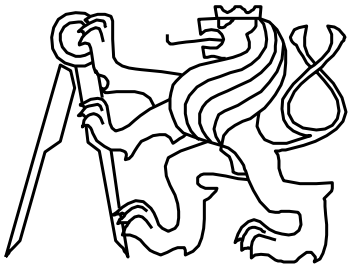
Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.
Praha & EU: Investujeme do vaší budoucnosti

Funkce, intuitivní chápání složitosti



BI-PA1 Programování a algoritmizace 1, ZS 2012-2013
Katedra teoretické informatiky
© Miroslav Balík
Fakulta informačních technologií
České vysoké učení technické

Funkce

- definice funkce
- parametry funkce
- deklarace funkce
- variace na téma největší společný dělitel
- procedury
- výstupní parametry
- přidělování paměti

Funkce – faktoriál

```
/* prog5-1a.c */   /* vypocet faktorialu */
#include <stdio.h>
 #include <stdlib.h>
int main(void) {
      int n;
      int i = 1, f = 1;
      printf("zadejte prirozene cislo: ");
      scanf("%d", &n);
      if (n<1) {
                printf("%d neni prirozene cislo\n", n);
                 return 0;
      }
      while (i<n) {
                i = i+1;
                f = f*i;
      }
      printf("%d! = %d\n", n, f);
      return 0;
}
```

čtení přirozeného čísla

výpočet faktoriálu

Faktoriál pomocí funkcí

- Funkce pro čtení přirozeného čísla

```
int ctiPrirozene(void) {
        int n;
        printf("zadejte prirozene cislo: ");
        scanf("%d", &n);
        if (n<1) { printf("%d neni prirozene cislo\n", n); exit(0);  }
        return n;
}
```

  - Hlavička funkce

```
int ctiPrirozene(void)
```

vyjadřuje, že funkce nemá parametry a že výsledkem volání funkce je hodnota typu *int*

- `return n;`

předepisuje návrat z funkce, výsledkem volání je hodnota *n*

- Příklad volání funkce:

```
int n = ctiPrirozene();
```

Faktoriál pomocí funkcí

- Funkce pro výpočet faktoriálu

```
int faktorial(int n) {
    int i = 1, f = 1;
    while (i<n) {
        i = i+1;
        f = f*i;
    }
    return f;
}
```

- Hlavička funkce vyjadřuje, že funkce má jeden parametr typu *int* a že výsledkem je hodnota typu *int*
- Příklad volání funkce
  - `int f = faktorial(4);  // vysledek se ulozi do f`

Faktoriál pomocí funkcí

```
/* prog5-1b.c */ /* vypocet faktorialu */
#include <stdio.h>
#include <stdlib.h>
int ctiPrirozene(void) {
   int n;
   printf("zadejte prirozene cislo: ");
   scanf("%d", &n);
   if (n<1) {
    printf("%d neni prirozene cislo\n", n);
    exit(0);
    }
   return n;
   }
```

definice funkce

Proměnnou *n* ve funkci *main* lze vynechat:

```
    int main(void) {
       printf("%d! = %d\n", n, faktorial(n));
       ...
    }
```

Definice funkce

- Funkce je podprogram (zápis dílčího algoritmu), který vrací hodnotu (výsledek)
- Definici funkce tvoří

  hlavička funkce *tělo funkce*
- Hlavička funkce v jazyku C má tvar
  - *typ jméno*(*specifikace parametrů*)
    - *typ* je typ výsledku funkce
    - *jméno* je identifikátor funke
    - *specifikací parametrů* se deklarují parametry funkce, každá deklarace má tvar *typ_parametru* *jméno_parametru*

      a oddělují se čárkou
    - nemá-li funkce parametry, specifikace parametrů je void
    - Tělo funkce je složený příkaz nebo blok, který se provede při volání funkce
- Tělo funkce musí dynamicky končit příkazem

  return *x*;

  kde *x* je výraz, jehož hodnota je výsledkem volání funkce

Parametry funkce

- Parametry funkce jsou lokální proměnné funkce, kterým se při volání funkce přiřadí hodnoty skutečných parametrů.
- Jestliže parametr funkce je typu *T*, pak přípustným skutečným parametrem je výraz, jehož hodnotu lze přiřadit proměnné typu *T* - tedy stejná podmínka, jako u přiřazení.

```
/* prog5-1c.c */
#include <stdio.h>
#include <stdlib.h>
int max(int x, int y) {
    if (x>y) return x;
    return y;
}
int main(void) {
    int a = 10, b = 20;
    printf("%d\n", max(a, b));
    printf("%d\n", max(32.4, b));
    return 0;
}
```

Parametry funkce

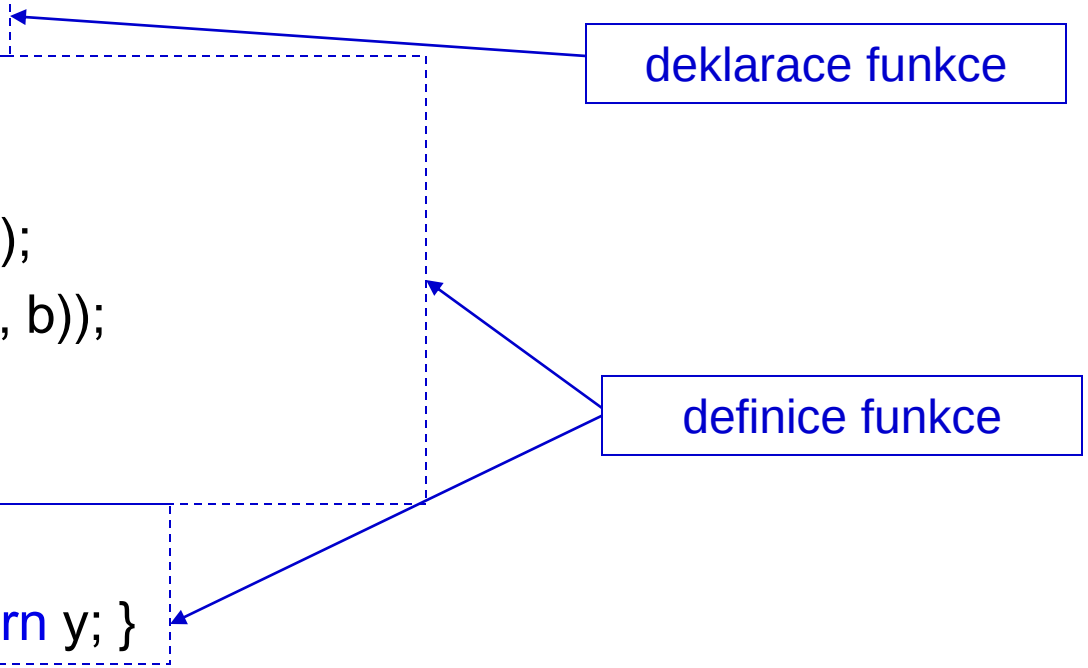
- Parametry funkce předávají vstupní data algoritmu, který funkce realizuje
- Častá chyba začátečníka: funkce, která čte hodnoty parametrů pomocí operace vstupu dat

```
int max(int x, int y) {
   scanf("%d%d", &x, &y); // nesmyslný příkaz
   if ( x > y ) return x;
   else return y;
}
```

Deklarace funkce

- Deklarace funkce je tvořena hlavičkou funkce a je zakončena středníkem
- Deklarací je funkce zavedena a může být použita

```
/* prog5-1d.c */
...
int max(int x, int y);
int main(void)
{   int a = 10, b = 20;
    printf("%d\n", max(a, b));
    printf("%d\n", max(32.4, b));
    return 0;
}
int max(int x, int y)
{ if (x>y) return x; else return y; }
```



- Definice funkce může být v jiném souboru (uvidíme později)

Volání funkce

- Funkci *f* můžeme vyvolat z funkce *g*, jestliže ve funkci *g* je známa hlavička funkce *f*
- Hlavička funkce se zadá buď deklarací nebo definicí funkce. Zopakujme, že hlavičkou funkce je dáno jméno (identifikátor) funkce, počet a typy parametrů a typ výsledku funkce
- Funkci *f*, která má *n* parametrů a vrací výsledek typu *T*, můžeme vyvolat *zápisem funkce*, což je výraz ve tvaru
  - $f(p_1, p_2, \ldots, p_n)$

    kde $p_1, p_2, \ldots, p_n$ jsou výrazy udávající skutečné parametry

  Hodnoty skutečných parametrů se přiřadí parametrům funkce podle pravidel pro přiřazení a pak se provede tělo funkce. Hodnotou zápisu funkce je výsledek funkce (zadaný v příkazu return).
- Zápis funkce obvykle používáme jako výraz, tzn. v kontextu, ve kterém je uvedeno, co s výsledkem funkce udělat (např. přiřadit nějaké proměnné).
- Ze zápisu funkce uděláme příkaz, zakončíme-li ho středníkem (výsledek funkce se pak nepoužije – zapomene se)

Další příklad funkce

- Funkce pro zjištění, zda daný rok je přestupný

```
/* prog5-1e.c */
#include <stdio.h>
#include <stdlib.h>
int jePrestupny(int rok) {
    if (rok%4==0 && (rok%100!=0 || rok%400==0)) return 1;
    else return 0;
}
int main(void) {
    int rok;
    printf("zadejte rok: ");
    scanf("%d", &rok);
    printf("rok %d ", rok);
    if (jePrestupny(rok)) printf("je prestupny\n");
    else printf("neni prestupny\n");
    return 0;
}
```

Funkce pro výpočet NSD

- Jednoduchý algoritmus výpočtu nsd jsme již uvedli

- Efektivnější algoritmus lze sestavit na základě těchto vztahů:

  je-li $x = y$, pak $nsd(x, y) = x$

  je-li $x > y$, pak $nsd(x, y) = nsd(x-y, y)$

  je-li $x < y$, pak $nsd(x, y) = nsd(x, y-x)$

- Řešení 2:

```
int nsd(int x, int y) {
    while (x!=y)
            if (x>y) x = x-y;
            else y = y-x;
    return x;
}
```

Funkce pro výpočet NSD

- Do těla cyklu vnoříme místo podmíněného příkazu pro jediné zmenšení hodnoty *x* nebo *y* dva cykly pro opakované zmenšení hodnot *x* a *y*
- Řešení 3:

```
int nsd(int x, int y) {
    while (x!=y) {
      while (x>y) x = x-y;
      while (y>x) y = y-x;
    }
    return x;
}
```

Euklidův algoritmus pro výpočet NSD

- Vnitřní cykly řešení 3 počítají nenulový zbytek po dělení většího čísla menším
- Pro výpočet zbytku po dělení slouží operátor %
- Euklidův algoritmus lze slovně formulovat takto: určíme zbytek po dělení daných čísel, zbytkem dělíme dělitele a určíme nový zbytek, až dosáhneme nulového zbytku; poslední nenulový zbytek je nsd
- Řešení 4:

```
int nsd(int x, int y) {
    int zbytek = x%y;
    while (zbytek!=0) {
      x = y;
      y = zbytek;
      zbytek = x%y;
    }
    return y;
}
```

Procedury

- Procedura je podprogram (zápis dílčího algoritmu), který nevrací žádnou hodnotu
- V jazyku C se procedury definují jako funkce, jejichž typ výsledku je *void*
- Proceduru je možno dynamicky ukončit kdekoliv příkazem

```
return;
```

- Příkaz return není nutný, procedura končí po provedení posledního příkazu
- Příklad procedury: výpis většího ze dvou celých čísel

```
void vypisMax(int x, int y) {
    if (x>y) printf("%d", x);
    else printf("%d", y);
}
```

- Dále se budeme držet terminologie jazyka C: řekneme-li *funkce*, myslíme tím: jak funkci, která vrací hodnotu, tak proceduru, která hodnotu nevrací (typ výsledku je void).

Vstupní a výstupní parametry

- Parametrem funkce je obvykle hodnota, která slouží jako vstup dílčímu algoritmu, který je funkcí realizován. Tyto parametry nazýváme *vstupními parametry*
- Všechny předchozí funkce měly vstupní parametry
- Většina programovacích jazyků umožňuje, aby funkce (procedura) měla též *výstupní parametr*
- Výstupní parametr umožňuje, aby funkce (procedura) přiřadila hodnotu do proměnné, která je dána skutečným parametrem.
- Příklad (v pseudojazyku): procedura, která ze dvou vstupních parametrů *x* a *y* (celá čísla) uloží menší číslo do proměnné dané výstupním parametrem *mensi* a vetší číslo do proměnné dané výstupním parametrem *vetsi*

```
proc minMax(in int x, in int y, out int mensi, out int vetsi) {
    if (x<y) { mensi = x; vetsi = y; }
    else { mensi = y; vetsi = x; }
}
```

Výstupní parametry

- Výstupní parametry se v jazyku C realizují tak, že funkci (proceduře) se předá adresa proměnné, kterou má funkce (procedura) změnit
- Operátor, který dodá adresu proměnné, už známe. Je to &, který používáme ve volání funkce *scanf*
- Proměnná (parametr) *p*, jejíž hodnotou může být adresa proměnné typu *T,* se deklaruje zápisem
  - *T *p*

Typ *T** se nazývá typem *ukazatel na T*

- Příklad: deklarace procedury, která ze dvou vstupních parametrů *x* a *y* (celá čísla) uloží vetší číslo do proměnné dané výstupním parametrem *vetsi* a menší číslo do proměnné dané výstupním parametrem *mensi*

```
void minMax(int x, int y, int *mensi, int *vetsi);
```

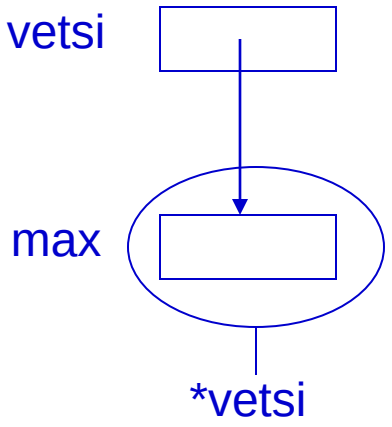
- Jak tuto proceduru vyvolat:

```
int a, b, min, max;
...
maxMin(a, b, &min, &max);
```

Výstupní parametry

- Pro přístup k proměnné, jejíž adresa je v proměnné (parametru) typu ukazatel, slouží unární operátor * (dereference)
- Příklad (prog5-2a.c):

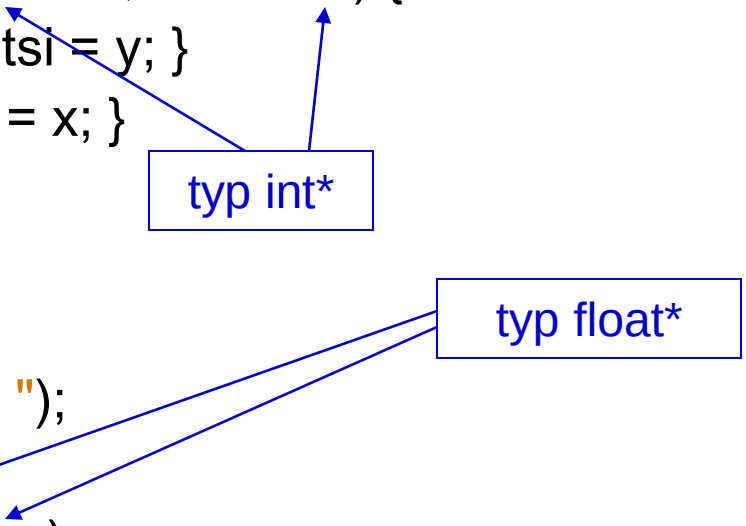
```
void minMax(int x, int y, int *mensi, int *vetsi) {
    if (x<y) { *mensi = x; *vetsi = y; }
    else { *mensi = y; *vetsi = x; }
}
int main(void) {
    int a, b, max, min;
    printf("zadejte dve cela cisla: ");
    scanf("%d%d", &a, &b);
    minMax(a, b, &min, &max);
    printf("min = %d, max = %d\n", min, max);
    ...
}
```



Parametry typu ukazatel

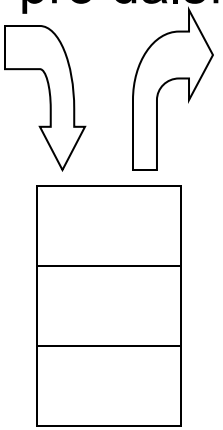
- Skutečným parametrem pro parametr typu *T** musí být adresa proměnné typu *T*. Je-li proměnná jiného typu, překladač vypíše varovné hlášení (warning), program se přeloží ale nebude správně fungovat
- Příklad (prog5-2b.c):

```
void minMax(int x, int y, int *mensi, int *vetsi) {
        if (x<y) { *mensi = x; *vetsi = y; }
        else { *mensi = y; *vetsi = x; }
}
int main(void) {
        float a, b, max, min;
        printf("zadejte dve cisla: ");
        scanf("%f%f", &a, &b);
        minMax(a, b, &min, &max);
        printf("min = %f, max = %f\n", min, max);
        ...
}
```



Přidělování paměti proměnným

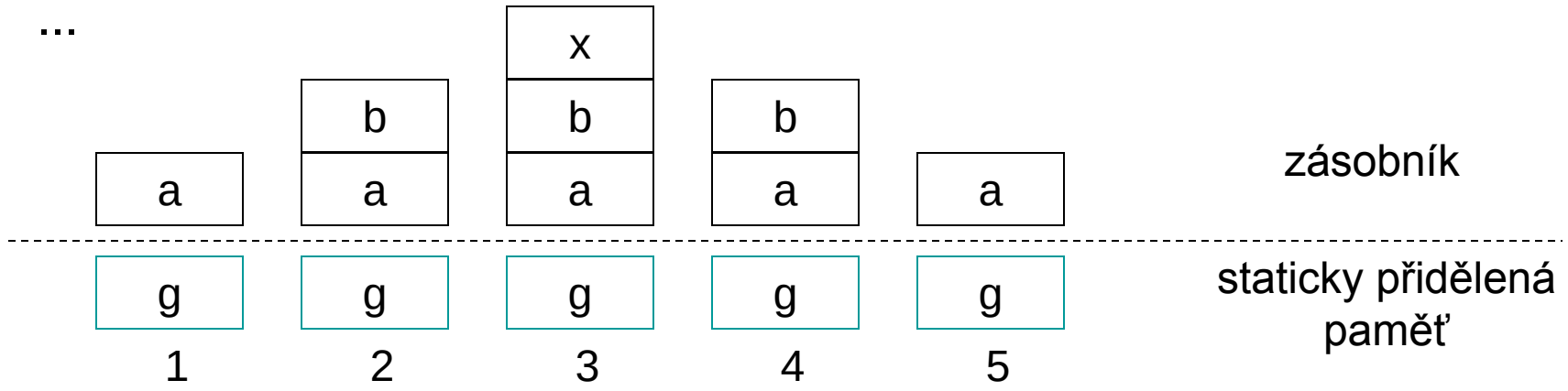
- Přidělením paměti proměnné rozumíme určení adresy umístění proměnné v paměti počítače
- Poznali jsme dva druhy proměnných:
  - globální proměnné (deklarované mimo funkci)
  - lokální proměnné funkcí (deklarované v bloku funkce a parametry funkce)
- Globálním proměnným se přidělí paměť při spuštění programu (statické přidělení paměti) a zůstane jim přidělena až do ukončení běhu programu
- Lokálním proměnným a parametrům funkce se paměť přidělí při volání funkce (dynamické přidělení paměti) a zůstane jim přidělena jen do návratu z funkce (po návratu se přidělené adresy uvolní pro další použití)
- Úseky paměti přidělované lokálním proměnným a parametrům tvoří tzv. zásobník (stack): úseky se přidávají a odebírají, přičemž se vždy odebere naposledy přidaný úsek (strategie LIFO)



zásobník

Přidělování paměti proměnným

```
int g;
void f(void) {
        int b; // 2
        h(4); // 4 ...
}
int h(int x) {
 ... // 3
}
int main(void) {
int a; // 1
f(); // 5
...
}
```



Problém: prvočísla

- Zjistěte počet **prvočísel** menších než zadané číslo ***n*** a určete čas, který na to budete potřebovat
- **prvočíslo: přirozené číslo** větší než 1, které je beze zbytku dělitelné **pouze** jedničkou a samo sebou.
- **přirozené číslo: kladné celé číslo** větší než 0
- Algoritmus:

  Pro každé číslo menší než n rozhodněte, zda jde o prvočíslo či ne

  raději pouze: pro každé **přirozené**

  Nasčítejte prvočísla

  podproblémy :

  1. zjištění zda dané číslo je prvočíslo
  2. měření času

funkce

Příklad: prvočísla I

Dle definice: Přirozené číslo **n** je **prvočíslo**, právě tehdy když jej beze zbytku dělí pouze číslo **n** a číslo **1**

```
int isPrvocislo1(long long n) {
    long long i;
    if (n < 2) { return 0; }
    int pocetDelitelu = 0;
    for (i = 1; i <= n; i++) {
        if (n % i == 0) { pocetDelitelu++; }
    }
    if (pocetDelitelu == 2) { return 1; }
    else { return 0; }
}
```

jak dlouho poběží tato metoda pro číslo 100 000 000 000 003?
a jak dlouho pro 100 000 000 000 000?

Příklad: prvočísla II

- Vylepšení 1. - najdeme-li prvního dělitele jiného než čísla **1** a **n,** již to není prvočíslo

```
int isPrvocislo2(long long n) {
     long long i;
    if (n < 2) { return 0; }
    for (i = 2; i < n; i++) {
            if (n % i == 0) { return 0; }
    }
    return 1;
}
```

jak dlouho poběží tato metoda pro číslo 100 000 000 000 003?
a jak dlouho pro 100 000 000 000 000?

Příklad: prvočísla III

- Vylepšení 2. Číslo složené lze napsat jako součin dělitelů, např. 35=7*5; jedno je „menší“ a druhé „větší“

```
int isPrvocislo3(long long n) {
        long long i;
        if (n < 2) { return 0; }
        for (i = 2; i <= n / 2; i++) {
                if (n % i == 0) { return 0; }
        }
        return 1;
}
```

jak dlouho poběží tato metoda pro číslo 100 000 000 000 003?
a jak dlouho pro 100 000 000 000 000?

Příklad: prvočísla IV

- Vylepšení 2. Číslo složené lze napsat jako součin dělitelů, např. 35=7*5; jedno je „menší“ a druhé „větší“, pokud nejsou stejné jako 49 = 7*7


```
int isPrvocislo(long long n) {
    long long i;
    if (n < 2) { return 0; }
    for (i = 2; i <= sqrt(n); i++) {
        if (n % i == 0) { return 0; }
    }
    return 1;
}
```

jak dlouho poběží tato metoda pro číslo 100 000 000 000 003?
a jak dlouho pro 100 000 000 000 000?

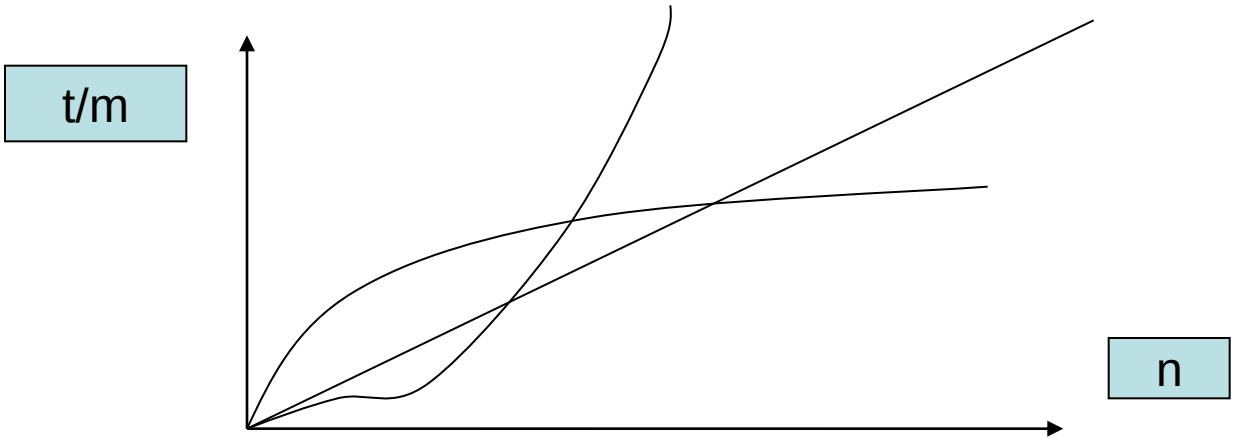
Příklad: prvočísla V

```
int isPrvocislo(long n) {
    long i;
    int max = (int)sqrt(n);
    if (n < 2) { return 0; }
    for (i = 2; i <= max; i++){
        if (n % i == 0) { return 0; }
    }
    return 1;
}
```

jak dlouho poběží tato metoda pro číslo 100 000 000 000 003?
a jak dlouho pro 100 000 000 000 000?

Příklad: prvočísla V - složitost

- **Složitost algoritmu určuje jeho časovou a prostorovou náročnost**



- **Určete složitost algoritmů (na základě odhadu času) pro všechny typy metod na zjištění, zda dané číslo je prvočíslo**