Suchen und Sortieren

1.) Für ein Adressverwaltungsprogramm soll eine Methode zum Suchen einer Adresse geschrieben werden.

Im Fenster „Suchen“ kann ein Name, eine Straße, eine PLZ oder ein Wohnort angegeben werden. Zur Vereinfachung soll zunächst davon ausgegangen werden, dass immer nur ein Such-Kriterium angegeben wird. Im Hauptfenster soll anschließend die erste Karte in der Liste angegeben werden, bei der das angegebene Kriterium übereinstimmt. (Falls es eine solche Karte nicht gibt, soll im Hauptfenster „Momentan kein Zugriff auf eine Karte!“ erscheinen.)

Entwickle am Beispiel einer Liste mit den Namen

„Franz“, „Anna“, „Cora“, „Emil“, „Berta“, „Erna“, „Dora“, „Hugo“

ein Verfahren, um einen Namen (zum Beispiel „Erna“) in diese Liste zu finden und implementiere eine entsprechende Methode `void sucheNamen (Strings s)` für die Anwendung (in der Klasse `SuchKlickEmpfaenger`).

(Die Anwendung lässt sich durch Ausführung der `main`-Methode der Klasse `KartenStart` starten. In der Klasse `SuchKlickEmpfaenger` steht die Liste unter der Bezeichnung `lis` zur Verfügung. Beim Suchen genügt es, dass die gesuchte Karte in der Liste das aktuelle Element wird. Durch den Methodenaufruf `haupt.setKarte();` wird diese dann angezeigt.)

2.) Entwickle eine Methode `Karte suche (Karte [] k, String pName),` mit der die Karte mit dem Namen `pName` in dem Array `k` gesucht wird. (Die Klasse `Karte` verfügt über einen Methode `String getName()`, mit der der Name, der auf der Karte gespeichert ist, erfragt werden kann.)

3.) Analysiere und erläutere die folgende Methode:

```
125  public Karte binaereSuche(Karte[] k, String pName){
126    return binaereSuche(k,pName,0,k.length-1);
127  }
128  private Karte binaereSuche(Karte[] k, String pName, int i, int j){
129    if((i<0)||(j>=k.length)||(i>j))
130      return null;
131    else{
132      int m=(i+j)/2;
133      if(k[m].getName().equals(pName))
134        return k[m];
135      else if(i==j)
136        return null;
137      else if(k[m].getName().compareTo(pName)>0)
138         return binaereSuche(k,pName,i,m-1);
139      else
140        return binaereSuche(k,pName,m+1,j);
141    }
142  }
```

Erläutere ihre Funktionsweise, indem du die rekursive Abarbeitung der Methodenaufrufe darstellst, wenn die Methode mit dem Array

„Anna“…	„Berta“…	„Cora“…	„Dora“…	„Emil“…	„Erna“…	„Franz“ …

und pName=“Anna“ aufgerufen wird. (Im Array sind jeweils der Einfachheit halber nur die Namen auf den Karten angegeben.)
Erläutere, inwieweit diese Methode effizienter als die lineare Suche ist.

Die Anwendung soll erweitert werden, so dass die Karteikarten sortiert werden könnten, damit bei der Eingabe nicht auf eine alphabetische Reihenfolge der Namen geachtet werden muss.

4.) Entwickle ein Sortierverfahren für die Liste der Karten mit den Namen „Franz“, „Anna“, „Cora“, „Emil“, „Berta“, „Erna“, „Dora“, „Hugo“, „Fritz“. Verwende dazu die Namenskarten (s.u.). Beachte, dass der Computer immer nur jeweils 2 Namen miteinander vergleichen kann. Implementiere es in Java.

5.) Implementiere das Verfahren für eine Sortierung des oben angegebenen Arrays k. Verwende dabei nur ein einziges Array.

6.) Entwickle ein rekursives Sortierverfahren für ein Array von Karten. Das Sortierverfahren soll (ähnlich wie die binäre Suche) das Problem zunächst in zwei möglichst gleich große Teilprobleme zerlegen und anschließend die Teillösungen (sortierte Teillisten oder Teile des Arrays) richtig zusammensetzen.

7.) Analysiere und erläutere die folgende Methode:

```
77   public void sortiere2(){
78        lis.toFirst();
79        lis=this.sortiereRekursiv(lis);
80   }
81
82   private List sortiereRekursiv(List pListe){
83      List lGroessere = new List();
84      List lKleinere = new List();
85      if (!pListe.isEmpty()){
86        pListe.toFirst();
87        Karte lTrennelement = (Karte)pListe.getObject();
88        pListe.remove();
89        if(!pListe.isEmpty()){
90          pListe.toFirst();
91          while ((!pListe.isEmpty())&&pListe.hasAccess()){
92            if (((Karte)pListe.getObject()).getName().compareTo(lTrennelement.getName())<0){
93                lKleinere.append((Karte)pListe.getObject());
94            }
95            else{
96              lGroessere.append((Karte)pListe.getObject());
97            }
98            pListe.remove();
99            pListe.toFirst();
100         }
101
102       }
103         lGroessere=this.sortiereRekursiv(lGroessere);
104         lKleinere=this.sortiereRekursiv(lKleinere);
105         lKleinere.append(lTrennelement);
106         lKleinere.concat(lGroessere);
107     }
108    return lKleinere;
109  }
```

8.) Analysiere und erläutere das folgende Sortierverfahren. Die Liste `lis` enthalte anfangs die Namen „Franz“, „Anna“, „Cora“, „Emil“, „Berta“, „Erna“, „Dora“, „Hugo“, „Fritz“ (in dieser Reihenfolge). Stelle die Veränderungen der Listen grafisch dar.

```
55   public void sortierel(){
56      List hilfsliste = new List();
57      while (!lis.isEmpty()){
58        lis.toFirst();
59        Karte lKarte = (Karte)lis.getObject();
60
61            hilfsliste.toFirst();
62            while (hilfsliste.hasAccess() && lKarte.getName().compareTo( ((Karte)hilfsliste.getObject()).getName())>0){
63                hilfsliste.next();
64            }
65            if (hilfsliste.hasAccess()){
66                hilfsliste.insert(lKarte);
67            }
68            else{
69                hilfsliste.append(lKarte);
70            }
71            lis.remove();
72        }
73        lis = hilfsliste;
74   }
75
```

9.) Vergleiche und beurteile die unterschiedlichen Verfahren hinsichtlich Zeitaufwand und Speicherbedarf.

Franz
Anna
Berta
Emil
Erna
Fritz
Hugo
Dora
Cora