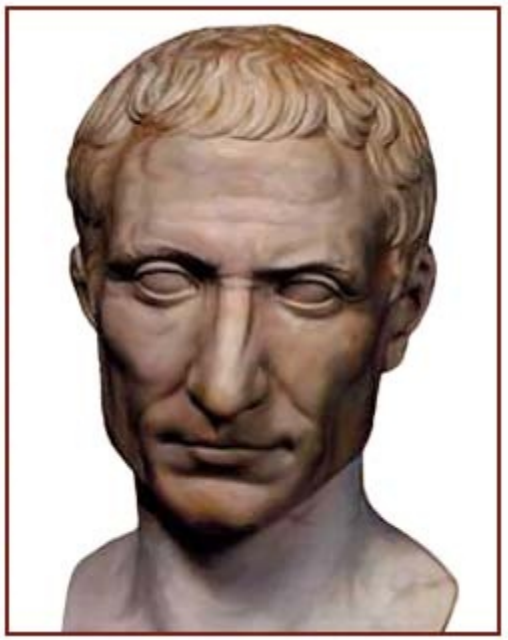
Algoritmid ja andmestruktuurid

- Jaga ja Valitse - algoritmide disaini paradigma
- Sorteerimisalgoritmid
- Rekursiivsete algoritmide keerukuse analüüs

Divide et Impera

CAESAR



Napoleon

Jaga ja Valitse strateegia

- Jaga
  - Jaga ülesanne lihtsamateks alamülesanneteks
- Valitse
  - Lahenda alamülesanded
- Ühenda
  - Leia lahendus alamülesannete lahenduste kaudu

- Kõige loomulikum on seda strateegiat realiseerida rekursiivsete algoritmidega
  - alamülesanne lahendatakse sama algoritmi järgi lihtsamate sisendandmetega

Rekursiivse Jaga ja Valitse algoritmi loomine

1. Leia kuidas ülesannet lihtsamateks alamülesanneteks jagada
2. Leia kuidas alamülesannete lahendustest käesoleva ülesande lahendus tuletada
3. Leia lõpetamistingimus mille poole jagamine viib
4. Määra lahendus kui lõpetamistingimus on täidetud

Binaarne otsing

- Saab kasutada otsimiseks sorteeritud andmetest
  - **Jaga**

    Jagame andmed pooleks
  - **Valitse**

    Otsime ainult sellest poolest, kuhu otsitav peaks jääma
  - **Ühenda**

    Vastuseks on alamülesande vastus

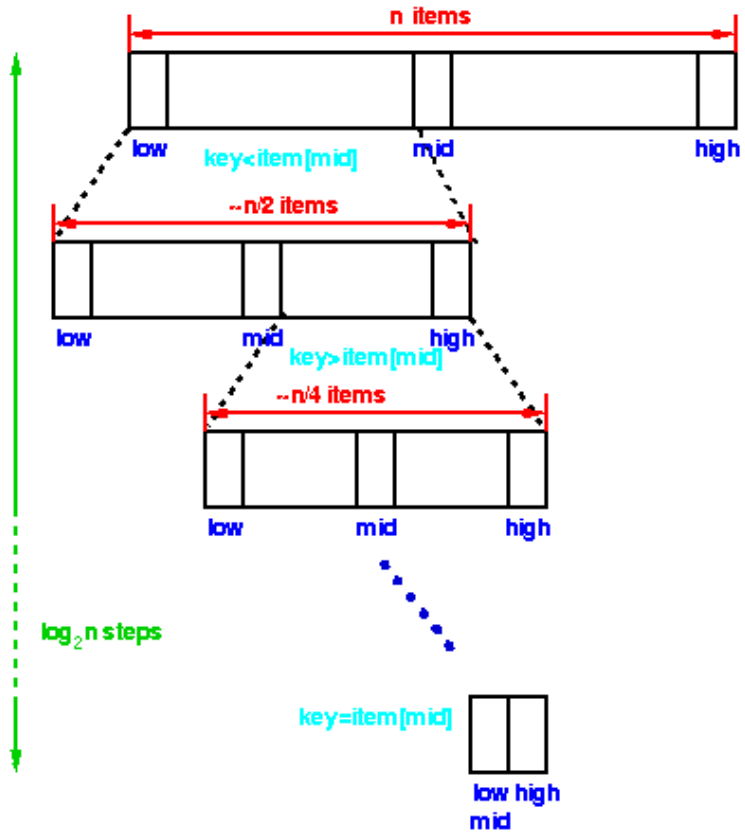
Rekursiooni lõpetamistingimus

Jah kui andmete keskmine element vastab küsitule

Ei kui andete maht on 0

Binaarne otsing rekursiivselt

```
binary_search(x,L)
{
  n = length of L;
  mid = n/2;
  if (n == 0)
    return no match
  else if (L[mid] matches x)
    return L[mid]
  else if (L[mid] > x)
    binary_search(x,L[1..mid-1])
  else
    binary_search(x,L[mid+1..n])
 }
```



Binaarse otsingu (rekursiivne) analüüs

- Rekursiivsete algoritmide keerukus avaldub rekurrentse avaldisena

  W(n) = W(n/2) + 3

  W(0) = 1

Funtsiooni kehas on 3 võrdlust

Rekursiooni lõpp

Rekursiivne väljakutse 1/2 suurusega ülesandele

- Otsime lahendust W(n) = ?, mis poleks rekurrentne

```
binary_search(x,L){
  n = length of L; mid=n/2;
  if (n = 0)     return no match
  else if (L[mid] matches x)   return L[mid]
  else if (L[mid] > x)   binary_search(x,L[1..mid-1])
  else                   binary_search(x,L[mid+1..n])
 }
```

Merge sort

1. Jagame massiivi pooleks

   Sorteerime mõlemad väiksemad massiivid
2. Ühendame sorteeritud massiivid üheks sorteeritud massiiviks võrreldes omavahel mõlema massiivi väiksemaid elemente
3. Jagamise lõpetamistingimuseks on 1 elemendiga massiiv
4. Ühe elemendiga massiivi võib lugeda sorteerituks, tagastame sellesama massiivi

Mergesort

```
mergesort (int n, keytype S[ ]) {
   if (n>1) {
      int h=[n/2],       m = n - h;
      keytype U[1 ..h], V[1 ..m];
      copy S[1] through S[h] to U[1] through U[h];
      copy S[h+1] through S[n] to V[1] through V[m];
      mergesort(h, U);
      mergesort(m, V);
      merge (h, m, U, V, S);
} }
```

$W(n)=W(n/2)+W(n/2)+(n/2+n/2)$

sort(U) sort(V) merge()

$W(n) = 2W(n/2) + n$, kui $n = 2^k$

$W(n) = W(\lfloor n/2 \rfloor) + W(\lceil n/2 \rceil) + n$, kui $n \neq 2^k$

$W(n) \in O(?)$

```
merge (int h, int m, keytype U[ ], keytype V[ ], keytype S[ ]) {
   index i, j, k;
   i = 1; j = 1; k = 1;
   while (i <= h && j <= m) {
      if (U[i] < V[j]) { S[k] = U[i]; i++; }
      else { S[k] = V[j]; j++; }
      k++; }
   if (i>h) copy V[j] through V[m] to S[k] through S[h+m];
   else copy U[i] through U[h] to S[k] through S[h+m]; }
```

Quicksort

1. Valime ühe elemendi teljeks (pivot)

   Sorteerime kõik teljest väiksemad temast vasakule ja suuremad paremale

   Sorteerime teljest paremale ja vasakule jääva massiivi
2. Ühendame vasaku massiivi, telje ja parema massiivi üksteise järgi
3. Jagamise lõpetamistingimuseks on 1 elemendiga massiiv
   Ühe elemendiga massiivi võib lugeda sorteerituks, tagastame sellesama massiivi

Quicksort

```
quicksort (index low, index high)
   index pivot
   if (high > low)
      pivot = partition(low, high)
      quicksort(low, pivot - 1)
      quicksort(pivot + 1, high)
```

```
partition(int low, int high)
   pivotitem = S[low]
   p = low   //pivot location
   for (i=low+1; i<=high; i++)
      if (S[i] < pivotitem){
         p++
         exchange S[i] and S[p]
   exchange S[low] and S[p]
   return p
```

Halvima juhu keerukus

$W(n) = W(0) + W(n - 1) + (n - 1)$

$W(n) = W(n - 1) + (n - 1)$

$W(0) = 0$

Keskmise juhu keerukus

$$A(n) = \sum_{p=1}^{n} 1/n[A(p-1)+A(n-p)]+(n-1)$$

Leia suurusel *k*- s element

- Lihtne algoritm

  leia *k* korda suurim ülejäänutest

  – keerukus O(*kn*)

  – mediaanelemendi (*k*=(*n*+1)/2) leidmisel O(n^2)

  – kas saaks paremini?

- Sorteerime andmed

  – mediaanelemendi saab keerukusega O(*n* lg *n*)

  – meil pole vaja sorteeritud andmeid, kas saaks paremini?

- *Quicksort* telje asukoht (*rank*) on peale *partition* tegemist teada!

  – tegeleme edasi ainult poolega, kuhu jääb *k*

  – jaga ja valitse!

k-select algoritm

```
keytype selection (index low, index high, index k)
  if (low == high)
    return S[low]
  else {
    index pivot = partition(low, high)
  if (k == pivot)
    return S[pivot]
  else if (k < pivot)
    return
      selection (low, pivot − 1, k)
    else return
      selection (pivot + 1, high, k)
```

```
partition(index low, index high)
   pivotitem = S[low]
   p = low   //pivot location
   for (i=low+1; i<=high; i++)
      if (S[i] < pivotitem){
         p++
         exchange S[i] and S[p]
   exchange S[low] and S[p]
   return p
```

k-select analüüs

- Parimal juhul

  $O(n)$
- Halvimal juhul

  $O(n^2)$
- Keskmiselt andmete ühtlase jaotuse korral $3n$ võrdlust - $O(n)$

- kavalam telje valik aitab saavutada keskmist keerukust
  - mediaanvalik
  - juhuslik valik

Rekurrentsete võrrandite lahendamine

http://www.geekarmy.com/cool/703/miniglobe-and-recursive-photography/

Rekurrentsed võrrandid

- Näiteks

$T(n) = T(n - 1) + (n - 1)$

$T(n) = 2T(n/2) + (n - 1)$

- Lahendusmeetodid
  - Üldmeetod: äraarvamine + induktiivne tõestus
  - Iterratsioonimeetod
  - Domeeni (muutujate) vahetus
  - Rekursioonipuu meetod
  - Spetsiaalmeetodid erikujulistele võrranditele

    Lineaarsed võrrandid: $\sum_i(a_i T(n - i)) = f(n)$

    Murruga võrrandid: $T(n) = aT(n / b) + f(n)$

Üldmeetod

- Lahendite arvutamine mitme *n* väärtuse jaoks
- Lahendi hüpoteesi püstitamine
- Hüpoteesi tõestamine induktsiooni abil

- Sobib kui suudame püstitada mõistliku hüpoteesi lahendi kohta (keerukusfunktsiooni)
  - analoogia mõne teadaoleva võrrandiga
  - rekursioonipuu analüüsist

Matemaatiline induktsioon

- Induktsiooni kasutatakse teoreemide tõestamiseks lõpmatutel loenduvatel hulkadel, näiteks

$$\sum_{i=1}^{n} i = n(n+1)/2$$

- Induktsiooni baas

  tõestus, et väide kehtib baasjuhul (n=1)

- Induktsiooni hüpotees

  eeldus, et väide kehtib suvalise $n \geq 1$ korral

- Induktsiooni samm

  tõestus, et kui eeldus kehtib *n* korral, siis kehtib ka *n*+1 korral

Iterratsioonimeetod

- Võib sobida võrranditele kujul
  $T(n) = T(n-a) + f(n)$
- On olemas üldisem meetod lineaarsete rekurrentsete võrrandite lahendamiseks (õpik lisa B)

$T(n) = T(n - 1) + (n - 1)$
$T(0) = 0$

$T(n) = T(n - 2) + (n - 2) + (n - 1)$

$T(n) = T(n - 3) + (n - 3) + (n - 2) + (n - 1)$

$T(n) = 0 + 0 + 1 + \ldots + (n - 2) + (n - 1)$

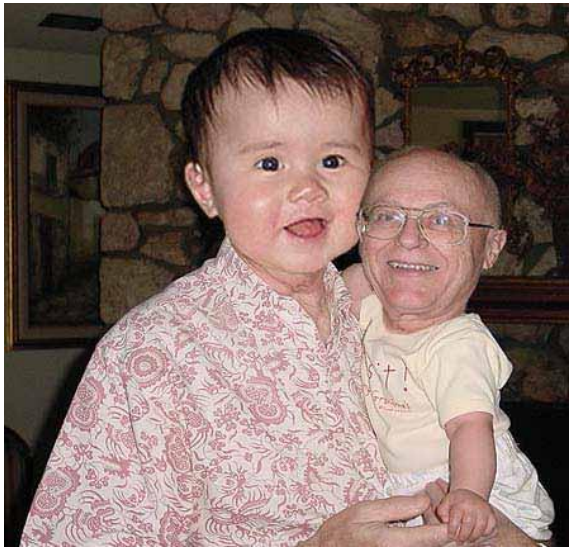
$T(n) = \sum^{n-1} i = (n - 1)\, n / 2$

Muutujate vahetuse meetod

Binaarse otsingu keerukus

$W(n) = W(n/2) + 3$

$W(1) = 1$



- Lahendades avaldise kui vahetuse $n = 2^k$ (ehk $k = \lg n$) korral saame

$W(2^k) = W(2^{k-1}) + 3$

kui $w(k) = W(2^k)$, siis $w(k) = w(k-1) + 3$ ja $w(0) = 1$

$w(k) = 3k + 1$ ehk $W(2^k) = 3k + 1$

$W(n) = 3 \lg n + 1 \in O(\log n)$

- Üldkujul

$W(n) = 3 \lfloor \lg n \rfloor + 1 \in O(\log n)$

Rekursioonipuu meetod

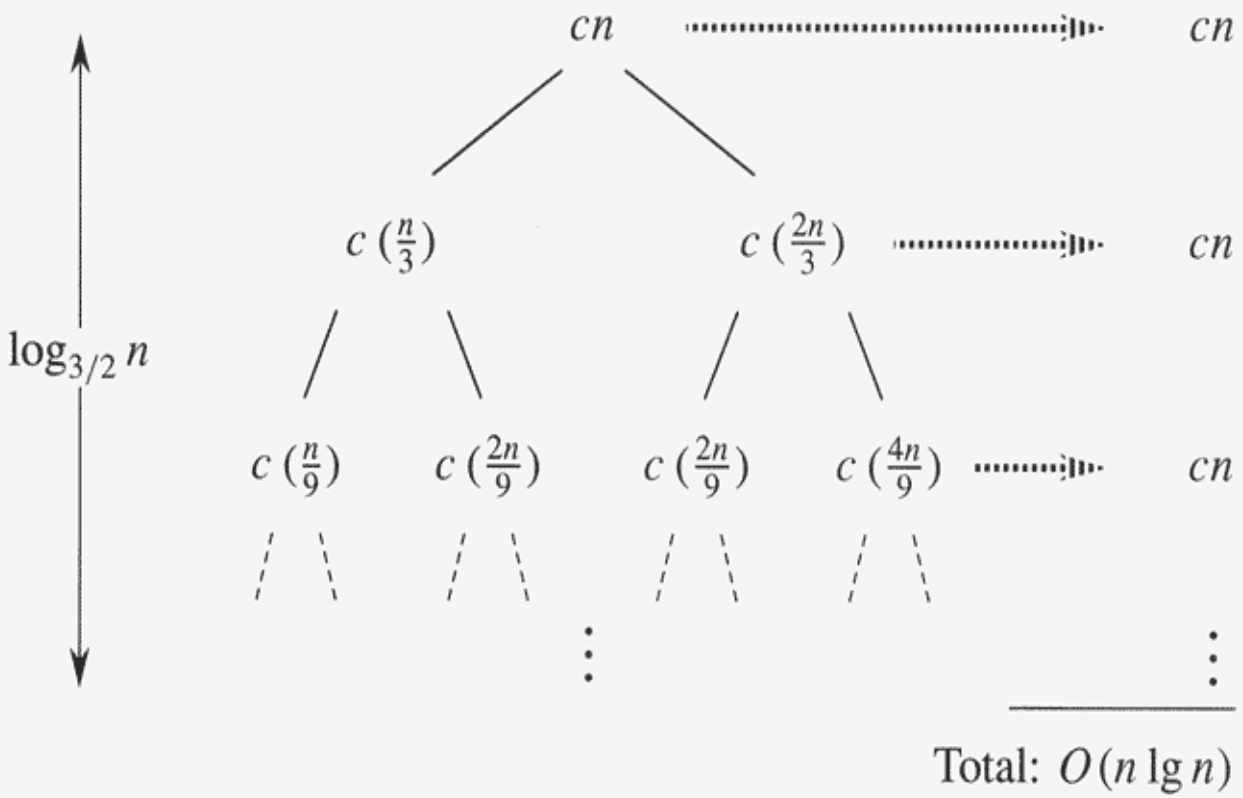


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

Quicksort keerukus

- Halvima juhu keerukus

  $W(n) = W(0) + W(n - 1) + (n - 1)$

  $W(n) = W(n - 1) + (n - 1)$
  $W(0) = 0$

  $$W(n) = n(n - 1) / 2 \in O(n^2)$$

- Keskmise juhu keerukus

  $$A(n) = \sum_{p=1}^{n} 1/n[\, A(p - 1) + A(n - p) \,] + (n - 1)$$

$$A(n) \approx (n+1)\, 2 \ln n \approx 1.38(n + 1) \lg n \in O(n \lg n)$$

Keerukuse põhiteoreem

Põhiteoreemist (*Master Theorem*) võib järeldada, et kui keerukuse võrrand on kujul

$$T(n) = aT(n / b) + c\, n^k$$

a - *alamülesannete arv*

b - *alamülesande kahanemise kordaja*

n^k - *funktsiooni keha keerukusaste*

siis on keerukus

$$T(n) \in \begin{cases} \Theta(n^k) & \text{if } a < b^k \\ \Theta(n^k \lg n) & \text{if } a = b^k \\ \Theta(n^{\log_b a}) & \text{if } a > b^k \end{cases}$$

$T(n) = 2T(n/2) + (n - 1)$

$T(n) \in O(n \lg n)$

Intuitsioon

Võrreldakse keerukust, mis tuleb rekursioonist $O\left(n^{\log_b a}\right)$
ja mis tuleb rekursiivse funktsiooni kehast $f(n)$

1. kui $O\left(n^{\log_b a}\right)$ domineerib, on keerukus

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

2. kui $f(n)$ domineerib, on keerukus

$$T(n) = \Theta\left(f(n)\right)$$

3. kui nad on võrreldavad, siis tuleneb lg n lisakeerukus

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

Merge sort keerukus

$W(n) = W(n/2) + W(n/2) + (n/2 + n/2)$
msort(left) msort(right) merge(left,right)

$W(n) = 2W(n/2) + n$ kui $n = 2^k$

$W(n) = W(\lfloor n/2 \rfloor) + W(\lceil n/2 \rceil) + n$ kui $n \neq 2^k$

$W(n) \in O(n \lg n)$

Sorteerimisalgoritmide võrdlus

	Halvimal juhul	Keskmisel juhul
Bubble sort	$O(n^2)$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$
Quicksort	$O(n^2)$	$O(n \log n)$

http://www.cs.ubc.ca/spider/harrison/Java/

http://www.cs.rit.edu/~atk/Java/Sorting/sorting.html

Suurte täisarvude aritmeetika

- Täpseteks arvutusteks suurte täisarvudega, mis ületavad riistvaralised piirid (2-8 baiti)

  Näiteks RSA krüptoalgoritm

  RSA võti: 1024 bitti = 128 baiti > 10^{300}

- Arvude lihtne esitus:
  numbri kaupa massiivis, madalamad järgud eespool

  658791 = [1][9][7][8][5][6]

- Lihtsad tehted O(*n*), kus *n* on järkude arv

  u + v u - v u * 10^m u DIV 10^m u REM 10^m

Jaga ja valitse: Suurte täisarvude korrutamine

1. Jagame arvu kaheks osaks

   $u = a * 10^m + b$

   $v = c * 10^m + d$

   u = [1][9][7][8][5][6]

   b | m | a

   $658791 = 658 * 10^3 + 791$

2. Korrutamiseks kasutame rekursiivselt sama algoritmi

   $uv = (a * 10^m + b)(c * 10^m + d)$

   $uv = ac * 10^{2m} + (ad + bc) * 10^m + bd$

3. Rekursiooni lõpetamistingimuseks on tegurite mahtumine olemaolevasse täisarvu andmetüüpi

4. rakendame riistvaralist (kompilaatori) korrutamist

Suurte täisarvude korrutamine

```
large_int prod (large_int u, large_int v)
{ large_int a, b, c, d;
  int n, m;
  n = max( length(u), length(v))
  if( u==0 || v==0 )
    return 0;
  else if ( n < threshold )
    return (u * v in usual way);
  else
  { m = n div 2;
    a = u div 10^m; b = u rem 10^m;
    c = v div 10^m; d = v rem 10^m;
    return (prod(a,c)*10^2m +
     (prod(a,d)+prod(b,c))*10^m +
     prod(c,d));  }}
```

Algoritmi keerukus

- Põhioperatsioonid

  lineaarse keerukusega opertsioonid (+, -, * 10^m)

  nende summaarse keerukuse tähistame *cn*

- Halvima juhu keerukus

  $W(n) = 4W(n/2) + cn$

  Lahendame Põhiteoreemi abil:

  $W(n) \in O(n^{\lg 4}) = O(n^2)$

Parem korrutamismeetod

- Vajasime tulemust avaldisele (4 korrutamist)

  $uv = ac * 10^{2m} + (ad + bc) * 10^{m} + bd$

- Avaldist (ad + bc) on võimalik leida teisiti

  $r = (a + b)(c + d) = ac + (ad + bc) + bd$

  $(ad + bc) = r - ac - bd$

  $(ad + bc) = (a + b)(c + d) - ac - bd$ (3 korrutamist!)

- Halvima juhu keerukus

  $W(n) \approx 3W(n/2) + cn$

  $W(n) \in O(n^{\lg 3}) \approx O(n^{1.58})$

Mõned järeldused

- Täisarvude korrutamine (ja jagamine) ei ole konstantse, ega isegi lineaarse keerukusega
  - 64-bit korrutamine on keerukam kui 2*32-bit korrutamine
- Jaga ja valitse
  aga oluline on kuidas jagada!

Lävega (*threshold*) segameetod

- Mingist *n* väärtusest (lävest) alates võtab jagamine rohkem resurssi kui annab võitu. Sel puhul võib kasutada mõnda põhimõtteliselt suurema keerukuse, aga väikeste n väärtuste korral kiiremat meetodit
- Sorteerimine
  - Merge ja bubble (või insertion) sort
- Suurte täisarvude korrutamine
  - Tarkvaraline ja riistvaraline korrutamine
- Parim läve väärtus sõltub mõlema algoritmi konkreetsest realisatsioonist.

Jaga ja valitse

- Jaga
  - Jaga ülesanne lihtsamateks alamülesanneteks
- Valitse
  - Lahenda alamülesanded
- Ühenda
  - Leia lahendus alamülesannete lahenduste kaudu

Millal “jaga ja valitse” ei sobi

- Ülesanne suurusega *n* jagatakse kaheks või enamaks peaaegu samasuureks ülesandeks (näiteks *n*-1)
  - fib(n) = fib(n-1)+fib(n-2)
  - Tulemuseks on eksponentsiaalne keerukus
- Ülesanne suurusega *n* jagatakse peaegu *n*-ks alamülesandeks suurusega *n/c,* kus *c* on konstant
  - $T(n) = k(n)\ T(n/e) + f(n), \qquad k(n) = O(n)$
  - Tulemuseks on $O(n^{\lg n})$ algoritm

Rekurrentsed võrrandid ja lahendid, mida võiks teada

- $T(n) = T(n - O(1)) + O(1)$ → $O(n)$
- $T(n) = T(n - O(1)) + O(n)$ → $O(n^2)$
- $T(n) = T(n/2) + O(1)$ → $O(\lg n)$
- $T(n) = 2T(n/2) + O(1)$ → $O(n)$
- $T(n) = 2T(n/2) + O(n)$ → $O(n \lg n)$

Kokkuvõtteks

- Jaga ja valitse –
  - jagame probleemi järjest väiksemateks tükkideks, kuni oskame lahendada triviaalset probleemi
  - Võit tavaliselt $O(n)$-st $O(\log n)$-ks
  - tulemuseks tihti rekursiivne algoritm
- Rekursiivse algoritmi keerukust väljendab rekurentne võrrand
- Rekurrentse võrrandi lahendamise meetodid
  - Iteratsioonimeetod
  - Muutuja asendamine
  - Rekursioonipuu
  - Põhiteoreem