

1▲ Pro následující třídící algoritmus na následujících vstupních datech ověřte jeho chování a spočtěte počet volání funkce swap a počet porovnání:

```
procedure bubblesort (A[1..n])
1.  for i := 1 to n - 1 do
2.      for j := 1 to n - i do
3.          if A[j] > A[j+1] then
4.              swap A[j], A[j+1];
```

a) $A = [5,3,1,6]$

b) $A = [2,3,7,9]$

2▲ Rozhodněte, zda jsou následující tvrzení pravdivá nebo nepravdivá:

a. $3n^5 - 16n + 2 \in O(n^5)$

b. $3n^5 - 16n + 2 \in O(n)$

c. $3n^5 - 16n + 2 \in O(n^{17})$

d. $3n^5 - 16n + 2 \in \Omega(n^5)$

e. $3n^5 - 16n + 2 \in \Theta(n^5)$

f. $3n^5 - 16n + 2 \in \Theta(n)$

g. $3n^5 - 16n + 2 \in \Theta(n^{17})$

3▲ Dokažte, že platí následující tvrzení:

1. $\log n \in O(n)$

2. $2^{n+1} \in O\left(\frac{3^n}{n}\right)$

4▲ Seřadte následující funkce podle rychlosti jejich růstu:

$$n^2 + \log n$$

$$\log \log n$$

$$\left(\frac{3}{2}\right)^n$$

$$n$$

$$n!$$

$$6$$

$$\log n!$$

$$7n^5 - n^3 + n$$

$$2^n$$

$$n \log n$$

$$n^n$$

$$n^2$$

$$\log n$$

$$\log^{14} n$$

5▲ Pro exaktní řešení problému obchodního cestujícího je znám algoritmus v $O(2^n)$. Při jeho řešení na starém počítači trval výpočet pro 36 měst téměř jeden den. Nyní máme k dispozici nový počítač, který je 1000-krát rychlejší. Určete, kolik měst můžeme zpracovat, aby výpočet nepřesáhl jeden den.

6▲ Určete časovou složitost následujícího algoritmu:

```
procedure bubblesort (A[1..n])  
1. for i := 1 to n - 1 do  
2.     for j := 1 to n - i do  
3.         if A[j] > A[j+1] then  
4.             swap A[j], A[j+1];
```

7▲ Zkuste modifikovat algoritmus bubblesort tak, aby se zlepšila jeho složitost na příznivých datech (nápodvěda: nejvíce příznivými data jsou již setřizené posloupnosti).

8▲ Určete časovou složitost následujícího algoritmu, který vrátí součin dvou přirozených čísel y a z .

```
function multiply (y, z)
```

1. $x := 0;$
2. while $z > 0$ do
3. if z is odd then $x := x + y;$
4. $y := 2y;$
5. $z := \lfloor \frac{z}{2} \rfloor;$
6. return $(x);$

9▲ Určete časovou složitost algoritmu lineárního vyhledávání pro vyhledání klíče (vrací index nalezeného prvku v poli D)

```
function ls (n, k: Int, D:array[Int] of Int): Int;  
    { n je pocet prvku v poli D, k je hledaný klíč}  
    var index: Int;
```

1. index := -1;
2. for i := 1 to n do
3. if k = D[i] then index := i;
4. if (index = -1) then "prvek nenalezen"
5. else return(index);

10▲ Určete časovou složitost algoritmu binárního vyhledávání pro vyhledání klíče v setřazeném poli D

D: array[Int] of Int;

```
function bs (l, r, k: Int ): Int
    { l, r jsou levý a pravý konec pole D, k je hledaný klíč }
    var m, index: Integer;
1.  if l > r then return(-1) {nenalezeno}
        else begin
2.      m := (l + r) div 2;
3.      if k < D[m] then return(bs(l, m-1, k))
4.          else if k > D[m] then return (bs(m+1, r, k))
5.          else return(m);
        end
    end
```

11▲ U následujících dvou algoritmů počítajících mocninu čísla určete jejich časovou složitost.

```
function power1 (z, n)    {Předpokládá se  $z > 0, n \geq 0$  }
```

```
1.  r := 1;
2.  for i := 1 to n do
3.      r := r * z;
4.  return(r);
```

```
function power2 (z, n)
```

```
1.  r := 1;
2.  t := z;
3.  if n = 0 then return(1);
4.  while n > 0 do
4.      if n is odd then
6.          r := r * t;
7.          n := n div 2;
8.          t := t * t;
      endwhile
9.  return(r);
```

12▲ Tabulka časů výpočtu algoritmů o složitostech $\log n$, n , n^2 , 2^n a n^n pro vstup délky 10, 20, 50 a 1000. Předpokládejme, že jedna iterace algoritmu trvá $1\mu\text{s}$.

	10	20	50	1000
$\log n$	0,000001s	0,000001s	0,000002s	0,000003s
n	0,00001s	0,00002s	0,00005s	0,001s
n^2	0,0001s	0,0004s	0,0025s	1s
2^n	0,001024s	1,048576s	35,7 let	$3,4 \cdot 10^{287}$ let*
n^n	2,8 hodiny	$3 \cdot 10^{12}$ let*		

* Stáří vesmíru je odhadováno na $13,7 \cdot 10^9$ let.