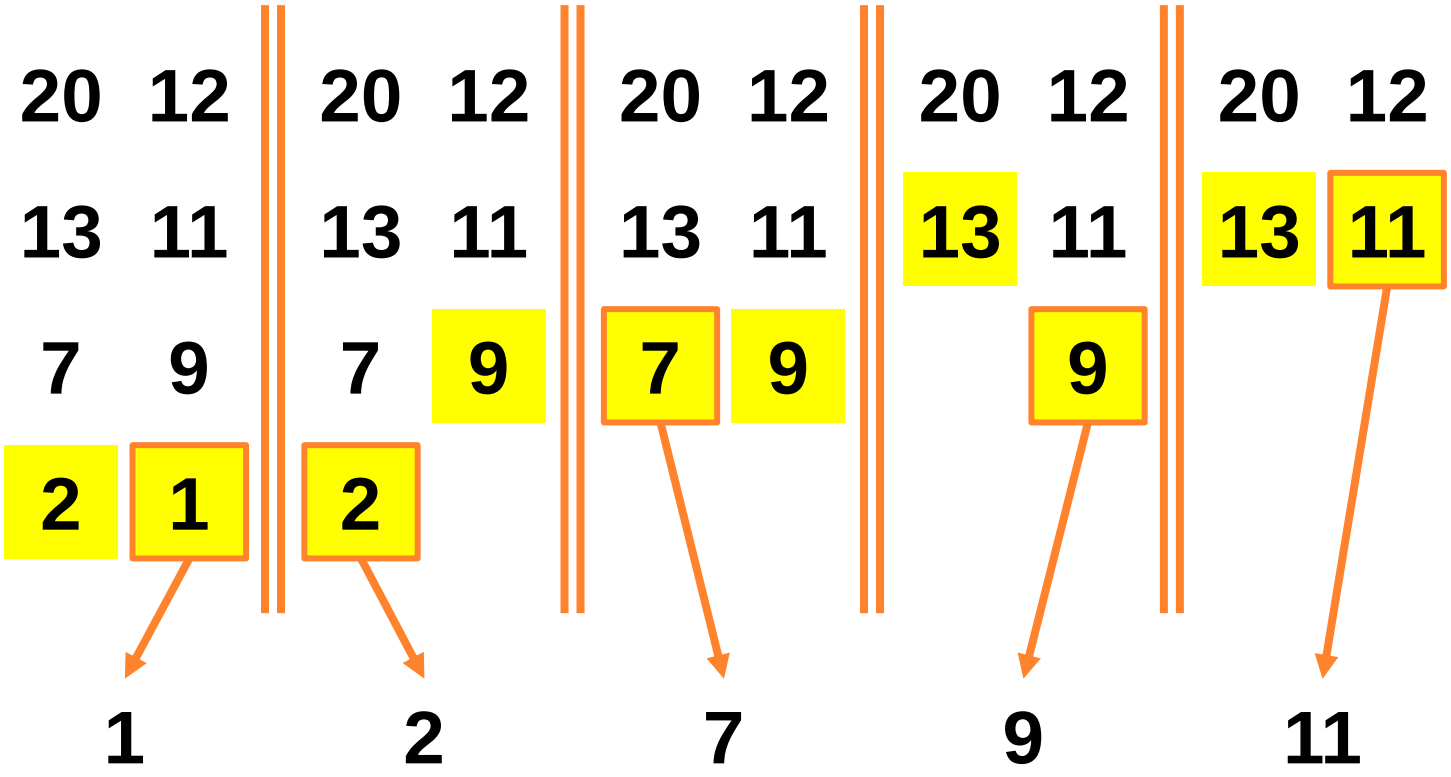
… MERGE SORTIRANJE …



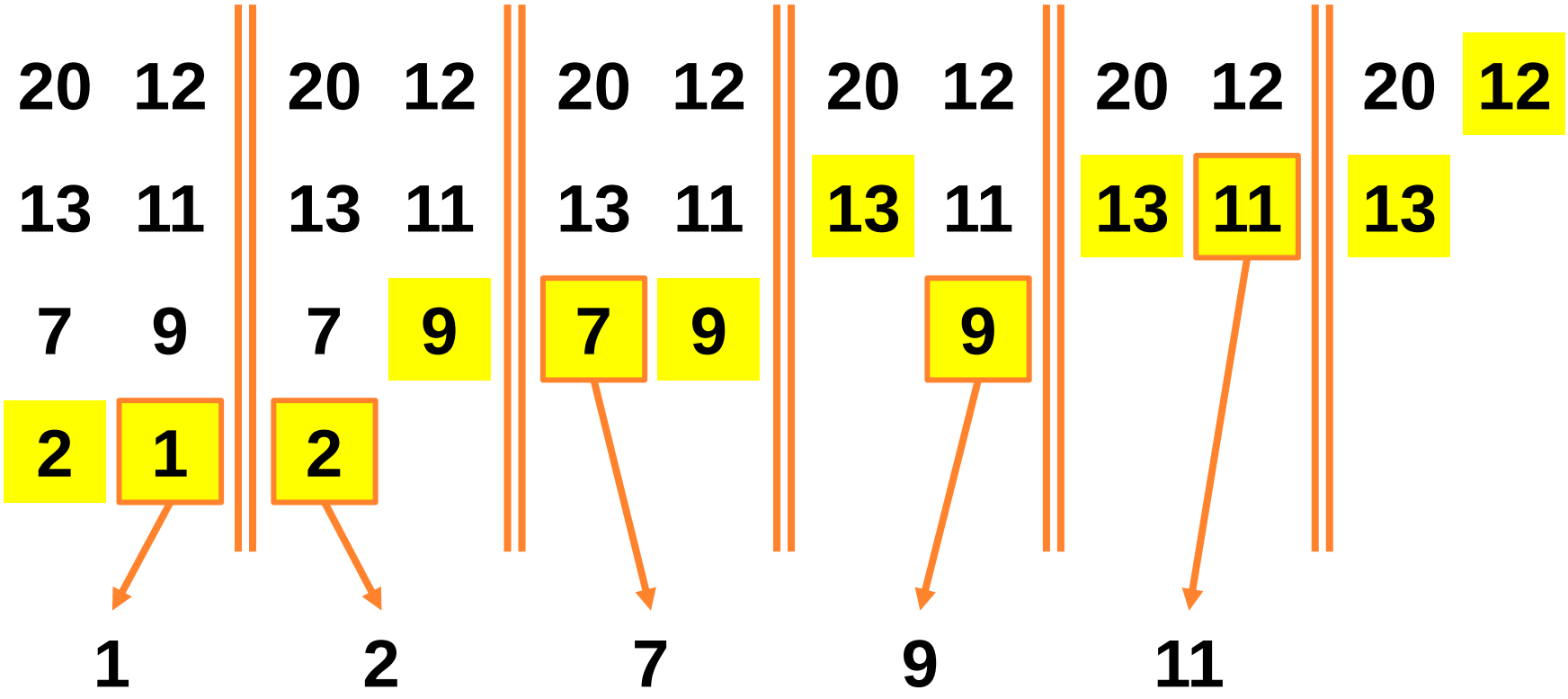
… MERGE SORTIRANJE …



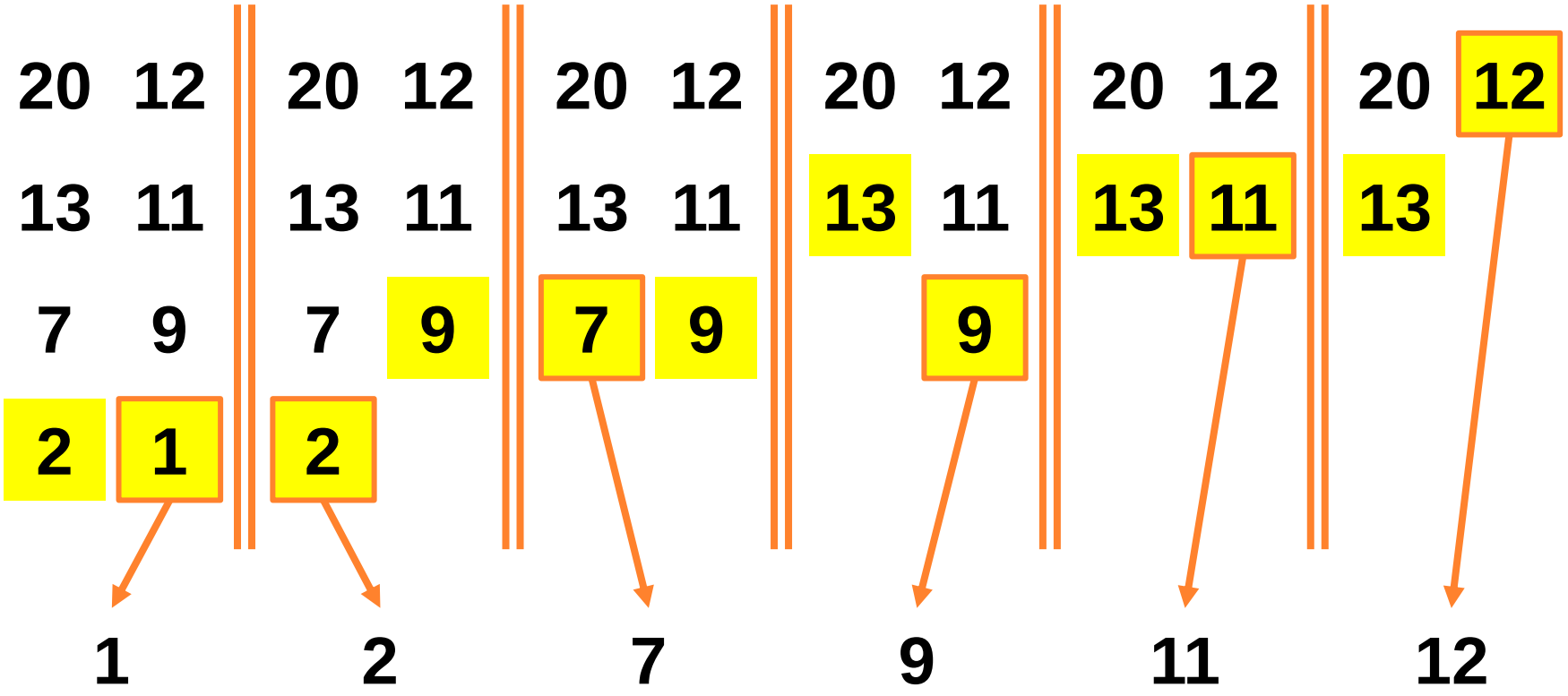
… MERGE SORTIRANJE …



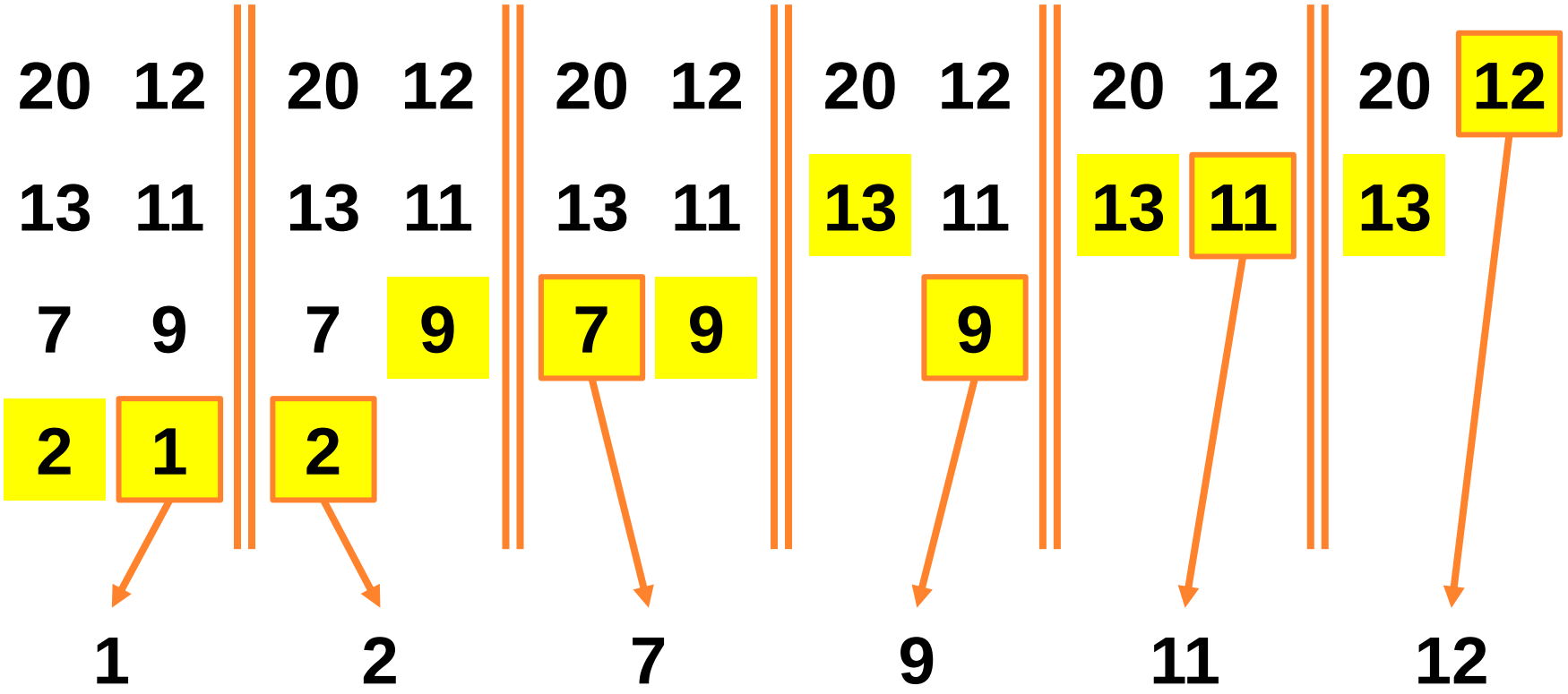
… MERGE SORTIRANJE …



… MERGE SORTIRANJE …



… MERGE SORTIRANJE …



1 2 7 9 11 12 13 20

… MERGE SORTIRANJE

- Što se tiče kompleksnosti operacija za spajanje je vrlo prosta i uzima konstantno vreme, jer se porede samo dve gornje vrednosti

- Vreme izvršavanja je

  $\Theta(n)$,

  jer se u suštini za n elemenata vrši najviše n osnovnih koraka

MERGE SORTIRANJE – ALGORITAM …

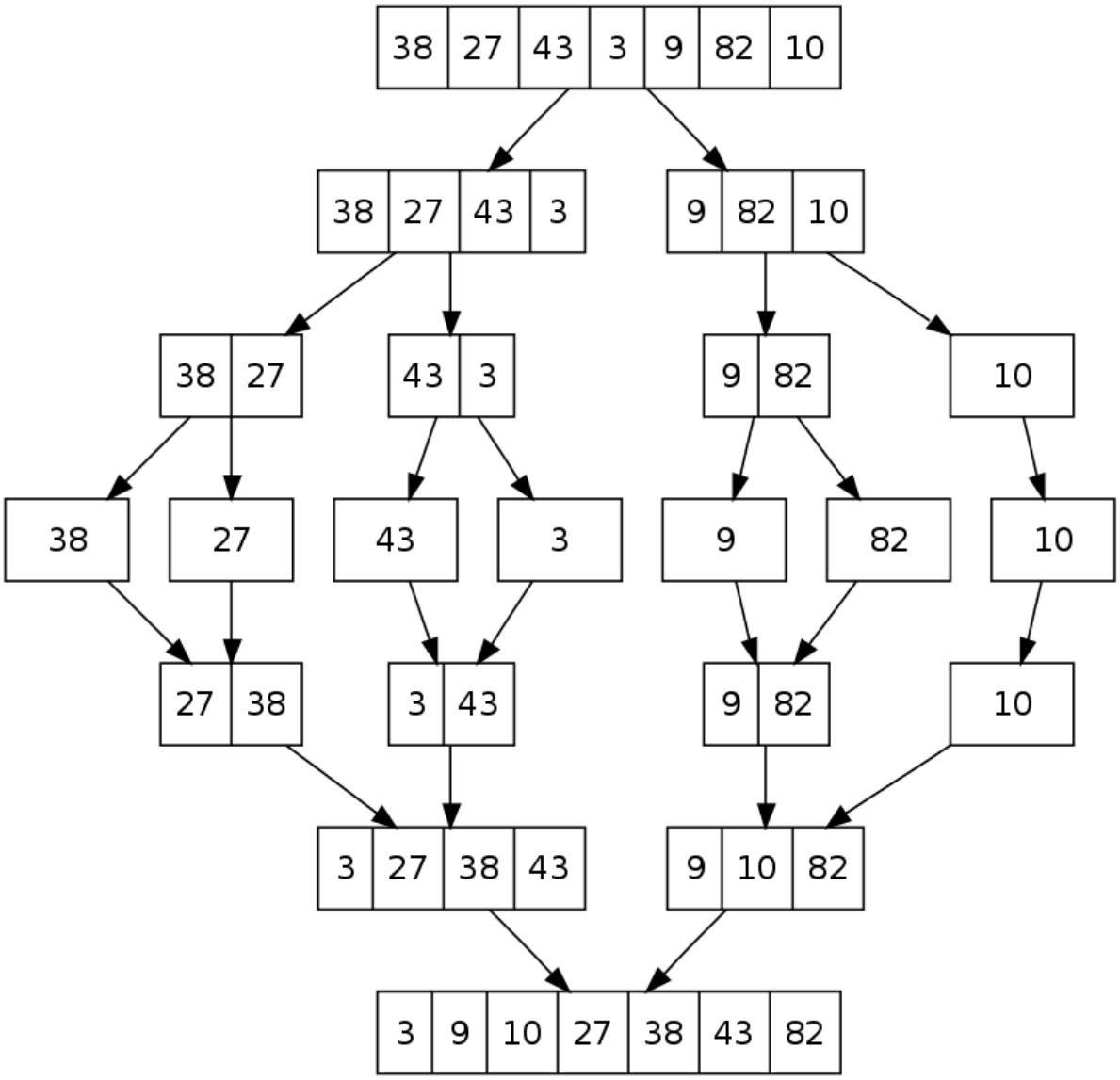

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
 4  for i = 1 to n1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

… MERGE SORTIRANJE – ALGORITAM …

- ***Merge(A, p, q, r)***,
  - ***A*** je niz koji se sortira
  - ***p, q, r*** su indeksi iz niza koji zadovoljavaju uslov: ***p ≤ q < r***
  - podrazumeva se da su podnizovi ***A[p .. q]*** i ***A[q+1 .. r]*** sortirani
  - Procedura ***Merge*** spaja podnizove u jedan sortirani niz koji zamenjuje niz ***A[p .. r]***
  - Šta radi vrednost ∞ ?
  - Kako se ne bi stalno proveravalo da li se stiglo do kraja podniza, uvodi se jedinstvena vrednost ∞ (ne može da se redovno nađe kao vrednost u podnizu)
  - **zašto baš ∞?**
  - Mora biti najveća moguća vrednost

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1]
 4  for i = 1 to n1
 5       L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7       R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13       if L[i] ≤ R[j]
14            A[k] = L[i]
15            i = i + 1
16       else A[k] = R[j]
17            j = j + 1
```

… MERGE SORTIRANJE – ALGORITAM …



… MERGE SORTIRANJE – ALGORITAM …

- ***Merge(A, p, q, r)***, zadržava invarijantnost petlje (koraci 12 – 17)
- Koja je tvrdnja koja se dokazuje?
- Na početku svake iteracije ***for*** petlje:
  - podniz ***A[p .. k– 1]*** sadrži ***k – p*** *sortiranih* najmanjih elemenata podnizova ***L[1 .. n_1+1]*** i ***R[1 .. n_2+1]***
  - ***L[i]*** i ***R[j]*** su najmanje vrednosti svojih podnizova koje još uvek nisu kopirane nazad u ***A***
- **Dokaz?**

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1]
 4  for i = 1 to n1
 5       L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7       R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13       if L[i] ≤ R[j]
14            A[k] = L[i]
15            i = i + 1
16       else A[k] = R[j]
17            j = j + 1
```

… MERGE SORTIRANJE – ALGORITAM …

- ***Inicijalizacija*** petlje (koraci 12 – 17)
  - Pre prve iteracije ***k = p***, što znači da je ***A[p .. k– 1]*** prazan
  - ***k – p = 0*** , što znači i da niz sadrži najmanje elemente iz ***L*** i ***R***
  - Kako je ***i = j = 1, L[i]*** i ***R[j]*** su definitivno najmanje vrednosti svojih podnizova koji još uvek nisu kopirane nazad u ***A***

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1]
 4  for i = 1 to n1
 5       L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7       R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13       if L[i] ≤ R[j]
14            A[k] = L[i]
15            i = i + 1
16       else A[k] = R[j]
17            j = j + 1
```

… MERGE SORTIRANJE – ALGORITAM …

- ***Održivost*** petlje (koraci 12 – 17)
  - Da bi se ovo dokazalo pretpostaviti prvo da je ***L[i] ≤ R[j]***, što znači da je ***L[i]*** najmanji element koji još nije kopiran u ***A***
  - Podniz ***A[p .. k– 1]*** po definiciji sadrži ***k – p*** najmanjih elemenata originalnog niza, te posle 14 koraka, kopira se ***L[i]*** u ***A[k]***, pa sada podniz ***A[p .. k]*** sadrži ***k – p + 1*** najmanjih elemenata originalnog niza
  - Inkrementiranje promenljive ***i*** u 15 koraku i inkrementiranje promenljive ***k*** na kraju ***for*** petlje, invarijantnost petlje se zadržava
  - Da je kojim slučajem ***L[i] > R[j]***, logika ostaje ista, ali se sve dešava u koracima 16 i 17

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1]
 4  for i = 1 to n1
 5       L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7       R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13       if L[i] ≤ R[j]
14            A[k] = L[i]
15            i = i + 1
16       else A[k] = R[j]
17            j = j + 1
```

… MERGE SORTIRANJE – ALGORITAM …

- ***Završavanje*** petlje (koraci 12 – 17)
  - Po završetku petlje, ***k = r + 1***
  - Podniz ***A[p .. k– 1]*** je zapravo ***A[p .. r]*** i sadrži ***k – p = r – p + 1*** najmanjih elemenata iz ***L[1 .. n_1+1]*** i ***R[1 .. n_2+1]*** u sortiranom redosledu (na osnovu invarijantnosti petlje)
  - ***L*** i ***R*** zajedno sadrže ***n_1+ n_2+ 2 = r – p + 3***, svi osim poslednja dva elementa su kopirana u ***A***, a ta dva su ∞ oznake

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1]
 4  for i = 1 to n1
 5       L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7       R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13       if L[i] ≤ R[j]
14            A[k] = L[i]
15            i = i + 1
16       else A[k] = R[j]
17            j = j + 1
```

… MERGE SORTIRANJE – ALGORITAM …

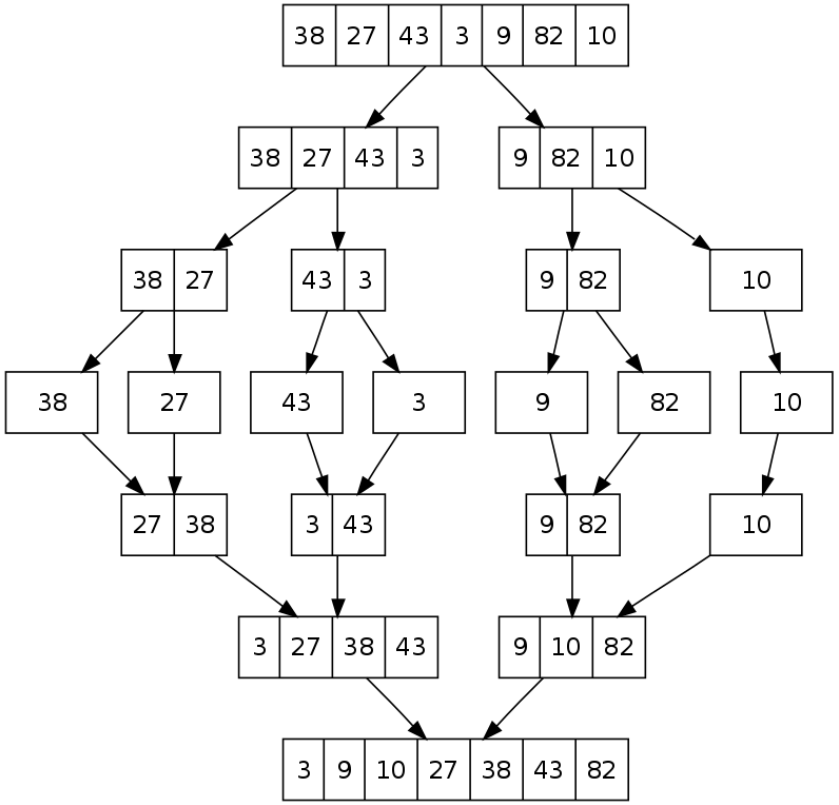
- Zašto Merge procedura ima ***Θ(n)*** vreme izvršavanja?
  - Koraci od 1–3 i 8–11 se izvršavaju u konstantnom vremenu
  - Koraci 4-7 u ***for*** petlji se izvršavaju u ***$\Theta(n_1 + n_2) = \Theta(n)$***
  - Postoji n iteracija ***for*** petlje od 12-17 koraka i svaki traje konstantno vreme

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1]
 4  for i = 1 to n1
 5       L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7       R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13       if L[i] ≤ R[j]
14            A[k] = L[i]
15            i = i + 1
16       else A[k] = R[j]
17            j = j + 1
```

… MERGE SORTIRANJE – ALGORITAM

- Merge procedura se može upotrebiti da se od nje napravi algoritam za Merge sortiranje (sada formalno)

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```



ZAVADI-PA-VLADAJ – ANALIZA …

- Kada algoritam koristi rekurziju, onda se može koristiti rekurentna jednačina (recurrence equation) ili jednostavno ponavljanje (recurrence)
- Opisuje kako se vreme izvršavanja rešenja problema veličine ***n*** prenosi (predstavlja) kroz vremena izvršavanja njegovih manjih delova
- Postoje matematički alati koji ovo rešavaju i pomoću kojih se mogu odrediti ograničenja u izvršavanju algoritma

$$T(n) = \begin{cases} \Theta(1), & za\ n \le c \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) & za\ sve\ preostale\ slučajeve \end{cases}$$

… ZAVADI-PA-VLADAJ – ANALIZA

- Kako se dolazi do te formule?
- Neka je ***T(n)*** vreme izvršavanja nekog algoritma za ulaz veličine ***n***

$$T(n) = \begin{cases} \Theta(1), \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) \end{cases}$$

- Ako je ***n*** dovoljno malo, npr. ***n ≤ c***, gde je ***c*** neka konstanta, direktno rešenje uzima neko konstantno vreme što se piše ***Θ(1)***
- Pretpostaviti da deljenjem problema nastaje ***a*** potproblema iste veličine ***1/b*** (što je retko u praksi)
- Potrebno je ***T(n/b)*** vremena da se reši potproblem veličine ***n/b***, pa je potrebno ***aT(n/b)*** vremena da se oni reše
- Potrebno je ***D(n)*** vremena da bi se problem podelio na potprobleme i ***C(n)*** vremena da se rešenja potproblema spoje u rešenje originalnog problema

MERGE SORTIRANJE – ANALIZA …

- Zbog jednostavnosti pretpostaviti da je broj elemenata u nizu koji se sortira paran (algoritam radi i za neparan broj)
- Jasno je da ako niz ima 1 element, vreme potrebno za MERGE sortiranje je ***Θ(1)***
- Ako je broj elemenata ***n > 1***, vreme izvršavanja se može podeliti na sledeći način:
  - ***Divide***: podela na 2 podniza po sredini uzima konstanto vreme ***D(n) = Θ(1)***
  - ***Conquer***: rekurzivno se rešava svaki od 2 potproblema (sortitanje podnizova), što je vreme izvršavanja od ***2T(n/2)***
  - ***Combine***: kao što je već izračunato spajanja podnizova od ***n*** elemenata traje ***C(n) = Θ(n)*** vremena
- ***D(n) + C(n) = Θ(1) + Θ(n) = Θ(n)***,
  rezultat sumiranja je linearna funkcija od ***n*** što je jednako ***Θ(n)***
- Tako se dobija **worst-case** vreme izvršavanja algoritma za MERGE sortiranje

… MERGE SORTIRANJE – ANALIZA …

- Tako se dobija **worst-case** vreme izvršavanja algoritma za MERGE sortiranje

$$T(n) = \begin{cases} \Theta(1), & za\ n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & za\ n \geq 1 \end{cases}$$

- Matematički se može dokazati da je **worst-case** vreme izvršavanja algoritma: ***Θ(n · log n)***
- **worst-case** vreme izvršavanja od ***Θ(n · log n)*** je brže od ***Θ(n²)***
- Da je **worst-case** vreme izvršavanja algoritma za MARGE sortiranje ***Θ(n · log n)*** može se dokazati i intuitivno, ako se pretpostavi da je:

$$T(n) = \begin{cases} c, & za\ n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & za\ n \geq 1 \end{cases}$$

- gde ***c*** konstanto vreme potrebno da se reši problem veličine 1

… MERGE SORTIRANJE – ANALIZA …

- Rešenje ***T(n) = 2T(n/2) + cn***, za ***c > 0*** (pretpostavka: n je paran broj)

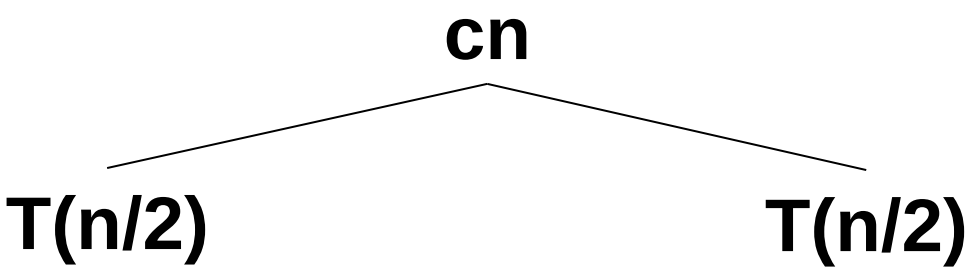
… MERGE SORTIRANJE – ANALIZA …

- Rešenje ***T(n) = 2T(n/2) + cn***, za ***c > 0*** (pretpostavka: n je paran broj)

T(n)

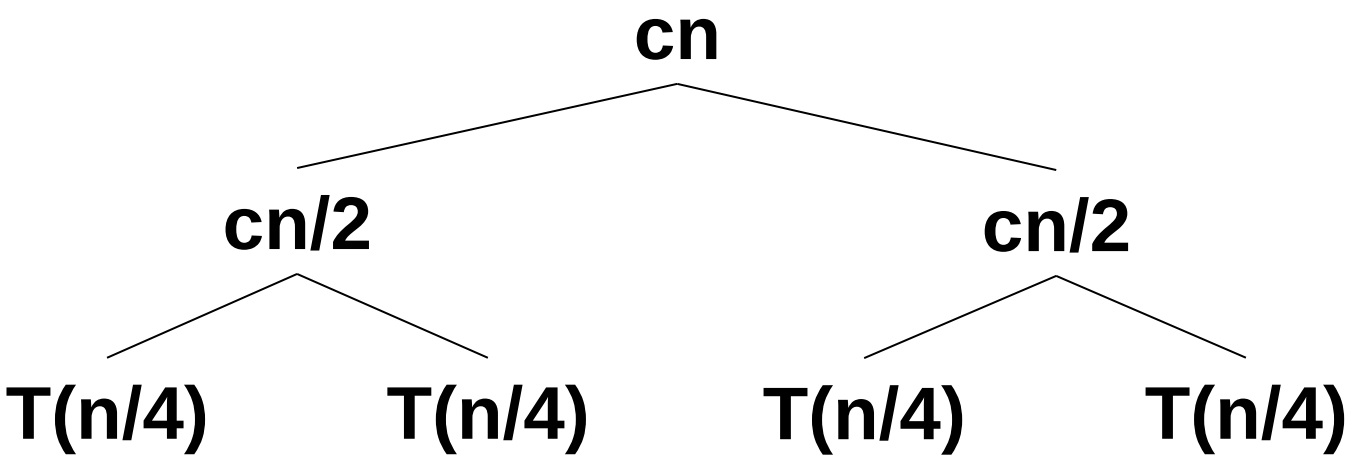
… MERGE SORTIRANJE – ANALIZA …

- Rešenje ***T(n) = 2T(n/2) + cn***, za ***c > 0*** (pretpostavka: n je paran broj)



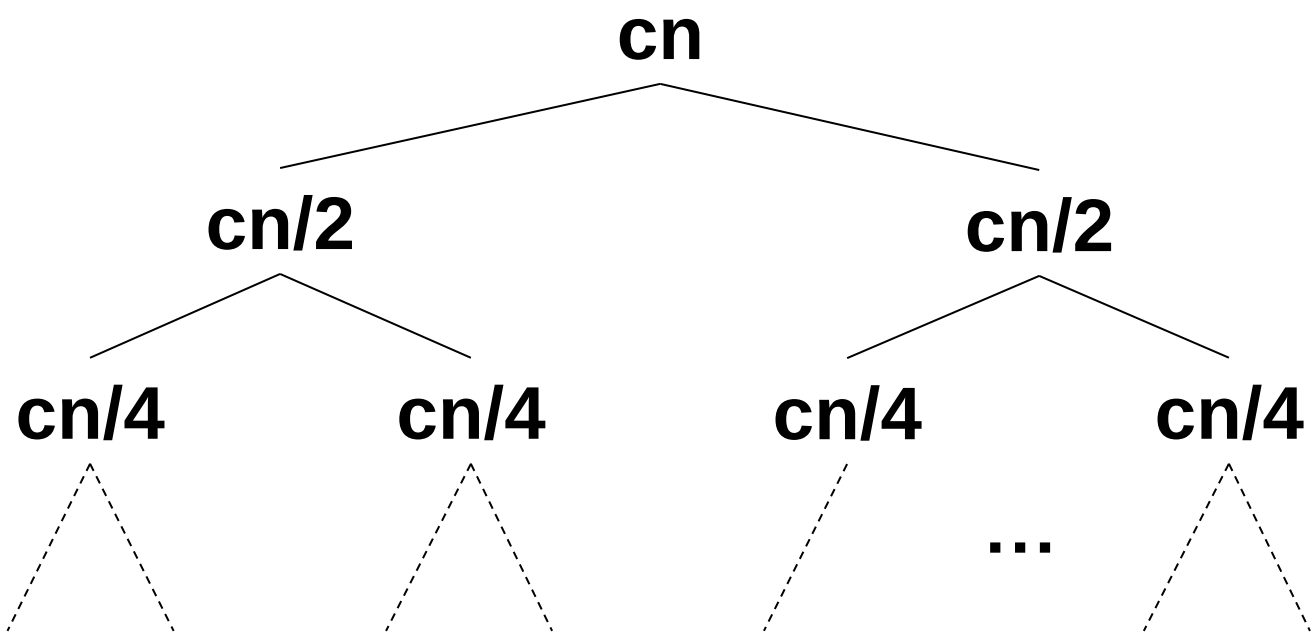
… MERGE SORTIRANJE – ANALIZA …

- Rešenje ***T(n) = 2T(n/2) + cn***, za ***c > 0*** (pretpostavka: n je paran broj)



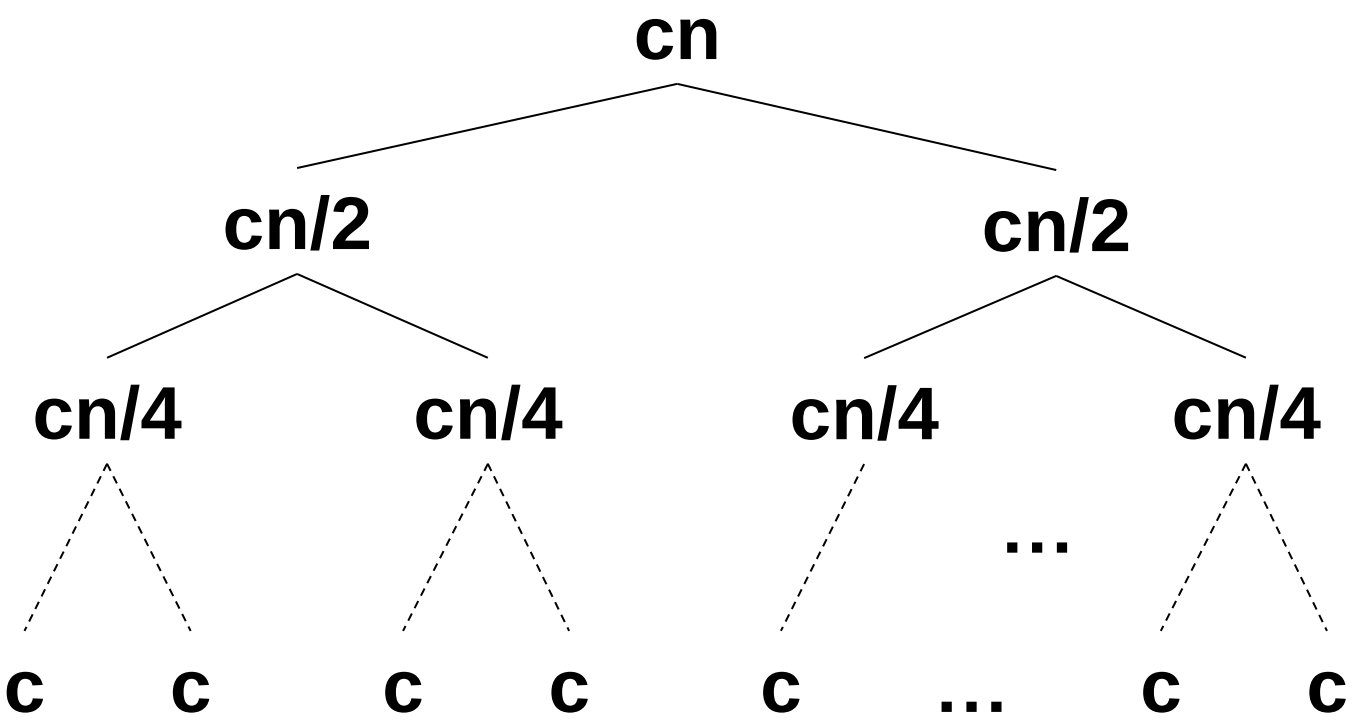
… MERGE SORTIRANJE – ANALIZA …

- Rešenje ***T(n) = 2T(n/2) + cn***, za ***c > 0*** (pretpostavka: n je paran broj)



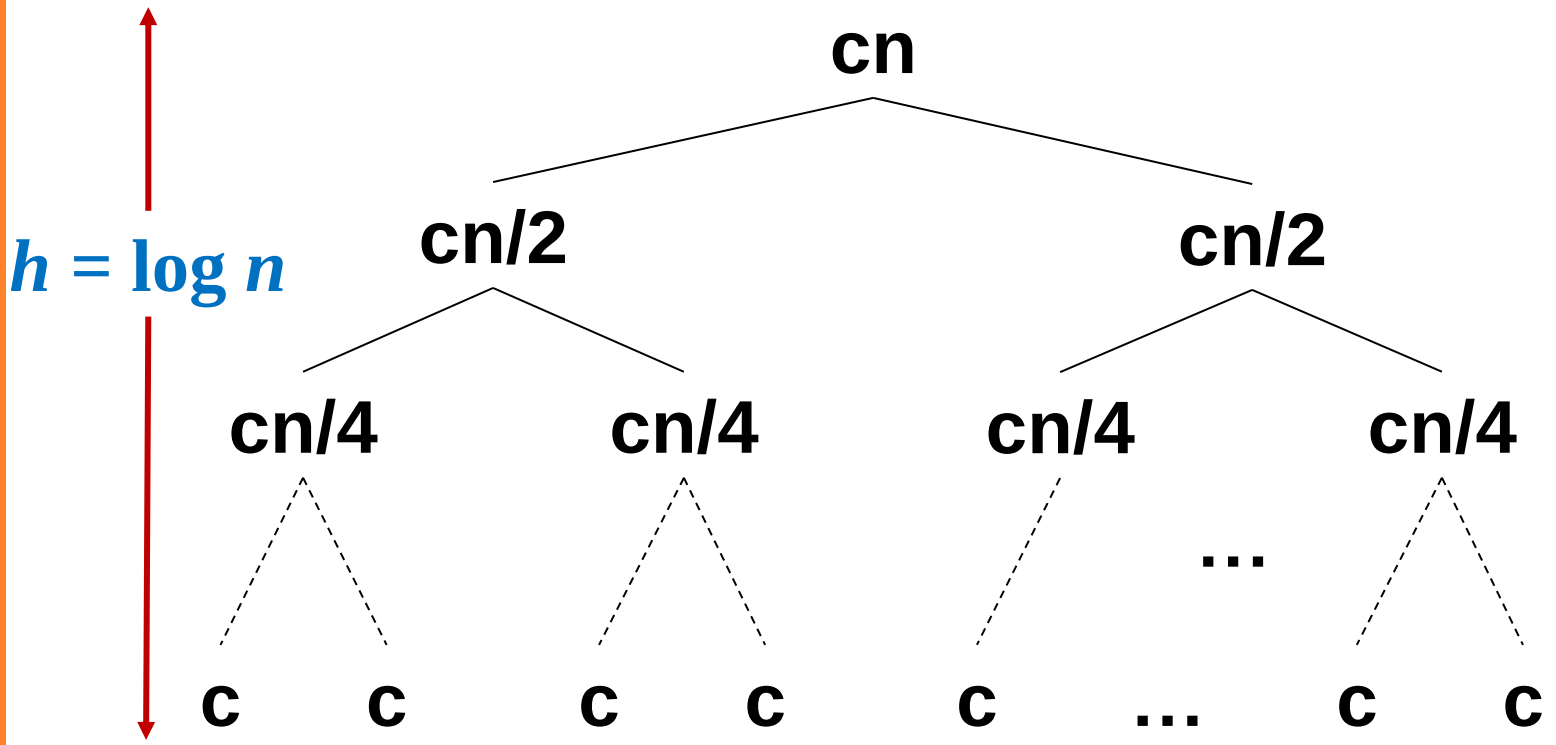
… MERGE SORTIRANJE – ANALIZA …

- Rešenje ***T(n) = 2T(n/2) + cn***, za ***c > 0*** (pretpostavka: n je paran broj)



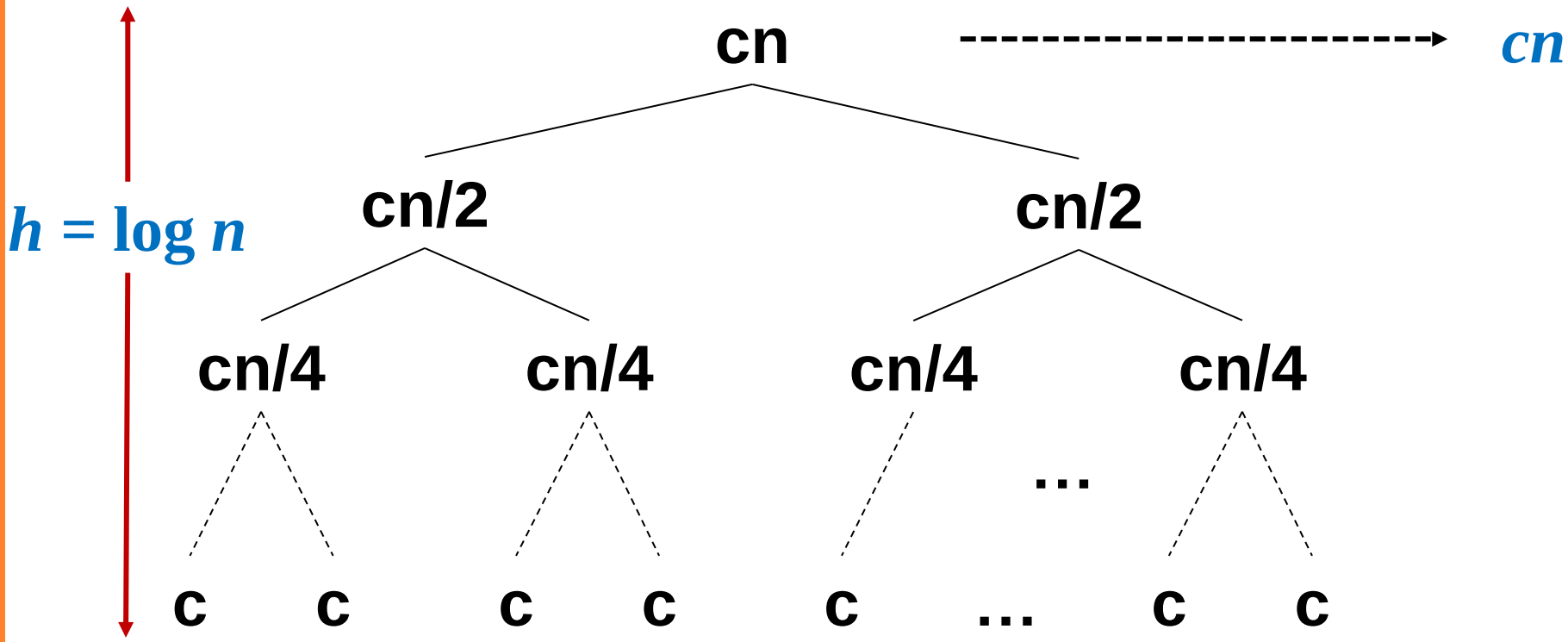
… MERGE SORTIRANJE – ANALIZA …

- Rešenje ***T(n) = 2T(n/2) + cn***, za ***c > 0*** (pretpostavka: n je paran broj)



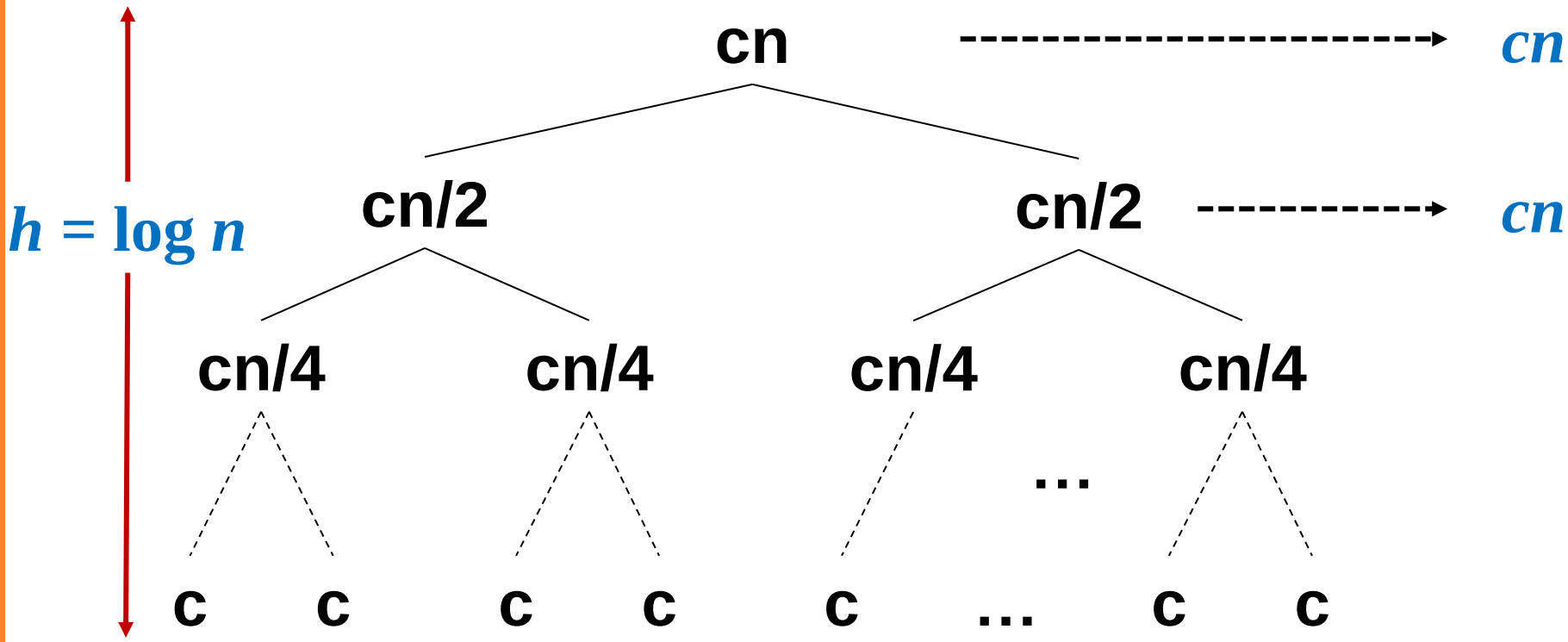
… MERGE SORTIRANJE – ANALIZA …

- Rešenje ***T(n) = 2T(n/2) + cn***, za ***c > 0*** (pretpostavka: n je paran broj)



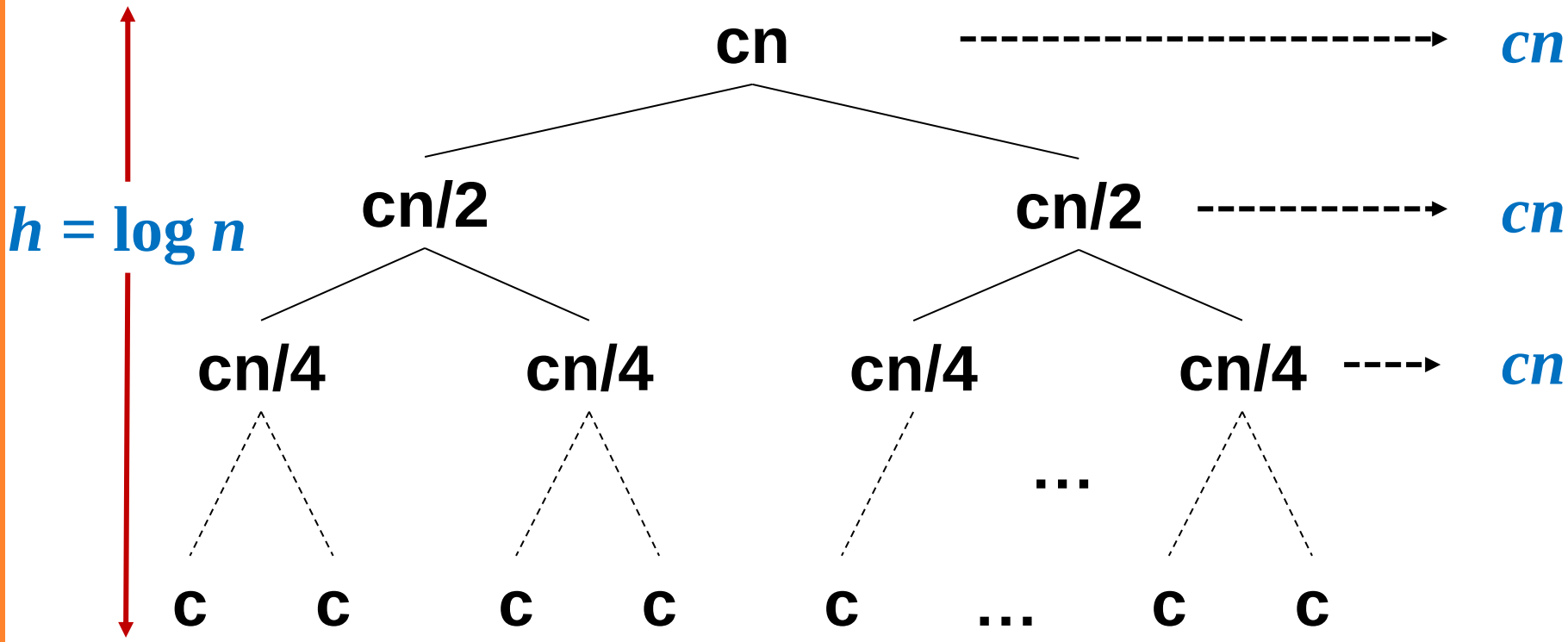
… MERGE SORTIRANJE – ANALIZA …

- Rešenje ***T(n) = 2T(n/2) + cn***, za ***c > 0*** (pretpostavka: n je paran broj)



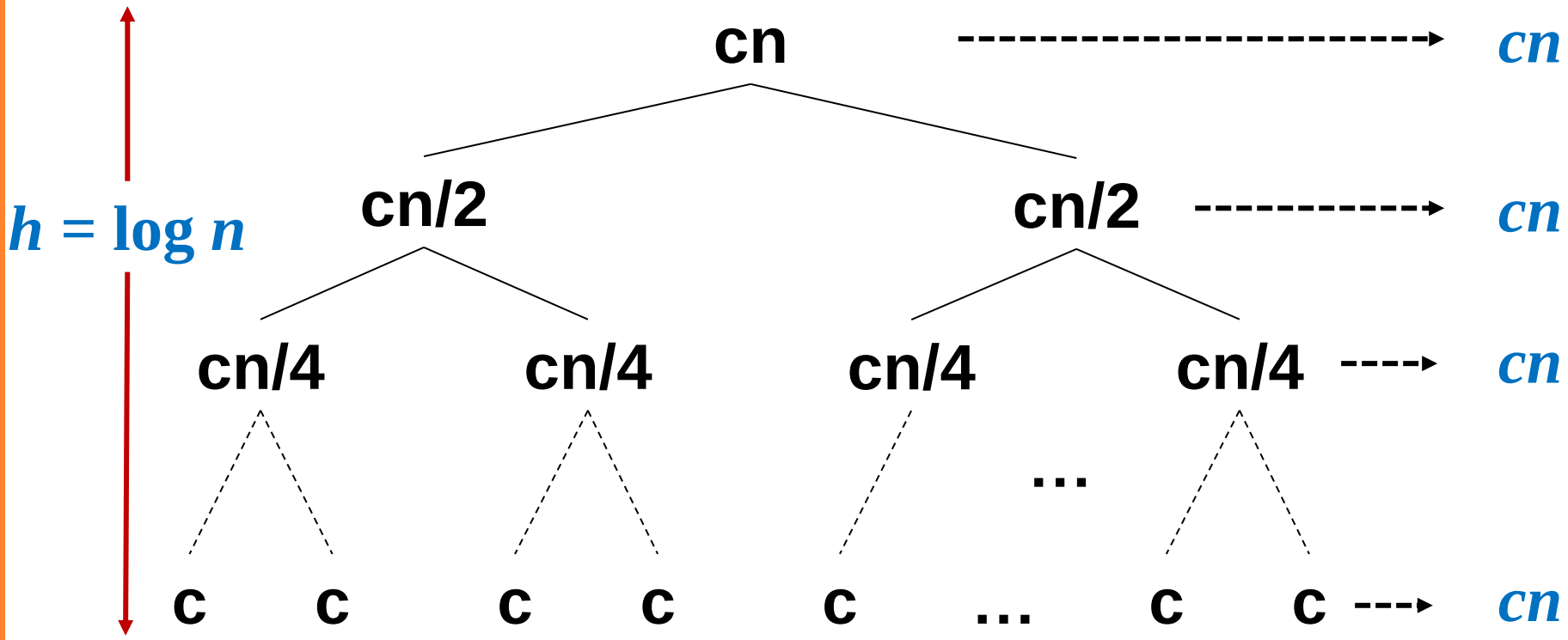
… MERGE SORTIRANJE – ANALIZA …

- Rešenje ***T(n) = 2T(n/2) + cn***, za ***c > 0*** (pretpostavka: n je paran broj)



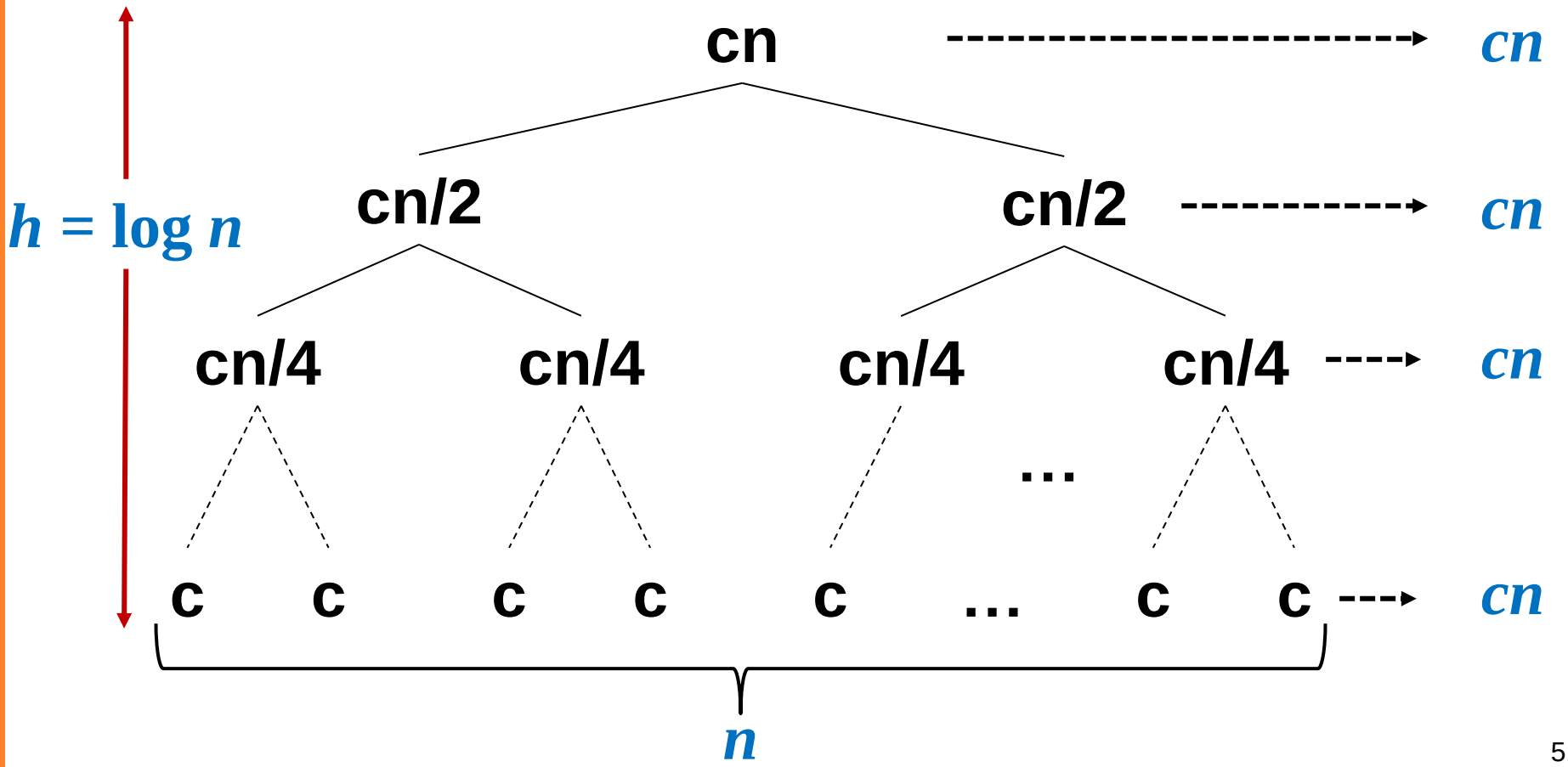
… MERGE SORTIRANJE – ANALIZA …

- Rešenje ***T(n) = 2T(n/2) + cn***, za ***c > 0*** (pretpostavka: n je paran broj)



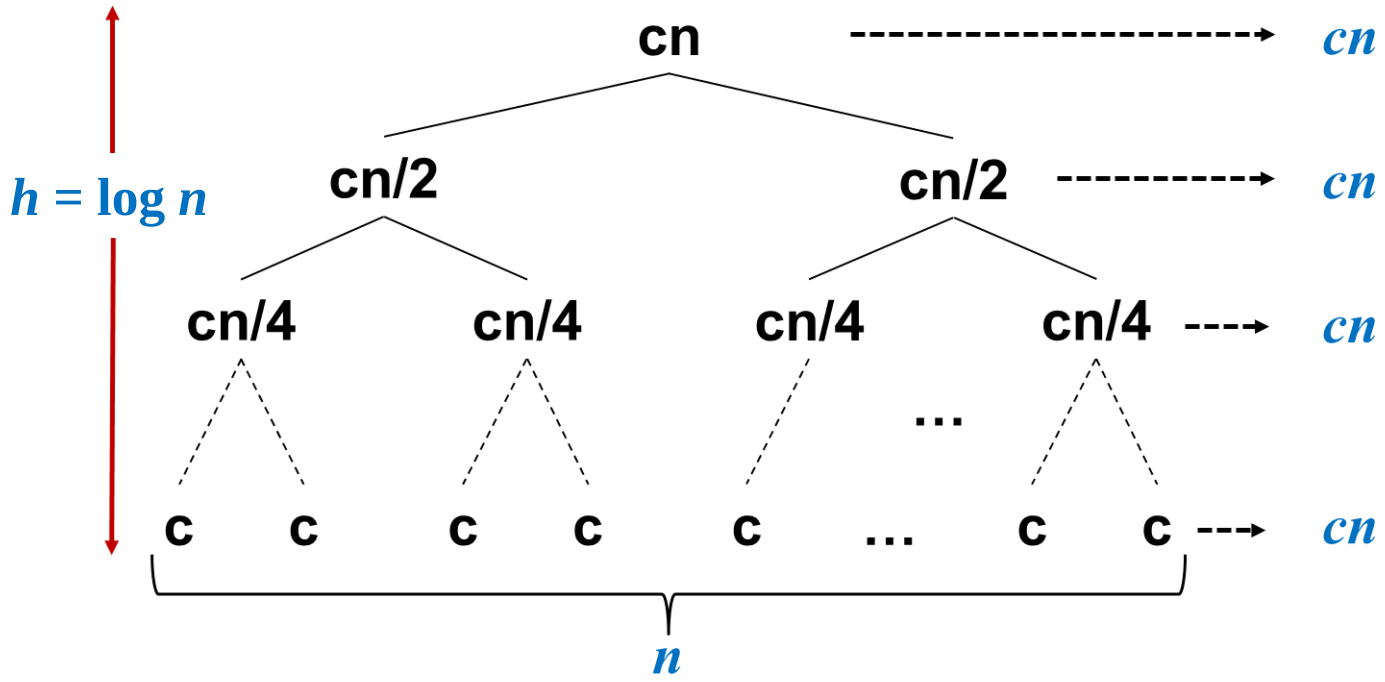
… MERGE SORTIRANJE – ANALIZA …

- Rešenje ***T(n) = 2T(n/2) + cn***, za ***c > 0*** (pretpostavka: n je paran broj)



… MERGE SORTIRANJE – ANALIZA …

- Rešenje ***T(n) = 2T(n/2) + cn***, za ***c > 0*** (pretpostavka: n je paran broj)



- Potpuno razvijeno stablo rekurzije ima **log *n* + *1*** nivoa (odnosno ima visinu **log *n***, kao što je naznačeno), a zbir vremena u svim čvorovima na istom nivou je ***cn***. Ukupna suma je, prema tome, ***cn*** · **log *n* + *cn***, što je ***n*** · **log *n*** . Te je kompleksnost algoritma: ***Θ(n*** · **log *n)***

… MERGE SORTIRANJE – ANALIZA

- Zaključak
  - $\Theta(n \cdot \log n)$ ima stepen rasta manji od $\Theta(n^2)$
  - Asimptotski gledano, MERGE sortiranje je brže od INSERTION sortiranja u najgorem slučaju
  - U praksi, MERGE sortiranje je brže od INSERTION sortiranja za svako $n > 30$
  - Testirajte sami