Rekursion

Tema: Rekursiva algoritmer och metoder.

U 1. Man kan definiera potensfunktionen x^n (n heltal ≥ 0) rekursivt enligt:
$x^0 = 1$
$x^n = x * x^{n-1}$

a) Skriv en rekursiv metod `public static double power(double x, int n)` som beräknar x^n enligt denna definition.

b) Beräkningen kan göras effektivare om man i stället gör följande definition:
$x^0 = 1$
$x^n = (x^{n/2})^2$ om n>0 och jämnt
$x^n = x * (x^{n/2})^2$ om n>0 och udda
Gör en ny implementering av metoden i a enligt denna definition. (Med $n/2$ avses här heltalsdivision.)

c) Ändra lösningen från uppgift b så att den även klarar negativa värden på n.
Ledning: Skriv en metod som kontrollerar om värdet på n är negativt eller ej och sedan anropar den rekursiva metoden från b. Tänk på att $x^{-n} = \frac{1}{x^n}$.

U 2. Betrakta följande class:

```
public class MysteryClass {

    public static void mystery(int n) {
        if (n > 0){
            mystery(n-1);
            System.out.print(n * 4);
            mystery(n-1);
        }
    }

    public static void main(String[] args) {
        MysteryClass.mystery(3);
    }
}
```

Vad skrivs ut när man kör programmet?

U 3. I rekursiva metoder ska det finnas minst ett basfall. Förklara begreppet basfall. Förklara också varför det måste vara med.

U 4. En palindrom är ett ord eller en mening som kan läsas på samma sätt både fram- och baklänges. Exempel: DALLASSALLAD, NITALARBRALATIN.

Skriv en metod med rubriken `public static boolean isPalindrome(String s)` som med rekursiv teknik undersöker om en sträng är en palindrom.

Ledning: Låt `isPalindrome` anropa en annan rekursiv metod som förutom strängen har två heltalsparametrar som anger index för första och sista tecken i den delsträng som ska undersökas.

U 5. Skriv en rekursiv metod som räknar ut hur mycket ett kapital växer med ränta på ränta:

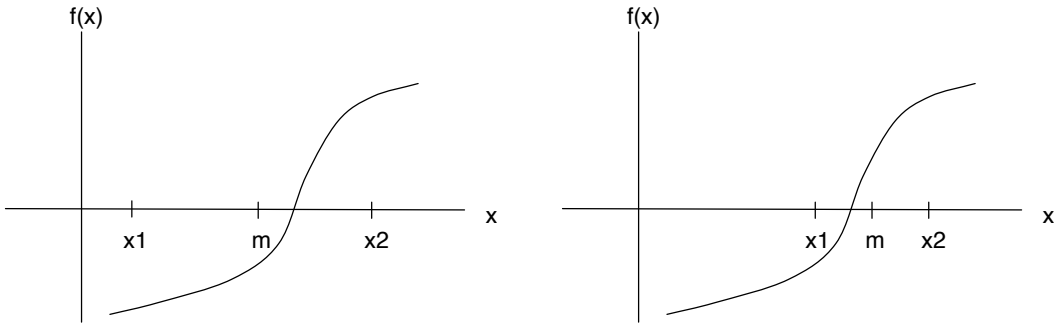
```
static double computeCapital(double capital, int years, double interestRate);
```

De data vi har är startkapital i kronor, antal år ($\geq$0) och räntesats (i procent). Exempel: Om startkapitalet är 1000 kr och räntesatsen är 10% så får man 100 kr i ränta under första året och kapitalet har alltså vuxit till 1100 kr. År två får man 110 kr i ränta och kapitalet har vuxit till 1210 kr. Efter tre år har man 1331 kr etc...

U 6. Antag att vi vet att en funktion $f(x)$ har ett nollställe i ett visst intervall $[x1, x2]$. Då kan vi beräkna detta nollställe med en teknik som liknar binärsökning. Vi ska alltså "stänga" in nollstället genom upprepade halveringar av intervallet $[x1, x2]$ tills intervallet är tillräckligt litet (mindre än `eps` som t ex kan vara lika med 0.0001).

En intervallhalvering går till så att man först beräknar mittpunkten, m, och därefter ersätter $x1$ eller $x2$ med m. För att avgöra om det är $x1$ eller $x2$ om ska ersättas jämförs tecknen på $f(m)$ med $f(x1)$ och $f(x2)$. Om $f(m)$ har samma tecken som $f(x1)$ ska mittpunkten ersätta $x1$, annars ska mittpunkten ersätta $x2$.

Exempel: i figuren nedan är $f(x1)$ och $f(m)$ båda negativa och sökningen fortsätter i högra halvan av interfallet.



Metoden `getZero` i klassen `Bisection` löser problemet ovan:

```
public class Bisection {
    /** Beräknar nollstället för funktionen f i intervallet [low, high] med
        precisionen eps.
        Förutsätter att det finns ett nollställe i intervallet. */
    public static double getZero(double low, double high, double eps, Function f) {
        ...
    }
}
```

`Function` är ett interface med en metod för att beräkna ett funktionsvärde:

```
public interface Function {
    double evaluate(double x);
}
```

a) Implementera metoden `getZero` med rekursiv teknik.

b) Skriv en klass som implementerar interfacet `Function` och som motsvarar funktionen $e^{-x} - 1 + cos(x)$.

c) Skriv programrad/rader som beräknar nollstället för funktionen $e^{-x} - 1 + cos(x)$ i intervallet $[0, 1.6]$ med noggrannheten 0.00001.

U 7. Vid implementeringen av en listklass har man tänkt använda följande klasser:

```
public class SingleLinkedList<E> {
    private ListNode<E> first;

    public SingleLinkedList() {
        first = null;
    }
    ...

    private static class ListNode<E> {
        private E element;
        private ListNode<E> next;

        private ListNode(E e) {
            element = e;
            next = null;
        }
    }
}
```

Listan ska alltså implementeras med en enkellänkad struktur. Den första metod som implementeras i listklassen är en metod för insättning. Följande rekursiva utformning föreslås:

```
public void add(E x) {
    add(first, x);
}

private void add(ListNode<E> node, E x) {
    if (node == null) {
        node = new ListNode<E>(x);
    } else {
        add(node.next, x);
    }
}
```

Vid test visar det sig att detta inte fungerar alls. Förklara varför. Ändra också implementeringen så att den blir korrekt (och fortfarande är rekursiv).

Kritisk granskning av rekursiva lösningsförslag

I Proceedings of The 7th Annual Conference on Innovation and Technology in Computer Science Education Aarhus, Denmark June 2002 finns en artikel med titeln "The Case of Base Cases: Why are They so Difficult to Recognize? Student Difficulties with Recursion" av Bruria Haberman och Haim Averbuch. Genom att samla in studenters rekursiva lösningar på olika problem beskrivs hur författarna försökte göra sig en bild av vilka misstag som var vanligast. Inte bara felaktiga lösningar var i detta sammanhang intressanta utan också sådana som är onödigt komplicerade. Låt oss knyta an till detta genom att studera de rekursiva lösningsförslagen (pseudokod) till några problem i uppgifterna U8 – 9. Försök i första hand för varje lösning avgöra om den är korrekt eller ej. För lösningar som du anser korrekta kan du sedan gå vidare och ge ytterligare synpunkter t ex om de kan förenklas eller göras elegantare.

U 8. Granska fyra följande förslag för att rekursivt beräkna n!:

a)
```
fac(n)
      om n == 0 return 1
      annars return n*fac(n-1)
```

b)
```
fac(n)
      om n==0 return 1
      annars
          om n == 1 return 1
          annars return n*fac(n-1)
```

c)
```
fac(n)
      om n==1 return 1
      annars return n*fac(n-1)
```

d)
```
fac(n)
      return n*fac(n-1)
```

U 9. Granska följande två förslag till att undersöka om elementet x ingår i en lista. (`list.tail()` används för att beteckna den lista som består av alla element utom det första i `list`):

a)
```
isMember(x,list)
      om list är tom return false
      annars
          om x == första elementet i list return true
          annars return isMember(x,list.tail())
```

b)
```
isMember(x,list)
      om x == första elementet i list return true
      annars return isMember(x,list.tail())
```

Rekursiva algoritmer – effektivitet

U 10. I denna uppgift behandlas problemet att beräkna största talet i en vektor med n heltal ($n \geq 1$). Följande implementering föreslås:

```
/** Tag reda på vilket av de n första elementen i v som är störst. */
public static int findMax(int[] v, int n) {
    if (n == 1) {
        return v[0];
    } else if (v[n-1] > findMax(v, n-1)) {
            return v[n-1];
    } else {
        return findMax(v, n-1);
    }
}
```

Denna algoritm är mycket ineffektiv. Förklara varför. Det går att göra en väsentligt effektivare rekursiv implementering genom en enkel ändring. Utför ändringen och ange sedan tidskomplexiteten.

U 11. Vad är dynamisk programmering och vad har man det till.

U 12. Följande båda algoritmer (pseudokod) föreslås för att skriva ut elementen i en enkellänkad lista i omvänd ordning. Jämför algoritmerna med avseende på enkelhet (att förstå och implementera) och effektivitet.

```
Algoritm1 : (letar först upp sista elementet, sedan näst sista etc...)
    leta upp sista noden i listan och låt q referera till denna
    så länge q != null
        skriv ut innehållet i q;
        om q = första noden i listan, avbryt
        p = första noden i listan:
        så länge p.next != q
            p = p.next;
        q = p;

Algoritm2 (rekursiv, skriver ut innehållet i den lista som n
refererar till i omvänd ordning)
public void printreverse(ListNode n)
    om (n != null)
        printreverse(n.next);
        skriv ut innehållet i n;
```