FACIT /SOLUTIONS

Tentamen - Datastrukturer, algoritmer och programkonstruktion.
DVA104

----------LÖSNINGSFÖRSLAG----------

Akademin för innovation, design och teknik
Måndag 2020-01-13

Skrivtid: 14.30-19.30
Hjälpmedel: Valfritt icke-elektroniskt material
Lärare: Caroline Uppsäll
(kan nås på telefon om du frågar tentavakten)

Preliminära betygsgränser:
Betyg 3: 17p
Betyg 4: 25p
Betyg 5: 30p
Max: 33p

Allmänt

- På uppgifter där du ska skriva kod ska koden skrivas i C.
- Markera tydligt vilken uppgift ditt svar avser.
- 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!
- Tips: Läs igenom hela tentan innan du börjar skriva för att veta hur du ska disponera din tid.
- Förklaringar/svar som till viss del är rätt och till viss del är fel ger inga poäng.

Lycka till!

FACIT /SOLUTIONS

Uppgift 1 [3p]

Vilka av följande påståenden är sanna? Du får maximalt svara att 6 påståenden är korrekta, anger du fler är det endast de 6 första som kommer att bedömas.

a) Ett annat namn för en stack är FIFO-kö
b) En arrayimplementation av en kö kan vara cirkulär
c) En ADT tas fram med utgångspunkt i vilket data som ska lagas
d) En oriktad graf kan vara starkt sammanhängande.
e) I ett set bryr man sig inte om i vilken ordning datat är lagrat.
f) Sökning i en länkad lista har bättre komplexitet än sökning i ett balanserat binärt sökträd.
g) I ett binärt träd kan varje nod ha maximalt två barn.
h) Ett minimum spanning tree (av en graf) ger den lägsta vikten som sammanknyter alla noder i grafen.
i) Quick sort har i sämsta fall komplexitet $O(n \log_2 n)$
j) Insättningssortering (insertion) och urvalssortering (selection) har samma komplexitet i värsta fall.
k) En hashtabell bör vara 1.3 gånger större än mängden data som ska sparas samt vara ett primtal.
l) Sökning i en hashtabell har alltid konstant komplexitet.

Uppgift 2 [3p]

Antag följande program:

```
int funk(int n)
{
    if (n == 0)
        return 1;
    else
        return n*funk(n-1);
}
```

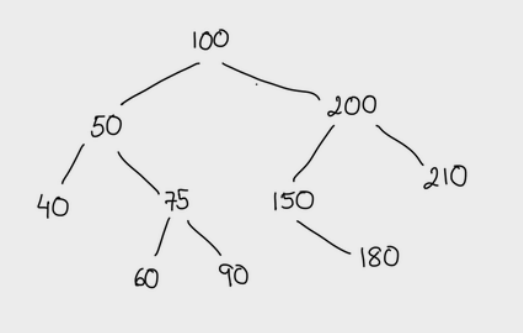
Vilken blir resultatet av följande anrop?

a) funk(5); → 120
b) funk(0); → 1
c) funk(-2); → når aldrig basfallet och kommer därför göra rekursiva anrop tills det krashar

Uppgift 3 [5p]

a) [1p] Rita upp ett binärt sökträd med elementen i nedanstående lista (i angiven ordning från vänster till höger).
100, 50, 75, 60, 200, 150, 180, 40, 90, 210

Svar



b) [1p] Vilken blir utskriften om du skriver ut det träd du byggde i a) med postorder (LR)?
40. 60. 90. 75. 50. 180. 150. 210. 200. 100

c) [3p] Antag att du har följande nod och typdeklaration.

```
struct treeNode
{
      int data;
      struct treeNode* left;
      struct treeNode* right;
};

typedef struct treeNode* BSTree;
```

Implementera en funktion som skriver ut alla noder som har exakt ett barn – utskriften ska ske i storleksordning från minsta till största.

Pekaren till trädet heter root och är deklarerad som
`BSTree root`
Du kan anta att root pekare på ett korrekt binärt sökträd.

Visa också hur anropet till din funktion ser ut.

```
void printNodesWithOneChild(BSTree tree)
{
    if(tree == NULL)
        return;
    printNodesWithOneChild(tree->left);
    if((tree->left != NULL && tree->right == NULL)||
        (tree->left == NULL && tree->right != NULL))
        printf("%d  ", tree->data);
    printNodesWithOneChild(tree->right);
}

Anrop:
printNodesWithOneChild(root);
```

FACIT /SOLUTIONS

Uppgift 4 [2p]

Gör en klassificering av urvalssorteringsalgoritmens egenskaper (komplexitet i bästa/värsta fall, minnesåtgång, naturlig, stabil). Självklart ska alla klassificeringarna ha tillhörande förklaringar.

Komplexiteten är alltid $O(n^2)$ eftersom samma arbete alltid genomförs – oavsett hur mängden ser ut. För alla tal (n) så måste den minsta hittas i H (n).
Minnesåtgång – inPlace – inget extra minne behövs.
Naturlig – Nej – ej snabbare för redan sorterad mängd.
Stabil – Ja – om man vid lika tal inte byter antagande om vilket som är minst.

Uppgift 5 [3p]

Vilken komplexitetsklass hamnar följande funktioner/beskrivningar av funktioner i? Självklart krävs både korrekt komplexitetsklass och en korrekt förklaring/beskrivning för att erhålla poäng på deluppgiften.

a)

```
int funk(int n, int m)
 {
     if(n < 1)
         return m;
     else if(n > 10)
         return silly(n-1, m);
     else
         return silly(n/2, m);
}
```

Stora n är de som spelar roll – O(n) – om man tänker att det är rekursivt anrop (vilket var tanken med uppgiften).
O(1) eftersom det inte är rekursiva anrop och vi inte vet vad silly gör.

b) En algoritm som söker efter ett data i en sorterad array genom att först titta på det mittersta elementet och därefter söka på samma sätt i den vänstra arrayhalvan om det eftersökta talet är mindre än det mittersta och i höger arrayhalva om det är större. Sorteringen upprepas på samma sätt till datat hittas eller det inte finns några element kvar i delarrayen.
Binärsökning – $O(\log_2 n)$

c)

```
void fillList(float *A, int n, struct node *head)
{
      for(int i = 0; i < n; i++)
          addNodeLast(head, A[i]);
 }
```

head är en pekare på en tom enkellänkad lista. A är en array som innehåller de värden som ska läggas in i listan och n är antalet element i arrayen. Funktionen addNodeLast lägger till noden sist i listan.

För alla n, gå igenom alla i listan för att placera sist – $O(n^2)$

Uppgift 6 [5p]

OBS! Svar ges på ett blad som rivs loss från denna tentamen – sista bladet.
Nedan finns en möjlig implementation av algoritmen insättningssortering.

```
void insertionSort(int a[], int n)
{
    int i, location, j, k, selected;

    for(i = 1; i < n; ++i) /*Gå igenom alla element i den osorterade
                             delarrayen*/
    {
        j = i-1; /*sätt gränsen mellan den sorterade och osorterade
                   arraydelen*/
        selected = a[i]; /*det tal som ska sorteras*/
        for(k = 0; k <= j; k++) /*Ta reda på var talet ska ligga*/
            if(a[k] > selected)
                break;
        location = k; /*talets position i den sorterade halvan är
                        hittad och lagrad i location*/
        location = binarySearch (a, selected, 0, j); /*Det går också
   bra att sätta k = binarySearch(…) och inte stryka location = k;*/

        while(j >= location) /*Flytta alla efterföljande för att
                               skapa en lucka för talet*/
        {
            a[j+1] = a[j];
            j--;
        }
        a[location] = selected; /*Sätt in talet på sin rätta plats*/
    }
}
```

Man kan effektivisera insättningssorteringen genom att kombinera den med binärsökning. Var i algoritmen ska binärsökningen i så fall göras?

Antag att du har en funktion som utför binärsökning som har följande funktionshuvud.

```
int binarySearch(int a[], int item, int low, int high);
```

binarySearch returnerar en position i arrayen a, item är det tal som ska sorteras och low och high används för att ange gränsvärden för en delarray.

a) [3p] Du ska ersätta del av koden ovan med anrop till binärsökningsfunktionen och på så sätt effektivisera algoritmen för insättningssortering.

b) [2p] Du ska också (på angivet svarspapper) förklara varför detta blir en effektivisering. Använd gärna komplexitetsklasserna.

Istället för att leta igenom (i värsta) fall hela vänsterdelen för att hitta rätt position (vilket är O(n)) så hittar binärsökning rätt position med komplexiteten O(log n)

FACIT /SOLUTIONS

Uppgift 7 [2p]

Antag att du har ett balanserat binärt sökträd med 5000 noder. Vilket är det minsta möjliga djupet för ett sådant träd?

Ange antalet nivåer och en motivering på maximalt 3 meningar alternativt en eller flera beräkningar.

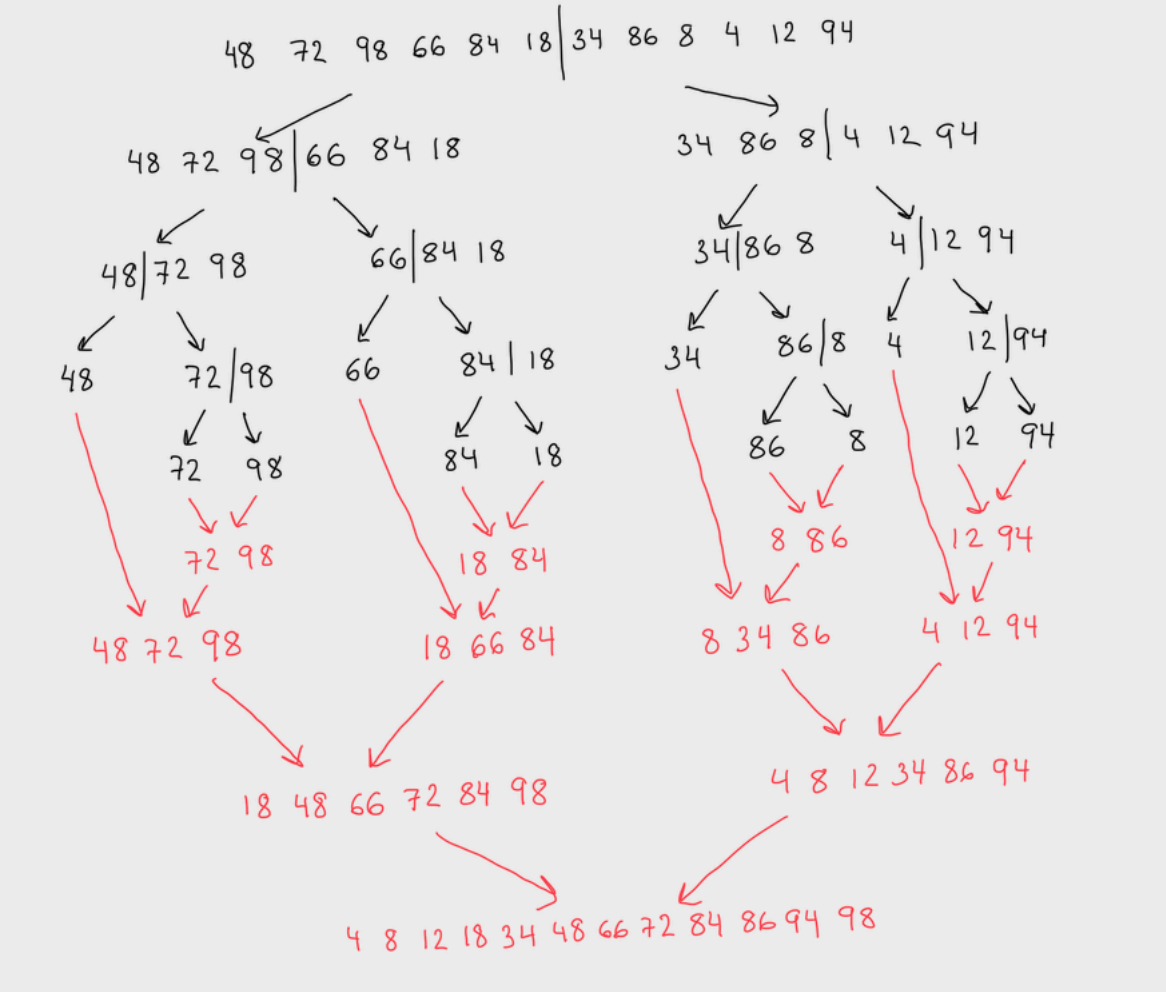
Nivå	Noder på nivån	Totalt antal noder i trädet
1	1	1
2	2	3
3	4	7
4	8	15
5	16	31
6	32	63
7	64	127
8	128	255
9	256	511
10	512	1023
11	1024	2047
12	2048	4095
13	4096	8191

Ett sådant träd måste ha minst 13 nivåer – djup 13

Uppgift 8 [2p]

Visa med en (eller flera) bild hur följande array av data sorteras med hjälp av algoritmen Merge sort.

48	72	98	66	84	18	34	86	8	4	12	94



FACIT /SOLUTIONS

Uppgift 9 [4p]

a) [2p] Antag att du har hashtabellen nedan som använder linjär sondering (linear probing) för att lösa krockar. Hashfunktionen som används gör key%SIZE.

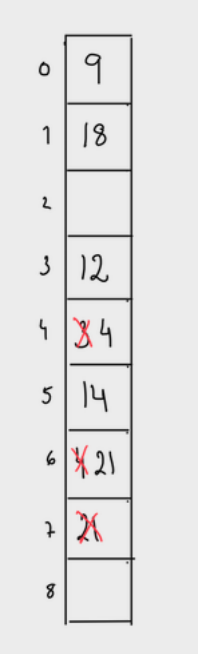
I vilken ordning kan elementen ha lagts till i tabellen? Det finns flera alternativ nedan som är korrekta och för full poäng ska du ange samtliga korrekta alternativ. Felaktigt angivet alternativ kvittas mot korrekt angivet alternativ.

0	9
1	18
2	
3	12
4	3
5	14
6	4
7	21
8	

a) 9, 14, 4, 18, 12, 3, 21
b) 12, 3, 14, 18, 4, 9, 21
c) 12, 14, 3, 9, 4, 18, 21
d) 9, 12, 14, 3, 4, 21, 18
e) 12, 9, 18, 3, 14, 21, 4

I A skulle 4 sättas in på index 4 istället för på index 6
I B skulle 18 sättas in på index 0 istället för index 1
I E skulle 21 sättas in på index 6 istället för index 7

b) [1p] Ta bort data 3 från tabellen och rita hur den ser ut efter borttagningen är färdig.

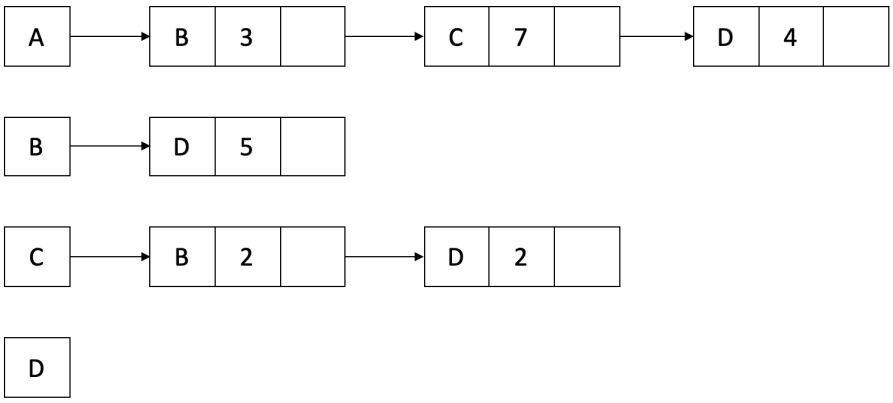


c) [1p] Om vi vill ha en hashtabell som ska lagra strängar, en möjlig hashfunktion är att göra följande: `strlen(key)%hashsize;` där key är själva strängen.

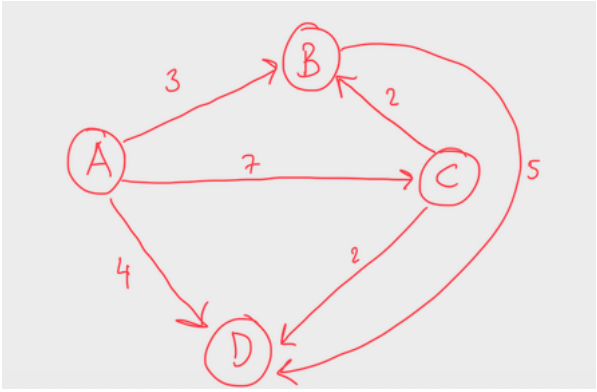
Är det en bra hashfunktion? Motivera ditt svar.
Nej, strängar med samma längd skulle hamna på samma index. Om man sätter in många strängar som är lika långa (vilket är troligt) så skulle det leda till många krockar och en komplexitet på O(n)

Uppgift 10 [4p]

Nedan ser du en listrepresentation av en riktad och viktad graf.



a) [2p] Rita upp hur grafen ser ut baserat på listrepresentationen ovan.



b) [2p] Ett annat sätt att representera en graf på är med hjälp av en matris. Rita matrisen för den graf du har ritat i a).

	A	B	C	D
A	0	3	7	4
B	0	0	0	5
C	0	2	0	2
D	0	0	0	0

FACIT /SOLUTIONS

Anonymiseringskod

Uppgift 6 [5p] - Svarspapper

```
/*Funktionsdeklaration för funktionen som genomför binärsökning.
Returvärdet är en positivon i arrayen a. item är det tal som ska
sorteras och low och high används för att ange gränsvärden för en
delarray.*/
int binarySearch(int a[], int item, int low, int high);


void insertionSort(int a[], int n)
{
    int i, location, j, k, selected;

    for(i = 1; i < n; ++i) /*Gå igenom alla element i den osorterade
                             delarrayen*/
    {
        j = i-1; /*sätt gränsen mellan den sorterade och osorterade
                   arraydelen*/
        selected = a[i]; /*det tal som ska sorteras*/
        for(k = 0; k <= j; k++) /*Ta reda på var talet ska ligga*/
            if(a[k] > selected)
                break;
        location = k; /*talets position i den sorterade halvan är
                        hittad och lagrad i location*/
        while(j >= location) /*Flytta alla efterföljande för att
                               skapa en lucka för talet*/
        {
            a[j+1] = a[j];
            j--;
        }
        a[location] = selected; /*Sätt in talet på sin rätta plats*/
    }
}
```

a) [3p] Stryk de satser i funktionen ovan som ska ersättas med binärsökning och skriv anropet till binärsökning här:

b) [2p] Varför gör införandet av binärsökning i insättningssortering att sorteringen blir effektivare? Använd gärna komplexitetsklasserna.