▲ Pro následující třídicí algoritmus na následujících daných vstupních datech ověřte jeho chování a spočítejte počet porovnání a počet volání funkce swap:

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

a) $A = [5, 3, 1, 6]$ b) $A = [2, 3, 7, 9]$

▲ Rozhodněte, zda jsou následující tvrzení pravdivá nebo nepravdivá:

1. $3n^5 - 16n + 2 \in O(n^5)$
2. $3n^5 - 16n + 2 \in O(n)$
3. $3n^5 - 16n + 2 \in O(n^{17})$
4. $3n^5 - 16n + 2 \in \Omega(n^5)$
5. $3n^5 - 16n + 2 \in \Theta(n^5)$
6. $3n^5 - 16n + 2 \in \Theta(n)$

7. $3n^5 - 16n + 2 \in \Theta(n^{17})$

▲ Pro následující funkce f a g nakreslete jejich graf a určete dvojici konstant c a n_0, která je svědkem toho, že platí $f \in O(g)$, resp. $g \in \Omega(f)$. Kolik existuje takových dvojic?

1. $f(n) = \frac{1}{2}n + 5$; $g(n) = n^2 - 4n + 7$
2. $f(n) = \log_2 n$; $g(n) = 2^{n-1} - 2$

▲ Dokažte, že platí následující tvrzení:

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

▲ Seřaďte následující funkce podle rychlosti jejich růstu:

$n^2 + \log n$	$7n^5 - n^3 + n$	n^2
$\log \log n$	2^n	$\log n$
$(\frac{3}{2})^n$	$n \log n$	$n!$
n	n^n	6
$\log n!$	$\log^{14} n$	

▲ 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.

▲ Určete časovou složitost následujících algoritmů vzhledem k velikosti n:

```
10 procedure printer1 (A[1..n])
11 1.   for i := 1 to 100000 do
12 2.       if i < n then print A[i];
13
14 procedure printer2 (A[1..n])
15 1.   for i := 1 to n - 1 do
```

```
16 2.        for j := i to i + 1 do
17 3.            print A[j];
18
19 procedure bubblesort (A[1..n])
20 1.   for i := 1 to n − 1 do
21 2.        for j := 1 to n − i do
22 3.            if A[j] > A[j+1] then
23 4.                swap A[j], A[j+1];
```

▲ Zkuste modifikovat algoritmus bubblesort tak, aby se zlepšila jeho složitost na příznivých datech (nápověda: nejvíce příznivými daty jsou již setřízené posloupnosti).

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

```
10 function multiply (y, z)
11 1.   x := 0;
12 2.   while z > 0 do
13 3.       if z is odd then x := x + y;
14 4.       y := 2y;
15 5.       z := \(\lfloor\frac{z}{2}\rfloor\);
16 6.   return (x);
```

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

```
10 function ls (n, k: Int, D:array[Int] of Int): Int;
11     \{ n je pocet prvku v poli D, k je hledaný klíč\}
12     var index: Int;
13
14 1.   index := -1;
15 2.   for i := 1 to n do
16 3.       if k = D[i] then index := i;
17 4.   if (index = -1) then "prvek nenalezen"
```

```
18 5.        else return(index);
```

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

```
10 D: array[Int] of Int;
11
12 function bs (l, r, k: Int ): Int
13     \{ l, r jsou levý a pravý konec pole D, k je hledaný klíč \}
14     var m, index: Integer;
15 1.  if l > r then return(−1) \{nenalezeno\}
16         else begin
17 2.          m := (l + r) div 2;
18 3.          if k < D[m] then return(bs(l, m−1, k))
19 4.              else if k > D[m] then return (bs(m+1, r, k))
20 5.                  else return(m);
21         end
```

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

```
10 function power1 (z, n)     \{Předpokládá se z > 0, n \(\geq\) 0 \}
11 1.   r := 1;
12 2.   for i := 1 to n do
13 3.       r := r * z;
14 4.   return(r);
15
16 function power2 (z, n)
17 1.   r := 1;
18 2.   t := z;
19 3.   if n = 0 then return(1);
20 4.   while n > 0 do
21 4.       if n is odd then
22 6.           r := r * t;
23 7.       n := n div 2;
24 8.       t := t * t;
25      endwhile
26 9.   return(r);
```

▲ Co počítají následující dvě funkce? Jaká je jejich složitost?

```
10 function kralicek(n : integer)
11 begin
12  if n < 0 then return "Chyba"
13  if n < 2 then return n;
14  else
15    return kralicek(n-1) + kralicek(n-2);
16  end if;
17 end;
18
19 function mysicka(n : integer)
20 begin
21  if n < 0 then return "Chyba"
22  first, second, tmp: integer;
23  first  := 0;
24  second := 1;
25  for i in 1..n do
26    tmp    := first + second;
27    first  := second;
28    second := tmp;
29  end
```

```
30   return first;
31 end;
```

▲ 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 1μs.

	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 $13,7.10^9$ let.