# TENTAMEN 20140602 DVA104 LÖSNINGSFÖRSLAG

Här visas lösningar på de uppgifter/deluppgifter som inte behandlas av labbarna eller tagits upp i lösningsförslag för andra tenter i kursen.

## UPPGIFT 1

a) För att underlätta modifiering, återanvändning, konstruktion, testning, vidareutveckling m.m. Genom att "kapsla in" implementationen kan man undvika onödiga bindningar.

e) LL: om vi vill kunna lägga till/ta bort data vart som helst i strukturen
Array: om det räcker med access längst bak, och om vi ofta vill kunna hämta data direkt på index. Kräver lite mindre minne.

g) In-place och naturlig: Bubblesort (optimerad) och Insättningssortering.

h) För att få en grov bild av algoritmens effektivitet. För att kunna jämföra olika algoritmer/lösningar på ett problem. Om det finns krav på systemet, för att kunna se till att algoritmen möter dessa.

i) FIFO = First In First Out = som en vanlig kö, det element som varit längst tid i kön plockas ut först.

j) Att pekarna till första elementet och nästa lediga plats cirkulerar

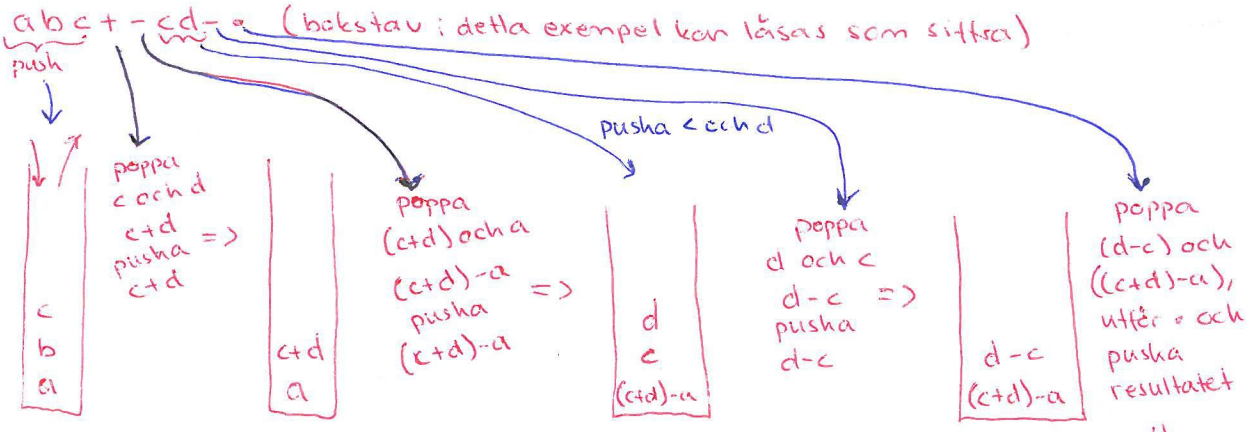

## UPPGIFT 4

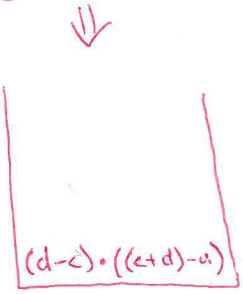
a) $O(n)$ - loopar ligger sekventiellt. Den första kör n ggr, den andra kör 2n ggr, koden kör alltså $n + 2n = 3n$ loopvarv. Konstanter försummas => $O(n)$.

b) $O(n^3)$ - loopar ligger nästlade. Den yttre kör n ggr, den inre kör $n^2$ ggr, koden kör alltså $n \cdot n^2 = n^3$ loopar. => $O(n^3)$

## UPPGIFT 5

a) siffra pushas på stacken, Vid operand, poppa två från stacken, applicera operanden och pusha resultatet till stacken.



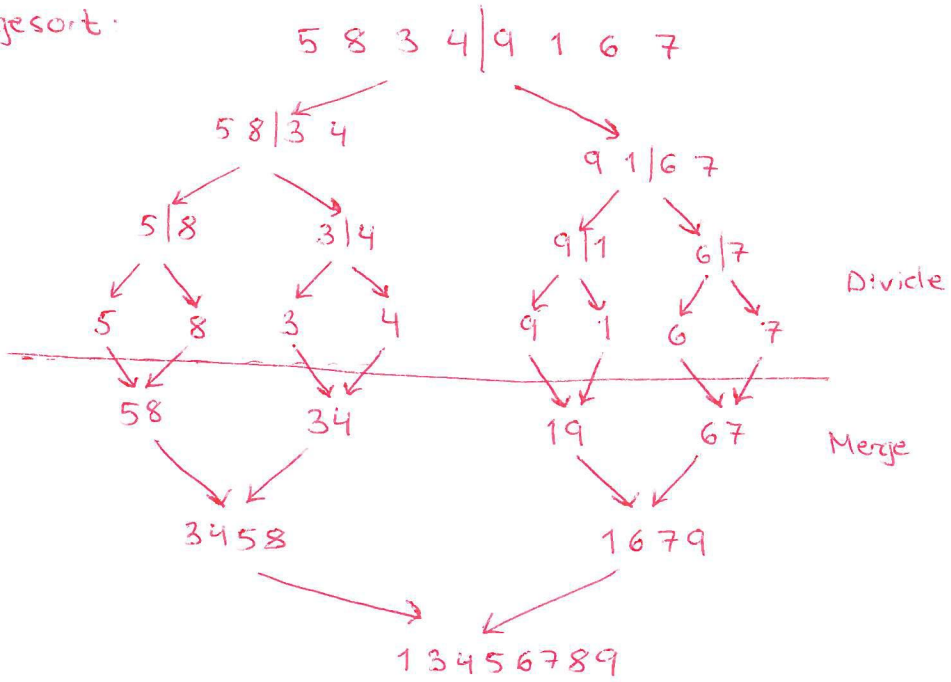




b) Vid operand om det inte finns två tal att poppa.
När uttrycket är slut ska enda resultatet ligga på stacken.

TENTAMEN 20140602 DVA104
LÖSNINGSFÖRSLAG

## UPPGIFT 6

a) Mergesort.




b) Mergesort har komplexitet $O(n \log n)$, Quicksort har i medelfall komplexiteten $O(n \log n)$ men i värsta fallet $O(n^2)$. Värsta fallet inträffar då ett pivot väljs som gör så att algoritmen degenererar.

ex: (1) 2 3 4

(2) 3 4

(3) 4

## UPPGIFT 7 (ej hela lösningen)

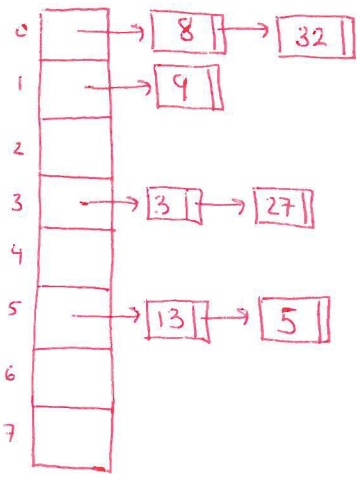
Sorteringsalgoritmer som inte använder divide & conquer är (bla) bubblesort, insertionsort och selectionsort.

TENTAMEN 20140602 DVA104
LÖSNINGSFÖRSLAG

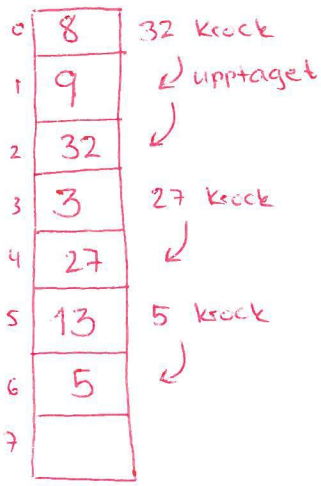
## UPPGIFT 8

| x: | 3 | 13 | 9 | 27 | 8 | 32 | 5 |
|---|---|---|---|---|---|---|---|
| x%8: | 3 | 5 | 1 | 3 | 0 | 0 | 5 |

a)

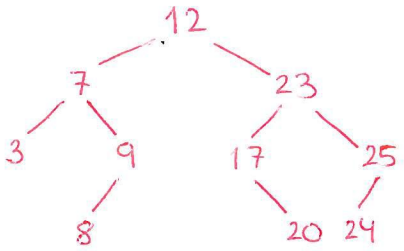



b)




c) Känner man till datamängdens storlek och det är ok att komplexiteten går mot O(n) i vissa fall (mycket krockar) bör hashtabell användas, annars binärt sökträd (balanserat).

## UPPGIFT 9




a) Binärt: Ja - varje nod har max två barn
Sökträd: Ja; mindre värden till vänster och större till höger i varje delträd
Balanserat: Ja - skillnaden i nivå/djup mellan delträd går inte att förbättra.

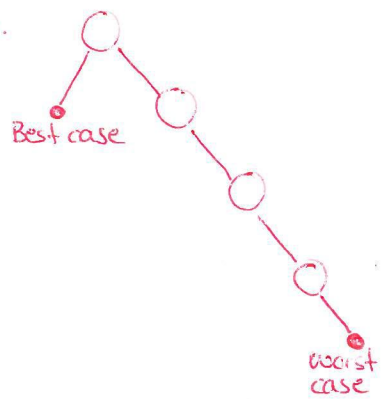
b) Preorder - hantera, vänster, höger - 12 7 3 9 8 23 17 20 25 24
Inorder - vänster, hantera, höger - 3 7 8 9 12 17 20 23 24 25
Postorder - vänster, höger, hantera - 3 8 9 7 20 17 24 25 23 12

c)

| Insättning | Balanserat | Obalanserat |
|---|---|---|
| Best case | O(log n) | O(1) → degenererat träd, högst upp i andra delträdet |
| Worst case | O(log n) | O(n) → degenererat träd, längst ner |

vid sökning efter rätt plats halveras hela tiden sökmängden

ex.



TENTAMEN 20140602 DUA104
LÖSNINGSFÖRSLAG

## UPPGIFT 9 FORTS.

e) Ta bort löv

Ta bort nod med ett barn - länka rätt av förälderns pekare till barnet så att sorterat upprätthålls.

Ta bort nod med två barn - upprätthålla sortering och inte tappa delträd. Kan göras genom att hitta största i vänster delträd eller minsta i höger delträd, kopiera detta till noden som ska tas bort och ta bort den kopierade noden som har ett eller inget barn.

f) Balansering kan göras genom rotation (AVL-träd). Om delträds djup skiljer med mer än ett efter insättning / borttagning roteras delträdet för att upprätthålla balansering.

Man kan också göra enligt följande algoritm:

- Skriv trädet sorterat till en array
- töm trädet
- bygg trädet rekursivt från arrayen
  - mittersta elementet bildar root
  - vänster delarray bygger vänster delträd
  - höger delarray bygger höger delträd