1.
Stabilní řazení je charakterizováno následujícím tvrzením

a) řazení nemá asymptotickou složitost větší než $\Theta(n \cdot \log(n))$
b) řazení nemá asymptotickou složitost menší než $\Theta(n \cdot n)$
c) řazení nemění pořadí prvků s různou hodnotou
d) řazení nemění pořadí prvků se stejnou hodnotou
e) kód řazení nemusí kontrolovat překročení mezí polí

Řešení
Stabilita řazení nesouvisí s rychlostí řazení, varianty a) a b) odpadají. Kdyby algoritmus neměnil pořadí prvků s různou hodnotou, nic by neseřadil, varianta c) nemá smysl. Kontrola mezí pole je záležitostí pouze implementační, s formulací algoritmu samotného nemá nic společného, varianta e) odpadá rovněž. Zbývá tak jen varianta d), jež je ostatně víceméně definicí stabilty.

2.
V určitém problému je velikost zpracovávaného pole s daty rovna rovna $2n^3 \cdot \log(n)$ kde n charakterizuje velikost problému. Pole se řadí pomocí Selection sort-u.
Asymptotická složitost tohoto algoritmu nad uvedeným polem je tedy

a) $\Theta(n^2)$
b) $\Theta(n^6 \cdot \log^2(n))$
c) $\Theta(n^3 \cdot \log(n))$
d) $\Theta(n^3 \cdot \log(n) + n^2)$
e) $\Theta(n^5 \cdot \log(n))$

Řešení
Máme-li velikost pole rovnu k, Selection sort jej zpracuje v čase $\Theta(k^2)$. Velikost našeho pole je
$k = 2n^3 \cdot \log(n)$, takže bude zpracováno v čase $\Theta(k^2) = \Theta((2n^3 \cdot \log(n))^2) =$
$= \Theta(4n^6 \cdot \log^2(n)) = \Theta(n^6 \cdot \log^2(n))$.
Platí tedy možnost b).

3.
Pole n různých prvků je uspořádáno od druhého prvku sestupně, první prvek má nejmenší hodnotu ze všech prvků v poli.
Zatrhněte všechny možnosti, které pravdivě charakterizují asymptotickou složitost Selection Sortu (třídění výběrem) pracujícího nad tímto konkrétním polem.
$O(n)$ $\Omega(n)$ $\Theta(n)$ $O(n^2)$ $\Omega(n^2)$ $\Theta(n^2)$

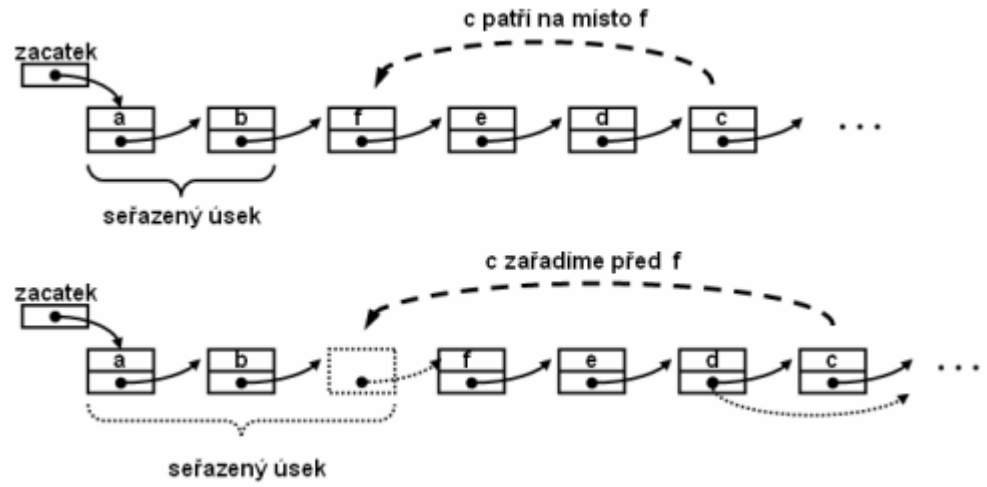
Řešení
Pro výběr aktuálního minima musí Select Sort pokaždé zkontrolovat všechny prvky od aktuálního až do konce pole, takže jeho složitost je je vždy právě $\Theta(n^2)$ nezávisle na datech v poli. V zadání je tedy nutno zatrhnout možnosti $\Omega(n)$, $O(n^2)$, $\Omega(n^2)$, $\Theta(n^2)$

4.
Implementujte Selection sort pro jednosměrně zřetězený spojový seznam.

Řešení
(Pro přehlednost nejprve kód pro řazení v poli)

```
for (i = 0; i < n-1; i++) {
 // select min
  jmin = i;
  for (j = i+1; j < n; j++) {
    if (a[j] < a[jmin]) {
      jmin = j;
    }
  }
 // make place for min
  min = a[jmin];
  for (j = jmin; j > i; j--) {
    a[j] = a[j-1];
 // put min
  a[i] = min;
}
```

Vnější cyklus algoritmu obsahuje tří fáze: Nalezení minima, uvolnění prostoru pro přesouvané minimum a vložení minima na správné místo. V zřetězeném seznamu je myšlenka jednodušší: Nalezení minima, vyjmutí minima ze seznamu a jeho vložení na příslušné místo. Ostatní prvky seznamu se automaticky posunou. Pracujeme totiž s celými prvky seznamu, nikoli jen s jejich hodnotami, jak to – celkem zbytečně – činil nejeden respondent této otázky. Obrázek snad naznačuje myšlenku:



Protože máme k dispozici jen jednosměrný seznam, budeme se potýkat s určitými potížemi.
Zaprvé musíme na daný prvek ukazovat nikoli přímo ale pomocí jeho předchůdce v seznamu, abychom mohli korektně prvek vyjímat a vkládat z/do seznamu. Díky neexistenci předchůdce prvního prvku budeme navíc muset nalezení nejmenšího prvku v seznamu obsloužit zvlášť.

Vystačíme si jen se čtyřmi dalšími ukazateli: **uk_i** pro vnější cyklus, **uk_j** pro vnitřní cyklus a pomocnými ukazateli **presouvany**, resp. **predPresouvany**, pomocí kterých budeme manipulovat s přesouvaným prvkem, resp ukazovat na předchůdce prvku, který má být přesunut. Doplnění chybějících deklarací v uvedeném pseudokódu necháváme na čtenáři.

```
prvekSeznamu selectSort (prvekSeznamu zacatek) {

if (zacatek == null) return;
if (zacatek.next == null) return;

// celkove minimum:
// najdeme jej v seznamu pocinaje druhym prvkem,
// pamatujme, ze na nic nemuzeme ukazovat primo diky jednosmernemu zretezeni
uk_j = zacatek;
predPresouvany = zacatek;
while (uk_j.next != null)
   if (uk_j.next.value < predPresouvany.next.value)
      predPresouvany = uk_j;
    // minimum v useku od druheho prvku nalezeno
    if (zacatek.value <= predPresouvany.next.value) { // minimum je na zacatku,
    }                                                 // nebudeme delat nic
    else { // presun minima  na zacatek
     // nejprve jej vyjmeme ...
     presouvany = predPresouvany.next;
     predPresouvany.next = presouvany.next;
     // ... a pak zaradime
     presouvany.next = zacatek;
     zacatek = presouvany;
}

// nyni jiz muzeme nasadit obecny cyklus
uk_i = zacatek;
while (uk_i.next.next != null) {
  // najdeme minimum v neserazene casti seznamu
  // uk_i ukazuje na posledni prvrk v serazene casti
  predPresouvany = uk_i;
  uk_j = predPresouvany.next;
  while(uk_j.next != null) {
    if (uk_j.next.value < predPresouvany.next.value)
       predPresouvany = uk_j;

  // predPresouvany ted ukazuje na prvek pred minimem, nastane presun minima
  if (predPresouvany == uk_i)  {} // neni co delat,
  else {
     // vyjmeme presouvany ze seznamu...
     presouvany = predPresouvany.next;
     predPresouvany.next = presouvany.next
     // ... a zaradime ZA uk_i
     presouvany.next = uk_i.next;
     uk_i.next = presouvany;
  } // konec presunu
  uk_i = uk_i.next;    // posuneme uk_i pro dalsi beh vnejsiho cyklu
} // konec vnejsiho cyklu
return zacatek; // ktery se mohl v prubehu razeni zmenit
}
```

5.
Nahradíme-li Selection sort Insert sort-em, pak asymptotická složitost řazení

a) nutně klesne pro každá data
b) nutně vzroste pro každá data
c) může klesnout pro některá data
d) může vzrůst pro některá data
e) zůstane beze změny pro libovolná data

Řešení
Tady je patrně nutné si pamatovat, že asymptotická složitost Selection sortu je $\Theta(n^2)$, zatímco asymptotická složitost Insert sortu je $O(n^2)$. To znamená, že existují případy, kdy Insert sort svá data seřadí v čase asymptoticky kratším než n^2. Na těchto datech však asymptotická složitost Selection sortu zůstane stejná, protože ta je rovna $\Theta(n^2)$ vždy. To znamená, že při přechodu od Selection sortu k Insert sortu nad týmiž daty může asymptotická složitost někdy klesnout, což odpovídá variantě c).

6.
Čtyři prvky řadíme pomocí Insert Sortu. Celkový počet vzájemných porovnání prvků je

a) je alespoň 4
b) je nejvýše 4
c) je alespoň 3
d) může být až 10
e) je vždy roven $4*(4-1)/2 = 6$.

Řešení
Nejmenší možný počet testů je:
při zařazení 2 prvku - 1
při zařazení 3 prvku - 1
při zařazení 4 prvku - 1
Tedy celkem 3 porovnání, pokud je posloupnost prvků rostoucí již před seřazením.
Největší možný počet testů je
při zařazení 2 prvku - 1
při zařazení 3 prvku - 2
při zařazení 4 prvku - 3
Tedy celkem 6 porovnání, pokud je posloupnost prvků klesající před seřazením.
Pro 4 prvky tedy Insert sort vykoná 3 až 6 porovnání prvků, podle toho, jaká má data.
S tímto zjištěním koresponduje pouze varianta c).

7.
V určitém problému je velikost zpracovávaného pole s daty rovna rovna $3n^2 \cdot \log(n^2)$ kde n charakterizuje velikost problému. Pole se řadí pomocí Insert sort-u.
Asymptotická složitost tohoto algoritmu nad uvedeným polem je tedy

a) $O(n^2 \cdot \log(n))$
b) $O(n^2 \cdot \log(n^2))$
c) $O(n^4 \cdot \log^2(n))$
d) $O(n^2 \cdot \log(n) + n^2)$
e) $O(n^2)$

Řešení
Máme-li velikost pole rovnu k, Insert sort jej zpracuje v čase $O(k^2)$. Velikost našeho pole je $k = 3n^2 \cdot \log(n^2)$, takže bude zpracováno v čase $O(k^2) = O((3n^2 \cdot \log(n^2))^2) =$
$= O(9n^4 \cdot \log^2(n^2)) = O(9n^4 \cdot 4 \cdot \log^2(n)) = O(n^4 \cdot \log^2(n))$.
Platí tedy možnost c).

8.
Insert sort řadí (do neklesajícího pořadí) pole o n prvcích, kde jsou stejné všechny hodnoty kromě poslední, která je menší. Jediné nesprávné označení asymptotické složitosti výpočtu je

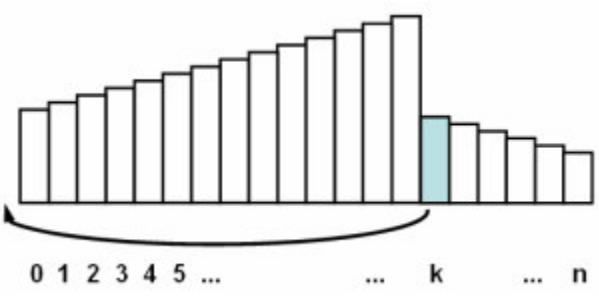
a) $O(n)$
b) $\Theta(n)$
c) $\Omega(n)$
d) $O(n^2)$
e) $\Omega(n^2)$

Řešení
Zpracování prvních $(n-1)$ prvků pole proběhne v čase úměrném $(n-1)$, protože žádný prvek se nebude přesouvat a zjištění toho, že se nebude přesouvat, proběhne v konstantním čase. Poslední prvek se pak přesune na začátek, což opět proběhne v čase úměrném n. Celkem tedy veškeré řazení proběhne v čase úměrném n. Platí ovšem
$konst \cdot n \in O(n)$
$konst \cdot n \in \Theta(n)$
$konst \cdot n \in \Omega(n)$
$konst \cdot n \in O(n^2)$
Jediná nepravdivá varianta je e).

9.
Insert sort řadí pole seřazené sestupně. Počet porovnání prvků za celý průběh řazení bude

a) n
b) n–1
c) n*(n–1)/2
d) n*n
e) $\log_2(n)$

Řešení
V k-tém kroku řazení je nutno prvek na pozici k zařadit na jemu odpovídající místo v seřazeném úseku nalevo od k (předpokládáme pole indexované 0.. n–1). Tento prvek je však díky původnímu sestupnému seřazení menší než všecny prvky nalevo od něj, tudíž musí být zařazen na úplný začátek, což znamená, že bude porovnán se všemi prvky nalevo od něj jichž je právě k. Viz obrázek.



V k tém kroku tedy bude provedeno k porovnání.
Celkový počet porovnání tedy je 0 + 1 + 2 + 3 + ... + (n–1) = n*(n–1)/2. Platí varianta c).

10.
Napište rekurzivní verzi algoritmu Insert sort. Při návrhu algoritmu postupujte metodou shora dolů.

Řešení
Toto je vůbec triviální úloha.
Celý algoritmus Insert sortu se skládá ze dvou vnořených cyklů. Aby se stal rekurzivním, změníme prostě vnější cyklus na rekurzivní volání, vše ostatní zachováme beze změny.

```
void insertSortRek(int k, int n) {     // n je počet prvků v poli
    vlož prvek na pozici k na jemu příslušné místo jako v nerekurzivní verzi;
   if (k < n)  insertSortRek(k+1,n);
}
```

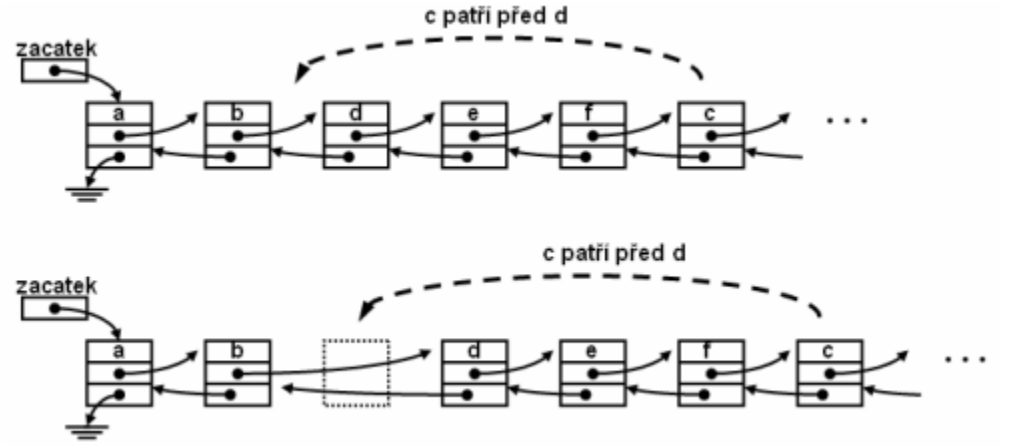
Jak probíhá vkládání prvku na jemu příslušné místo pro jistotu popište také.
Celé řazení pak proběhne po zavolání insertSortRek(1,n);.

11.
Implementujte Insert sort pro obousměrně zřetězený spojový seznam.

Řešení
(Pro přehlednost nejprve kód pro řazení v poli)

```
( 1) for (i = 1; i < n; i++) {
( 2)   // find & make place for a[i]
( 3)   insVal = a[i];
( 4)   j = i-1;
( 5)   while ((j >= 0) && (a[j] > insVal)) {
( 6)     a[j+1] = a[j];
( 7)     j--;
( 8)   }
( 9)   // insert a[i]
(10)   a[j+1] = insVal;
(11) }
```

Ve vnitřním cyklu (řádky 6—8) se vyhledává patřičné místo po vkládanou hodnotu insVal a přitom se ostatní hodnoty posunují o jednu pozici doprava, aby tak vkládané hodnotě uvolnily místo v poli. Mnozí respondenti zachovali tuto strategii i při použití zřetězeného seznamu, přestože je v tomto případě naprosto zbytečná. Stačí pouze přesunout celý prvek seznamu (i s jeho hodnotou) na patřičné místo a ostatní relevantní prvky seznamu (i se svými hodnotmi) se posunou o jednu pozici doprava zcela automaticky. Snad to dostatečně ilustruje následující obrázek.



Budou nám stačit ukazatele **uk_i** pro vnější cyklus a **uk_j** pro vnitřní cyklus a pomocny ukazatel **presouvany**, pomocí kterého budeme manipulovat s přesouvaným prvkem. Doplnění chybějících deklarací v uvedeném pseudokódu necháváme na čtenáři.

```
prvekSeznamu insertSort (prvekSeznamu zacatek) {
if (zacatek == null) return;
if (zacatek.next == null) return;
uk_i = zacatek.next;
while (uk_i != null) {
  uk_j = uk_i.prev;
  // najdi misto pro zarazovany prvek, na nejz ukazuje uk_i
  while ((uk_j != null) && (uk_j.value > uk_i.value))
```

```
      uk_j = uk_j.prev;
   // misto nalezeno, vkladej:

   if (uk_j.next == uk_i)
       // pokud uz byl prvek pod uk_i na spravnem miste
       uk_i = uk_i.next;  // tak jen posunem uk_i

   else { // nastane opravdovy presun
      // nejprve ovsem prvek z jeho mista korektne vyjmeme...:
      presouvany = uk_i;
      uk_i = uk_i.next;   // nutny posun pro dalsi beh vnejsiho cyklu

      // uz jen manipulaci s ukazateli
      // presuneme presouvany prvek kam patri
         // uvolnime stavajici vazby...
      presouvany.prev.next = presouvany.next;
      if (presouvany.next != null)
         presouvany.next.prev = presouvany.prev;
         // ... a vkladame jinam:
         // ale co kdyz musime vkladat na uplny zacatek?
      if (uk_j == null) {
         presouvany.prev = null;
         presouvany.next = zacatek.next;
         zacatek.prev = presouvany;
         zacatek = presouvany;
      }
      else { // vkladame ZA uk_j
         presouvany.next = uk_j.next;
         presouvany.prev = uk_j;
         uk_j.next = presouvany;
         preouvany.next.prev = presouvany;
      } // konec vkladani ZA uk_j
   } // konec presunu
} // konec vnejsiho cyklu
return zacatek; // ktery se mohl v prubehu razeni zmenit
}
```

Poznámka. Také by příkaz **`uk_i = uk_i.next;`** mohl být v kódu spíše jen jednou před koncem vnějšího cyklu...

12.

Insert sort řadí (do neklesajícího pořadí) pole o n prvcích, kde v první polovině pole jsou pouze dvojky, ve druhé polovině jsou jen jedničky. Jediné nesprávné označení asymptotické složitosti výpočtu je

a) $\Theta(n)$
b) $\Omega(n)$
c) $O(n^2)$
d) $\Theta(n^2)$
e) $\Omega(n^2)$

Řešení
Na obrázku máme znázorněnou situaci před prvním přesunem jedničky první a pak situaci před některým dalším přesunem jedničky.

V každém kroku je nutno jednu jedničku přesunout o $n/2$ pozic doleva a těchto kroků bude $n/2$ (protože jedniček je $n/2$). Počet provedených operací bude tedy úměrný hodnotě výrazu $n/2 \cdot n/2 = n^2/4 \in \Theta(n^2)$.
Z toho, že $n^2/4 \in \Theta(n^2)$, vyplývá bezprostředně také, že $n^2/4 \in \mathrm{O}(n^2)$, $n^2/4 \in \Omega(n^2)$, $n^2/4 \in \Omega(n)$ a navíc $n^2/4 \notin \mathrm{O}(n)$. Pravdivé jsou tedy varianty b) – d), nepravdivá je jedině varianta a).

13.
V poli velikosti n mají všechny prvky stejnou hodnotu kromě posledního, který je menší. Zatrhněte všechny možnosti, které pravdivě charakterizují asymptotickou složitost Insert Sortu (třídění vkládáním) pracujícího nad tímto konkrétním polem.
$\mathrm{O}(n)$ $\Omega(n)$ $\Theta(n)$ $\mathrm{O}(n^2)$ $\Omega(n^2)$ $\Theta(n^2)$

Řešení
Insert Sort postupuje od začátku pole a každý prvek zařadí do již seřazeného počátečního úseku pole. protože ale jsou všechny prvky stejně velké, zařazení každého prvku se redukuje na to, že bude ponechán na místě. To je operaces konstantní časovou složitostí, takže prvních n–1 prvků bude zpracováno v čase $\Theta(n)$. Poslední prvek v poli je sice menší, takže patrně nebude zpracován v konstantním čase. Zpracování každého jednotlivého prvku při řazení Insert Sort-em má však složitost $\mathrm{O}(n)$, takže celková složitost řazení bude $\Theta(n) + \mathrm{O}(n) = \Theta(n)$. V zadání je tedy nutno zatrhnout možnosti $\mathrm{O}(n)$, $\Omega(n)$, $\Theta(n)$, $\mathrm{O}(n^2)$.

14.
Společná vlastnost Insert sort-u, Select sort-u a Bubble sort-u je následující:

a) všechny tyto algoritmy jsou vždy stabilní
b) všechny tyto algoritmy jsou vždy nestabilní
c) všechny tyto algoritmy mají složitost $\mathrm{O}(n \cdot \log_2(n))$
d) všechny tyto algoritmy mají složitost $\mathrm{O}(n^2)$
e) všechny tyto algoritmy mají složitost $\Theta(n^2)$

Řešení
První z uvedených řazení — Insert sort — lze pouhou záměnou ostré a neostré nerovnosti v testu vnitřního cyklu učinit stabilním či nestabilním. Tvrzení variant a) a b) o stabilitě/nestabilitě za všech okolností tak patrně neplatí. Všechny zmíněné algoritmy v nejhorším případě řadí v čase úměrném n^2, funkce $n \cdot \log_2(n)$ roste asymptoticky pomaleji než funkce n^2, tudíž varianta c) rovněž neplatí. Insert sort může řadit v čase úměrném n, má-li vhodná data, takže tvrdit o něm, jako ve variantě e), že pokaždé řadí v čase úměrném n^2, není korektní. Zbývá tak jen správná varianta d).