Procedūrinio programavimo pagrindai

Algoritmai

lekt. Irmantas Radavičius
irmantas.radavicius@mif.vu.lt
Informatikos institutas, MIF, VU

Turinys

Algoritmai

Algoritmas – tinkamai apibrėžta žingsnių seka,
iš tam tikrų duomenų (įvestis)
leidžianti gauti tam tikrą rezultatą (išvestį)

Programavimo etapai

- kūrimas (projektavimas)
- kodo rašymas
- testavimas
- palaikymas

Algoritmo pseudokodas – būdas neprisirišti prie konkrečios kalbos

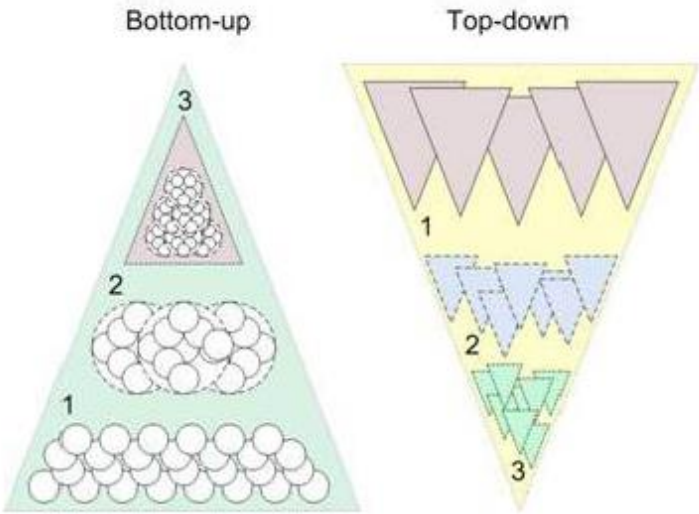
Projektavimas

Projektavimas

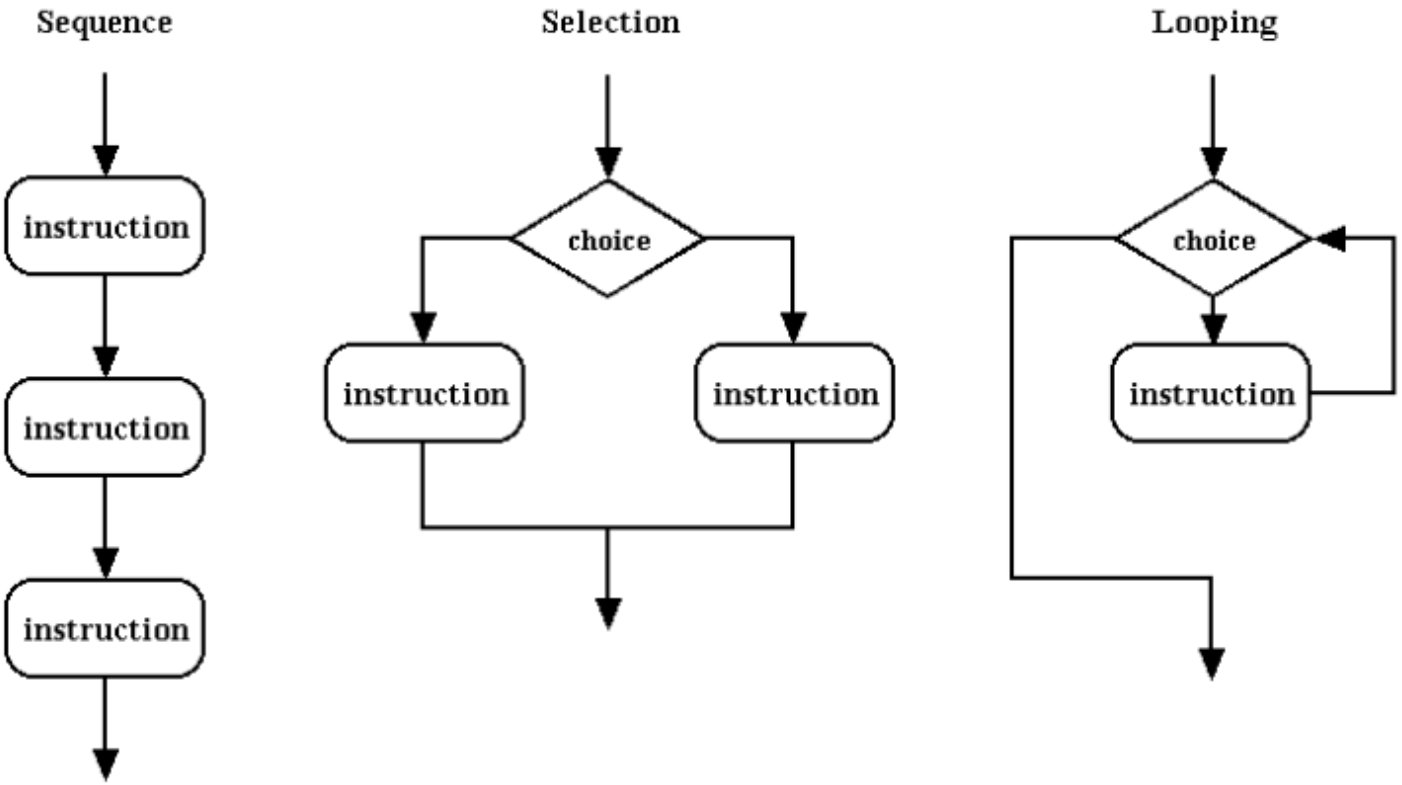
- problemos suvokimas (ką turime)
- problemos sprendimas (ko reikia)
- algoritmo kūrimas/parinkimas (kaip tai gauti)
- algoritmo korektiškumo tikrinimas (ar tai veikia)

Dvi projektavimo strategijos

- iš viršaus žemyn
- iš apačios aukštyn



Struktūrinio programavimo teorema



Pavyzdys: DBD algoritmas

Algorithm 1 Euclids algorithm

```
1: procedure EUCLID(a, b)                    ▷ The g.c.d. of a and b
2:     r ← a mod b
3:     while r ≠ 0 do                        ▷ We have the answer if r is 0
4:         a ← b
5:         b ← r
6:         r ← a mod b
7:     end while
8:     return b                              ▷ The gcd is b
9: end procedure
```

Ar galime apibendrinti?

Pavyzdys: MBK algoritmas

Algoritmo kūrimas

Algoritmo testavimas

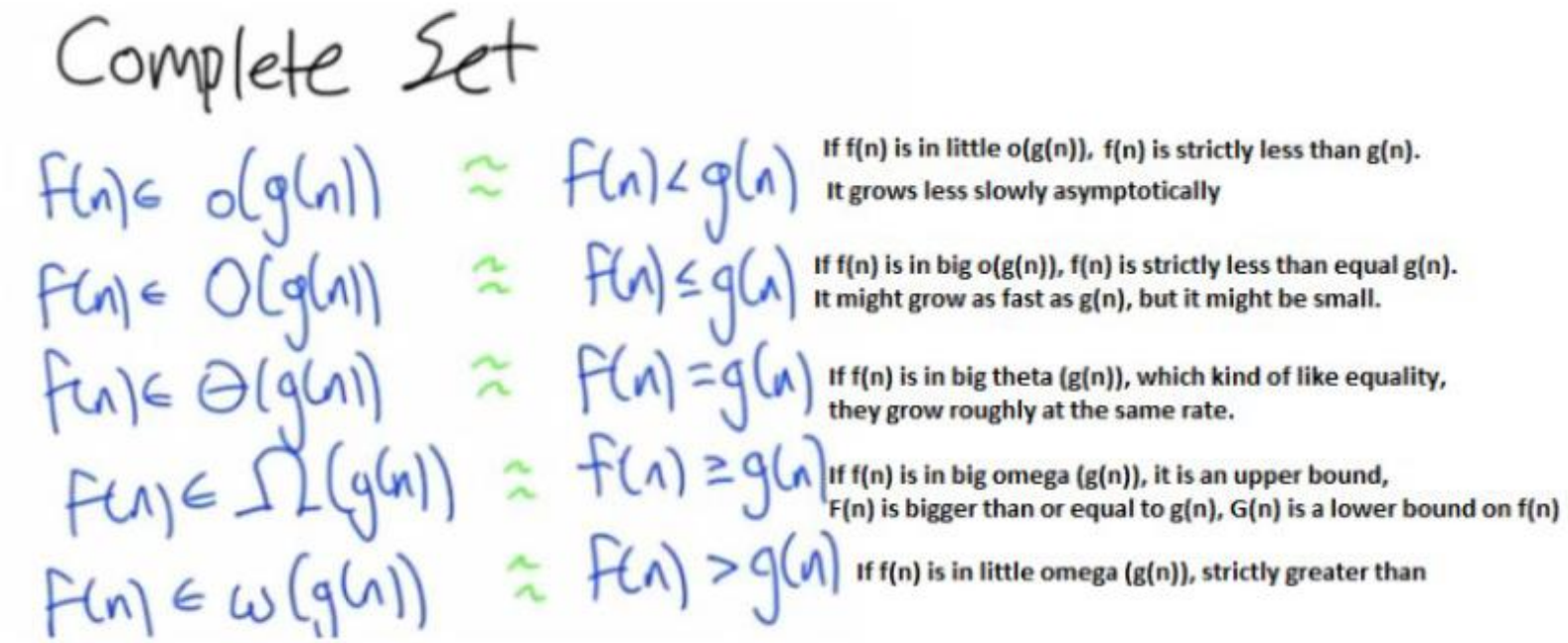
Įvairūs pastebėjimai (ar reikia naujo algoritmo?)

$$\operatorname{lcm}(n,m) = \frac{m \cdot n}{\gcd(m,n)}$$

Ar galime apibendrinti?

Algoritmų sudėtingumas

Asimptotinis algoritmų sudėtingumo įvertinimas skirtas suvokti kaip **algoritmo darbo laikas** priklauso nuo **duomenų kiekio** kai duomenų kiekis labai labai labai didelis.



Algoritmų vertinimas

Algoritmo sudėtingumo priklausomybė nuo duomenų:

- sudėtingumas geriausiu atveju (angl. best case)
- sudėtingumas blogiausiu atveju (angl. worst case)
- sudėtingumas vidutiniu atveju (angl. average case)

Kokie tai duomenys?

Rėžiai sudėtingumui vertinti:

- Rėžis iš apačios (angl. lower bound)
- Rėžis iš viršaus (angl. upper bound)

O-didžiojo notacija

Rašoma $f(n) = O(g(n))$ jeigu galima teigti jog:

(neformaliai) $|f(n)| \leq k \cdot |g(n)|$ for some positive k

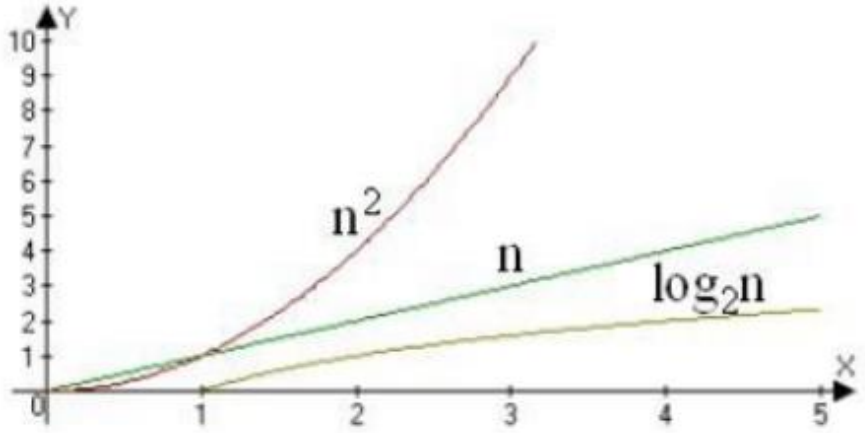
(formaliai) $\exists k > 0 \; \exists n_0 \; \forall n > n_0 \; |f(n)| \leq k \cdot |g(n)|$

Expression	Dominant term(s)	$O(\dots)$
$5 + 0.001n^3 + 0.025n$	$0.001n^3$	$O(n^3)$
$500n + 100n^{1.5} + 50n \log_{10} n$	$100n^{1.5}$	$O(n^{1.5})$
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$	$2.5n^{1.75}$	$O(n^{1.75})$
$n^2 \log_2 n + n(\log_2 n)^2$	$n^2 \log_2 n$	$O(n^2 \log n)$
$n \log_3 n + n \log_2 n$	$n \log_3 n$, $n \log_2 n$	$O(n \log n)$
$3 \log_8 n + \log_2 \log_2 \log_2 n$	$3 \log_8 n$	$O(\log n)$
$100n + 0.01n^2$	$0.01n^2$	$O(n^2)$
$0.01n + 100n^2$	$100n^2$	$O(n^2)$
$2n + n^{0.5} + 0.5n^{1.25}$	$0.5n^{1.25}$	$O(n^{1.25})$
$0.01n \log_2 n + n(\log_2 n)^2$	$n(\log_2 n)^2$	$O(n(\log n)^2)$
$100n \log_3 n + n^3 + 100n$	n^3	$O(n^3)$
$0.003 \log_4 n + \log_2 \log_2 n$	$0.003 \log_4 n$	$O(\log n)$

Tipiškai žymimas tikslus viršutinis rėžis (!) blogiausiam atvejui (!)

Algoritmų optimizavimas

Geras algoritmas yra geriau už gerą kompiuterį



n	$\log_2 n$	$n \log_2 n$	n^2	n^3	2^n
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4096	65536
32	5	160	1024	32768	4294967296
128	7	896	16384	2097152	3.4×10^{38}
1024	10	10240	1048576	1073741824	1.8×10^{308}
65536	16	1048576	4294967296	2.8×10^{14}	Forget it!

Iteraciniai algoritmai

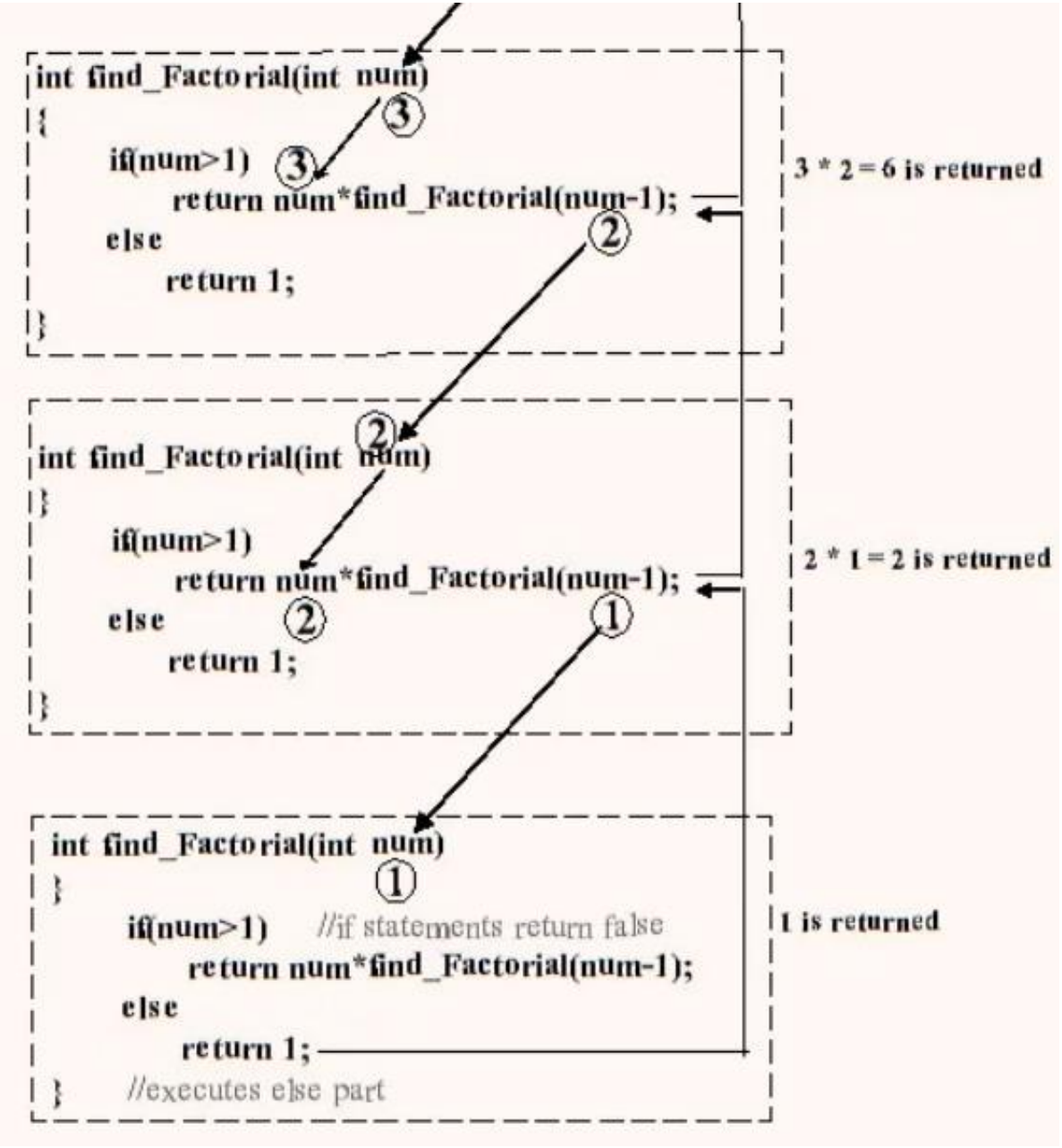
Blokas kartojamas, kol (ne)teisinga sąlyga.

```
#include<stdio.h>
int main()
{
    int n,fact=1;
    printf("enter the number\n");
    scanf("%d",&n);
    while(n)
    {
        fact=fact*n;
        n--;
    }
    printf("Factorial:%d",fact);
}
```

Rekursiniai algoritmai

Funkcijos viduje kreipiamasi į ją pačią.

```
int find_Factorial(int num) //define the function according to declaration
{
   if(num<1)
      return 1;
   else
      return num*find_Factorial(num-1);//find_Factorial function calls itself
}
```


int find_Factorial(int num)
{
if(num>1)
return num*find_Factorial(num-1);
else
return 1;
}
3 * 2 = 6 is returned
int find_Factorial(int num)
{
if(num>1)
return num*find_Factorial(num-1);
else
return 1;
}
2 * 1 = 2 is returned
int find_Factorial(int num)
{
if(num>1) //if statements return false
return num*find_Factorial(num-1);
else
return 1;
} //executes else part
1 is returned

Iteraciniai vs rekursiniai algoritmai

Baigtinumo problema: amžinas ciklas vs amžina rekursija

Rekursija

- gali supaprastinti realizaciją (skaldyk ir valdyk)
- reikalauja daugiau atminties (ar reikia įsiminti praeitį?)
- tipiškai lėtesnė (funkcijų kreipiniai)

Iteracijos

- efektyvios (nereikia papildomos atminties ar f-jų kreipinių)
- gali būti sudėtingesnės (skaitomumas ir įgyvendinimas)

Nuosekli (tiesinė) paieška

Nuosekliai ieškomas elementas duomenų rinkinyje

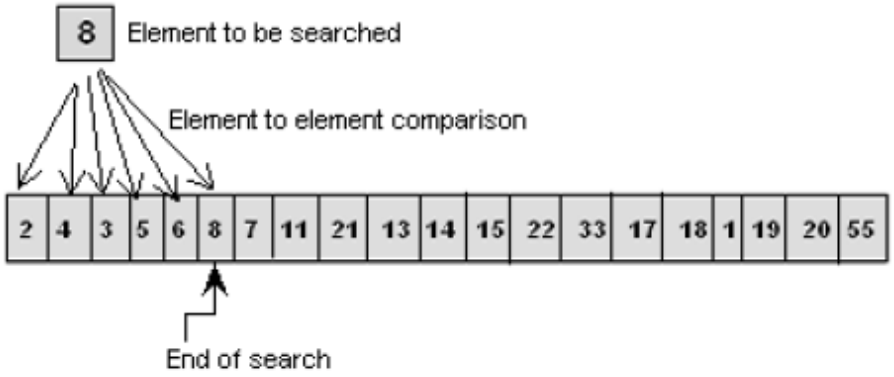


Fig.7.1(a) Successful search

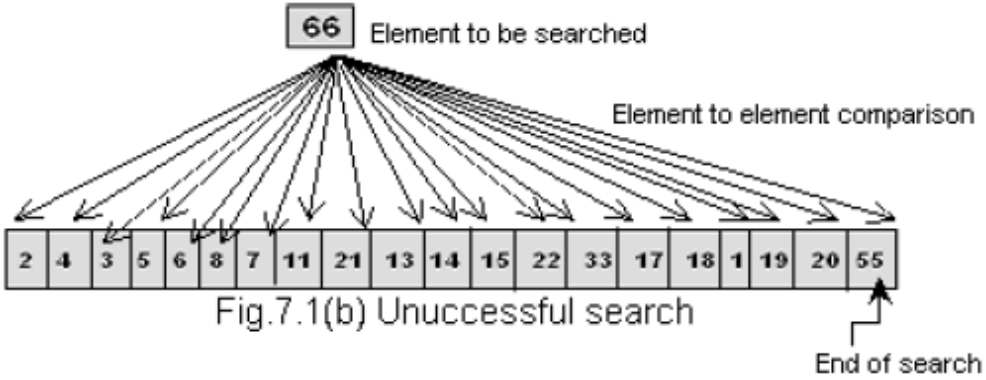


Fig.7.1(b) Unuccessful search

Nuosekli (tiesinė) paieška

Viena iš galimų algoritmo realizacijų. Sudėtingumas?

```
int linearsearch (int a [ ], int first, int last, int key)
{
        for (int i = first; i < = last; i ++)
        {
                if (key = = a [i])
                {
                return i;          // successfully found the
                }          // key and return location
        }
        return – 1;                  // failed to find key element
}
```

Nuosekli (tiesinė) paieška

Kitokia realizacija (rekursija)? Privalumai, trūkumai. Sudėtingumas?

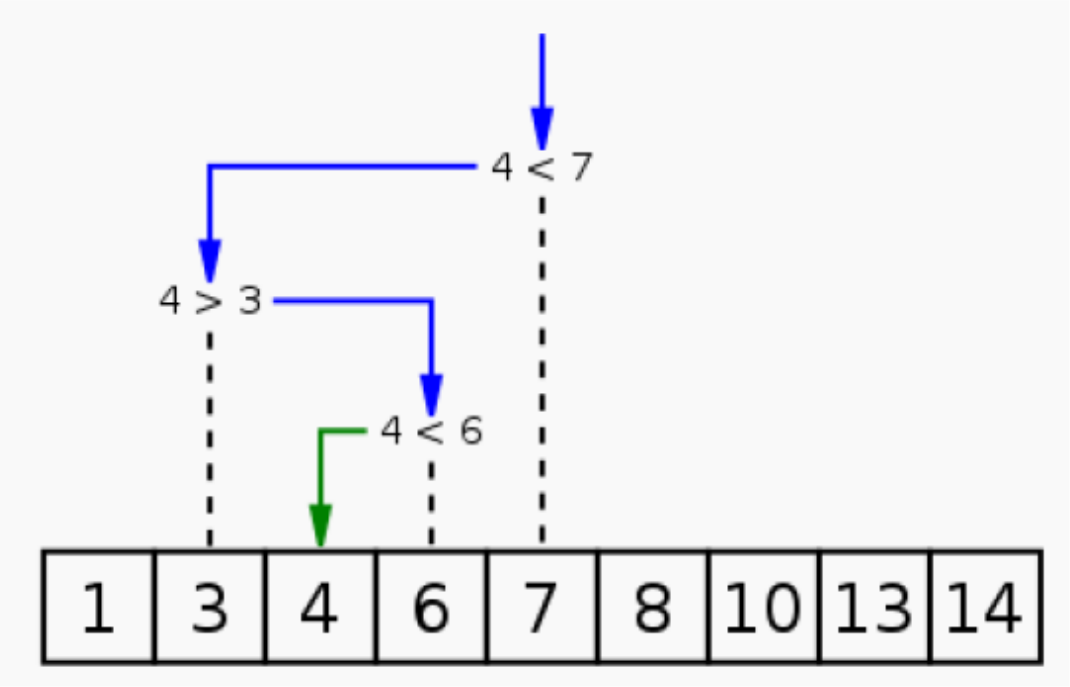
ALGORITHM 5 A Recursive Linear Search Algorithm.

procedure *search*(i, j, x: i, j, x integers, $1 \leq i \leq j \leq n$)
if $a_i = x$ **then**
    return i
else if $i = j$ **then**
    return 0
else
    return *search*($i + 1, j, x$)
{output is the location of x in $a_1, a_2, \ldots, a_n$ if it appears; otherwise it is 0}

Dvejetainė paieška

Žaidimas „atspėk skaičių“...

leškoma surikiuotame masyve – tai papildoma informacija, leidžianti pagreitinti paiešką.



Dvejetainė paieška

Iteracijomis paremta algoritmo versija. Sudėtingumas?

Pseudocode (using iteration)

```
BinarySearch(list[], min, max, key)
while min ≤ max do
   mid = (max+min) / 2
   if list[mid] >key then
      max = mid-1
   else if list[mid] <key then
      min = mid+1
   else
      return  mid
   end if
end while
return  false
```

Dvejetainė paieška

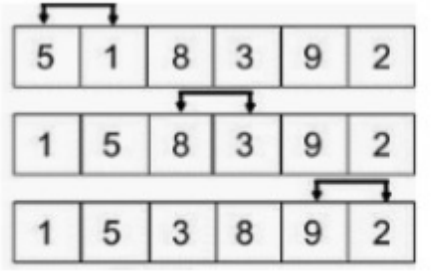
Rekursija paremta algoritmo versija. Sudėtingumas?

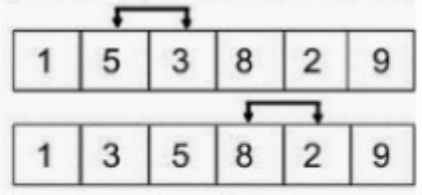
Pseudocode (using recursion)

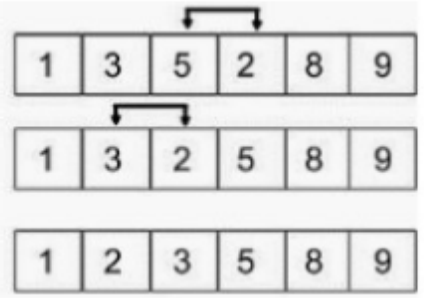
```
BinarySearch(list[], min, max, key)
if max <min then
   return  false
else
   mid = (max+min) / 2
   if list[mid] >key then
      return  BinarySearch(list[], min, mid-1, key)
   else if list[mid] <key then
      return  BinarySearch(list[], mid+1, max, key)
   else
      return  mid
   end if
end if
```

Burbuliuko rikiavimo metodas

Algoritmo idėja





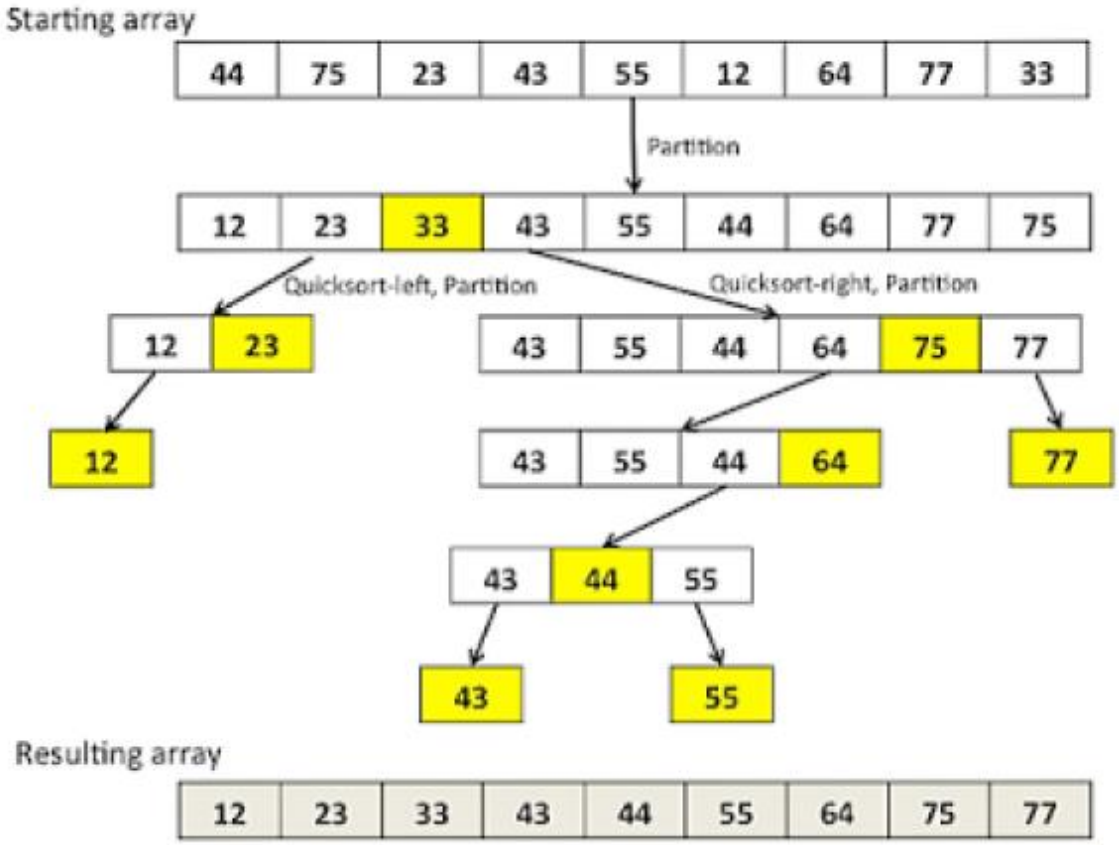


Viena iš galimų realizacijų. Sudėtingumas? Optimizavimo galimybės?

```
for i ← 1 to length[A]
    do for j ← length[A] downto i + 1
        do if A[j] < A[j -1]
            then exchange A[j] ↔ A[j-1]
```

Greitojo rikiavimo metodas

Algoritmo idėja



Greitojo rikiavimo metodas

Algoritmo realizacija, paremta rekursija. Sudėtingumas?

Pseudocode for quicksort

```
QUICKSORT(A, p, r)
  if p < r
    then q ← PARTITION(A, p, r)
      QUICKSORT(A, p, q–1)
      QUICKSORT(A, q+1, r)
```

Initial call: QUICKSORT(A, 1, n)

$O(b \cdot y \cdot e)$