Tentamen - Datastrukturer, algoritmer och programkonstruktion.
DVA104

Akademin för innovation, design och teknik
Onsdag 2019-01-16

Skrivtid: 08.30-13.30
Hjälpmedel: Handskrivna anteckningar (obegränsad mängd) samt ordbok/lexikon
Lärare: Caroline Uppsäll
(kan nås på telefon om du frågar tentavakten)

Preliminära betygsgränser:
Betyg 3: 22p
Betyg 4: 33p
Betyg 5: 39,5p
Max: 44p

Allmänt

- Kod skriven i tentauppgifterna är skriven i C-kod.
- 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, börja ny uppgift på nytt papper.
- 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.

Lycka till!

Uppgift 1 [3p]

Vilka av följande påstående är sanna?
Felaktigt svar kvittas mot angivet korrekt svar.

1) När man skapar en ADT utgår man från vad man vill kunna göra med datatypen.
2) När man skapar en ADT utgår man ifrån hur datatypen ska fungera.
3) ADT är ett begrepp specifikt för programspråket C.
4) I en ADT vill man synliggöra så många detaljer som möjligt för användaren.
5) En ADT består av två olika komponenter, interface och implementation.
6) Allt som går att lösa iterativt går också att lösa rekursivt, och vice versa.
7) Många problem som går att lösa iterativt går också att lösa rekursivt, men inte alla.
8) En rekursiv lösning kräver generellt mindre minne än en iterativ lösning.
9) En rekursiv lösning kräver generellt mer minne än en iterativ lösning.
10) Rekursion innebär att en funktion anropar en annan funktion med samma argument/parametrar.
11) Linjär komplexitet är bättre än logaritmisk komplexitet.
12) Komplexitetsklasserna beskriver en algoritms minnesåtgång.
13) Komplexitetsklasserna beskriver en algoritms effektivitet i termer av tid.
14) Det måste finnas en implementerad lösningar på ett problem (alltså kod) för att komplexitetsklassen ska kunna tas fram.
15) FIFO är ett annat namn på en stack.
16) Binärsökning kräver att den linjära mängden som sökningen ska utföras i är sorterad.
17) Linjärsökning kan endast användas på en sorterad mängd.
18) Quick sort och Merge sort är alltid lika effektiva (oberoende av mängdens sortering).
19) Urvalssortering (Selection sort) är naturlig.
20) Inget av ovanstående är sant.

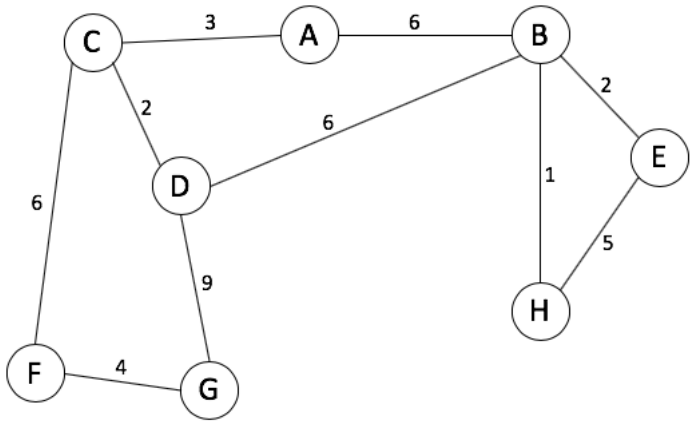
Uppgift 2 [5p]

a) *[2p]* Implementera en arraybaserad cirkulär kö, kön ska kunna spara/innehålla 10 heltal. Du ska alltså skapa själva kön samt nödvändiga variabler för att arbeta med kön. Kön ska från början innehålla endast 0:or, samtliga variabler som skapas ska vara initierade till korrekta startvärden.

b) *[3p]* Implementera funktionen enQueue som lägger till ett data (heltal) i kön du skapade i a. Om kön är full ska självklart ingenting läggas till i den, du får själv välja hur du vill hantera det.

Uppgift 3 [12p]

Du ska i följande deluppgifter arbeta med följande graf.



a) Är grafen riktad eller oriktad?
b) Är grafen en multigraf eller inte?
c) Är grafen viktad eller oviktad?
d) Innehåller grafen några cykler? Om ja, ge ett exempel.
e) Är grafen sammanhängande eller osammanhängande?
f) Vilken grad har nod D?

[a-f ger totalt 3p]

g) *[2p]* Rita upp hur en arrayrepresentation av grafens bågar ser ut, använd matrisen på sista sidan i tentamen (du kan riva bort sidan från tentamen och lämna in den tillsammans med dina svar om du inte vill skriva av den på annat papper).

h) *[2p]* Förklara vad ett shortest path tree är samt ge ett exempel på när ett sådant skulle vara lämpligt att ta fram. Använd max 5 meningar.

i) *[3p]* Rita det minimum spanning tree med start i noden A man får genom att följa Prim's algoritm. Om du vill kan du använda svarspappret längst bak i tentamen för att göra anteckningar i grafen. Det går även bra att du ger ditt svar där, glöm dock inte att riva loss det och lämna in det tillsammans med dina övriga svar.

j) *[2p]* Använd en kö (FIFO) för att skriva ut grafen enligt bredden först med start i A. Då flera noder ska läggas till i kön samtidigt ska de läggas in i bokstavsordning.

Uppgift 4 [3p]

Antag att du har en dubbellänkad lista bestående av följande nodtyp och listtyp

```
typedef struct node
{
    int data;
    struct node *next;
    struct node *prev;
}Node;

typedef struct list
{
    Node *head;
    Node *tail;
}List;
```

Listan består av en head-pekare till första noden samt en tail-pekare till sista noden, den sista nodens next är satta till NULL och likaså den första nodens prev. head och tail måste alltid peka på första respektive sista noden. För en tom lista är dessa satta till NULL.

När en nod skapas sätts alltid dess next och prev till NULL.

Skriv funktionen som lägger till data sist i listan. Följande funktionshuvud ska användas

```
int addLast(List *list, int data);
```

Om det går att lägga till datat sist i listan ska 1 returneras, annars 0.

Du kan anta att det finns en funktion

```
static struct Node* createListNode(const int data);
```

som skapar en ny nod med angivet data (next och prev är satta till NULL) samt returnera noden. Om det inte går att skapa noden returnerar funktionen NULL.

Uppgift 5 [3p]

Ett och samma binära sökträd har skrivits ut med ordningarna Preorder (LR), Inorder (LR) och Postorder (LR). Vilket av nedanstående alternativ är utskrivet med vilken traverseringsordning? Notera att tre av alternativen inte använder någon av ovanstående ordningar.

a) 25, 27, 14, 13, 12, 10
b) 27, 25, 14, 13, 12, 10
c) 10, 12, 14, 13, 27, 25
d) 10, 12, 13, 14, 25, 27
e) 10, 12, 14, 13, 25, 27
f) 25, 13, 10, 12, 14, 27

Uppgift 6 [5p]

När man arbetar med hashtabeller så accepterar man att krockar kommer att ske då olika nycklar genererar samma hashindex, frågan är hur vi löser dessa krockar. Ett sätt att lösa krockar på är med hjälp av öppen adressering vilket innebär att man vid krock letar vidare i tabellen tills en ledig plats hittas. Du ska i denna uppgift göra detta på två olika sätt.

I båda deluppgifterna nedan så ska du utgå ifrån en tom hashtabell med 10 platser (SIZE). De nyckel-värdepar som ska sättas in är (i ordning):
13 - banan
27 - kiwi
2 - gurka
22 - melon
33 - apelsin
10 - tomat
16 - potatis
26 - majs
19 - morot
18 – citron
33 – clementin (OBS! denna ska endast läggas till i deluppgift a)
Siffran är nyckeln och frukten/grönsaken är värdet

Glöm inte att visa beräkningen av hashindexet.

a) *[2p]* Använd nu linjär sondering (linear probing) för att sätta in ovan nämna nyckel-värdepar i ovan nämnda hashtabell. Hashfunktionen som ska användas gör
`Nyckel % SIZE`
Visa hur tabellen ser ut efter insättningen (både nyckeln och värdet ska visas), visa också beräkningarna av index.

b) *[3p]* Ett annat sätt att lösa krockar på i öppen adressering är med hjälp av en så kallad dubbel hashning (double hashing). Den går till såhär:

Istället för att endast använda en hashfunktion så används två. Den första (hash1) är den som i första hand används, om krock uppstår vid användning av hash1 så anropas den andra hashfunktionen (hash2). När hash2 har beräknats så används följande formel för att hitta ett index:
`(hash1 + i*hash2)%SIZE`
Där i är en räknare som börjar på 1 och ökar ett steg i taget (alltså 1-2-3-4 osv) tills en ledig plats i tabellen hittas (eller samma nyckel påträffas).

Hashfunktion 1 (hash1) är likadan som i deluppgift a):
`Nyckel % SIZE`

Hashfunktion 2 (hash2) ser ut såhär:
`7-(Nyckel%7);`

Ett exempel:
Lägg in 19, 27, 36, 10 i en tabell som är 13 element stor

hash1(19) = 19%13 = 6
hash1(27) = 27%13 = 1
hash1(36) = 36%13 = 10
hash1(10) = 10%13 = 10 KROCK – gör dubbel hashning
hash2(10) = 7-(10%7) = 7-3 = 4
(hash1(10) + 1*hash2(10))%13 → (10 + 1*4)%13 → 14%13 = 1 KROCK, öka i
(hash1(10) + 2*hash2(10))%13 → (10 + 2*4)%13 → 18%13 = 5

Lägg nu in samma värden som i a) (förutom 33-clementin) i tabellen (med storlek 10, tabellen är från början tom) med hjälp av ovan beskrivna dubbla hashning, använd de båda hashfunktionerna beskrivna ovan. Visa hur tabellen ser ut efter insättningen, visa också beräkningarna av index. Det räcker i den här deluppgiften att du visar nycklarna (själva värdet kan utelämna).

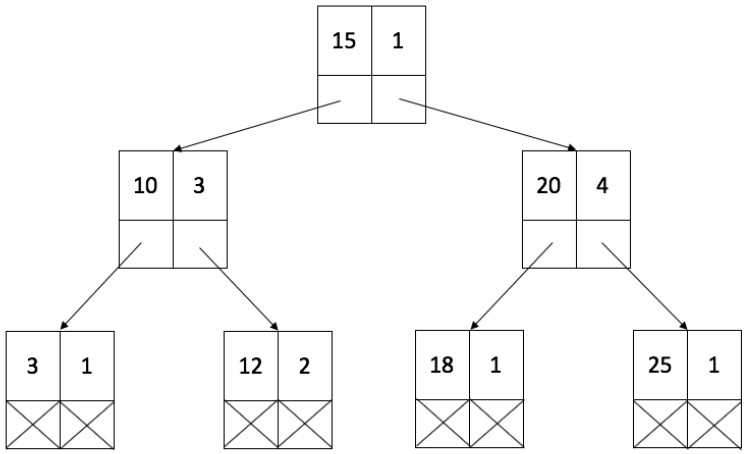
Uppgift 7 [4p]

Antag att vi har ett helt vanligt binärt sökträd som hanterar heltal men förutom själva data så innehåller varje nod också information om hur många gånger det datat existerar i trädet. Se posterna samt exemplet nedan:

```
struct treenode
{
    struct treenode* left;
    struct treenode* right;
    int data;
    int nrOfData;//anger hur många gånger det datat lagst till i trädet
};

typedef struct treenode* BSTree;
```

I följande exempel har datat 15, 3, 18 och 25 lagts till 1 gång vardera. 10 har lagts till 3 gånger, 12 har lagts till 2 gånger och 20 har lagts till 4 gånger. Insättning av dubbletter hanteras alltså genom att fältet `nrOfData` ökas med 1.



Du ska skriva den rekursiva funktion som räknar hur många data (enheter) som totalt finns i trädet (notera att detta inte nödvändigtvis är samma sak som antalet noder i trädet, i exemplet ovan finns det t.ex. 7 noder och 13 dataenheter). Du ska använda dig av följande funktionshuvud:

```
int nrOfDataInTree(BSTree tree);
```

Uppgift 8 [6p]

a) *[2p]* Visa hur följande mängd sorteras med algoritmen Merge Sort.

5	9	7	2	4	8	3

b) *[1p]* Vilken komplexitetsklass tillhör Merge Sort?

c) *[2p]* Beskriv hur valet av pivot påverkar komplexiteten på sorteringsalgoritmen Quick sort. Använd max 7 meningar, du får gärna använda bilder i din förklaring.

d) *[1p]* Vilken komplexitetsklass tillhör Quick sort i bästa/medelfall samt i värsta fall?

Uppgift 9 [3p]

Vilken komplexitetsklass tillhör följande kodsegment (med avseende på n)? Motivera dina svar med max 2 meningar vardera.

a)

```
void funkA(int n, int m)
{
        for(int i = 0; i < 1000; i++)
        {
              for(int k = n; k > 0; k--)
                   printf("%d", m);
        }
}
```

b)

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

c)

```
int funkC(int n)
{
    for(int i = 0; i < n; i++)
        funkA(n, n);
}
```

Exam – Datastructures, algorithms and program design
DVA104

School of Innovation, design and technology

Wednesday 2019-01-16

Writing time:	08.30-13.30
Aids:	Hand written notes (amount not restricted) and dictionary.
Examiner:	Caroline Uppsäll (Can be reached by telephone if you ask the exam guard)

Preliminära betygsgränser:

Betyg 3:	22p
Betyg 4:	33p
Betyg 5:	39,5p
Max:	**44p**

Generally

- All code should be written in C.
- Write clearly what task/sup-task your answers consider.
- Do only use **one** side of the paper, start new question on new paper.
- Do not refer between answers.
- If you are unsure of a meaning of a question, write down your assumption.
- *Unreadable/incomprehensible answers will not be marked.*
- Comment your code!
- It is not allowed to use goto-statements or global variables
- Hint: To know how to allocate your time, read through the entire exam before you start writing.

Good luck!
/Caroline

Question 1 [3p]

Which of the following statements is true?
An incorrect given answer together with a correct given answer gives a total of 0 points.

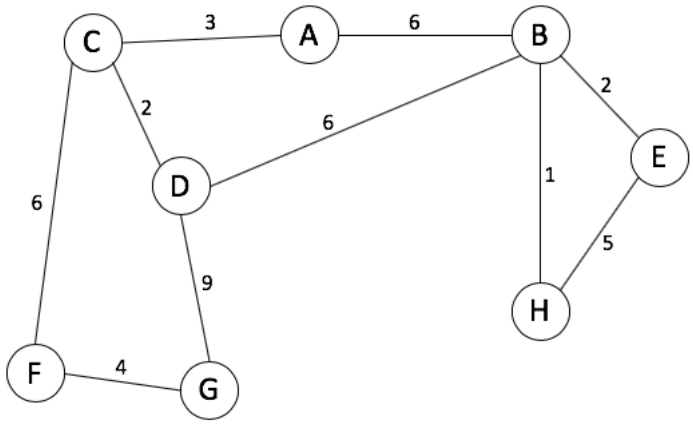
1) When you create an ADT, you start with thinking about what you want to do with the new type.
2) When you create an ADT, you start from how the data type should be implemented.
3) ADT is a concept specific to the programming language C.
4) In an ADT, you want to make as many details as possible viewable for the user.
5) An ADT consists of two different components, an interface and an implementation.
6) Every problem that can be solved with iteration can also be solved with recursion, and vice versa.
7) Many problems that can be resolved with iteration can also be solved with recursion, but not all.
8) A recursive solution generally requires less memory than an iterative solution.
9) A recursive solution generally requires more memory than an iterative solution.
10) Recursion means that a function calls another function with the same arguments/parameters.
11) Linear complexity is better than logarithmic complexity.
12) The complexity classes describes an algorithm's memory usage.
13) The complexity classes describes an the effectiveness of an algorithm in terms of time.
14) There must be an implemented solution to a problem (i.e. code) in order to determine the complexity of the solution.
15) FIFO is another name for a stack.
16) Binary search requires that the linear sequence of data is sorted.
17) Linear search can only be used on a sorted linear sequence.
18) Quick sort and Merge sort does always have the same effectiveness (regardless of what the data looks like).
19) Selection sort is natural.
20) None of the above is correct.

Question 2 [5p]

a) *[2p]* Implement an array-based circular queue, the queue should be able to save/contain 10 integers. You should declare the queue as well as any variables you will need to manage the queue. The queue should, from the beginning, contain 0:s, all variables created should be initialized with correct starting values.

b) *[3p]* Implement the function enQueue which adds a data (an integer) to the queue you created in a). If the queue is full, nothing should be added – it's up to you to decide how to handle that case.

Question 3 [12p]
In the following sub-tasks you will be working with the following graph.



a) Is the graph directed or undirected?
b) Is the graph a multigraph or not?
c) Is the graph weighted or not weighted?
d) Does the graph contain any cycles? If yes, give one example.
e) Is the graph connected or is it a disjoint graph?
f) Which is the degree of node D?

[a-f gives a total of 3p]

g) *[2p]* Draw out how an array representation of graphs edges looks like, use the matrix on the last page of the exam (if you like to you can tear of the last page of the exam, give your answer on that page and submit it along with the rest of your answers).

h) *[2p]* Explain what a shortest path tree is and give an example of when such should be useful to calculate. Answer with at maximum of 5 sentences.

i) *[3p]* Draw the minimum spanning tree starting in node A you get by following Prim's algorithm. If you like you can use the last page of the exam to give your answer on, tear it of and submit it together with the rest of your answers.

j) *[2p]* Use a queue (FIFO) to print out the graph with breath-first traversal (starting in node A). When more than one node should be added to the queue at the same time they should be added alphabetically.

Question 4 [3p]

Suppose you have a double-linked list consisting of the following node type and list type.

```
typedef struct node
{
    int data;
    struct node *next;
    struct node *prev;
}Node;

typedef struct list
{
    Node *head;
    Node *tail;
}List;
```

The list consists of a head-pointer to the first node and a tail-pointer to the last node, the last node's next and the first node's prev is set to NULL. Head and tail must always point to the first and last node of the list, if the list is empty both head and tail are set to NULL.

When a node is created, it's next and prev is always set to NULL.

Write the function that adds data at the last position in the list. Use the following function declaration:

```
int addLast(List *list, int data);
```

If it's possible to add data to the last position in the list the function should return 1, if it's not possible the function should return 0.

You can assume that there is a function:

```
static struct Node* createListNode(const int data);
```

That creates a new node with the specified data (next and prev is set to NULL) and returns the new node. If it's not possible to create the new node, the function will return NULL.

Question 5 [3p]

One and the same binary search tree has been printed with the orders Preorder (LR), Inorder (LR) and Postorder (LR). Which of the following options is printed with which traversal function? Not that tree of the options does not use any of the functions above.

a) 25, 27, 14, 13, 12, 10
b) 27, 25, 14, 13, 12, 10
c) 10, 12, 14, 13, 27, 25
d) 10, 12, 13, 14, 25, 27
e) 10, 12, 14, 13, 25, 27
f) 25, 13, 10, 12, 14, 27

Question 6 [5p]

When working with hash tables we accept that collisions will occur when different keys generate the same hash index, the question is how to resolve these collisions. One way to solve them is to use open addressing with means that you look at other indexes in the table (according to some rule) until you find the correct index. In this task you will do this in two different ways.

In both of the sub-tasks below you should start with an empty hash table with 10 slots (SIZE). The key-value pairs to be inserted in the table are the following (in order):
13 - banana
27 - kiwi
2 - cucumber
22 - melon
33 - orange
10 - tomato
16 - potato
26 - corn
19 - carrot
18 – lemon
33 – clementine (NOTE! This should only be added to sub-task a)
The number is the key and the fruit/vegetable is the value.

Don't forget to show the calculation of the hash index.

a) *[2p]* Use linear probing to put the data above in the hashtable. The hash function is:
   `Key % SIZE`
   Show what the table looks like after inserting all of the data above (both key and value). Do also show the calculation of the hash index.

b) *[3p]* Another way to solve crashes in open addressing is with the help of a so called double hashing. It works like this:

Instead of using a single hash function two different hash functions are used. The first one (hash1) is the one used primarily, if a collision occurs when using hash1, the second hash function (hash2) is called. When hash 2 has been calculated, the following formula is used to fins an index:
`(hash1 + i*hash2)%SIZE`
Where i is a counter starting at 1 and increases one step at a time (i.e. 1-2-3-4 etc) until a free space in the table is found (or the key is found).

Hash function 1 (hash1) is the same as in sub-task a):
`Key % SIZE`

Hash function 2 (hash2) looks like this:
`7-(Key%7);`

An example:
Insert the following keys: 19, 27, 36, 10 into a hash table with 13 slots (SIZE is 13)

hash1(19) = 19%13 = index 6
hash1(27) = 27%13 = index 1
hash1(36) = 36%13 = index 10
hash1(10) = 10%13 = 10 COLLISION – make double hashing
hash2(10) = 7-(10%7) = 7-3 = 4
(hash1(10) + 1*hash2(10))%13 → (10 + 1*4)%13 → 14%13 = 1 COLLISION, increase i
(hash1(10) + 2*hash2(10))%13 → (10 + 2*4)%13 → 18%13 = index 5

Your task is to insert the same key-value pairs as in sub-task a) (except 33-clementine) into a hash table (SIZE 10, empty from the beginning) resolving collisions with double hashing (as described above).
Show what the table looks like after insertion. Do also show the calculation of the hash index. In this task it's ok to only show the key (the value can be omitted.

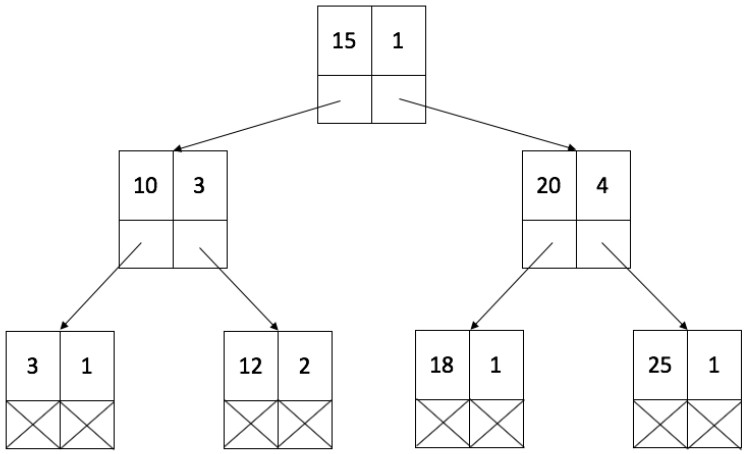
Question 7 [4p]

Suppose we have a common binary search tree that handles integers, but in addition to the data itself, each node also contains information about how many times the data exists in the tree. See the implementation and example below:

```
struct treenode
{
    struct treenode* left;
    struct treenode* right;
    int data;
    int nrOfData;//indicates how many times that data has been added to the tree.
};

typedef struct treenode* BSTree;
```

In the following example data 15, 3, 18 and 25 has been added 1 time each. Data 10 has been added 3 times, data 12 has been added 2 times and data 20 has been added 4 times. Insertion of duplicates is as can be seen handled by increasing the field `nrOfData` with 1.



Write the recursive function that counts how many data entries that are stored in the tree (note that this is not necessarily the same as the number of nodes in the tree. In the example above there are 7 nodes but 13 data entries). Use the following function declaration:

```
int nrOfDataInTree(BSTree tree);
```

Question 8 [6p]

a) *[2p]* Show how the following data is sorted using Merge sort.

5	9	7	2	4	8	3

b) *[1p]* State the complexity class for Merge Sort.

c) *[2p]* Describe how the choice of pivot affects the complexity of the sorting algorithm Quick sort. Use a maximum of 7 sentences in your answer, you may use pictures in your explanation.

d) *[1p]* Which class of complexity does Quick sort belong to at best/average and in the worst case?

Question 9 [3p]

Which complexity class does the following code segments belong to (with respect to n)? Motivate your answer with a maximum of 2 sentences each.

a)

```
void funkA(int n, int m)
{
        for(int i = 0; i < 1000; i++)
        {
              for(int k = n; k > 0; k--)
                   printf("%d", m);
        }
}
```

b)

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

c)

```
int funkC(int n)
{
    for(int i = 0; i < n; i++)
        funkA(n, n);
}
```

Anonymiseringskod/code

Uppgift 3 / Question 3
Svarspapper / Answer sheet

	A	B	C	D	E	F	G	H
A								
B								
C								
D								
E								
F								
G								
H