Lösningsförslag till Algoritmer och datastrukturer.

Uppgifter

a) *Nämn 3 sätt att representera en algoritm.*

Pseudokod, naturligt språk, programspråk, flödesdiagram

b) *Förklara skillnaden mellan begreppen algoritm, program och process.*

En algoritm är en beskrivning för hur ett problem skall lösas. Ett program är en algoritm uttryckt med hjälp av ett programspråk. En process är den aktivitet som datorn utför när den exekverar programmet.

c) *Ge exempel på hur man kan få en "foot in the door" när man försöker lösa ett problem.*

Att få en "foot in the door" handlar om att komma över en slags tröskel som gör att problemet kan lösas. Det finns olika metoder att använda. Ett exempel är att börja lösa problemet bakifrån. Ett annat exempel är att leta efter liknande problem som antingen är enklare eller som man känner lösningen till, och sedan utgå från detta exempel vid lösandet av det aktuella problemet.

d) *Visa hur insättningssortering skulle sortera strängarna*

Erland, Stina, Birgit, Christer, Olle

Osorterad sekvens	Sorterad sekvens
Erland, Stina, Birgit, Christer, Olle	-
Stina, Birgit, Christer, Olle	Erland
Birgit, Christer, Olle	Erland, Stina
Christer, Olle	Birgit, Erland, Stina
Olle	Birgit, Christer, Erland, Stina
-	Birgit, Christer, Erland, Olle, Stina

e) *Antag att du har ett fält med n st sorterade poster. Hur lång tid tar det att söka efter en post x i fältet om du använder sekventiell sökning (sequential search)? Antag att en jämförelse tar 1 millisekund. Hur lång tid tar det om du använder binärsökning (binary search)?*

Hur mycket längre tid tar det om fältet har längden 4n? Om fältet har längden 8n?

Med med rak sökning tar det i värsta fall (worst case) n tidsenheter att finna posten x och i genomsnitt n/2 tidsenheter. Med binärsökning tar det $\log_2$ (n) tidsenheter både i värsta fall och i genomsnitt.

Om fältet innehåller 4n poster tar rak sökning värsta fall 4n tidsenheter och i genomsnitt 2n tidsenheter, medan binärsökning tar $\log_2$ (4n) = 2 + $\log_2$ (n) tidsenheter både i värsta fallet och i genomsnitt. För sekventiell sökning tar det alltså 4 gånger så lång tid, medan det för binärsökning tar 2 millisekunder längre tid.

Om fältet innehåller 8n poster tar rak sökning värsta fall 8n tidsenheter och i genomsnitt 4n tidsenheter, medan binärsökning tar $\log_2$ (8n) = 3 + $\log_2$ (n) tidsenheter både i värsta fallet och i genomsnitt. För sekventiell sökning tar det alltså 8 gånger så lång tid, medan det för binärsökning tar 3 millisekunder längre tid.

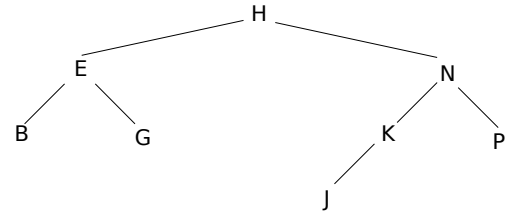
f) *Varför kan man inte bevisa att ett program är korrekt genom att testköra det?*

För att bevisa att ett program är korrekt genom testning måste en uttömmande testning göras, dvs att man testar alla möjliga uppsättningar indata som kan ges till programmet och därigenom garanterar att alla möjliga exekveringsvägar i programmet blivit genomlöpta. Även för ganska triviala program är detta ogenomförbart eftersom antalet indatauppsättningar är alltför många.

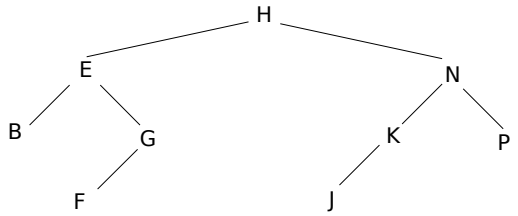
g) *Vad kännetecknar ett binärt sökträd?*

Ett binärt sökträd är ett binärt träd för vilket det gäller att värdet i alla noder i det vänstra delträdet är mindre värdet i roten, och att värdet i alla noder i det högra delträdet är större än värdet i roten. Det måste också gälla att det vänstra delträdet och det högra delträdet är binära sökträd.

h) Sätt in 'F' på rätt ställe i det binära sökträdet



Utseende efter insättning:



i) *Man kan beräkna x^n genom att beräkna uttrycket x*x*x*...*x*x, dvs genom att göra n upprepade multiplikationer av x. "Kostnaden" för denna metod är således n multiplikationer.*

Konstruera en effektivare metod, som endast behöver 2log n *multiplikationer, genom att använda* divide and conquer. *Beskriv metoden med hjälp av pseudokod.*

Man delar upp x^p i halvor tills man som minsta byggsten får x*x. Sedan multiplicerar man denna byggsten med sig själv och erhåller en större byggsten, som multipliceras med sig själv. Denna process upprepas tills man beräknat det ursprungliga uttrycket x^p . Enklast implementeras lösningen med användning av rekursion. Nedan ges dock en iterativ metod.

```
public static double  power(int  x, int  n) {
    // om n är udda så multiplicerar vi resultatet med x och minskar n med 1
    // om n är jämn så kvadrear vi x och delar n med 2
    // observera att n alltid blir udda till slut när n=1 och då,
    // om inte förr, updateras result
    int  result = 1;
    while  (n != 0) {
        if  ( (n % 2) != 0 ) {  //n är udda
            result = result * x;
            n = n-1;
        }
        else  {    // n är jämn
            x = x*x;
            n = n/2;
        }
    }
    return  result;
}
```