

# Tentamen - Datastrukturer, algoritmer och programkonstruktion.

**DVA104**

*Akademin för innovation, design och teknik*

*Onsdag 2014-01-15*

**Skrivtid:** 08.30-13.30

**Hjälpmedel:** Inga

**Lärare:** Caroline Uppsäll, 021-101456

## Betygsgränser

Betyg 3: 25p - varav minst 6 poäng är P-uppgifter

Betyg 4: 35p

Betyg 5: 42p

Max: 46p - varav 12 poäng är P-uppgifter

## Allmänt

- Kod skriven i tentauppgifterna är skriven i pseudokod.
- På uppgifter där du ska skriva kod (P-uppgifter) ska koden skrivas i det språk som var aktuellt då du tog kursen (C eller Ada). Ange högst upp på bladet vilket språk koden är skriven i.
- Skriv endast en uppgift per blad
- Skriv bara på ena sidan av pappret.
- Referera inte mellan olika svar.
- Om du är osäker på vad som avses i någon fråga, skriv då vad du gör för antagande.
- Oläsliga/Oförståeliga svar rättas inte.
- Kommentera din kod!
- Eventuellt intjänade bonuspoäng kan endast användas på ordinarie tentamenstillfälle.
- Tips: Läs igenom hela tentan innan du börjar skriva för att veta hur du ska disponera din tid.

*Lycka till!*

## Uppgift 1: (8p)

a) Vad menas med divide & conquer?

Deler upp problemet i delproblem tills delproblemets  
är tillräckligt trivialt för att enkelt kunna lösas.  
Sätt sedan ihop lösningarna på delproblemens till en  
lösning på originalproblemet.

b) Beskriv ett exempel på en algoritm som använder divide & conquer och på vilket sätt den  
...använder paradigmen.

Mergesort - delar i hälften tills mängden är 1. Sätter sedan  
ihop delmängderna sorterat.

Quicksort - lägger allt mindre än pivot till vänster och allt större  
till höger, gör sedan Quicksort på delmängderna ... tills  
de är 1 element stora.

c) Vad är ett annat namn på en LIFO kö? Stack

d) Varför vill du kunna mäta en algoritms komplexitet?

för att få ett estimat mått på dess effektivitet  
för att jämföra olika lösningar.

e) Vad menas med att en sorteringsalgoritm är naturlig?

att det gör snabbare att sortera en nedan  
sorterad mängd.

f) Ge ett exempel på en sorteringsalgoritm som är in-place och som är naturlig.

Bubble sorteras  
Insättning - om vi letar från höger i vänsterdelen.

g) Beskriv en funktion på en array som har linjär komplexitet – förklara också varför den har  
...linjär komplexitet.

sökning  
utskrift  
hitta min/max  
etc. } går igenom alla element i arrayen

h) Varför är basfallet viktigt i rekursion?

För att avsluta den rekursiva loopen. (utan brash)

## Uppgift 2: (7p)

Antag att följande funktion (funk) är given i pseudokod och att node är en nod i en enkellänkad lista (tecknet ! betyder *skiljt från*).

- a) Vad gör denna funktion exakt? (1p)

```
node
{
    Integer data
    node next
}

funk(node, x)
    current = node
    while current != NULL do
        if current.data == x then
            return true
        end if
        current = current.next
    end while
    return false
```

*en pekare till början*  
*så länge det finns noder kvar*  
*om vi har hittat rätt*  
*stege fram till nästa nod.*

Funktionen söker i listan som börjar vid node efter datat x. Returnerar true om det finns i listan, annars false.

- b) Implementera denna funktion i kod (2p) **P-uppgift**

```
int funk(node *n, int x)
{
    node *current = n;
    while (current != NULL)
        if (current->data == x)
            return 1;
        current = current->next;
    }
    return 0;
}
```

```
node * funk(node *n, int x)
{
    if (n != NULL)
        if (n->data == x)
            return n;
        return funk(n->next, x);
    }
    return NULL;
```

- c) Implementera en rekursiv version av funktionen som gör precis samma sak ..... (4p) **P-uppgift**

### Uppgift 3: (3p)

Du har ett uttryck som innehåller parenteser '(' och ')'. Uttrycket kan också innehålla andra tecken.

Beskriv hur du kan med hjälp av nedanstående stack-ADT kontrollera om alla parenteser i uttrycket matchar. Alltså om varje '(' har en matchande ')'. Du väljer själv om du vill beskriva detta genom att skriva kod, pseudokod, beskrivande text och/eller genom att rita. Din lösning ska på alla steg vara tydliga och enkla att följa.

Exempel på korrekt uttryck är: (x(x)(x(x(x))))

Exempel på icke korrekt uttryck är: (x)(x(x))

Den stack ADT du ska använda dig av kan lagra tecken (characters). Nedan finns dennes interface. Antag att stacken inte kan bli full.

*Stacken är baserad på en länkad lista där varje nod håller datadel (character) och pekare till nästa nod (next)*

```
Stack  
{  
    LinkedList *myStack  
}
```

*Precondition: stacken är inte full*

*Postcondition: c läggs högst upp på stacken*

*Returvärde: ingenting*

*Push(character c, stacken)*

*Precondition: det finns tecken på stacken*

*Postcondition: det översta tecknet på stacken är borttaget*

*Returvärde: ingenting*

*Pop(stacken)*

*Precondition: det finns tecken på stacken*

*Postcondition: stacken är oförändrad*

*Returvärde: det översta tecknet*

*character Peek(stacken)*

*Precondition: -*

*Postcondition: stacken är oförändrad*

*Returvärde: true om stacken är tom, annars false.*

*Boolean isEmpty(stacken)*

Gå igenom uttrycket tecken för tecken. Vid ( - push vid ) - pop, finns det inget att poppa är uttrycket fel. Om stacken inte är tom när uttrycket är slut är uttrycket fel.

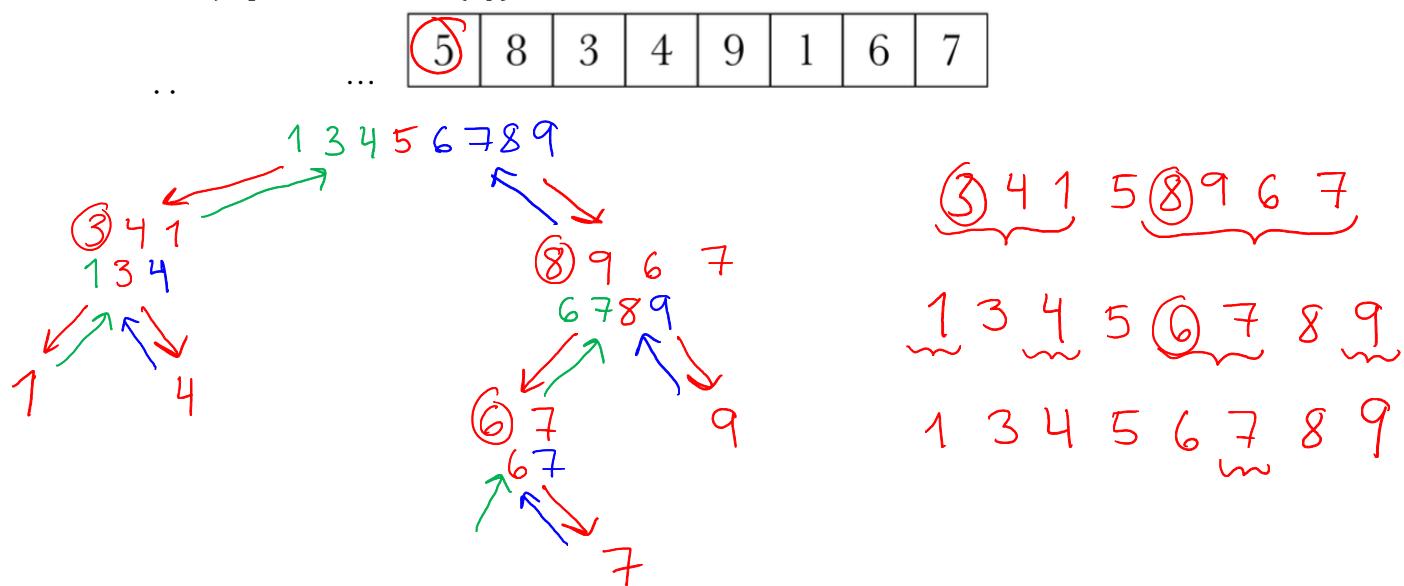
#### Uppgift 4: (7p)

a) Beskriv hur Quicksort fungerar (1p)

Välj ett pivot.

Flytta mindre värden så att de ligger till vänster om pivot och större värden så att de ligger till höger om pivot. pivot läggs på sin rätta plats mellan de två delarna. Sortera vänster & höger delmängd

b) Visa med bild/bilder (och text om det behövs) hur följande sekvenser sorteras med hjälp av Quicksort. (3p)



c) Hur påverkar valet av pivot värde Quicksort algoritmens effektivitet? - diskutera (2p)

Om ett pivot som hela tiden delar mängden ungefärlig i mitten  
väljs för vi en effektiv algoritm  $O(n \log n)$ .

Om vi väljer ett dåligt pivot som går att de flesta (eller alla)  
värden hamnar till vänster eller höger för vi samma effekter  $O(n^2)$

Att beräkna vilket data som ska vara pivot är dyrt ( $O(n^2)$ )

d) Vilken komplexitet har algoritmen i bästa fallet och varför? (1p)

Bästa:  $O(n \log n)$  - halverar datat och "står ihop".

- \* För varje delmängd gå igenom alla för att lägga mindre till vänster och större till höger  $\rightarrow O(n)$
- \* Delmängderna halveras vid varje steg (med ett bra pivot)  $\Rightarrow O(\log n)$

## Uppgift 5: (8p)

- a) Beskriv hur en sekvens av data sorteras med hjälp av urvalssortering. (1p)

Två delar, V och H. V är från början tom och kommer på slutet innehålla den sorterade mängden.

Hitta det minsta i H. Lägg det först i H och flytta fram gränsen mellan V och H.

⇒ Leta upp det minsta och lägg sist i sorterade

- b) Skriv koden för urvalssortering (4p) **P-uppgift**

```
void selectionSort (int arr[])
{
    int H=0, smal;
    while (H < SIZE)
    {
        smal = H;
        for (int i = H+1, i < SIZE, i++)
        {
            if (arr[i] < arr[smal])
                smal = i;
        }
        swap (arr, H, smal);
        H++;
    }
}
```

- c) Vilken komplexitet har urvalssortering i bästa fall samt sämsta fall – vilka fall är det som är bäst och sämst? (2p)

Sämsta == bästa ⇒ algoritmen gör alltid samma arbete.

$$O(n^2)$$

- d) Är algoritmen stabil? – motivera. (1p)

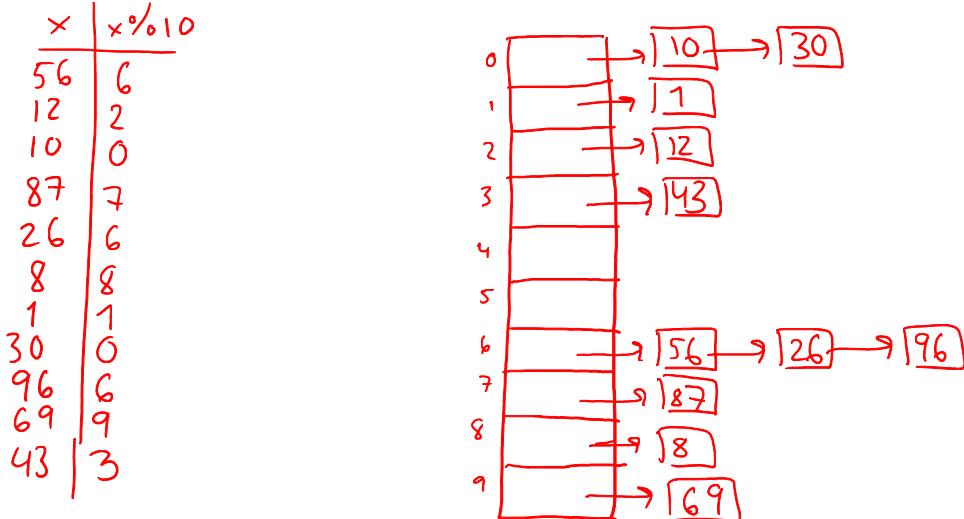
Ta - om man inte tar ett nytt antagande om minsta tal  
då de är lika. - annars Nej.

## Uppgift 6: (5p)

Antag att du har en hashtabell med plats för 10 element- Hashfunktionen är  $h(x) = (x \bmod 10)$  (tips: i praktiken blir detta alltså sista siffran i  $x$ ).

- a) Antag att tabellen använder chained-strategin (länkad lista) för att hantera krockar.  
 Visa hur hashtabellen ser ut när följande element sätts in i ordning: (2p)

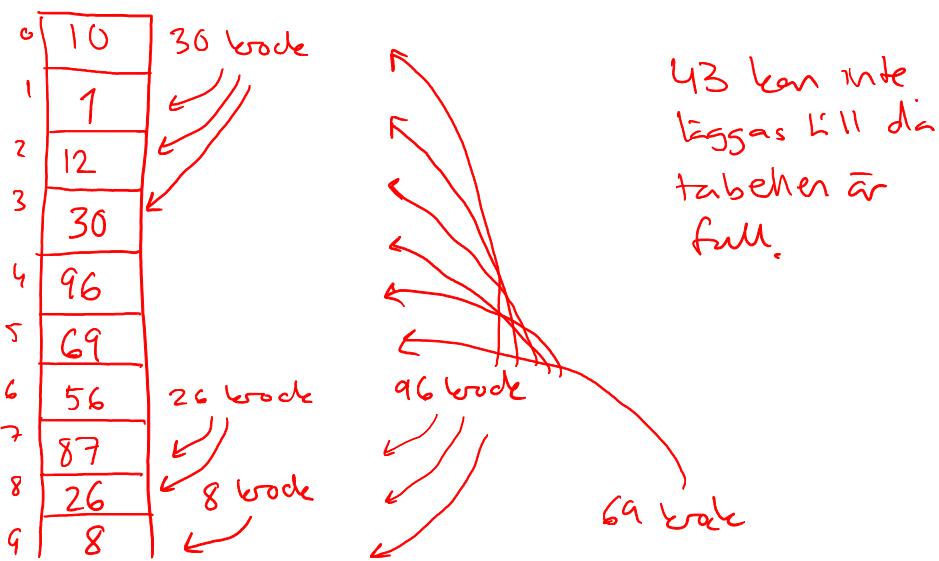
56 12 10 87 26 8 1 30 96 69 43



- b) Beskriv hur sökning går till i en listad hashtabell (1p)

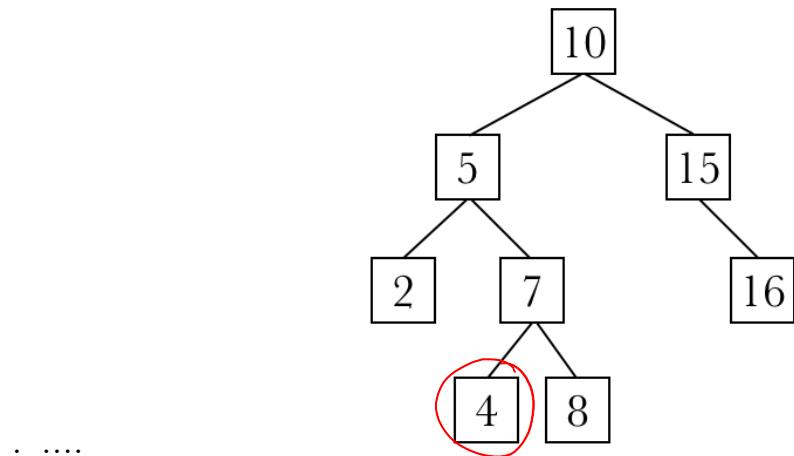
Anropa hashfunktionen för att få nyckeln (vilken plats i hasharranget). Leta igenom listan på det givna indexet.

- c) Antag nu istället att tabellen använder öppen adressering för att hantera krockar.  
 Visa hur hashtabellen ser ut när samma sekvens som i a) sätts in (i samma ordning).  
 (2p)



### Uppgift 7: (8p)

Antag följande träd:



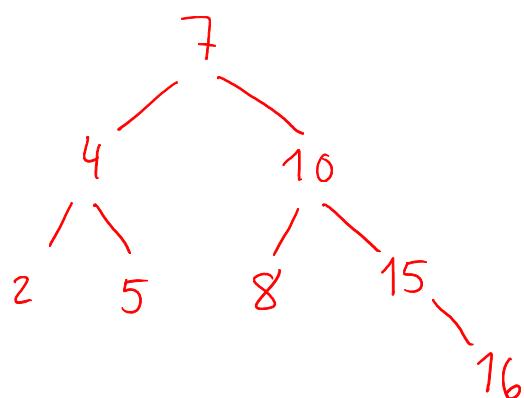
- a) Är trädet ovan ett binärt sökträd? - motivera (1p)

Binärt - Ja

Söktred - särskrat - Nej - 4 missar fel.

- b) Rita om trädet så att det blir ett balanserat binärt sökträd. (1p)

2 4 5 ⊕ 8 10 15 16



c) Antag följande träd och nod:

```
BTree  
{  
X node root  
}  
node  
{  
X: integer data  
X: node leftChild  
X: node rightChild  
}
```

```
void PreOrder(node *subTree)  
{  
    if (subTree != NULL)  
    {  
        printf("%d ", subTree->data);  
        PreOrder(subTree->leftChild);  
        PreOrder(subTree->rightChild);  
    }  
}
```

Skriv koden för att skriva ut trädet i Preorder (rekursivt). (2p) **P-uppgift**

d) Vilken är komplexiteten i det bästa fallet för sökning i ett binärt sökträd? – motivera

(1p)  $O(1)$  - hittar vid rooten.

e) Vilken är komplexiteten i det sämsta fallet för sökning i ett binärt sökträd? –

motivera (1p)

$O(\log n)$  - data finns inte.  
Halverar sökmängden vid varje ny nivå (varje steg).

f) Om man vill söka efter ett data i en stor mängd. När bör man använda hashtabell och när behöver man använda binärt sökträd – diskutera. (2p)

Om sökningen aldrig får leda till  $O(n)$  ska ett balanserat binärt sökträd användas.

Likaså om ni inte vet mängden data (det är därför att hitta en bra storlek på hashtabellen).