Tentamen i Algoritmer & Datastrukturer i Java

Hjälpmedel: Skrivhjälpmedel, miniräknare.
Ort / Datum: Halmstad / 2011-03-15
Skrivtid: 4 timmar
Kontakt person: Mattias Wecksten 7396
Poäng / Betyg: Max poäng 40
_ >=20 poäng ger betyg 3
_ >=28 poäng ger betyg 4
_ >=37 poäng ger betyg 5

Övrigt: Det finns två olika sorters frågekategorier, de som skall besvaras genom programmerings och de som skall besvaras genom förklaringar med text och illustrationer.
Programmeringslösningarna skall i största mån vara skrivna i korrekt Java-syntax.
Koden skall vara välstrukturerad och lättläst.
Koden skall innehålla lämpliga ledtexter (utskrifter) som instruerar användaren vad som krävs av honom/henne.
Om en lösning är uppenbart ”klumpig” och det anses att tentanden skall känna till en smidigare lösning kan den ”klumpiga” lösningen medföra poängavdrag.
Förklaringslösningar bör, om tillämpningsbart, innehålla illustrationer på relevanta datastrukturer och använda algoritmer.
Tänk på att vara noggrann och strukturerad. Det är Du som skall visa vad Du kan!

Lycka Till!

Uppgift 1 – Tidskomplexitet (4p)

a) För nedanstående delar av olika algoritmer ange en uppskattning av exekveringstiden i form av Big-Oh notation. Motivera också din uppskattning.

A1)
```
public class Statistics{


public static double average( double da[ ])
  {
     double n=da.length;
     return sum(da)/n;

  }

public static double sum (double da[ ])
{
  double s=0;
  for(int i=0; i<da.length;i++)
    s=s+da[i];
 return s;
}
```

A2)
```
while(i>=1){
     for( int j=1;j<n;j++){
            x=x+1;
     }
     i=i/2;

    }
  for(int i=0; i<n;i++)
     x=x+1;
```

Uppgift 2 – Datastrukturer (2p+4p+6p+4p)

a) På föreläsningen pratade vi om en sk. *”wrap-around”* implementation av en kö.
Förklara kort datastrukturen kö samt vad som menas med *”wrap-around”* implementation av kö. Vilka är fördelarna med *”wrap-around”* implementation?

b) Förklara principen med de datastrukturer som vi kallar **”Symbol tabeller”**. Din förklaring skall innehålla bl.a. svar på följande frågor:
Vad är dessa för typ av datastrukturer? Vilka olika sätt att representera datastrukturen känner du till? Vilka operationer är vanligt förekommande? Vad kan du säga om tidskomplexiteten for operationerna. Vilka är fördelar/nackdelar med de olika sett att representera ”Symbol tabeller”?

c) I en multipop-stack tillåter man förutom de vanliga stackoperationerna även en som tar bort
ett antal av de översta elementen (utan att returnera något). Antalet element som skall tas bort är in parameter till metoden.

Implementera en klass för en sådan stack. Stacken skall vara arraybaserd, och innehålla metoderna, ***push(Object x) pop(), och multipop(int n)***

OBS! Tänk på att metoden multipop(int n) in parameter **n** kan vara större än stackens storlek. Du får hitta ett sätt att hantera det fallet. Metoden skall returnera true om multipop kan genomföras annars false.

Klassen behöver inte nödvändigtvis vara precis som den som finns i boken eller pratat om på föreläsningen vad gäller exception-hantering. Det viktiga är att alla väsentliga metoder finns med och att de är utformade så att rimliga resultat alltid ges.

d) Du jobbar på en implementation av en enkel *min heap* i Java. Du använder dig av en array som innehåller alla element i heapen. Här är en del av koden som du har skrivit hittills:

```
public class MINSpecialHeap
{
private final int maxSize;
private Comparable[] heap;
private int currentSize;

public MINSpecialHeap(int maxSize)
{
this.maxSize = maxSize;
heap = new Comparable[maxSize];
currentSize = 0;
}
...
}
```

Din uppgift är att **lägga till och implementera** två metoder med följande signatur:

```
// lägger till värdet x på rätt plats i heapen
public void add (Comparable x)

// returnerar en lista som innehåller alla värden från
//heapen som är mindre än x. Du väljer själv vilken typ av
Collection datastruktur som metoden skall returnera.
public Collection getLessThan(Comparable x)
```

Tips 1: Som du vet en heap är en ”sorterad datastruktur”. I detta fallet är det en MIN heap, d.v.s minsta värde finns på första platsen.
Tips 2: Välj bland klasserna som implementerade interfacen Collection. Vilka var dem?

Uppgift 3 – Kända algoritmer (1p+1p+1p+2p)

a) Anta att vi måste söka efter ett element bland en stor mängd data som är lagrad i en array. Vad måste vi veta om datan som skall sökas igenom för att kunna tillämpa binärt sökning?

b) Varför är **INTE** binärt sökning lämpligt för länkade listor?

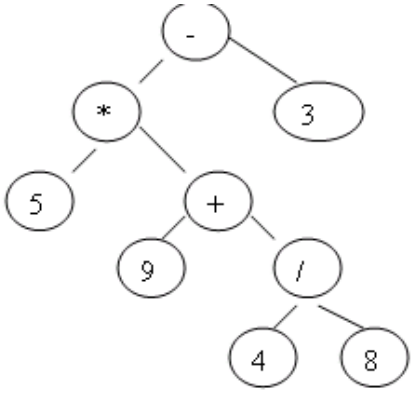
c) I vanligt fall är *quicksort* algoritmen en mycket effektiv sorterings algoritm.
Förklara i vilka fall kan *quicksort* algoritmen ge exekveringstid i O(N^2).

d) Förklara hur exekveringstiden påverkas om arrayen som skall sorteras är *”nästan”* sorterad. Vilken sorterings algoritm skulle du använda om du vet att data i arrayen är *”nästan”* sorterad.

Uppgift 4 – Träd och HashTabell (2p+2p+2p)

Vi kan använda binära träd för att representera algebriska uttryck. Dessa träd kallas ”expressions tree”.
Om man använder metoden printPostOrder() för att skriva ut trädet får vi en aritmetiskt utryckt i postfix notation.
Se nedan implementation av metoden samt ett träd.

```
public void printPostOrder( )
    {
        if( left != null )
            left.printPostOrder( );          // Left
        if( right != null )
            right.printPostOrder( );         // Right
        System.out.println( element );       // Node
    }
```



a) Vilken blir uttrycket för trädet ovan.

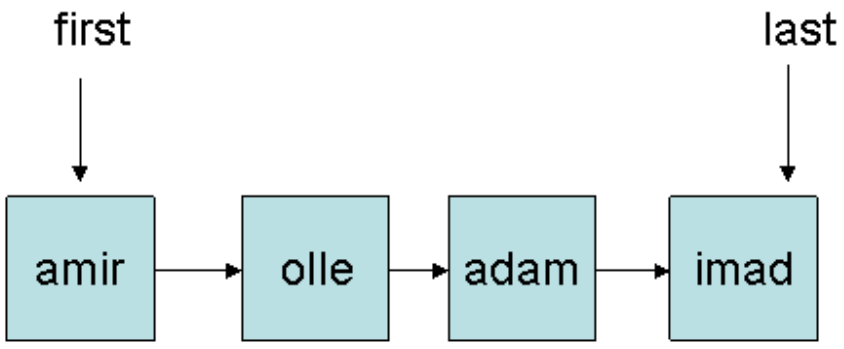
b) För att beräkna aritmetiska uttryck i postfix notation används datastrukturen **stack**. Vissa hur stacken förändras när du beräknar *postfix uttrycket* som du fått fram i uppgift a. Lyckas du inte lösa uppgift a välj själv ett aritmetisk utryckt och visa hur beräkningen med hjälp av stacken sker.

c) Förklara vad en hash funktionen används till samt varför nedanstående hashfunktion är dålig.

```
public static int badHashFunc(String data) {
  int val = 0;
  for(int i = 0; i < data.length(); i++)
    val += data.chatAt(i);
  return val;
}
```

Uppgift 5 – Länkad lista (9p)

En s.k. ”double-ended linked list” man tillåter insättning och borttagning både i början och i slutet av listan. Antag att listan är en enkellänkad och i klassen finns två referenser, en till första noden och en till sista.



Noden i listan definieras som vanligt, se nedan.

```
class Node {

Object element; // innehållet i listnoderna
Node next; // referens till nästa nod i listan
public Node(Object data, Node n)  // konstruerare
{
  element=data;
  next=n
}
public Node(Object data)
{
  element=data;
  next=null;
}
}
```

Implementera klassen enligt nedanstående specifikation.

```
public class DoubleEndedLinkedList {

 /** Obs! first och last pekar på första och sista noden i
listan. Inga noder är ”tomma” */
private ListNode first, last;

public DoubleEndedLinkedList {
  first=last=null;
}

/** Sätt in x sist i listan */
public void insertLast(Object x);

/** Sätt in x först i listan */
public void insertFirst(Object x);

/** Tag bort första noden i listan och returnera dess
innehåll.Returnera null om listan är tom */
public Object removeFirst();

/** Tag bort sista noden i listan och returnera dess
innehåll. Returnera null om listan är tom */
public Object removeLast();

/** Returnerar objeket som hittas på en viss index,
returnerar null om idex är större än listan storlek
public Object getElement(int index);

}
```

```
/**  Returnerar innehållet i listan, returnera null om
listan är tom */
public String toString();

}
```