Tentamen i Algoritmer & Datastrukturer i Java

Hjälpmedel: Skrivhjälpmedel, miniräknare.
Ort / Datum: Halmstad / 2010-03-16
Skrivtid: 4 timmar
Kontaktperson: Nicolina Månsson
Poäng / Betyg: Max 44 poäng
_ >=20 poäng ger betyg 3
_ >=30 poäng ger betyg 4
_ >=40 poäng ger betyg 5

Övrigt: Det finns två olika sorters frågekategorier, de som skall besvaras genom programmering 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.
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 (2p+2p+4p+2p)

a) Betrakta följande Javametod

Vad är tidskomplexiteten för ett anrop f(n)? Motivera ditt svar.

```
public int f(int n) {
int sum = 0;
for (int i = n; i >=1; i=i/2)
sum = sum + i;
return sum;
}
```

b) Binärsök algoritmen är tänkt för arrayer där data är sorterad.

```
public static int binarySearch( int [ ] a, int x )
  {
     // här kommer Tony att lägga till kod ( se uppgift d)

        int NOT_FOUND=-1;
        int low = 0;
        int high = a.length - 1;
        int mid;

       while( low <= high )
        {
            mid = ( low + high ) / 2;
            if ( x> a[ mid ]  )
                low = mid + 1;
            else if(x< a[ mid ] )
                high = mid - 1;
            else
                return mid }
                return NOT_FOUND;
        }
```

Tony har inte gått Algoritmer och Datastruktur kursen och anropar metoden men en osorterad array. Vad kommer det att hända? Kraschar programmet? Hur påverkar detta sökningen? Motivera med exempel.

c) Då Tony inser att ”något” inte är som det skall när han kör programmet med osorterad input, tänker han ut en lösning: Att komplettera binarySearch metoden med några rader kod som **testar om input arrayen är sorterad.** Implementera en algoritm med bästa tänkbara tidskomplexitet som testar om en array är sorterad.

d) Hur påverkar din algoritm tidskomplexiteten för binarySearch? Vad tycker du om Tonys lösning?

Uppgift 2 – Datastrukturer (2p+4p+4p)

a)Förklara vad som är olämpligt i följande exempel. Beskriv också hur man ska göra istället. Motivera.

1)En idrottsförening vill ha ordning på alla sina medlemmar. Idrottsföreningen väljer att lagra medlemmarna (som är av typen Person-objekt) i en hashtabell. Som hashkod väljer man de två första siffrorna i medlemmens personnummer.

2) En stack implementeras genom att elementen lagras i en array. Vid push() sätts elementet in alltid på position 0 i arrayen och vid pop() hämtas elementet från position 0.

b) I en dator vill man att många processer ska kunna köra samtidigt även om man bara har tillgång till en CPU. Detta åstadkommer man genom att varje process får köra en liten stund och sedan bytas ut mot nästa. Denna fördelning av CPU-tid kan lösas på många olika sätt. Ett av de enklaste är Round Robin: Processerna placeras i en kö och får tillgång till CPUn i tur och ordning. Alla processer får lika lång CPU-tid på sig. När den process som står i tur har använt sin tilldelade CPU-tid hamnar den sist i kön igen, såvida den inte exekverat färdigt.

Antag att vi har fyra processer med namn, p1, p2, p3, p4 som behöver exekveringstiderna 250 ms, 50 ms, 400 ms respektive 400 ms. Antag också att den CPU-tid varje process får (kallas tidskvantum) är 100 ms. När alla processer sätt in i kön ser kön med ut på följande sätt.

Vid varje dequeue() minskas exekveringstiden för processen med tidskvantum 100 ms.

Q: p1 250 ms, p2 50 ms, p3 400 ms, p4 400 ms
f b

Visa utvecklingen av kön (med front och back pekarna) efter varje enqueue() och dequue() tills alla processer har exekverat färdig. Antag att du använder dig av en array baserad kö, se bilaga.

c) Implementera ett Simuleringsprogram som simulerar de beskriva scenariet. Ett Process-objekt beskrivs av klassen Process, se nedan.
Programmet börjar med att du skapar 4 processer. Välj själv namn för dina processer, men slumpa processtiden(timeleft) till ett heltasvärde i intervallet 100-400 med
new Random().nextInt(int maxValue);
Använd för tidskvantum 100 ms.
För att simulera att programmet ”låter processorn exekvera” använd Thread.sleep(tidskvantum);
Efter varje enqueue ()/ dequeue ()skall processinformationen skivas ut.

```
public class Process {
private String processName; // processnummer
private int timeLeft;  // tid kvar att exekvera

// Skapar en process med namen processName och med timeLeft
ms kvar att exekvera.

public Process(String processN, int timeLeft) {

processName = processN;
timeLeft = timeLeft;
}

/** Returnerar tiden kvar att exekvera (i ms). */
public int getTimeLeft() {
return timeLeft;
}
/** Sätter tiden kvar att exekvera till timeLeft ms. */
public void setTimeLeft(int timeLeft) {
this.timeLeft = timeLeft;
}
```

```
/** Returnerar en sträng som representerar processen. */
public String toString() {
return "p" + processNbr + " " + timeLeft + " ms";
}
}
```

Uppgift 3 - Länkade datastrukturer(10 p)

En buffert kan vara en länkad lista med begränsad storlek. I denna uppgift skall en klass BoundedBuffer, som representerar en buffert, implementeras. Den övre gränsen för antalet element som samtidigt får befinna sig i bufferten anges som parameter till konstruktorn. Om man anropar insättningsoperation när bufferten innehåller det maximalt tillåtna antalet element så sker ingen insättning. I stället genereras en exception.

Elementen i bufferten skall placeras i en enkellänkad lista. Klassen har två attribut (head och tail) som refererar till första respektive sista noden i denna lista. Båda är ”tomma” noder som finns för att markera början och sluten av listan. Dessutom finns det ett heltalsattribut (maxSize), som anger maximalt antal tillåtna element samt ett heltalsattribut (size), som anger hur många element som finns i bufferten:
Klassen Nod är välbekant dig och representerar en Nod i länkade listan.

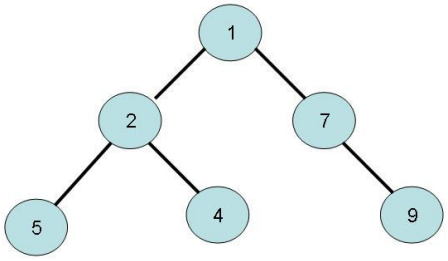
```
class Node {
Object element;
Node next;

public Node(Object  x) {
element = x;
next = null;
}

}
```

```
public class BoundedBuffer {
Node head;
Node tail;
int maxSize;
int size;


/** Konstruktor, skapar två "toma" noder som head och tail
pekar på, dvs, noderna head och tail får inte innehålla
data. Sätter maxSize värdet för bufferten */
public BoundedBuffer(int thesize) {...}


/** Sätter in elementet x i bufferten om den inte är full.
Om bufferten innehåller maximalt antal tillåtna element
genereras
BufferOverflowException. */

public void insert(Object x) {...}

/** Returnerar och tar bort första elementet i bufferten.
Om bufferten är tom returneras null. */
public Object getFirst() {...}


/** Undersöker om bufferten innehåller maximalt antal
tillåtna element. */
public boolean isFull() {...}
}

/** Undersöker om bufferten är tom */
public boolean isEmpty() {...}
}

/**Skriver ut innehållet i bufferten*/
public void print{...}
```

a) Implementera konstruktorn och alla operationer enligt beskrivningen.
Du får anta att klassen BufferOverflowException finns redan implementerad.

Uppgift 4 - Träd och annat (6p+4p+2p+2p)

a) Studera följande träd:



1) Är trädet ett binärt sökträd? Motivera ditt svar genom att kort beskriva datastrukturen binärt sökträd.
2) Är trädet ett balanserat träd? Motivera ditt svar genom att kort beskriva begreppet balanserad träd.
3) Är trädet en heap? Motivera ditt svar genom att kort beskriva datastrukturen heap.

b) För att omvandla aritmetiska utryck till ”postfix” notation används **operand stack** och för att beräkna aritmetiska uttryck i ”postfix” notation används en **operator stack.**
Beskriv dessa algoritmer med hjälp av ett enkelt uttryck som **$5+3*4^2/6$.**

c) I denna uppgift behandlas hashtabeller.

1) Antag att man har en hashtabell där elementen lagras i en array av storlek 11 och en hash-funktion **h(x)=x%11.**

Linjär teknik används för att hantera kollisioner. Antag att man satt in följande element 34,45,3, 67, 65, 19,1,17. Hur ser tabellen ut efter dessa insättningar?

2) Om man önskar ta bort elementet 34 och bara lämnar dess plats i tabellen tom uppstår det problem. Vilket är problemet?

d) Förklara varför Quick sort är en så kallad subkvadratisk sorterings algoritm. Visa för följande array **7 13 5 2 18 10** hur sorteringen sker.

Bilaga 1.

```
public class ArrayQueue<AnyType>
{
   public ArrayQueue( int size )
   {
      theArray = (AnyType []) new Object[ size];
      makeEmpty( );
   }

      public boolean isEmpty( )
   {
      return currentSize == 0;
   }

   public void makeEmpty( )
   {
      currentSize = 0;
      front = 0;
      back = -1;
   }

   public AnyType dequeue( )
  {
      if( isEmpty( ) )
         throw new UnderflowException( "ArrayQueue dequeue" );
      currentSize--;

      AnyType returnValue = theArray[ front ];
      front = increment( front );
      return returnValue;
   }
```

```
public void enqueue( AnyType x )
   {
      if( currentSize == theArray.length )
         doubleQueue( );
      back = increment( back );
      theArray[ back ] = x;
      currentSize++;
   }



private int increment( int x )
   {
      if( ++x == theArray.length )
         x = 0;
      return x;
   }

   private void doubleQueue( )
   {
      AnyType [ ] newArray;

      newArray = (AnyType []) new Object[ theArray.length * 2 ];

         // Copy elements that are logically in the queue
      for( int i = 0; i < currentSize; i++, front = increment( front ) )
         newArray[ i ] = theArray[ front ];

      theArray = newArray;
      front = 0;
      back = currentSize - 1;
   }

   private AnyType [ ] theArray;
   private int         currentSize;
   private int         front;
   private int         back;

}
```