

Tentamen - Datastrukturer, algoritmer och programkonstruktion.

DVA104

Akademin för innovation, design och teknik

Måndag 2015-08-17

Skrivtid: 14.30-19.30

Hjälpmedel: Bok relaterad till ämnet – ej digital (inga föreläsningsanteckningar eller egna anteckningar är tillåtna)

Lärare: Caroline Uppsäll, 021-101456

Betygsgränser

Betyg 3: 25p - varav minst 8 poäng är P-uppgifter

Betyg 4: 36p

Betyg 5: 42p

Max: 47p - varav 14 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 0: (0p)

Läs noga igenom instruktionerna på förstasidan och följ dem!

Uppgift 1: (9p)

- a) Varför är basfallet viktigt i rekursion? (1p)

För att avsluta den rekursiva loopen.

Utan ett basfall som nås kommer programmet fastna i rekursiva årsningar - tills programmet kraschar.

- b) Vad innebär det att en sorteringsalgoritm är stabil? (1p)

Att den relativira ordningen mellan lika data behålls.

- c) Vad innebär det att en algoritm har kvadratisk komplexitet? (1p)

$O(n^2)$ - Exekveringstiden blir proportionell mot $n \times n$ där n är mängden data. (Ofta nästlade loopar som båda är beroende av n)

Ju större n är desto snabbare växer exekveringstiden.



- d) Nämnn två sätt på vilka man kan optimera standardlösningen av en bubblesort. (2p)

- Avsluta om ett helt varv har gjort utan några byten
- Jämför inte med data/platser som blivit korrektata på tidigare varv.

- e) Beskriv hur en stack (LIFO-kö) fungerar. (1p)

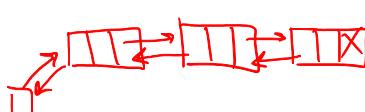
Last In First Out. Lägg in på toppen och ta bort från toppen.

Det element som varit längst tid i stacken tas bort först

- f) Vad används ordobegreppet (O-notation) till? (1p)

Komplexitet - beskriva en algoritms effektivitet

- g) Beskriv kortfattat vad en dubbellänkad lista är och hur en sådan fungerar. (2p)



En linjär struktur för att lagra data.

Vareje nod innehåller data & två länkar (till föregående nod och till nästa nod).

Man kan "vandra" både framåt & bakåt i listan.

Uppgift 2: (6p)

- a) Beskriv hur binärsökning i en linjär sekvens går till, finns det några förutsättningar som måste vara uppfyllda för att binärsökning ska kunna tillämpas? (1p) *Listan måste vara sorterad.*

titta på mitten.

↳ om det vi söker => klara!

↳ om det vi söker är mindre => utför binärsökning på vänster delsekvens.

↳ om det vi söker är större => utför binärsökning på höger delsekvens.

- b) Skriv koden för en funktion som utför binärsökning på en sorterad linjär sekvens. Du kan anta att sekvensen är en array/ett fält av heltal. Sekvensen och det eftersökta värdet ska tas som inparametrar till funktionen. (3p) ---P-uppgift

```
int binSearch ( int arr[], int value, int low, int high )
{
    if (low <= high)
    {
        int mid = (low+high)/2;
        if (arr [mid] == value)
            return 1;
        else if (arr [mid] > value)
            return binSearch (arr, value, low, mid-1);
        else
            return binSearch (arr, value, mid+1, high);
    }
    return 0;
}
```

- c) Vilken komplexitet har funktionen (i bästa samt sämsta fall) och varför? (2p)

Bästa: hitta direkt i mitten => O(1)

Sämsta (samt medel): hittar "längst ner i hierarkin" => O(log₂ n) => halvera till endast 1 element återstår.

*eller
inte
finns*

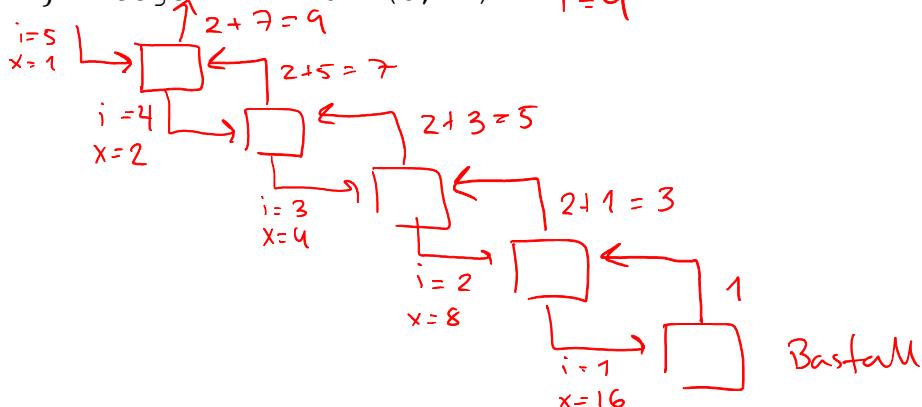
Uppgift 3: (5p)

Antag följande rekursiva funktion (skriven i pseudokod)

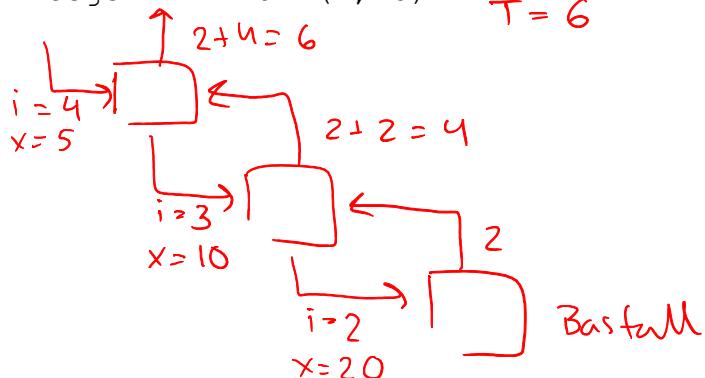
```
Integer Funk(Integer i, Integer x)
    If x <= 10 AND i > 0 then
        Return 2 + Funk(i-1, x*2)
    Else then
        Return i
End Funk
```

- a) Beskriv vilket resultatet blir efter följande anrop (vad ligger i variabeln T efter att anropet har körts) och motivera varför (med bild och/eller text). (3p)

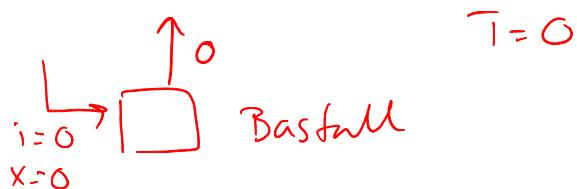
1) Integer T = Funk(5, 1) $T=9$



2) Integer T = Funk(4, 5) $T=6$



3) Integer T = Funk(0, 0)



```

Integer Funk(Integer i, Integer x)
    If x <= 10 AND i > 0 then
        Return 2 + Funk(i-1, x*2)
    Else then
        Return i
End Funk

```

- b) Skriv om ovanstående rekursiva funktion till en iterativ algoritm som gör samma sak.
(2p) ---P-uppgift

```

int funk(int i, int x)
{
    int sum = 0;
    while (x <= 10 && i > 0)
    {
        i = i - 1;
        x = x * 2;
        sum = sum + 2;
    }
    sum = sum + i;
    return sum;
}

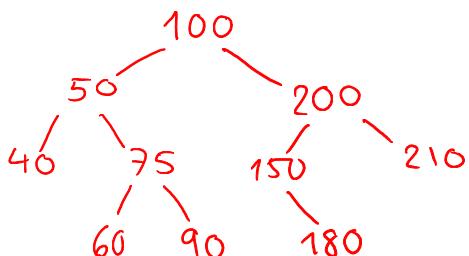
```

Uppgift 4: (8p)

- a) Vad innebär det att ett binärt träd är balanserat? (1p)

All varje delträdets vänster och höger delträd är lika "tunga" (i den mån det är möjligt).

- b) Rita upp ett binärt sökträd med elementen i nedanstående lista. Ta elementen ett och ett från vänster till höger ur listan nedan och lägg in i trädet. (2p)
100, 50, 75, 60, 200, 150, 180, 40, 90, 210



- c) Vilken traverseringsmetod bör användas på ditt träd ovan (från uppgift b) för att skriva ut elementen i storleksordning (från minsta till största)? (1p)

Inorder (LR)

- d) Visa hur utskriften av ditt träd ovan (från uppgift b) ser ut om traverseringsmetoden postorder används. (1p)

(LR)

Postorder : 40 60 90 75 50 180 150 210 200 100
Vänster
Höger
Besök

- e) Antag att du har noden nedan (skriven i pseudokod). Skriv funktionen som letar upp ett angivet data i trädet. Du väljer själv om funktionen ska vara rekursiv eller iterativ. Det eftersökta värdet ska tas som parameter till funktionen och 1/true eller 0/false ska returneras beroende på om datat hittats i trädet eller inte. Du kan anta att det inte finns några dubbletter i trädet samt att alla "tomma" länkar är satta till NULL. Länkarna (leftChild och rightChild) är pekare/referenser.

(3p) ---P-uppgift

```
typedef struct BTreenode * BSTree;
```

```
BTreenode
    Integer Node data
    BTreenode leftChild
    BTreenode rightChild
```

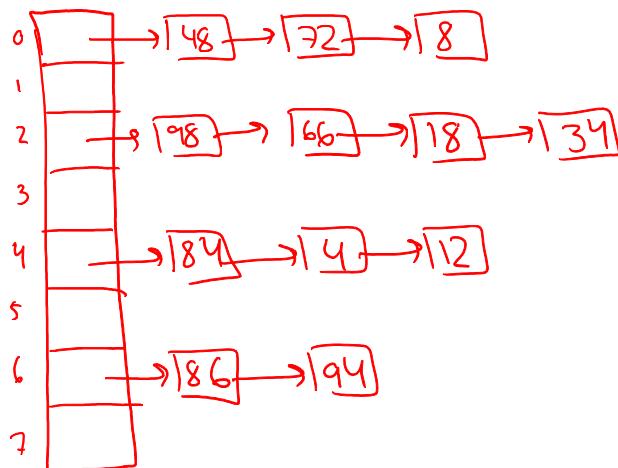


```
int search (BSTree tree, int data)
{
    if (tree == NULL)
        return 0;
    if (tree->data == data)
        return 1;
    else
        if (data < tree->data)
            return search (tree->leftChild, data);
        else
            return search (tree->rightChild, data)
}
```

Uppgift 5: (7p)

- a) Ta en från början tom listad (chained) hashtabell X vars hashfunktion är $x \bmod 8$. Hashtabellen har 8 platser. Rita en bild över vad som händer när följande sekvens av data sätts in (i ordning från vänster till höger). (2p)
- 48, 72, 98, 66, 84, 18, 34, 86, 8, 4, 12, 94

x	$x \% 8$
48	0
72	0
98	2
66	2
84	4
18	2
34	2
86	6
8	0
4	4
12	4
94	6



- b) Bedöm hashfunktionens kvalitet, diskutera med avseende på t.ex. minnesåtgång, komplexitet etc. (3p)

Ganska många krockar.

12 värden ska läggas in, 8 platser. \Rightarrow längre listor.

En lång lista ger $O(n)$ istället för $O(1)$

För kortare listor (färre krockar) bör tabellen vara större.

- c) Beskriv hur hashning med öppen adressering (linear probing) fungerar, vad händer om ovanstående sekvens av data läggs till i en hashtabell som använder just öppen adressering för att hantera krockar? (2p)

Vid krock letar man linjärt efter nästa lediga plats.

Om tabellen fortfarande är 8 platser så kan endast 8 element läggas till.

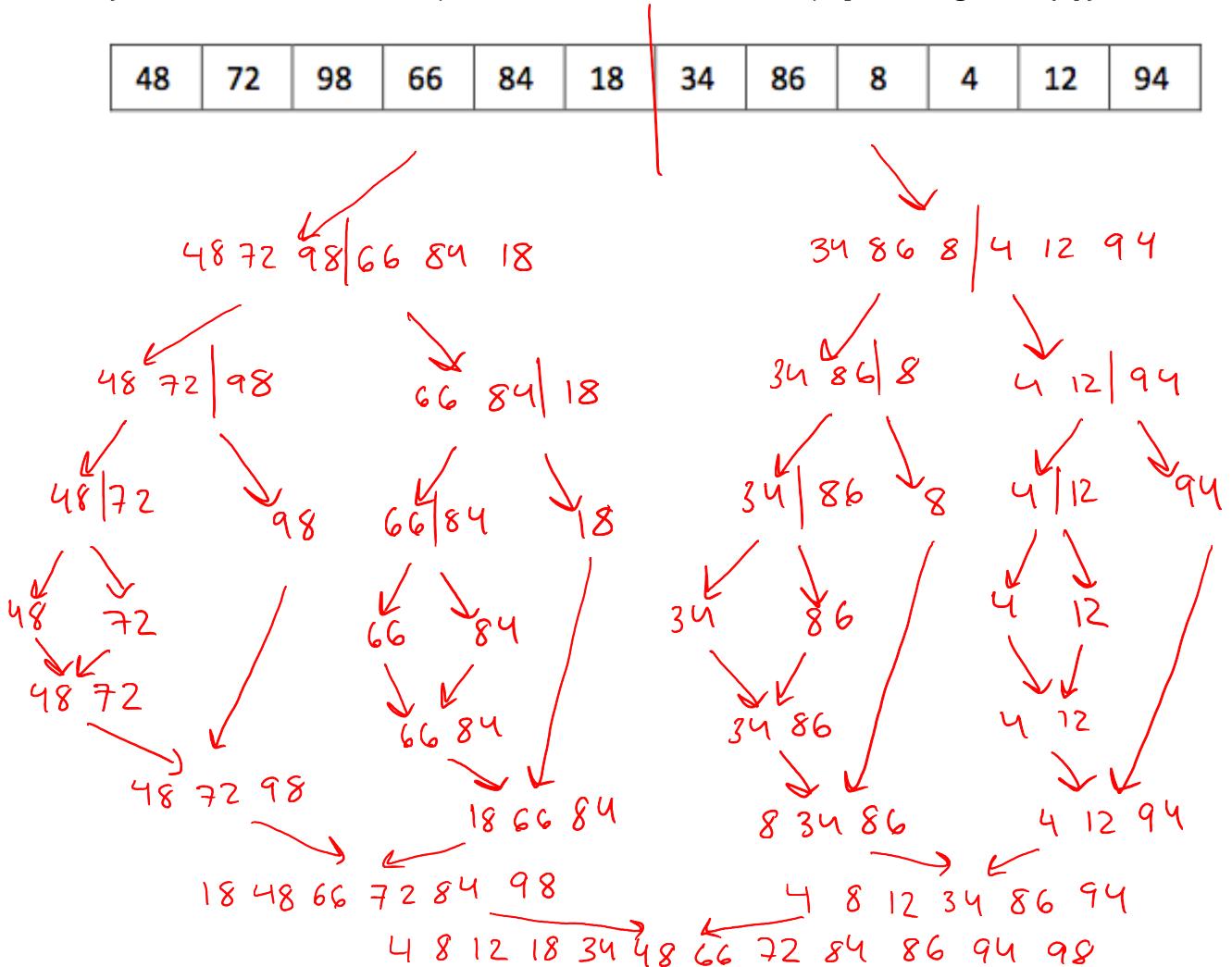
0	48
1	72
2	98
3	66
4	84
5	18
6	34
7	86

Storleken måste vara minst 12, gärna lite större för att minimera krockar.

Fultt!

Uppgift 6: (6p)

- a) Visa med en bild hur följande sekvensens sorteras med hjälp av MergeSort. (2p)



- b) Gör en klassificering av MergeSorts egenskaper (bästa komplexitet, sämsta komplexitet, minnesåtgång, naturlig, stabil). Glöm inte att förklara klassificeringen. (4p)

Komplexitet: $\hat{\text{Bästa}} = \hat{\text{sämsta}} = O(n \log n)$ - algoritmen gör alltid samma arbete

Vare nivå innehåller n element = $O(n)$

För varje nivå halveras delarrayerna = $O(\log n)$

Minne - ej in-Place - extra minne krävs vid ihopslagningen

Naturlig - Nej - samma jobb ska göras oavsett om mängden är sorterad eller inte

Stabil - Ja om mängd 1 (den vänstra) prioriteras vid lika värden (vid merge)

Uppgift 7: (3p)

Definiera en ADT för en kö (FIFO). Informationsdelen ska vara ett heltal. Glöm inte pre- och postconditions samt eventuellt returvärde. Kön ska använda sig av nedanstående länkade lista (skriven i pseudokod). Länkarna (next, previous, head och tail) är pekare/referenser. (3p) ---P-uppgift

```
LinkedListNode
    Integer data
    LinkedListNode next
    LinkedListNode previous
```

```
LinkedList
    LinkedListNode head
    LinkedListNode tail
```

typedef LinkedList Queue;

{post: en tom lista/kö är skapad
Queue create Queue (void);

{post: data läggs till i queue
void enqueue (Queue *queue, int data);

{pre: kö är inte tom
post: det första elementet är borttaget från queue
void dequeue (Queue *queue);

{pre: kö är inte tom
int front (Queue queue);

int isEmpty (Queue queue);

Uppgift 8: (3p)

Sorteringsfunktionen QuickSort använder sig av två funktioner. En (QuickSort) som hanterar den övergripande logiken och en (Partition) som plockar ut pivot-värde samt placerar mindre värden till vänster och större till höger. Den senare har följande funktionshuvud (skrivet i pseudokod) där sekvensen är en array/ett fält av heltal.

Integer Partition(Integer sekvens, Integer first, Integer last)
Din uppgift är att skriva pseudokoden (eller koden, du väljer själv) för den övergripande, rekursiva funktionen QuickSort. Visa också hur det initiala anropet till funktionen ser ut. (3p) ---P-uppgift

```
void QuickSort ( int arr[ ], int first, int last )
{
    if (first <= last)
    {
        int pivot_pos = Partition (arr, first, last);
        QuickSort (arr, first, pivot_pos - 1);
        QuickSort (arr, pivot_pos + 1, last);
    }
}
```

Anrop: QuickSort (arr, 0, n-1)
 ↑
 sizeof(arr)/4