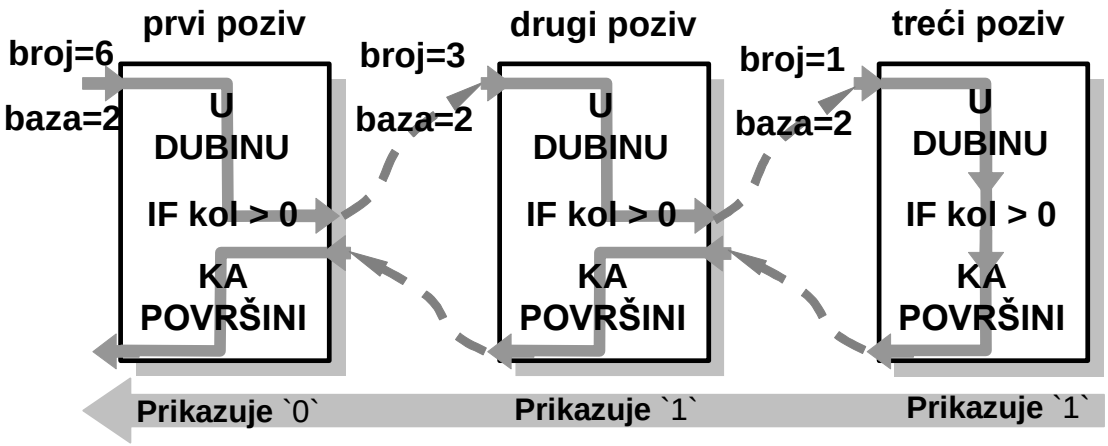
DIVIDE-AND-CONQUER

ZAVADI-PA-VLADAJ …

- Već smo se upoznali sa tim …
- Zavadi-pa-vladaj (divide-and-conquer) pristup znači da pojednostavimo problem, pa da ga rešavamo parče po parče
- Problem se razbija na podprobleme koji su slični originalnom, ali jednostavniji po svojoj prirodi
  - npr. da li je lakše sortirati 8 brojeva ili 4 ili 2 broja …
- 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

… ZAVADI-PA-VLADAJ

- Ako je problem dovoljno veliki, onda ima smisla rešavati ga rekurzijom
- Algoritam se rekurzivno poziva sve dok se ne dođe do najjednostavnijeg koraka (*base case*), pa se posle rekurzivno vraća ka prvom pozivu
- Nekada jednostavni koraci nisu iste prirode kao i ostatak rekurzije, pa se rešavaju zasebnim algoritmom koji je najčešće deo koraka kombinovanja (poziva u visinu)
- Kada se rekurzija rešava potprogramima, onda postoje 3 faze u izvršavanju potprograma:
  - Poziv u dubinu
  - Uslovno izvršavanje
  - Poziv ka površini



REKURENTNE JEDNAČINE

PONAVLJANJE (RECURRENCE) …

- Zavadi-pa-vladaj algoritmi idu ruku pod ruku sa ponavljanjem (*recurrence*) i rekurentnim jednačinama koje nam pružaju prirodan način da karakterišemo prirodu ovih algoritama i da odredimo vreme izvršavanja zavadi-pa-vladaj algoritama
- Opisuje kako se vreme izvršavanja rešenja problema veličine $\boldsymbol{n}$ prenosi (predstavlja) kroz vremena izvršavanja njegovih manjih delova
- Njihovo rešavanje je kao rešavanje integrala i diferencijalnih jednačina, mora se znati par trikova
- Primer rekurentne jednačine iz algoritma za MERGE sortiranje:

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

Na kraju je: $\boldsymbol{T(n) = n \cdot \log n}$

… PONAVLJANJE (RECURRENCE)

- Postoje različite situacije u kojima se pojavljuje rekurentnost, problem se može podeliti i na nejednake delove, npr. 2/3 prema 1/3, u tom slučaju jednačina dobija oblik: $\boldsymbol{T(n) = T\left(\frac{2n}{3}\right) + T\left(\frac{n}{3}\right) + \Theta(n)}$
- Potproblemi ne moraju biti konstantni deo velikog problema, npr. to može biti i linearna funkcija koja je za samo jedan element manja od originalnog problema: $\boldsymbol{T(n) = T(n-1) + \Theta(1)}$
- Tri metode za rešavanje rekurentnost, tj. za određivanje ***Θ*** i ***O*** graničnih funkcija
  - Metoda supstitucije – pogodimo graničnu funkciju i dokažemo je matematičkom indukcijom
  - Metoda stabla rekurzije – pretvara rekurziju u stablo gde svaki čvor predstavlja grananje u rekurziji i za svaki čvor se računa koliko to grananje košta, posebnom tehnikom sumiranja (bounding summations) se određuje ukupna cene algoritma
  - Master metoda – primena metode $\boldsymbol{T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)}$

REKURENTNE JEDNAČINE – METODA SUPSTITUCIJE

METODA SUPSTITUCIJE …

- Radi vrlo jednostavno:
  1. Pretpostaviti graničnu funkciju
  2. Dokazati da smo dobro pogodili matematičkom indukcijom
- Moćna metoda, samo jedan problem! Koji?
- Sve zavisi da li smo dobro pogodili!
- Može se iskoristiti za definisanje ili gornje ili donje asimptotske granice
- Neka je za primeri gornja asimptotska granica neke rekurentnosti:

$$\boldsymbol{T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n}$$

- Pretpostavimo (nagađamo) da je rešenje: $\boldsymbol{T(n) = O(n \cdot \log n)}$
- Da bi upotrebili metodu supstitucije, moramo dokazati da je $\boldsymbol{T(n) \leq c \cdot n \cdot \log n}$, za neku odgovarajuću konstantu $\boldsymbol{c > 0}$

… METODA SUPSTITUCIJE …

- Rešenje: $T(n) = O(n \cdot \log n)$
- Dokazati da je $T(n) \leq c \cdot n \cdot \log n$, za neko $c > 0$
- Prvo krećemo od toga da dokažemo da prethodna formula važi (da je to gornja granica) za svako $m < n$, naročito za $m = \lfloor n/2 \rfloor$, što dovodi do: $T(\lfloor n/2 \rfloor) \leq c \cdot \lfloor n/2 \rfloor \cdot \log(\lfloor n/2 \rfloor)$
- Kada se to zameni u rekurentnu jednačinu:

$$
\begin{aligned}
T(n) &\leq 2 \cdot \left(c \cdot \left\lfloor \frac{n}{2} \right\rfloor \cdot \log\left(\left\lfloor \frac{n}{2} \right\rfloor \right) \right) + n \\
&\leq cn \cdot \log\left(\frac{n}{2}\right) + n \\
&= cn \cdot \log(n) - cn \cdot \log(2) + n \\
&= cn \cdot \log(n) - (cn - n) \\
&\leq cn \cdot \log(n)
\end{aligned}
$$

željeno - ostatak

željeno, važi uvek kada je $n(c-1) > 1$

- Poslednji korak važi sve dok je $c \geq 1$

… METODA SUPSTITUCIJE …

$$T(n) \leq 2 \cdot \left(c \cdot \left\lfloor \frac{n}{2} \right\rfloor \cdot \log\left(\left\lfloor \frac{n}{2} \right\rfloor\right)\right) + n \leq cn \cdot \log\left(\frac{n}{2}\right) + n$$
$$= cn \cdot \log(n) - cn \cdot \log(2) + n = cn \cdot \log(n) - cn + n \leq cn \cdot \log(n)$$

- Matematička indukcija zahteva da se ovo dokaže za sve granične slučajeve
- To znači da se za granične slučajeve mora dokazati da postoji c dovoljno veliko da $T(n) \leq cn \cdot \log(n)$ važi i za granične uslove
- Ovaj uslov nekada može da izazove probleme. Npr. pretpostaviti da je $T(1) = 1$ jedini granični uslov rekurentnosti algoritma
- Za $n = 1$ sledi $T(1) \leq c \cdot 1 \cdot \log(1)$ što je $T(1) = 0$ a to je suprotno sa početnim uslovom
- Šta sad?

… METODA SUPSTITUCIJE …

$$T(n) \leq 2 \cdot \left(c \cdot \left\lfloor \frac{n}{2} \right\rfloor \cdot \log\left(\left\lfloor \frac{n}{2} \right\rfloor\right)\right) + n \leq cn \cdot \log\left(\frac{n}{2}\right) + n$$
$$= cn \cdot \log(n) - cn \cdot \log(2) + n = cn \cdot \log(n) - cn + n \leq cn \cdot \log(n)$$

- Iskoristićemo osobinu asimptotskih notacija da važe za svako $\mathbf{n} \geq n_0$, gde je n_0 vrednost koju mi biramo
- Ostaje granični uslov, ali ga ne posmatramo, te se za bazni slučaj dokaza indukcije može onda uzeti i $T(2)$ i $T(3)$
- Igrom slučaja, za $c \geq 2$ važe oba granična uslova i $T(2) \leq c \cdot 2 \cdot \log(2)$ i $T(3) \leq c \cdot 3 \cdot \log(3)$
- Za većinu primera, ovo je jednostavno dokazati i za malo n je lako pronaći neko n_0 za koje se može asimptotski dokazati da tvrdnja važi

… METODA SUPSTITUCIJE …

- Na žalost ne postoji opšti način da se pogodi korektno rešenje za rekurentnost, za to je potrebno iskustvo i ponekad i kreativnost
- Na sreću postoje heuristike koje mogu pomoći u pogađanju (može se iskoristiti i sledeća metoda koju ćemo raditi)
- Ako je rekurentnost slična prethodnoj, onda je korektno pretpostaviti da će i rešenje biti slično
- Npr: $\boldsymbol{T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor + 17\right) + n}$
- Da li je ovo teško? $\boldsymbol{\left\lfloor \frac{n}{2} \right\rfloor + 17}$
- Za dovoljno veliko $\boldsymbol{n}$ da li $\boldsymbol{17}$ pravi razliku?
- Tako da je korektno pretpostaviti da je $\boldsymbol{T(n) = O(n \cdot \log n)}$ rešenje i za ovaj slučaj
- Mogu se tražiti i ne striktne granice, $\boldsymbol{O}$ i $\boldsymbol{\Omega}$, pa onda to pooštravati

… METODA SUPSTITUCIJE …

- Često se dešava da se pogodi ograničavajuća funkcija, ali da se ona ne može dokazati ili pretpostavka nije dovoljno čvrsta (gornja granica nije dovoljno jaka)
- U takvim situacijama rešenje je često tehnika oduzimanja (*subtracting*), gde se monom nižeg nivoa oduzima od prve pretpostavke (i posle toga matematika proradi…)
- Npr. $\boldsymbol{T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1}$
- Pretpostavimo da je rešenje: $\boldsymbol{T(n) = O(n)}$
- Treba da se dokaže $\boldsymbol{T(n) \leq c \cdot n}$ za odgovarajuću konstantu $\boldsymbol{c}$
- Zamenom pretpostavke u rekurentnosti

$$\boldsymbol{T(n) \leq c \cdot \left\lfloor \frac{n}{2} \right\rfloor + c \cdot \left\lceil \frac{n}{2} \right\rceil + 1 = cn + 1}$$

- što ne implicira $\boldsymbol{T(n) \leq cn}$ za svako $\boldsymbol{c}$

… METODA SUPSTITUCIJE …

$$T(n) \leq c \cdot \left\lfloor \frac{n}{2} \right\rfloor + c \cdot \left\lceil \frac{n}{2} \right\rceil + 1 = cn + 1$$

- što ne implicira $T(n) \leq cn$ za svako c
- Može se probati $T(n) = O(n^2)$ kao rešenje, ali je prva pretpostavka zapravo dobra
- Inicijalna pretpostavka je tačna ali se hipoteza koja se dokazuje indukcijom mora učiniti strožijom
- Pretpostavka greši samo za jednu konstantu a koja je monom nižeg reda, ali matematička indukcija radi samo za precizno definisanu hipotezu (tu nema mnogo odstupanja)
- Ali ako se pretpostavi: $T(n) \leq c \cdot n - d$, gde je $d \geq 0$
- Sada je dokaz:

$$T(n) \leq \left(c \cdot \left\lfloor \frac{n}{2} \right\rfloor - d\right) + \left(c \cdot \left\lceil \frac{n}{2} \right\rceil - d\right) + 1 = cn - 2d + 1 \leq cn - d$$

… METODA SUPSTITUCIJE …

$$T(n) \leq \left(c \cdot \left\lfloor \frac{n}{2} \right\rfloor - d\right) + \left(c \cdot \left\lceil \frac{n}{2} \right\rceil - d\right) + 1 = cn - 2d + 1 \leq cn - d$$

- Ovo je tačno sve dok je $d \geq 1$
- Naravno, mora da se odabere dovoljno veliko c
- Teže je dokazati slabiju gornju granicu!
- Prilikom dokazivanja, mora da se vodi računa o dokazivanju
- Npr. „dokazivanje“ $T(n) = O(n)$ pretpostavljanjem da je $T(n) \leq cn$ i onda dokazati:

$$T(n) \leq 2\left(c \cdot \left\lfloor \frac{n}{2} \right\rfloor\right) + n \leq cn + n = O(n)$$ ← **Greška!**

- Da bi se dokazalo $T(n) = O(n)$ mora se dokazati $T(n) \leq cn$

… METODA SUPSTITUCIJE

- Ponekad mala suptilna manipulacija algebarskom formulom može transformisati rekurentnu jednačinu u nama već poznat oblik
- Npr. $\boldsymbol{T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \log n}$
- se može transformisati promenom promenljivih (ne brinemo o zaokruživanju promenljivih), tako da uvedemo promenljivu $\boldsymbol{m = \log n}$ onda se rekurentna jednačina može transformisati u oblik:

  $$\boldsymbol{T(2^m) = 2T(\lfloor 2^{m/2} \rfloor) + m}$$

- Ako sada preimenujemo: $\boldsymbol{S(m) = T(2^m)}$ dobijamo:

  $$\boldsymbol{S(m) = 2S(\lfloor m/2 \rfloor) + m}$$

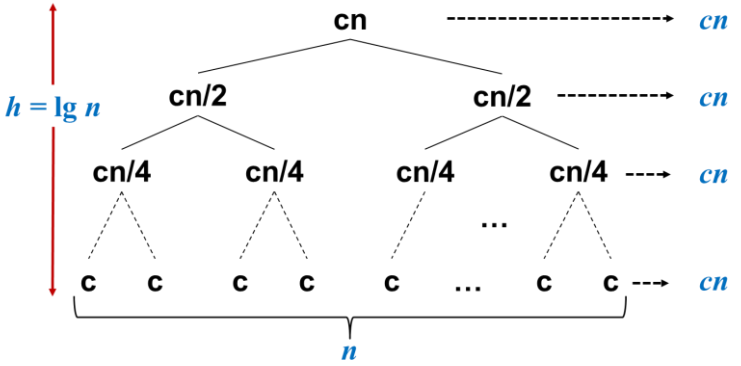
- Ovo sad daleko više liči na prethodnu rekurentnu jednačinu, te se može pretpostaviti rešenje: $\boldsymbol{S(m) = O(m \log m)}$
- Kada se zameni $\boldsymbol{S(m)}$ sa $\boldsymbol{T(n)}$ dobija se:

  $$\boldsymbol{T(n) = T(2^m) = S(m) = O(m \cdot \log m) = O(\log n \cdot \log(\log n))}$$

REKURENTNE JEDNAČINE – METODA STABLA REKURZIJE

METODA STABLA REKURZIJE …

- Sa metodom supstitucije, problem je kako doći do dobre pretpostavke
- Jedan način je da se nacrta stablo rekurzije kojim se iscrtavaju koraci izvršavanja algoritma i vizuelno prikazuje kako se deli algoritam na osnovu čega može da se odredi vreme (cena) izvršavanja algoritma
- Primer stabla rekurzije smo imali u algoritmu za MERGE sortiranje
- Svaki čvor stabla predstavlja vreme izvršavanja jednog potproblema negde u rekurzivnom algoritmu



- Vreme izvršavanja svih čvorova na jednom nivou se sumira kako bi se dobila cena celog nivoa
- A zatim se sumiranjem svih nivoa dolazi do vremena izvršavanja čitavog algoritma

… METODA STABLA REKURZIJE …

- Metoda stabla rekurzije je dobra da se dođe do pretpostavke o tome koja je granična funkcija vremena izvršavanja algoritma, pretpostavka se dalje dokazuje metodom supstitucije
- Ako se kombinuju metoda supstitucije i metoda stabla rekurzije, onda je dozvoljeno malo netačnosti (greške) u proceni vremena izvršavanja, jer će se ona kasnije verifikovati
- Ako se stablo crta vrlo pažljivo, sa velikim naporom kako bi se greškice izbegle, ono se može upotrebiti za određivanje rešenja rekurentne jednačine
- Primer: odrediti rešenje rekurentne jednačine:

$$\boldsymbol{T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)}$$

… METODA STABLA REKURZIJE …

$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$$

- Pošto zaokruživanje ne utiče na pronalaženje rešenja rekurentne jednačine, može se pretpostaviti sledeće (traljavost – uvodimo grešku):

$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + cn^2 \text{ za } c > 0$$

- Pretpostaviti da je n umnožak 4 . **Zašto?**
- Zato što će svi potproblemi biti celi brojevi
- Zatim polako krećemo deljenje problema na potprobleme i crtanje stabla rekurzije
- Izvršavanje na vrhu je cn^2 . **Zašto?**
- Neću da vam kažem! Razmislite sami…

… METODA STABLA REKURZIJE …

$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + cn^2$$

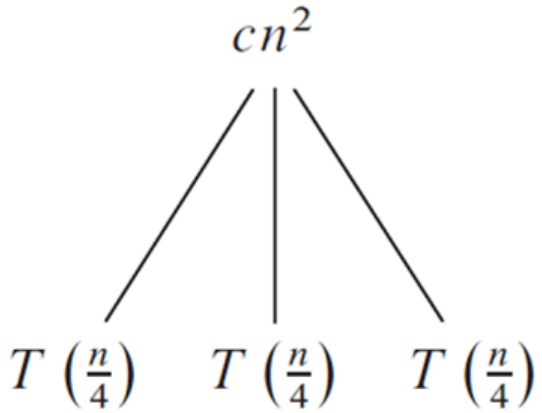
- Stablo rekurzije:

$$T(n)$$

… METODA STABLA REKURZIJE …

$$T(n) = 3T\left(\left\lfloor\frac{n}{4}\right\rfloor\right) + cn^2$$

- Stablo rekurzije:

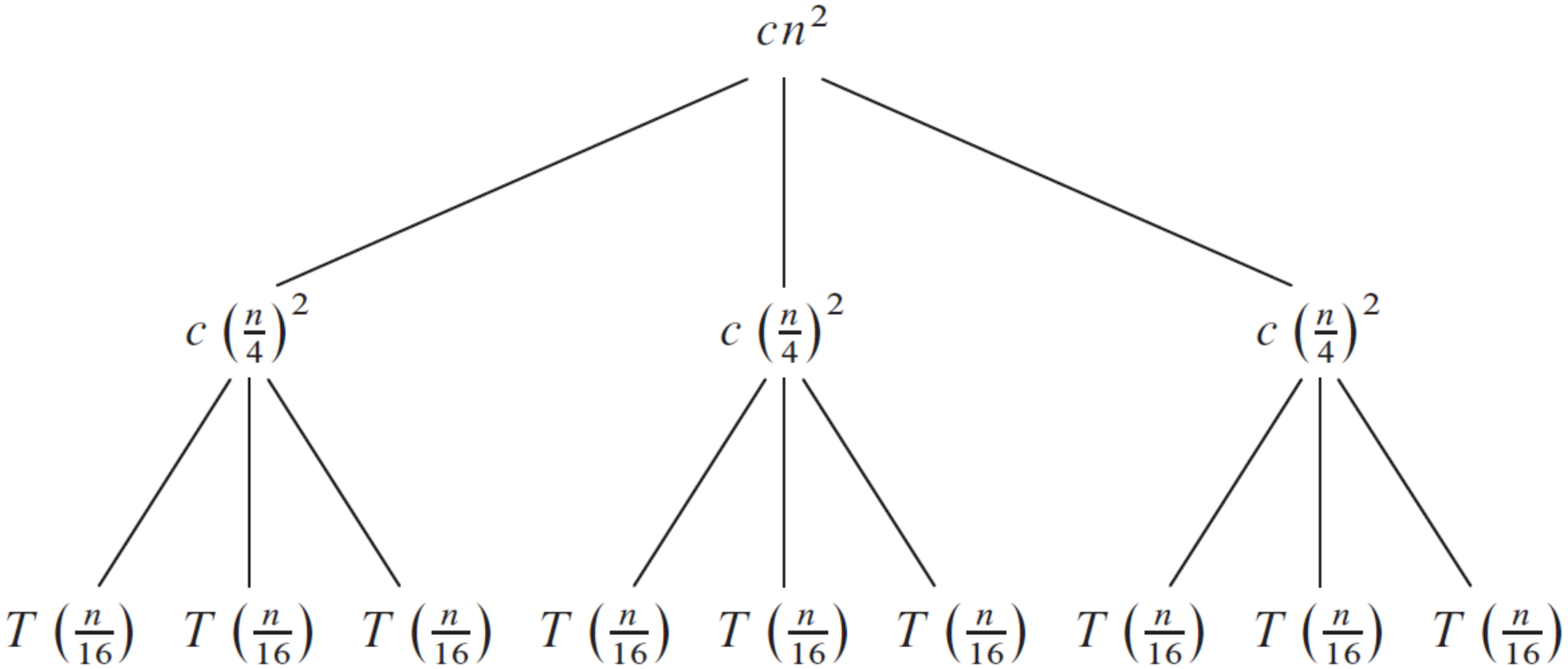


- Svaki problem delimo na 3 potproblema čije je vreme izvršavanja $T(n/4)$
- Vreme izvršavanja svakog čvora ispod korena je zapravo $c(\frac{n}{4})^2$, dalje nastavljamo da razgrađujemo problem ….

… METODA STABLA REKURZIJE …

$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + cn^2$$

- Stablo rekurzije:

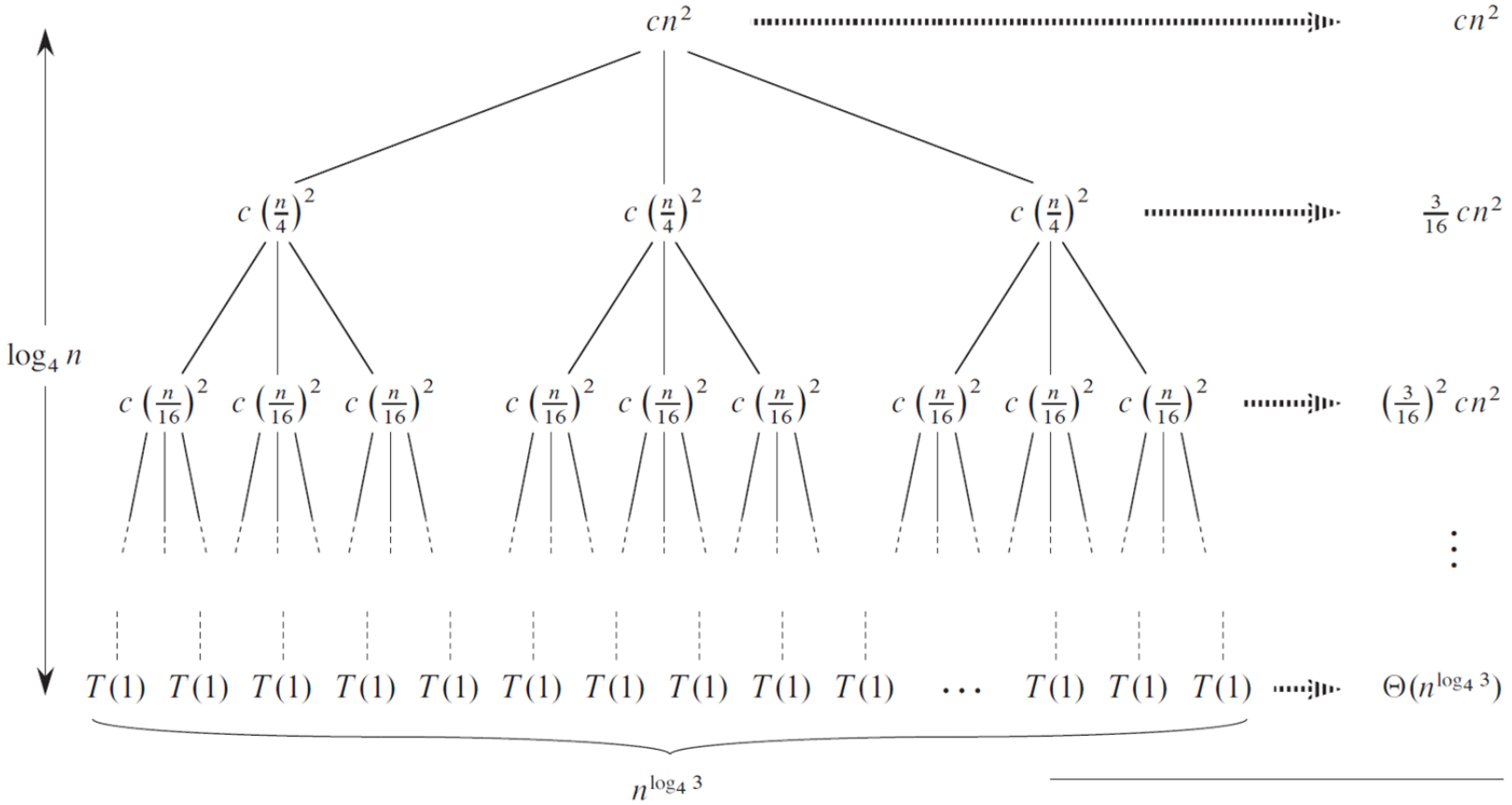


- Pošto se problem smanjuje za faktor 4 u svakoj iteraciji, moramo doći do graničnog slučaja izvršavanja

… METODA STABLA REKURZIJE …

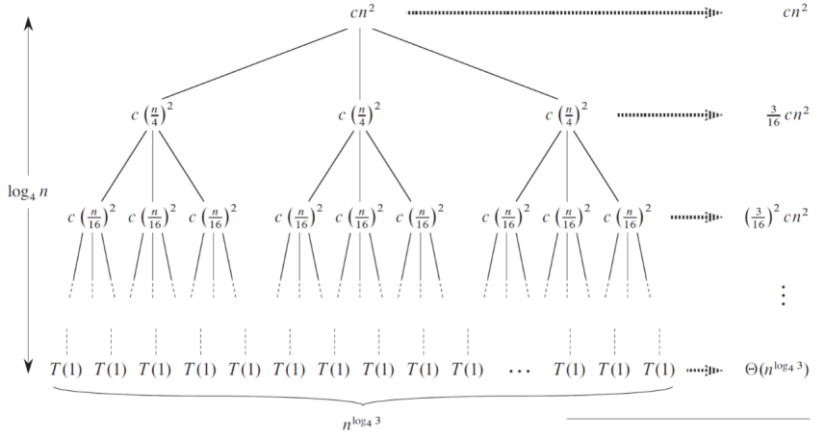
$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + cn^2$$

- Stablo rekurzije:



… METODA STABLA REKURZIJE …

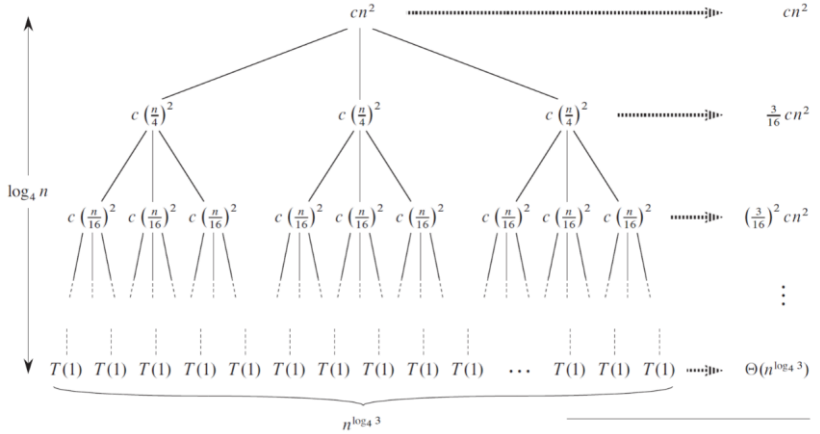
$$T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + cn^2$$



- Veličina potproblema koji nastaje dekompozicijom problema na nivou i je $n/4^i$, to znači do graničnog slučaja stižemo kada je $n = 1$, tj. za $n/4^i = 1$, ili ekvivalentno kada je $i = \log_4 n$
- Tako dolazimo do toga da stablo rekurzije ima $\log_4 n + 1$ nivoa,
$$(0, 1, 2, \ldots, \log_4 n)$$
- U sledećem koraku treba odrediti koliko košta svaki čvor
  - Na i-tom nivou ima 3^i čvora
  - Problem se smanjuje za faktor 4 krećući se od korena, što znači da svaki čvor na dubini i, za $i = 0, 1, 2, \ldots, \log_4 n - 1$, vreme izvršavanja je $c(n/4^i)^2$
  - Za sve čvorove jednog nivoa i dobija se da cena čitavog nivoa:

$$3^i c(n/4^i)^2 = (3/16)^i cn^2$$

… METODA STABLA REKURZIJE …

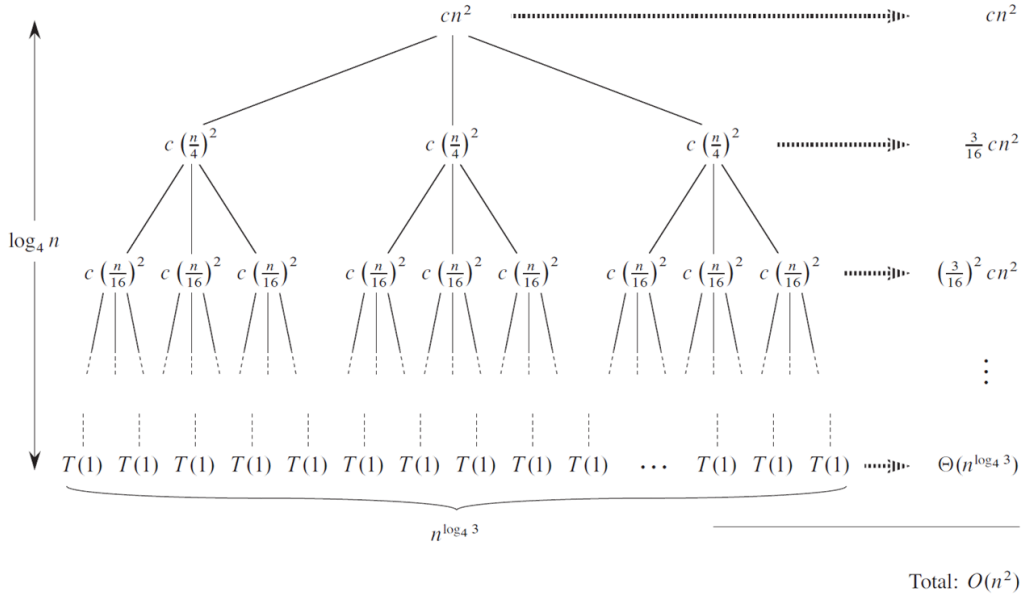


- U sledećem koraku treba odrediti koliko košta svaki čvor
  - Na i-tom nivou ima 3^i čvora
  - Svaki problem se smanjuje za faktor 4 krećući se od korena, što znači da svaki čvor na dubini i, za $i = 0, 1, 2, \ldots, \log_4 n - 1$, vreme izvršavanja je $c(n/4^i)^2$
  - Za sve čvorove jednog nivoa i dobija se da cena čitavog niova:
    $$3^i c(n/4^i)^2 = (3/16)^i cn^2$$
  - Na najnižem nivou stabla rekurzije, $\log_4 n$, ima $3^{\log_4 n} = n^{\log_4 3}$ čvorova koji se svaki izvršava $T(1)$, za ukupnu cenu izvršavanja: $n^{\log_4 3} \cdot T(1)$, što je zapravo $\Theta(n^{\log_4 3})$. Zašto? Zato što je $T(1)$ konstantna vrednost
- Na osnovu stabla rekurzije može se doći do cene (vremena) izvršavanja celekupnog stabla rekurzije (samim tim i rešenje rekurentne jednačine).
- Sabira se cena svih nivoa i svih čvorova

… METODA STABLA REKURZIJE …

- Sabira se cena svih nivoa i čvorova

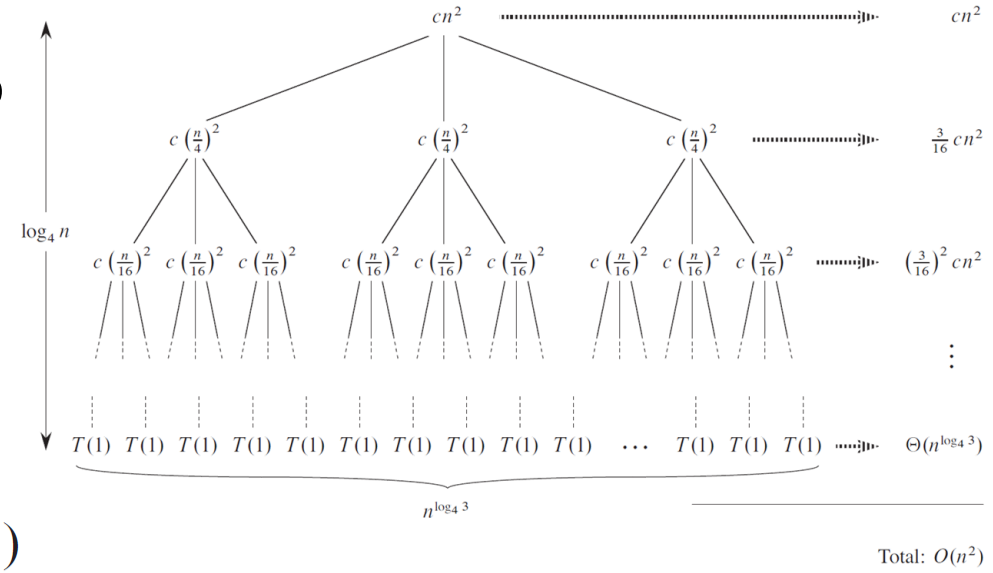
$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1}$$



$$\begin{aligned} T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\ &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{(3/16)^{\log_4 n} - 1}{(3/16) - 1} cn^2 + \Theta(n^{\log_4 3}) \end{aligned}$$

… METODA STABLA REKURZIJE …

- Koristićemo činjenicu da imamo smanjivanje u geometrijskoj progresiji (*infinite decreasing geometric series*) i odgovarajuće formule



$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2) \,.
\end{aligned}
$$

$$
\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}
$$

… METODA STABLA REKURZIJE …

- Tako smo došli do pretpostavke da je za rekurentnu jednačinu $\boldsymbol{T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)}$ rešenje $\boldsymbol{T(n) = O(n^2)}$
- $\boldsymbol{O(n^2)}$ je gornja granična funkcija, ali da li je ona i najbolja ocena?
- Jeste! Zašto?
- Kako prvi nivo doprinosi sa $\boldsymbol{\Theta(n^2)}$, a kako u najboljem slučaju ne idemo dalje od 1. nivoa, onda je i $\boldsymbol{\Omega(n^2)}$
- Metodom supstitucije se može dokazati validnost rešenja dobijenog metodom stabla rekurzije
- Kako?
- Ako je $\boldsymbol{T(n) = O(n^2)}$ gornja granica jednačine $\boldsymbol{T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)}$ onda treba dokazati da je $\boldsymbol{T(n) \leq dn^2}$ za neku konstantu $\boldsymbol{d > 0}$

… METODA STABLA REKURZIJE

- Ako je $\boldsymbol{T(n) = O(n^2)}$ gornja granica jednačine $\boldsymbol{T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)}$ onda treba dokazati da je $\boldsymbol{T(n) \leq dn^2}$ za neku konstantu $\boldsymbol{d > 0}$
- Za $\boldsymbol{c > 0}$

$$\begin{aligned} \boldsymbol{T(n)} &\boldsymbol{\leq 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + cn^2} \\ &\boldsymbol{\leq 3d\left\lfloor \frac{n}{4} \right\rfloor^2 + cn^2} \\ &\boldsymbol{\leq 3d(\frac{n}{4})^2 + cn^2} \\ &\boldsymbol{= \frac{3}{16}dn^2 + cn^2} \\ &\boldsymbol{\leq dn^2} \end{aligned}$$

- poslednja tvrdnja važi $\boldsymbol{d \geq \frac{16}{3}c}$

REKURENTNE JEDNAČINE – MASTER METODA

MASTER METODA …

- Predstavlja opštu metodu rešavanja rekurentne jednačine tipa

  $$\boldsymbol{T(n) = aT\left(\frac{n}{b}\right) + f(n)}\ ,$$

  gde su $\boldsymbol{a \geq 0}$ i $\boldsymbol{b > 0}$ konstante, a $\boldsymbol{f(n)}$ asimptotski pozitivna funkcija
- Upotreba master metode zahteva pamćenje tri slučaja, ali se njihovom primenom može rešiti većina rekurentnih jednačina, vrlo jednostavno, često i bez olovke i papira (ako je verovati autorima ItA knjige)
- Opisuje vreme izvršavanja algoritma koji deli problem veličine $\boldsymbol{n}$ na $\boldsymbol{a}$ potproblema veličine $\boldsymbol{n/b}$
- $\boldsymbol{a}$ potproblema se posle rekurzivno rešava, svaki u $\boldsymbol{T\left(\frac{n}{b}\right)}$ vreme
- Funkcija $\boldsymbol{f(n)}$ predstavlja vreme deljenja na potprobleme i kombinovanje rešenja potproblema

… MASTER METODA …

- Striktno govoreći $\frac{n}{b}$ ne mora biti ceo broj, ali zamena $\frac{n}{b}$ sa $\left\lfloor \frac{n}{b} \right\rfloor$ ili $\left\lceil \frac{n}{b} \right\rceil$ neće puno uticati na asimptotsko ponašanje rekurentne jednačine

- Suština je u poređenju $f(n)$ i $n^{\log_b a}$

… MASTER METODA …

- Teorema:

  Neka su $\boldsymbol{a \geq 1}$ i $\boldsymbol{b > 1}$ konstante, neka je $\boldsymbol{f(n)}$ funkcija, i neka je $\boldsymbol{T(n)}$ definisana nad nenegativnim celim brojevima određenim rekurentnom jednačinom: $\boldsymbol{T(n) = aT\left(\frac{n}{b}\right) + f(n)}$ gde se $\frac{n}{b}$ može interpretirati kao $\left\lfloor \frac{n}{b} \right\rfloor$ ili $\left\lceil \frac{n}{b} \right\rceil$. Onda $\boldsymbol{T(n)}$ ima sledeće asimptotske granice

  1. Ako je $\boldsymbol{f(n) = O(n^{\log_b a - \epsilon})}$ za neku konstantu $\boldsymbol{\epsilon > 0}$, onda je
     $$\boldsymbol{T(n) = \Theta(n^{\log_b a})}$$
  2. Ako je $\boldsymbol{f(n) = \Theta(n^{\log_b a})}$, onda je $\boldsymbol{T(n) = \Theta(n^{\log_b a} \log n)}$
  3. Ako je $\boldsymbol{f(n) = \Omega(n^{\log_b a + \epsilon})}$ za neku konstantu $\boldsymbol{\epsilon > 0}$, i ako je $\boldsymbol{a \cdot f\left(\frac{n}{b}\right) \leq c \cdot a \cdot f(n)}$ za neku konstantu $\boldsymbol{c < 1}$ i dovoljno veliko $\boldsymbol{n}$, onda je $\boldsymbol{T(n) = \Theta(f(n))}$

… MASTER METODA …

- U svakom od navedenih slučajeva, poredimo $\boldsymbol{f(n)}$ sa $\boldsymbol{n^{\log_b a}}$
- Veća od funkcija određuje asimptotsku granicu, slučajevi 1 i 3,
- Dok u slučaju 2, ako su funkcije jednake, onda množimo sa logaritamskim faktorom, tako da je rešenje:

$$\boldsymbol{T(n) = \Theta\left(n^{\log_b a} \log n\right) = \Theta(f(n) \log n)}$$

- Treba biti svestan nekih stvari:
  - Ne samo da $\boldsymbol{f(n)}$ mora biti manje od $\boldsymbol{n^{\log_b a}}$, nego mora biti polinomijalno manje, tj. asimptotski manje za neki faktor $\boldsymbol{n^{\epsilon}}$, pri čemu je konstanta $\boldsymbol{\epsilon > 0}$
  - U trećem slučaju $\boldsymbol{f(n)}$ mora biti polinomijalno veća od $\boldsymbol{n^{\log_b a}}$, ali mora biti zadovoljen i uslov $\boldsymbol{a \cdot f\left(\frac{n}{b}\right) \leq c \cdot a \cdot f(n)}$ (ovo zadovoljava većina polinomijalno ograničenih funkcija koje srećemo)

… MASTER METODA …

- Postoje situacije koje master metoda ne pokriva:
  - Slučaj između 1 i 2, kada je $\boldsymbol{f(n)}$ manja od $\boldsymbol{n^{\log_b a}}$ ali ne i polinomijalno
  - Slučaj između 2 i 3, kada je $\boldsymbol{f(n)}$ veće od $\boldsymbol{n^{\log_b a}}$ ali ne i polinomijalno
  - Slučaj 3 kada uslov u slučaju nije zadovoljen
- U tim slučajevima master metoda ne može rešiti problem rekurentne jednačine
- Master metoda se koristi tako što se jednostavno vidi da li se za datu rekurentnu jednačinu može primeniti neki od slučajeva master teoreme

… MASTER METODA …

- Primeri:

$$T(n) = 9T\left(\frac{n}{3}\right) + n$$

  - $a = 9$ i $b = 3$, $f(n) = n$ tako da imamo $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$
  - Kako je $f(n) = O\left(n^{\log_3 9 - \epsilon}\right)$ gde je $\epsilon = 1$, možemo primeniti slučaj 1 master teoreme, te je $T(n) = \Theta(n^2)$

$$T(n) = T\left(\frac{2n}{3}\right) + 1$$

  - $a = 1$ i $b = 3/2$, $f(n) = 1$ tako da imamo $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$
  - Važi 2 slučaj master teoreme, $f(n) = \Theta\left(n^{\log_b a}\right) = \Theta(1)$, te je rešenje $T(n) = \Theta(\log n)$

… MASTER METODA …

$$T(n) = 3T\left(\frac{n}{4}\right) + n\log n$$

- $a = 3$ i $b = 4$, $f(n) = n\log n$, tako da imamo $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$
- Kako je $f(n) = \Omega\left(n^{\log_4 3+\epsilon}\right)$ gde je $\epsilon \approx 0.2$, možemo primeniti slučaj 3 master teoreme, ako dokažemo da važi uslov $a \cdot f\left(\frac{n}{b}\right) \leq a \cdot f(n)$, $af\left(\frac{n}{b}\right) = 3\left(\frac{n}{4}\right)\log\left(\frac{n}{4}\right) \leq \frac{3}{4}n\log n = cf(n)$, za $c = \frac{3}{4}$
- te je $T(n) = \Theta(n\log n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

- $a = 2$ i $b = 2$, $f(n) = \Theta(n)$ tako da imamo $n^{\log_b a} = n^{\log_2 2} = n$
- Važi 2 slučaj master teoreme, $f(n) = \Theta(n)$, te je rešenje

$$T(n) = \Theta(n \cdot \log n)$$

… MASTER METODA …

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

- $a = 8$ i $b = 2$, $f(n) = \Theta(n^2)$, tako da imamo $n^{\log_b a} = n^{\log_2 8} = n^3$
- Kako je $f(n) = O\left(n^{3-\epsilon}\right)$ gde je $\epsilon = 1$, možemo primeniti slučaj 1 master teoreme, te je $T(n) = \Theta(n^3)$
- n^3 je polinomijalno veći od $f(n)$

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$

- $a = 7$ i $b = 2$, $f(n) = \Theta(n^2)$ tako da imamo $n^{\log_b a} = n^{\log_2 7}$
- Kako je $f(n) = O\left(n^{\log 7-\epsilon}\right)$ gde je $\epsilon = 0.8$, možemo primeniti slučaj 1 master teoreme, te je $T(n) = \Theta(n^{\log 7})$

… MASTER METODA

$$T(n) = 2T\left(\frac{n}{2}\right) + n \cdot \log n$$

- $a = 2$ i $b = 2$, $f(n) = n \log n$, tako da imamo $n^{\log_b a} = n^{\log_2 2} = n$
- Koji slučaj master teoreme ovde važi?
- Slučaj 3, jer je $n \cdot \log n$ asimptotski veće od n
- Da li je dovoljno veliko?
- Nije polinomijalno veće!
- Odnos $\frac{f(n)}{n^{\log_b a}} = \frac{n \log n}{n} = \log n$, nije veće od n^{ϵ} za bilo koju konstantu $\epsilon > 0$
- Rekurenta jednačina pada između slučaja 2 i 3

- Koga zanima, dokaz master teoreme pronaći u knjizi ItA

DIVIDE-AND-CONQUER ALGORITMI

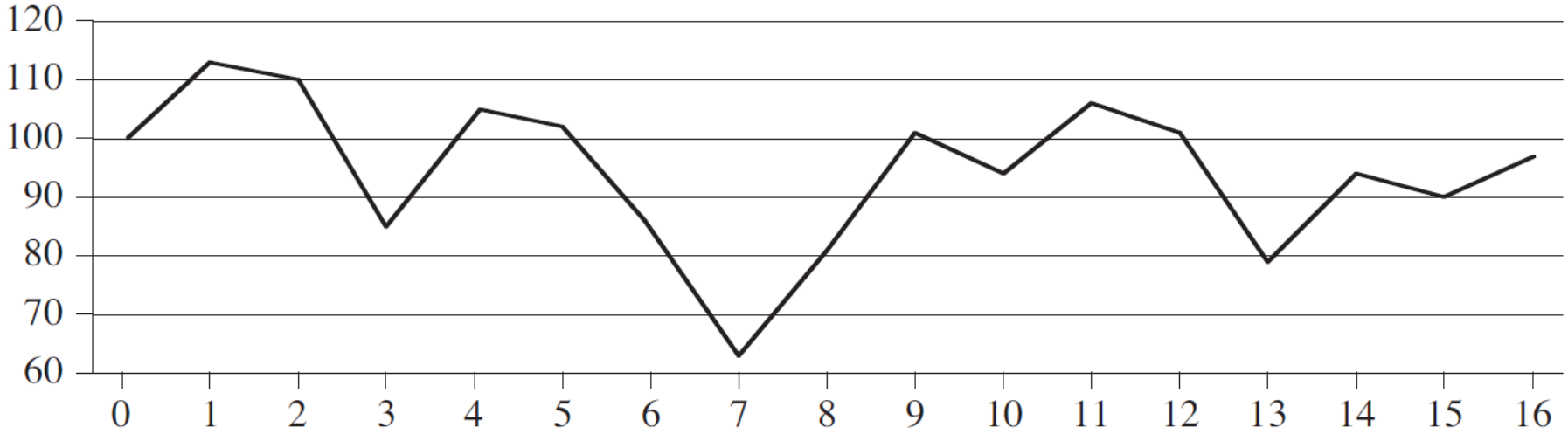
PREGLED ALGORITAMA

- Primeri primene zavadi-pa-vladaj algoritama:
  - Maksimalni podniz
  - Pretraživanje binarnog stabla
  - Stepenovanje brojeva
  - Fibonačijevi brojevi
  - Množenje matrica
  - Strassenov algoritma
  - VLSI raspoređivanje binarnog stabla u mrežu
  - Nalaženje dve najbliže tačke

PROBLEM NAJVEĆEG PODNIZA

PROBLEM NAJVEĆEG PODNIZA …

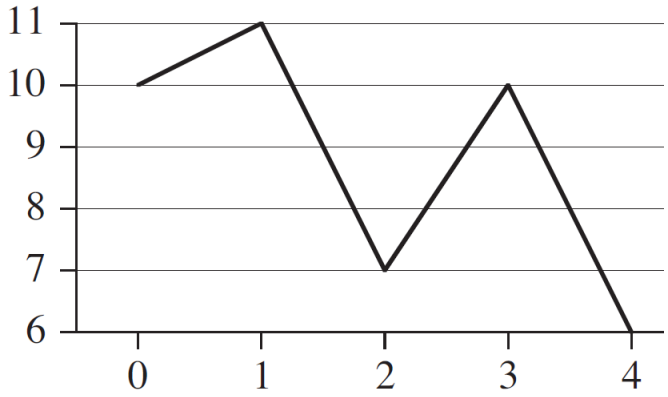
- Pronaći najveću razliku između dva člana niza idući od početka ka kraju niza bez vraćanja (primer kupovine prodaje na berzi)



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

… PROBLEM NAJVEĆEG PODNIZA …

- Kakav bi bio direktni pristup?
- Nađi najmanju vrednost, pa kreni u desno i traži najveću razliku
- Nađi najveću vrednost, pa kreni u levo i traži najveću razliku
- Da li je to najbolje rešenje? Da li je to uopšte rešenje?



Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	−4	3	−4

- Očigledno nije uvek rešenje!

… PROBLEM NAJVEĆEG PODNIZA …

- Šta bi bio brute force?
- Probaj svaki par, pa uzmi onaj sa najvećom razlikom!
- Niz on ***n*** elemenata ima $\binom{n}{2}$ takvih parova
- Koliko nam ***for*** petlji treba za ovo?
- **DVE!**
- Koliko je asimptotsko vreme izvršavanja?
- Za $\binom{n}{2}$ je $\Theta(n^2)$
- U najboljem slučaju nam treba da evoluiramo svaki par u konstantom vremenu $\Omega(n^2)$
- Može li bolje?

… PROBLEM NAJVEĆEG PODNIZA …

- Bolji način je putem rekurzije
- Prvo treba modifikovati problem i iskazati ga kao niz razlika između ***i***-tog i ***i-1*** – og
- Na šta se sada svodi problem?
- Sada se problem svodi na pronalaženje nepraznog, najdužeg podniza čija suma elemenata ima najveću vrednost

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

podniz čija suma elemenata je najveća (43)

- Na prvi pogled ova transformacija nije pomogla (ne ako se koristi brutalni pristup)
- I dalje mora da se proveri $\binom{n-1}{2}$ parova, što i dalje daje $\Theta(n^2)$
- Zašto?

… PROBLEM NAJVEĆEG PODNIZA …

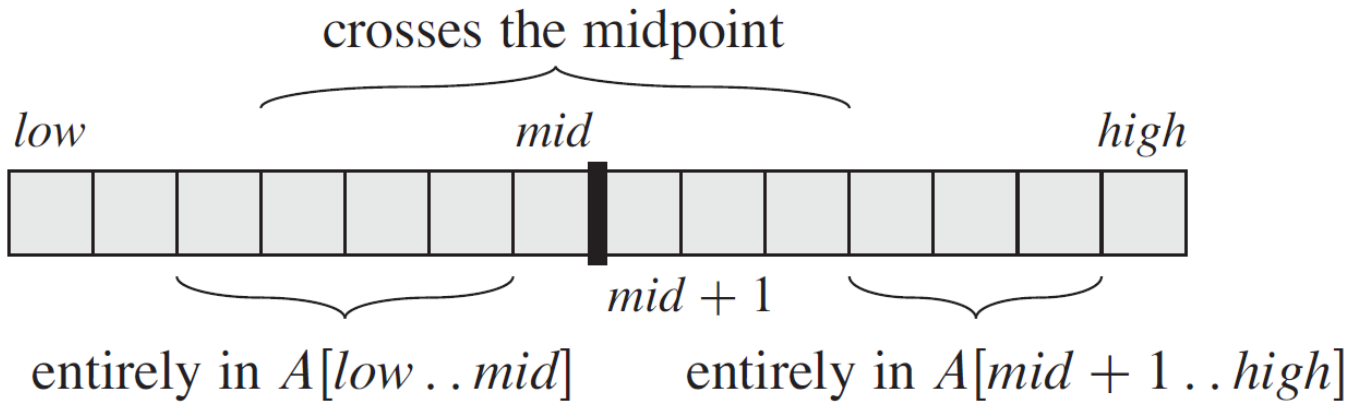
- Zašto je traženje najvećeg podniza takav izazov?

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

- Da su sve vrednosti pozitivne, da li bi bilo izazovno?
- Hajde da razmislimo kako ovo bože efikasnije?
- Koju tehniku proučavamo?
- Podeli-pa-zavladaj!
- Kako se može podeliti niz?
- Da, po sredini!
- Koji je problem?
- Gde se nalazi rešenje?

… PROBLEM NAJVEĆEG PODNIZA …

- Ako niz $A[low \ldots high]$ podelimo na dva podniza, $A[low \ldots mid]$ i $A[mid + 1 \ldots high]$
- Gde se nalazi najveći podniz?
- U jednoj od tri moguće kombinacije:
  - Ceo se nalazi u podnizu $A[low \ldots mid]$, $low \leq i \leq j \leq mid$
  - Ceo se nalazi u podnizu $A[mid + 1 \ldots high]$, $mid < i \leq j \leq high$
  - Prelazi preko sredine, malo u podnizu , $low \leq i \leq mid < j \leq high$

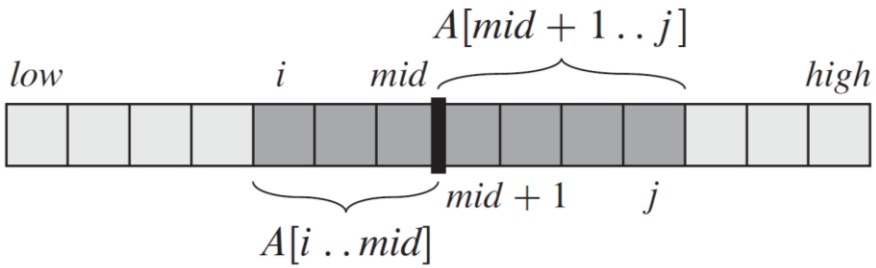


… PROBLEM NAJVEĆEG PODNIZA …

- Taj podniz treba bude najveći u odnosu na bilo koji drugi podniz bez obzira da li je u donjem, gornjem delu ili sredini
- Najveći podniz u $\boldsymbol{A[low \dots mid]}$ ili $\boldsymbol{A[mid + 1 \dots high]}$ je lako naći, primenom rekurzije
- Najveći podniz u preseku nizova $\boldsymbol{A[low \dots mid]}$ i $\boldsymbol{A[mid + 1 \dots high]}$ može se lako pronaći u linearnom vremenu
- Pronalaženje najvećeg niza u preseku nizova nije manji potproblem originalnog problema
- Zašto?
- Jer postoji ograničenje da podniz mora da prelazi polovinu

… PROBLEM NAJVEĆEG PODNIZA …

- Najveći niz u preseku nizova uvek ima dva dela, jedan iz $\boldsymbol{A[low \ldots mid]}$ i jedan iz $\boldsymbol{A[mid + 1 \ldots high]}$, to su podnizovi $\boldsymbol{A[i \ldots mid]}$ i $\boldsymbol{A[mid + 1 \ldots j]}$, gde su $\boldsymbol{low \leq i \leq mid < j \leq high}$



- U suštini samo treba da se pronađu ta dva podniza i da se oni kombinuju u jedan

… PROBLEM NAJVEĆEG PODNIZA …


```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4       sum = sum + A[i]
 5       if sum > left-sum
 6            left-sum = sum
 7            max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11       sum = sum + A[j]
12       if sum > right-sum
13            right-sum = sum
14            max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

… PROBLEM NAJVEĆEG PODNIZA …

- Koje je vreme izvršavanja ovog algoritma?
- Ako niz $A[low \ldots high]$ ima n elemenata, $n = high - low + 1$
- Vreme da se izvrši svaka ***for*** petlja je ***Θ(1)***, tako da samo treba prebrojati broj izvršavanja ***for*** petlji
- Od 3 – 7 koraka ima $mid - low + 1$ iteracija, a od 10 – 14 $high - mid$ koraka

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4       sum = sum + A[i]
 5       if sum > left-sum
 6            left-sum = sum
 7            max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11       sum = sum + A[j]
12       if sum > right-sum
13            right-sum = sum
14            max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

$$(mid - low + 1) + (high - mid) = high - low + 1 = n$$

… PROBLEM NAJVEĆEG PODNIZA …


```
FIND-MAXIMUM-SUBARRAY(A, low, high)
 1  if high == low
 2       return (low, high, A[low])          // base case: only one element
 3  else mid = ⌊(low + high)/2⌋
 4       (left-low, left-high, left-sum) =
              FIND-MAXIMUM-SUBARRAY(A, low, mid)
 5       (right-low, right-high, right-sum) =
              FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
 6       (cross-low, cross-high, cross-sum) =
              FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 7       if left-sum ≥ right-sum and left-sum ≥ cross-sum
 8            return (left-low, left-high, left-sum)
 9       elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10            return (right-low, right-high, right-sum)
11       else return (cross-low, cross-high, cross-sum)
```

… PROBLEM NAJVEĆEG PODNIZA …

- Vreme izvršavanja ovog algoritma
- Osnovni slučaj, kada je $\boldsymbol{n = 1}$ je linija 1, $\boldsymbol{T(n) = \Theta(1)}$
- Kada je $\boldsymbol{n > 1}$ svaki problem se deli na 2 potproblema koji je veličine $\boldsymbol{n/2}$, te treba $\boldsymbol{T(n/2)}$ vremena da bi se rešio
- Znači traženje najvećeg podniza na dve strane traje $\boldsymbol{2T(n/2)}$
- U svakoj iteraciji postoji i vreme potrebno da se odredi presečni niz koje smo pokazali da je $\boldsymbol{\Theta(n)}$
- I na kraju vreme potrebno da se to sve kombinuje, koje je konstantno i linearno $\boldsymbol{\Theta(1)}$
- Kada se sve to sabere dobija se:

$$\boldsymbol{T(n) = \Theta(1) + 2T\left(\frac{n}{2}\right) + \Theta(n) + \Theta(1) = 2T\left(\frac{n}{2}\right) + \Theta(n)}$$

- Koje je rešenje za ovu rekurentnu jednačinu?

… PROBLEM NAJVEĆEG PODNIZA

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

- $a = 2$ i $b = 2$, $f(n) = \Theta(n)$ tako da imamo $n^{\log_b a} = n^{\log_2 2} = n$
- Važi 2 slučaj master teoreme, $f(n) = \Theta(n)$, te je rešenje $T(n) = \Theta(n \cdot \log n)$

- Da li je rešenje bolje od brutalnog pristupa?

STEPENOVANJE BROJA

STEPENOVANJE BROJA …

- Koji je problem?

  $\boldsymbol{a^n}$, gde je $\boldsymbol{n} \in \boldsymbol{N}$

- Naivni pristup?

  $\boldsymbol{a} \cdot \boldsymbol{a} \cdot \cdots \cdot \boldsymbol{a}$ i tako ***n*** puta, tj. jedna ***for*** petlja

- Koliko je vreme izvršavanja?

  $\Theta(\boldsymbol{n})$

- Bolje rešenje?

- Rekurzija

… STEPENOVANJE BROJA

- Kako da podelimo problem?

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{, za parno } n \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{, za neparno } n \end{cases}$$

- Koliko je vreme izvršavanja?

$T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$, drugi slučaj master teoreme

$T(n) = \Theta(\log n)$

- Zašto?

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$$

  - $a = 1$ i $b = 2$, $f(n) = \Theta(1)$ tako da imamo $n^{\log_b a} = n^{\log_2 1} = 1$
  - Važi 2 slučaj master teoreme, $f(n) = \Theta(1)$, te je rešenje $T(n) = \Theta(\log n)$

MNOŽENJE MATRICA

MNOŽENJE MATRICA …

- Množimo dve matrice i rezultat pišemo u treću
- Ulaz: $A[a_{ij}]$ i $B[b_{ij}]$, za $i, j = 1, 2, \dots, n$
- Ulaz: $C[c_{ij}]$, za $i, j = 1, 2, \dots, n$

$$\begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ij} \cdot b_{ij}$$

… MNOŽENJE MATRICA …

- Kako izgleda standardni algoritam?

```
SQUARE-MATRIX-MULTIPLY(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          c_ij = 0
6          for k = 1 to n
7              c_ij = c_ij + a_ik · b_kj
8  return C
```

- Koje je vreme izvršavanja?
- $\Theta(n^3)$
- Zašto?

… MNOŽENJE MATRICA …

- Može li bolje?
- Zavadi-pa-vladaj?
- Može ovako?
- Matrica ***n*×*n*** se može podeliti na ***2*×*2*** matrice od ***(n/2)*×*(n/2)*** podmatrica

$$\left[\begin{array}{c|c} r & s \\ \hline t & u \end{array}\right] = \left[\begin{array}{c|c} a & b \\ \hline c & d \end{array}\right] \cdot \left[\begin{array}{c|c} e & f \\ \hline g & h \end{array}\right]$$

$$C = A \cdot B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\}$$

8 množenja ***(n/2)*×*(n/2)*** podmatrica **(rekurzija)**

4 sabiranja ***(n/2)*×*(n/2)*** podmatrica

… MNOŽENJE MATRICA …

SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)

1 $n = A.rows$
2 let C be a new $n \times n$ matrix
3 **if** $n == 1$
4 $\quad c_{11} = a_{11} \cdot b_{11}$
5 **else** partition A, B, and C as in equations (4.9)
6 $\quad C_{11} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{11}, B_{11})
$\quad\quad +$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{12}, B_{21})
7 $\quad C_{12} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{11}, B_{12})
$\quad\quad +$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{12}, B_{22})
8 $\quad C_{21} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{21}, B_{11})
$\quad\quad +$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{22}, B_{21})
9 $\quad C_{22} =$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{21}, B_{12})
$\quad\quad +$ SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_{22}, B_{22})
10 **return** C

… MNOŽENJE MATRICA

- Koliko je vreme izvršavanja?

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

  - $a = 8$ i $b = 2$, $f(n) = n^2$ tako da imamo $n^{\log_b a} = n^{\log_2 8} = n^3$
  - Kako je $f(n) = O(n^2)$, možemo primeniti slučaj 1 master teoreme, te je $T(n) = \Theta(n^3)$

- I nije neko poboljšanje …

STRASSENOV ALGORITAM

STRASSEN ALGORITAM …

- Modifikacija rekurzivnog pristupa
- Umesto da se koristi 8 rekurzivnih množenja nad ***(n/2)×(n/2)*** matrica, koristi se 7 rekurzivnih množenja
- Izbacivanje jednog množenja rezultuje u povećanju broja sabiranja i dodavanju par oduzimanja
  - Ipak, to su sve samo dodatne konstante operacije
- Kako algoritam funkcioniše?

$$\boldsymbol{C} = \begin{pmatrix} \boldsymbol{C_{11}} & \boldsymbol{C_{12}} \\ \boldsymbol{C_{21}} & \boldsymbol{C_{22}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{P_5 + P_4 - P_2 + P_6} & \boldsymbol{P_1 + P_2} \\ \boldsymbol{P_3 + P_4} & \boldsymbol{P_5 + P_1 - P_3 - P_7} \end{pmatrix}$$

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A_{11}} & \boldsymbol{A_{12}} \\ \boldsymbol{A_{21}} & \boldsymbol{A_{22}} \end{pmatrix} \boldsymbol{B} = \begin{pmatrix} \boldsymbol{B_{11}} & \boldsymbol{B_{12}} \\ \boldsymbol{B_{21}} & \boldsymbol{B_{22}} \end{pmatrix} \boldsymbol{C = A \cdot B}$$

… STRASSEN ALGORITAM …

- Drugi korak očekuje pravljenje 10 novih ***(n/2)×(n/2)*** matrica nekom kombinacijom (sabiranjem/oduzimanjem) ***(n/2)×(n/2)*** podmatrica dobijenih deljenjem inicijalnih matrica u prvom koraku
  - Pravljenje ovih 10 matrica traje $\Theta(\boldsymbol{n^2})$

$$\boldsymbol{S_1 = B_{12} - B_{22}} \qquad \boldsymbol{S_6 = B_{11} + B_{22}}$$

$$\boldsymbol{S_2 = A_{11} + A_{12}} \qquad \boldsymbol{S_7 = A_{12} - A_{22}}$$

$$\boldsymbol{S_3 = A_{21} + A_{22}} \qquad \boldsymbol{S_8 = B_{21} + B_{22}}$$

$$\boldsymbol{S_4 = B_{21} - B_{11}} \qquad \boldsymbol{S_9 = A_{11} - A_{21}}$$

$$\boldsymbol{S_5 = A_{11} + A_{22}} \qquad \boldsymbol{S_{10} = B_{11} + B_{12}}$$

… STRASSEN ALGORITAM …

- Treći korak podrazumeva rekurzivno pravljenje 7 novih ***(n/2)×(n/2)*** matrica primenom matrica iz koraka 1 i 2

$$P_1 = A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}$$

$$P_2 = S_1 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}$$

$$P_3 = S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}$$

$$P_4 = A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}$$

$$P_5 = S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}$$

$$P_6 = S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}$$

$$P_7 = S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}$$

… STRASSEN ALGORITAM …

- Četvrti korak je izračunavanje matrica C_{11}, C_{12}, C_{21}, C_{22} sabiranjem i oduzimanjem raznih kombinacija P_i matrica
  - Vreme izvršavanje ovih operacija je $\Theta(n^2)$

$$C_{11} = P_5 + P_4 - P_2 + P_6 = (A_{11} \cdot B_{22} + A_{12} \cdot B_{21})$$

$$C_{12} = P_1 + P_2 = (A_{11} \cdot B_{12} + A_{12} \cdot B_{22})$$

$$C_{21} = P_3 + P_4 = (A_{21} \cdot B_{11} + A_{22} \cdot B_{21})$$

$$C_{22} = P_5 + P_1 - P_3 - P_7 = (A_{22} \cdot B_{22} + A_{21} \cdot B_{12})$$

… STRASSEN ALGORITAM …

- Koliko je vreme izvršavanja algoritma?
  - Korak 1 – $\Theta(\mathbf{1})$
  - Korak 2 – $\Theta(\boldsymbol{n^2})$
  - Korak 4 – $\Theta(\boldsymbol{n^2})$
  - Korak 3 ?
    - Koliko imamo rekurzivnih množenja?

      7
    - Koliko se smanjuje problem?

      2
    - Znači $\mathbf{7}\boldsymbol{T}\left(\frac{\boldsymbol{n}}{\mathbf{2}}\right)$
- Kombinujući sve korake dobijamo:

$$\boldsymbol{T(n)} = \mathbf{7}\boldsymbol{T}\left(\frac{\boldsymbol{n}}{\mathbf{2}}\right) + \Theta(\boldsymbol{n^2})$$

… STRASSEN ALGORITAM

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$

- $a = 7$ i $b = 2$, $f(n) = n^2$ tako da imamo $n^{\log_b a} = n^{\log_2 7} \approx n^{2.81}$
- Kako je $f(n) = O(n^2)$, možemo primeniti slučaj 1 master teoreme, te je $T(n) = \Theta(n^{\log 7})$

- Na prvi pogled ovo ne deluje kao značajno unapređenje u odnosu na $\Theta(n^3)$, ali kako se radi o razlici u eksponentu, uticaj na vreme izvršavanja je značajan
- Na savremenim računarima za sve vrednosti $n > 32$ Strassenov algoritam pobeđuje klasičan pristup u množenju matrica

BINARNA PRETRAGA

BINARNA PRETRAGA …

- Neka je $\langle x_1, x_2, \ldots, x_n \rangle$ niz realnih brojeva takav da je $x_1 \leq x_2 \leq \cdots \leq x_n$
- Za zadati realni broj z treba ustanoviti da li se pojavljuje u nizu, a ako je odgovor "da", potrebno je pronaći indeks i takav da je $x_i = z$
- Zbog jednostavnosti, tražimo samo jedan indeks i takav da je $x_i = z$
- U opštem slučaju cilj može da bude pronalaženje svih takvih indeksa, najmanjeg ili najvećeg među njima i slično
- Ideja je prepoloviti prostor koji se pretražuje tako što se najpre proveri srednji član niza
- Pretpostavimo zbog jednostavnosti da je n paran broj. Ako je z manje od $x_{\frac{n}{2}+1}$, onda z može biti samo u prvoj polovini niza; u protivnom, z može biti samo u drugoj polovini niza

… BINARNA PRETRAGA …

- Pronalaženje z u prvoj ili drugoj polovini niza je problem sa veličinom ulaza $n/2$, koji se rešava rekurzivno
- Bazni slučaj $n = 1$ rešava se neposrednim upoređivanjem broja z sa elementom

```
BINARRY-SEARCH(X, n, z)

1:return FIND(z, X, 1, n)
```

… BINARNA PRETRAGA …

FIND(*z*, *X*, *Left*, *Right*)

```
1:if Left = Right then
2:   if X[Left] = z then
3:      return Left
4:   else return 0
5:else
6:   Middle = ⌈(Left + Rigt)/2⌉
7:   if z < X[Middle] then
8:      return FIND(z, X, Left, Middle-1)
9:   else
10:     return FIND(z, X, Middle, Right)
```

… BINARNA PRETRAGA

- Posle svakog upoređivanja opseg mogućih indeksa se polovi, pa je potreban broj upoređivanja za pronalaženje zadatog broja u nizu veličine $\boldsymbol{n}$ jednak $\boldsymbol{O}(\log \boldsymbol{n})$
- Ova varijanta binarne pretrage odlaže proveru jednakosti do samog kraja; alternativa je provera jednakosti sa $\boldsymbol{z}$ u svakom koraku
- Problem sa prikazanom varijantom je u tome što se pretraga ne može završiti pre nego što se opseg za pretragu suzi na samo jedan broj
- Prednost joj je pak da se u svakom koraku vrši samo jedno upoređivanje
- Ovakva pretraga je zbog toga obično brža
- Iako je jednostavnije napraviti rekurzivni program, ovaj program nije teško prevesti u nerekurzivni
- Binarna pretraga nije tako efikasna za male vrednosti $\boldsymbol{n}$, pa je tada bolje zadati niz pretražiti linearno, član po član

NALAŽENJE DVE NAJBLIŽE TAČKE

NALAŽENJE DVE NAJBLIŽE TAČKE

- Pretpostavimo da su zadate lokacije $\boldsymbol{n}$ objekata i da je zadatak proveriti da li među njima postoje dva objekta, koji su preblizu ili blizu jedan drugog
- Ovi objekti mogu biti, na primer, delovi mikroprocesorskog čipa, zvezde u galaksiji ili sistemi za navodnjavanje
- Problem: U zadatom skupu od $\boldsymbol{n}$ tačaka u ravni pronaći dve koje su na najmanjem međusobnom rastojanju
- Na istim koordinatama može se naći samo jedna tačka i koordinate su diskretni celi brojevi
- Rastojanje se računa kao: $\boldsymbol{d} = \sqrt{(\boldsymbol{x}_1 - \boldsymbol{x}_2)^2 + (\boldsymbol{y}_1 - \boldsymbol{y}_2)^2}$
- Slični ovom su problemi nalaženja najbliže tačke (ili $\boldsymbol{k}$ najbližih tačaka) za svaku tačku zadatog skupa, ili nalaženje najbliže tačke novododatoj tački

DIREKTNI PRISTUP

- Dva moguća rešenja:

1. Mogu se izračunati rastojanja između svake dve tačke, i zatim pronaći najmanje među rastojanjima
   - To obuhvata $\frac{n(n-1)}{2}$ izračunavanja rastojanja i $\frac{n(n-1)}{2} - 1$ upoređivanja
2. Direktno induktivno rešenje moglo bi se zasnivati na uklanjanju jedne tačke, rešavanju problema za $n - 1$ tačaka, i dodavanju nove tačke
   - Međutim, jedina korisna informacija koja se dobija rešavanjem problema za $n - 1$ tačaka je minimalno rastojanje, pa se moraju proveriti rastojanja nove tačke do svih prethodnih $n - 1$ tačak
   - Zbog toga ukupan broj $T(n)$ izračunavanja rastojanja za n tačaka zadovoljava diferencijalnu jednačinu $T(n) = T(n-1) + n - 1$, $T(2) = 1$, čije je rešenje $T(n) = (n^2)$

- Dva opisana rešenja suštinski su ekvivalentna. Cilj je pronaći efikasniji algoritam za velike n

PRISTUP ZASNOVAN NA REKURZIJI …

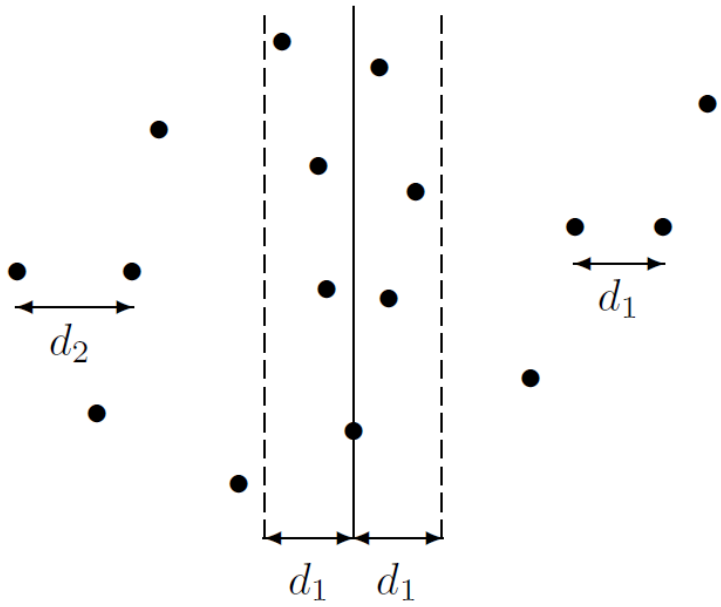
- Ne gleda se svaka tačka pojedinačno, već se skup tačaka deli na dva dela
- Problem se svodi na dva potproblema veličine $\frac{n}{2}$ tačaka
- Teži se da se iz manjih delova dobije što više korisnih informacija, tj. da što više tih informacija važi i za kompletan problem
- Šta ovde koristiti za podelu problema?
- Podeliti tačke ravnom, tako da dobijemo dva disjunktna dela, tako da svaki od njih sadrži polovinu tačaka
- Nakon što se pronađu najmanja rastojanja u svakom delu, potrebno je još razmotriti rastojanja između tačaka bliskih granici skupova

… PRISTUP ZASNOVAN NA REKURZIJI …

- Najjednostavniji način podele je sortirati tačke prema (na primer) x-koordinatama i podeliti ravan pravom paralelnom sa y-osom, koja deli skup na dva jednaka dela
- ako više tačaka leži na pravoj podele, tačke se mogu na proizvoljan način razdeliti između skupova
- Na ovaj način maksimalno se pojednostavljuje objedinjavanje rešenja manjih problema
- Sortiranje treba izvršiti samo jednom
- Zbog jednostavnosti ograničavamo se na nalaženje vrednosti najmanjeg rastojanja između tačaka (a ne i para tačaka za koje se ono dostiže)
- Šta tu treba da se doda?

… PRISTUP ZASNOVAN NA REKURZIJI …

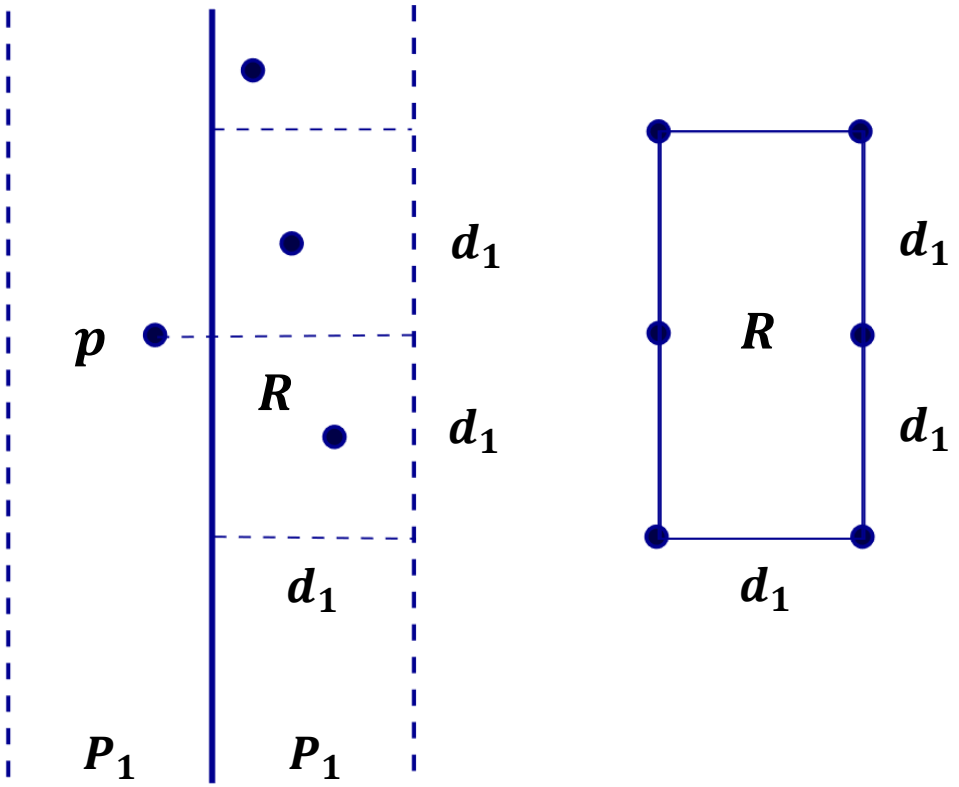
- Neka je $\boldsymbol{P}$ skup tačaka
- Najpre se skup $\boldsymbol{P}$ deli na dva podskupa $\boldsymbol{P_1}$ i $\boldsymbol{P_2}$ čije se veličine razlikuju najviše za jedan (ako je njihov broj neparan)
- Najmanje rastojanje u svakom podskupu pronalazi se na osnovu induktivne hipoteze
- Neka je $\boldsymbol{d_1}$ minimalno rastojanje u $\boldsymbol{P_1}$, a $\boldsymbol{d_2}$ minimalno rastojanje u $\boldsymbol{P_2}$
- Bez smanjenja opštosti može se pretpostaviti da je $\boldsymbol{d_1 \leq d_2}$
- Potrebno je pronaći najmanje rastojanje u celom skupu, odnosno proveriti da li u $\boldsymbol{P_1}$ postoji tačka na rastojanju manjem od $\boldsymbol{d_1}$ od neke tačke u $\boldsymbol{P_2}$



- Dovoljno razmatrati tačke u traci širine $\boldsymbol{2d_1}$, simetričnoj u odnosu na pravu podele

… PRISTUP ZASNOVAN NA REKURZIJI …

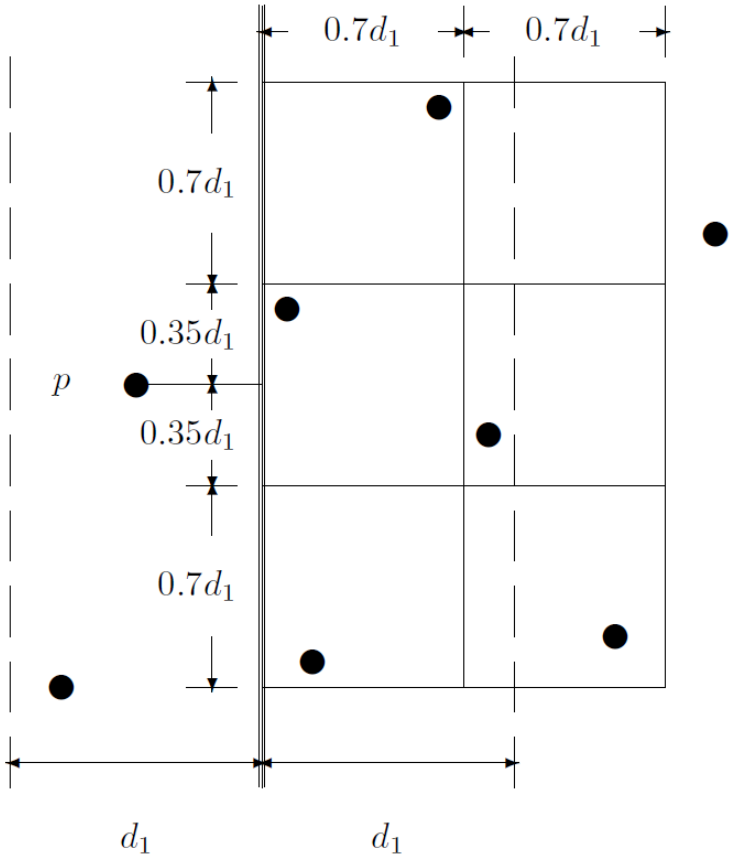
- Ostale tačke ne mogu biti na rastojanju manjem od d_1 od neke tačke iz drugog podskupa
- Ovim zapažanjem, može se eliminisati dosta veliki broj tačaka
- Za proizvoljnu tačku p u traci postoji samo mali broj tačaka u suprotnoj traci čije rastojanje može biti manje od d_1
- U suštini to su sve tačke iz pravougaonika R širine d_1 i visine $2d_1$ pri čemu je centar tog pravougaonika u pravoj koja prolazi kroz tačku p
- Koliko tačaka može biti u R ako je njihova udaljenost najmanje d_1?



… PRISTUP ZASNOVAN NA REKURZIJI …

- Još malo ovo da analiziramo…
- Važi $\frac{\sqrt{2}}{2} < \mathbf{0.7}$, i neka je y_p y-koordinata tačke p
- Možemo napraviti pravougaonik visine $\mathbf{2.1}d_1 < \frac{3\sqrt{2}}{2}d_1$ i širine $\mathbf{1.4}d_1 < \sqrt{2}d_1$, tako da naleže na pravu podele i da mu je y-koordinata preseka dijagonala jednaka y_p
- Podelimo pravougaonik jednom vertikalnom pravom i sa dve horizontalne prave na šest jednakih kvadrata stranice $\mathbf{0.7}d_1$

(kojima je dijagonala manja od d_1!)



… PRISTUP ZASNOVAN NA REKURZIJI …

- U svakom od malih kvadrata može se naći najviše jedna tačka, jer je rastojanje bilo koje dve tačke u kvadratu manje od $\boldsymbol{d_1}$
- Sve tačke u drugom podskupu, koje su na rastojanju od $\boldsymbol{p}$ manjim od $\boldsymbol{d_1}$, očigledno se moraju nalaziti u pravougaoniku
- Dakle, u drugom podskupu se može nalaziti najviše šest tačaka na rastojanju od $\boldsymbol{p}$ manjem od $\boldsymbol{d_1}$
- Tačka kandidat sa y-koordinatom $\boldsymbol{y_p}$ mora da zadovolji i uslov $|\boldsymbol{y_p} - \boldsymbol{y_q}| < \boldsymbol{d_1}$, pa se mora nalaziti u jednom od 6 kvadrata
- Ako su sve tačke u traci sortirane prema y-koordinatama i pregledaju se tim redom, dovoljno je za svaku tačku proveriti rastojanje osam suseda u suprotnoj traci, četiri ispod i četiri iznad (a ne od svih $\boldsymbol{n - 1}$ tačaka u najgorem slučaju)

… PRISTUP ZASNOVAN NA REKURZIJI …

- Sam algoritam

NAJBLIZI-PAR-ALG(P)

1: SORTIRANJE-PO-X(P)

2: d = NAJBLIZI-PAR$\left(P, 1, \frac{P.length}{2}, P.length\right)$

3: $//P[1 .. \frac{P.length}{2}], P[\frac{P.length}{2} + 1 .. P.length]$

… PRISTUP ZASNOVAN NA REKURZIJI …

- Sam algoritam

NAJBLIZI-PAR$(P, poc, sred, kraj)$

1: **if** $(kraj - poc) = 1$ **then**

2: return DIST$(P[poc], P[kraj])$

3: d_1 = NAJBLIZI-PAR$\left(P, poc, \frac{sred}{2}, sred\right)$

4: d_2 = NAJBLIZI-PAR$\left(P, sred + 1, \frac{kraj - (sred + 1)}{2}, kraj\right)$

5: $d = \min(d_1, d_2)$

6: S_y = PREUZMI-TACKE-KOD-PRAVE-PODELE$(P, poc, sred, kraj, d)$

7: SORTIRANJE-PO-Y(S_y)

8: d = ODREDI-NAJMANJE-RASTOJANJE-U-TRACI(S_y, d)

9: **return** d

… PRISTUP ZASNOVAN NA REKURZIJI …

- Sam algoritam

PREUZMI-TACKE-KOD-PRAVE-PODELE($P, poc, sred, kraj, d$)

1: $j = 1$
2: **for** $i = poc$ **to** $kraj$
3: **if** $|P[i].x - P[sred].x| < d$
4: $S_y[j] = P[i]$
5: $j = j + 1$
6: **return** S_y

… PRISTUP ZASNOVAN NA REKURZIJI

- Sam algoritam

```
ODREDI-NAJMANJE-RASTOJANJE-U-TRACI (S_y, d)
1:  for i = 1 to S_y.length
2:      j = i + 1
3:      while j < S_y.length and S_y[i].y − S_y[j].y < d
4:          // ova petlja se ne izvršava više od 6 puta
5:          if DIST(S_y[i], S_y[j]) < d
6:              d = DIST(S_y[i], S_y[j])
7:  return d
```

KOMPLEKSNOST REŠENJA …

- Sortiranje složenosti $O(\boldsymbol{n} \cdot \log \boldsymbol{n})$ izvršava se samo jednom
- Eliminacija tačaka van centralne trake može se izvesti za $O(\boldsymbol{n})$ koraka
- Za sortiranje tačaka u traci po y-koordinatama u najgorem slučaju potrebno je $O(\boldsymbol{n} \cdot \log \boldsymbol{n})$ koraka
- Potrebno je $O(\boldsymbol{n})$ koraka da se pregledaju tačke u traci, i da se provere rastojanja svake od njih od konstantnog broja suseda u sortiranom redosledu

```
NAJBLIZI-PAR-ALG(P)
1:  SORTIRANJE-PO-X(P)
2:  d = NAJBLIZI-PAR(P, 1, P.lengt h/2, P.length)
3:  //P[1 .. P.lengt h/2], P[P.lengt h/2 + 1 .. P.length]
```

```
NAJBLIZI-PAR(P, poc, sred, kraj)
1:  if (kraj − poc) = 1 then
2:     return DIST(P[poc], P[kraj])
3:  d1 = NAJBLIZI-PAR(P, poc, sred/2, sred)
4:  d2 = NAJBLIZI-PAR(P, sred + 1, kraj − (sred + 1)/2, kraj)
5:  d = min(d1, d2)
6:  Sy = PREUZMI-TACKE-KOD-PRAVE-PODELE(P, poc, sred, kraj, d)
7:  SORTIRANJE-PO-Y(Sy)
8:  d = ODREDI-NAJMANJE-RASTOJANJE-U-TRACI(Sy, d)
```

- Da bi se rešio problem veličine $\boldsymbol{n}$, treba rešiti dva potproblema veličine $\frac{\boldsymbol{n}}{\boldsymbol{2}}$ i izvršiti $O(\boldsymbol{n} \cdot \log \boldsymbol{n})$ koraka za kombinovanje njihovih rešenja (plus $O(\boldsymbol{n} \cdot \log \boldsymbol{n})$ koraka za jednokratno sortiranje tačaka po x-koordinatama na početku)

… KOMPLEKSNOST REŠENJA

- Dobijamo rekurentnu jednačinu:

$$\boldsymbol{T(n) < 2T\left(\frac{n}{2}\right) + cn\log n, T(2) = 1}$$

- pri čemu u $\boldsymbol{T(n)}$ nije uključeno početno sortiranje po x-koordinatama
- Indukcijom se pokazuje da je rešenje:

$$\boldsymbol{T(n) < cn \cdot \log^2 n},$$

naime za $\frac{n}{2}$ dobija se:

$$\boldsymbol{T\left(\frac{n}{2}\right) < c\left(\frac{n}{2}\right)\log^2\left(\frac{n}{2}\right)}$$

za $\boldsymbol{n > 2}$ važi

$$\boldsymbol{T(n) < cn \cdot (\log n - 1)^2 + cn \cdot \log n = cn(\log^2 n - \log n + 1) < cn \cdot \log^2 n}$$

- Složenost $\boldsymbol{O(n \cdot \log^2 n)}$ je asimptotski bolja od kvadratne

```
NAJBLIZI-PAR-ALG(P)
1:  SORTIRANJE-PO-X(P)
2:  d = NAJBLIZI-PAR(P, 1, P.length/2, P.length)
3:  //P[1..P.length/2], P[P.length/2 + 1..P.length]
NAJBLIZI-PAR(P, poc, sred, kraj)
1:  if (kraj − poc) = 1 then
2:     return DIST(P[poc], P[kraj])
3:  d1 = NAJBLIZI-PAR(P, poc, sred/2, sred)
4:  d2 = NAJBLIZI-PAR(P, sred + 1, kraj − (sred + 1)/2, kraj)
5:  d = min(d1, d2)
6:  Sy = PREUZMI-TACKE-KOD-PRAVE-PODELE(P, poc, sred, kraj, d)
7:  SORTIRANJE-PO-Y(Sy)
8:  d = ODREDI-NAJMANJE-RASTOJANJE-U-TRACI(Sy, d)
```

ALGORITAM SLOŽENOSTI $O(n \cdot \log n)$ ...

- U toku objedinjavanja rešenja potproblema izvodi se $O(n \cdot \log n)$ koraka zbog sortiranja tačaka po y-koordinatama
- Ideja je da se traženje najmanjeg rastojanja objedini sa sortiranjem po y-koordinatama
- To znači da će svaki podniz već biti sortiran i da prilikom njihovog objedinjavanja treba ukomponovati dva sortirana podniza u jedan niz
- Ovo jako liči na MERGE sortiranje, ali ovde nije potrebno izvesti kompletno sortiranje, nego samo objedinjavanje dva već sortirana podniza
- Kako se objedinjavanje sortiranih podnizova izvodi za $O(n)$ koraka, diferencijalna jednačina za složenost (bez početnog sortiranja po -koordinatama) postaje

$$T(n) < 2T\left(\frac{n}{2}\right) + cn, T(2) = 1$$

- Njeno rešenje je $O(n \cdot \log n)$

… ALGORITAM SLOŽENOSTI $O(n \cdot \log n)$

NAJBLIZI-PAR-ALG(P)

1: Sortirati tačke prema x-koordinatama

2: //ovo sortiranje izvršava se samo jednom, na početku

3: Podeliti skup na dva istobrojna podskupa

4: Rekurzivno izvršiti sledeće

5: Izračunati minimalno rastojanje u oba podskupa;

6: Sortirati tačke u svakom delu prema -koordinatama;

7: //Objediniti dva sortirana niza u jedan; Objedinjavanje se mora izvršiti pre eliminacije; sledećem nivou rekurzije mora se isporučiti sortiran kompletan skup}

8: Neka je d manje od dva minimalna rastojanja

9: Eliminisati tačke koje su na rastojanju većem od d od prave podele

10: Za preostale tačke proveriti njihova rastojanja od 4 prethodne i 4 naredne tačke iz suprotne trake

11: **if** neko od ovih rastojanja manje od d

12: popraviti d