Příkladky na složitost

1. Je dáno pole *n* celých čísel. Spočítejte střední hodnotu a rozptyl.
   Pro připomenutí: střední hodnota $\bar{x} = \frac{1}{n}\sum_{i=0}^{n-1} x_i$ a rozptyl $s^2 = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \bar{x})^2$.

```
int suma = 0;
for (int i = 0; i < n; i++)
   suma += pole[i];                                  //T1
double mean = ((double)suma) / n;

double dsuma = 0;
for (int i = 0; i < n; i++)
   dsuma += (pole[i] - mean)*(pole[i] - mean);     //T2
double dev = dsuma / n;
```

   Složitost: operace „`suma += pole[i];`“ trvá t_1 ms, operace „`dsuma += (pole[i] - mean)*(pole[i] - mean);` “ t_2 ms. Obě operace jsou volány *n* krát, tudíž celkový čas $T(n) = n*(t_1 + t_2)$. Z definice $O(f(n)) = g(n)$ pokud $\exists c \in R, \exists n_0 \in N, c>0, n_0>0$ takové, že $c*f(n) <= g(n)$ pro $n >= n_0$, vyplývá, že složitost je O(n) a platí pro $n_0=1$: stačí zvolit $c= t_1 + t_2$. Analogicky lze odvodit složitosti Θ a Ω.

2. V šachové partii dělá počítač jen tahy, které jsou vyhodnoceny, že jsou pro něj nejvýhodnější. Součástí vyhodnocení je stanovení míry ohrožení protivníka. Nechť máme šachovnici *n* x *n* reprezentovanou maticí celých čísel, přičemž kladná čísla jsou vyhrazena pro figury počítače, záporné pro figury protihráče (0 znamená, že tam nikdo nestojí) a absolutní hodnota čísla vyjadřuje důležitost šachové figury stojící na daném místě. Spočtěte míru ohrožení protivníka věží na pozici [x, y].

```
int ohrozeni = 0;
int i = x + 1;
while (i < n && matice[i][y] == 0)
   i++;
if (i < n && matice[i][y] < 0)
   ohrozeni += -matice[i][y];

i = x - 1;
while (i >= 0 && matice[i][y] == 0)
   i--;
if (i >= 0 && matice[i][y] < 0)
   ohrozeni += -matice[i][y];

int j = y + 1;
while (j < n && matice[x][j] == 0)
   i++;
if (j < n && matice[x][j] < 0)
   ohrozeni += -matice[x][j];

j = x - 1;
while (j >= 0 && matice[x][j] == 0)
   j--;
if (j >= 0 && matice[x][j] < 0)
   ohrozeni += -matice[x][j];
```

3. Nechť je dána matice *n* x *n*. Pro každé nezáporné číslo na diagonálách zavolejte metodu pocitej(int prvek).

```
for (int i = 0; i < n; i++) {
   if (mat[i][i] >= 0)
         pocitej(mat[i][i]);
}

for (int i = 0; i < n; i++) {
   if (mat[n-i-1][n-i-1] >= 0)
         pocitej(mat[n-i-1][n-i-1]);
}
```

4. Při zobrazování 3D modelů (např. ve hrách) je třeba transformovat jednotlivé body z 3D do 2D. Provedení transformace se provádí vynásobením souřadnic bodu maticí transformace. Určete složitost pro obecný případ, kdy je matice *n* x *n* a vektor *n* x 1.

```
int[] trans_bod = new int[bod.length];
for (int i = 0; i < n; i++) {
   trans_bod[i] = 0;
   for (int j = 0; j < n; j++) {
         trans_bod[i] += bod[j]*mat[j][i];
   }
}
```

Složitost: vnější cyklus proveden *n*-krát, operace ve vnitřním cyklu také *n*-krát => n^2

5. Určete složitost řazení pole *n* prvků metodou Select Sort.

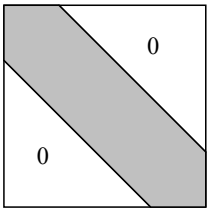
```
for (int i = 0;  i < pole.length - 1;  i++) {
      int iMin = i;
      for (int j = i + 1;  j < pole.length;  j++) {
        if (pole[j] < pole[iMin]) {          //T1
          iMin = j;
        }
      }
      if (i != iMin) {
        Prvek pom = pole[iMin];              //T2
        pole[iMin] = pole[i];                //T2
        pole[i] = pom;                       //T2
      }
}
```

Složitost: Nechť je čas potřebný pro provedení libovolné operace s výjimkou operací s řazenými hodnotami zanedbatelný; porovnání řazených hodnot vyžaduje čas T1 a přesun prvku čas T2. Vnější cyklus proveden *n*-1x, vnitřní cyklus se při první obrátce vnějšího provede *n*-1x, při další *n*-2x atd. až 1x, tudíž celkový čas potřebný pro Select Sort T(n)=3**m**T2 + [(*n*-1) + (*n*-2) + ... 1]*T1, kde *m* je počet, kolikrát byl nalezen nejmenší prvek na nesprávném místě (tj. i != iMin). Součet posloupnosti (*n*-1) + ... je *n**(*n*-1)/2, tudíž složitost T(*n*)=3**m**T2 + *n**(*n*-1)/2*T1. Nejlepší čas dostaneme v případě, že *m*=0, tj. vstupní pole prvků je již vzestupně seřazené, naopak nejhorší v případě, že *m*=*n*-1, tj. vstupní pole je seřazené sestupně. Složitost algoritmu pro libovolná vstupní data je tedy omezena zdola T(*n*) = *n**(*n*-1)/2*T1 a shora T(*n*)=3*(*n*-1)*T2 + *n**(*n*-1)/2*T1. Z definice „$T(n) = \Omega(f(n))$, existují-li kladné konstanty *c* a n_0 takové, že $0 \le cf(n) \le g(n)$ pro všechna $n \ge n_0$“ vyplývá T(*n*) = *n**(*n*-1)/2*T1 = $\Omega(n^2)$ – stačí zvolit *c*<T1/4 a n_0=2. Dále z definice „$T(n) = O(f(n))$, existují-li kladné konstanty *c* a n_0 takové, že $0 \le g(n) \le cf(n)$ pro všechna

$n \geq n_0$“ vyplývá T(n)=3*(n-1)*T2 + n*(n-1)/2*T1 = O(n^2) – stačí zvolit c>0 a n_0=1. Srovnáním a z definice „*g(n) = Θ(f(n))* existují-li kladné konstanty c_1, c_2 a n_0 takové, že $0 \leq c_1 f(n) \leq g(n) \leq c_2 f(n)$ pro všechna $n \geq n_0$“ dostáváme, že asymptotická složitost řazení je Θ(n^2) pro n_0=2, c_1<T1/4 a c_2>T1/4+3*T2/4.

6. Je dána pásová matice n x n, pás zasahuje do n/4 (viz obrázek). Vynásobte všechny prvky matice konstantou c.

```
for (int i = 0; i < n; i++) {
   for (int j = i; j < i + n/4; j++) {
         mat[i][j] *= c;
         mat[j][i] *= c;
   }
}
```



Složitost: třebaže řídící proměnná vnitřního cyklu je funkcí řídící proměnné vnějšího cyklu, vnitřní cyklus proběhne vždy n/4x => složitost je Θ(n^2)

7. Je dána pásová matice n x n, pás zasahuje do n/4 (viz příklad 6). Nastavte všechny prvky matice v obou trojúhelnících na náhodnou hodnotu.

```
Random rnd = new Random();
for (int i = n/4; i < n; i++) {
   for (int j = 0; j <= i – n/4; j++) {
         mat[i][j] = rnd.NextInt();          //T1
         mat[n-i][n-j] = rnd.NextInt();      //T1
   }
}
```

Složitost: vnější cyklus poběží ¾*n krát, vnitřní při první obrátce 1x, pak 2x ... až ¾*n, celková složitost T(n)=(1+2+...3/4*n)*2*T1 = (3/4*n+1)*3/4*n*T1 = (9/16*n^2+3/4*n)*T1 = Θ (n^2) – stačí zvolit n_0=1, c_1<21/16*T1 a c_2>21/16*T1.

8. Je dána množina n bodů ve 2D. Sestrojte konvexní polygon takový, že a) jehož vrcholy patří do podmnožiny zadaných bodů a b) žádný ze zadaných bodů neleží mimo polygon. Problém vyřešíme dekompozicí na menší problémy, který samostatně řešit již umíme: rekurzivně dělíme množinu na dvě části, sestrojíme polygony obou částí a poté provedeme sloučení těchto polygonů do jednoho. Dělení množiny se zastaví, když podmnožina má jediný bod, výsledkem pak bude sám bod.

Celkový čas algoritmu bude zřejmě silně závislý na čase potřebného pro sloučení. Čas sloučení bude určitě záviset na počtu vrcholů vstupních polygonů. Nejhorší případ tedy nastane, když všechny body jsou vrcholy výsledného polygonu (to bude v případě, že body leží na kružnici). Stanovme čas T(n) pro nejhorší případ. Nechť čas pro zpracování jednobodové podmnožiny je T1, pak T(n) = T(n/2) + T(n/2) + Tslouc(n); T(1) = T1. Začneme rekurentně dosazovat do formule: T(n) = Tslouc(n) + 2*T(n/2) = Tslouc(n) + 2*(Tslouc(n/2) + 2*T(n/4)) = Tslouc(n) + 2*(Tslouc(n/2) + 2*(Tslouc(n/4) + 2*T(n/8))) = Tslouc(n) + 2*(Tslouc(n/2) + 2*(Tslouc(n/4) + 2*(Tslouc(n/8) + 2*(....+ 2*(Tslouc(2) + 2*T(1)) ...)))). Provedeme roznásobení: T(n) = Tslouc(n) + 2*Tslouc(n/2) + 4*Tslouc(n/4) + ...n/2*Tslouc(2) + n*T1. Dá se ukázat, že čas na sloučení je lineární, tj. Tslouc(n) = n*T2. Dosazením dostaneme, že T(n) = n*T2*(1+1+...+1)+n*T1, přičemž těch jedniček je tam k a platí n/2=2^k, tedy k=log(n)-1. Výsledkem je:

T(n)=n*log(n)*T2+n*(T1-T2). Z definice O(f(n)) dostáváme, že složitost problému je O(n*log(n)) – volíme n_0=2 a c>T1