GENOMGÅNG AV GAMLA TENTOR

5DV149, 5DV150

Ola Ringdahl, Niclas Börlin, Nina Khairova

UMEÅ UNIVERSITET

TENTAN NÄSTA VECKA

- Skrivtid 8.00-12.00
- Plats: Östra paviljongen
- **Endast de som är anmälda kan skriva!**
- Du kan se på studentwebben om du är anmäld
  - Maila studexp@cs.umu.se omgående om du inte är anmäld men vill skriva tentan. De kanske kan hitta en extra plats åt dig.

INSPERA

- Digital salstentamen i Inspera, du **använder din egen dator** istället för papper och penna (men det är bra att ha ändå! ☺)
- För att det ska funka måste du
  - **Installera en säker webbläsare** på din dator innan tentan (finns för Mac och Win, men *inte* Linux)
  - Se till att **Eduroam** funkar på din dator (det är enda nätverket som finns i ÖP)
  - Gör en **demotenta** i Inspera för att kolla så allt funkar på din dator
- Om det inte går installera den säkra webbläsaren på din dator så finns det ett fåtal datorer att låna. Maila studexp@cs.umu.se om du behöver låna en.
- På den här sidan kan ni läsa allt om hur det går till att skriva digital tenta: https://www.umu.se/student/mina-studier/tentamen/digital-salstentamen
- Frågorna kommer vara i form av textsvar eller flervalsfrågor. Troligen lik fjolårets tentor.
- Om det behövs miniräkare kommer det finnas en i Inspera

VILKA FRÅGOR KAN KOMMA?

- Kursens lärmål
  - förklara grundläggande begrepp relaterade till datastrukturer och algoritmer,
  - beskriva vanliga abstrakta datatyper och algoritmer,
  - formulera lösningar till enkla problem i form av algoritmer inklusive att identifiera och använda lämpliga datatyper,
  - använda sig av grundläggande problemlösningsstrategier,
  - (experimentellt och) teoretiskt undersöka en mjukvaras tids- och minneskomplexitet, samt redogöra för och dokumentera utfallet enligt givna riktlinjer,
  - visa förståelse för hur struktur-, tids- och minnesaspekter påverkar kvalitet hos program
- Se Begrepp att kunna inför tentan på Canvas

GENOMGÅNG AV GAMLA TENTAFRÅGOR

- Notera!
  - Felaktigheter kan finnas.
  - Detta är endast lösningsskisser. Inte fullständiga svar.
  - Det finns andra korrekta svar också!
- Vi har summerat alla svar som kommit in i quizen och diskuterar de högst prioriterade frågorna i tur och ordning.

HUFFMAN

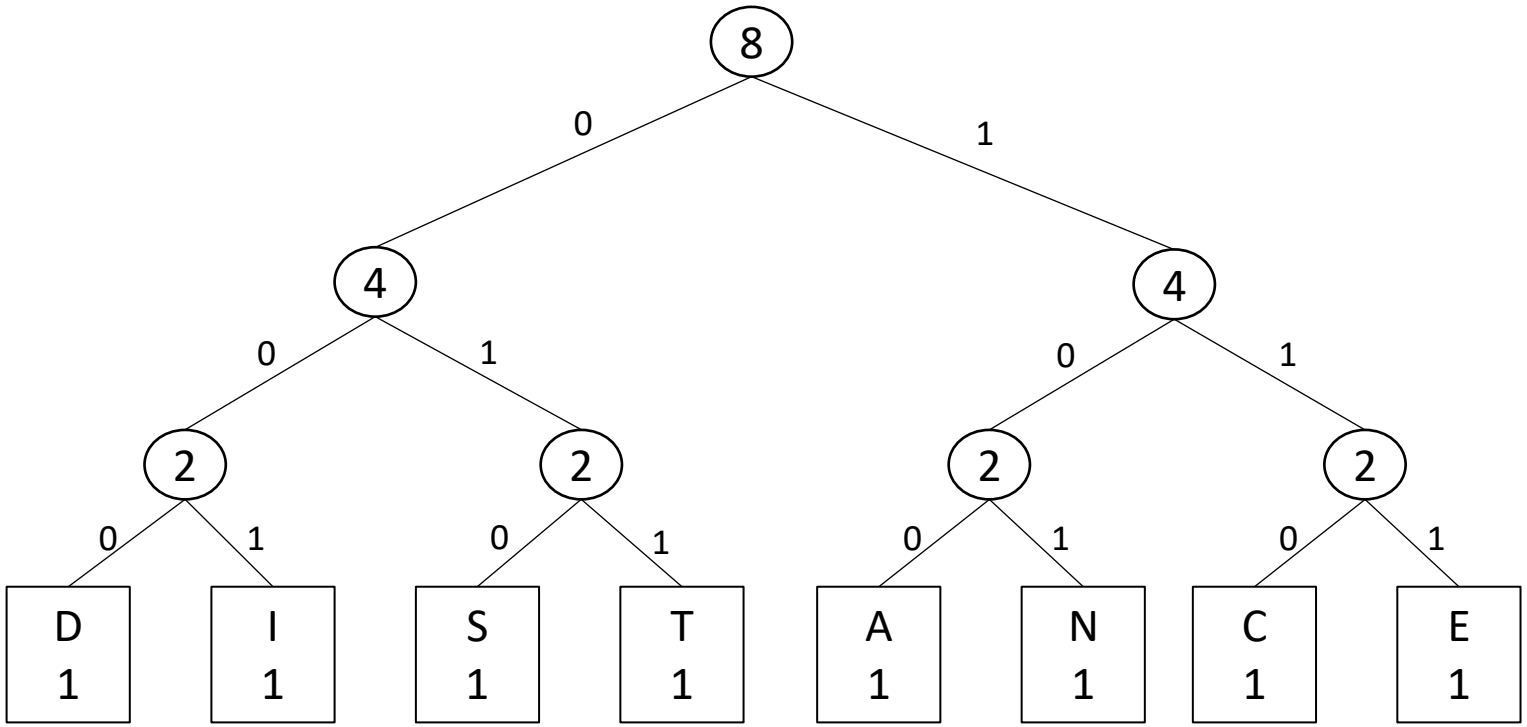
2022-04-20, Fråga 2

UMEÅ UNIVERSITET

Givet ett Huffmanträd för strängen DISTANCE:

- Hur lång blir den längsta sekvensen bitar som behövs för att koda ett tecken?
- Hur lång blir den kortaste sekvensen bitar som behövs för att koda ett tecken?

Både kortaste och längsta blir 3 bitar som vi kan se i trädet nedan



Uppgiten:

Sträng: “DUBBAR ÄR BRA ATT HA” (bortse från mellanslagen)

- Sortera löven i storleksordning, med största värdet till höger och minsta till vänster.
- Ordningen på två noder som slås ihop bestäms i första hand av deras vikt och i andra hand av bokstavsordningen (a först, ö sist).
    - Om det finns fler än två val med samma vikt när du ska slå samman noder skall du använda de två som har lägst höjd.
- Etiketten är 0 på vänster-grenar och 1 på höger-grenar.

Notera att den här uppgiften blir lite klurigare pga reglerna kring bokstavsordning. Det var ett försök att kunna automaträtta utan att behöva rita trädet.

Generellt är det vikt i första hand och höjd i andra hand (vid lika vikt) som gäller

Börja med att sortera löven i bokstavsordning och vikt:

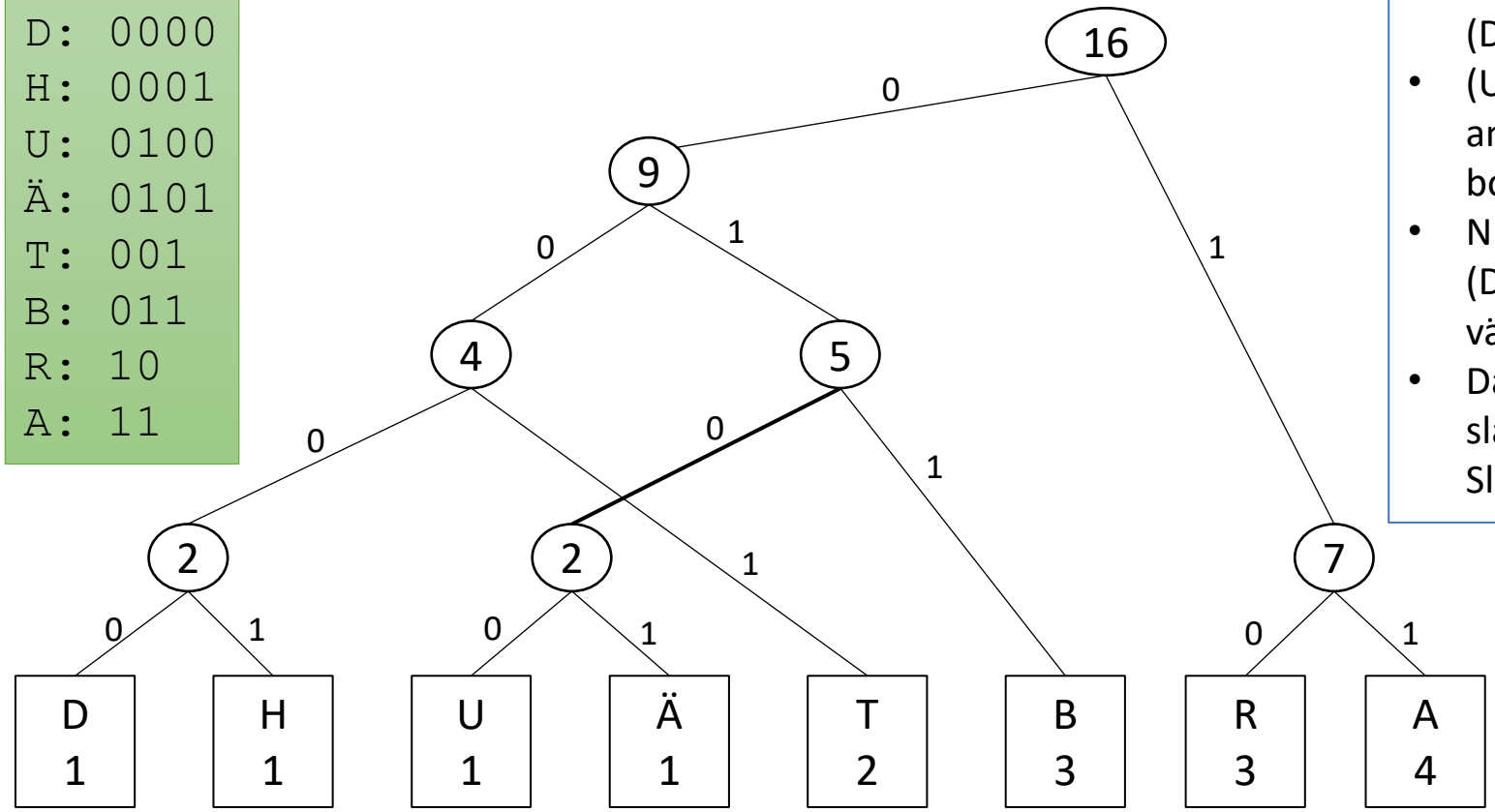
D	H	U	Ä	T	B	R	A
1	1	1	1	2	3	3	4

Uppgiten:

Sträng: “DUBBAR ÄR BRA ATT HA” (bortse från mellanslagen)

- Sortera löven i storleksordning, med största värdet till höger och minsta till vänster.
- Ordningen på två noder som slås ihop bestäms i första hand av deras vikt och i andra hand av bokstavsordningen (a först, ö sist).
  - Om det finns fler än två val med samma vikt när du ska slå samman noder skall du använda de två som har lägst höjd.
- Etiketten är 0 på vänster-grenar och 1 på höger-grenar.

```
D: 0000
H: 0001
U: 0100
Ä: 0101
T: 001
B: 011
R: 10
A: 11
```



Lösningsförslag

- Vi börjar med att slå ihop noderna med lägst vikt (Ä,U,H,D).
  - Vilken ska man ta först? Enligt uppgiften ska a slås ihop först och ö sist.
- Alltså ska vi slå ihop (D,H) först eftersom de kommer först i bokstavsordning. Därefter ska (U,Ä) slås ihop.
- Nu har vi tre st 2:or. Då ska vi slå ihop den med minst höjd (T) med den först i bokstavsordning (D,H).
- (U,Ä) har minst vikt nu (2). Den ska slås ihop med antingen R eller B som båda har 3. B är före R i bokstavsordning och därför slår vi ihop med den.
- Nu tar vi R och slår ihop med antingen A eller (T, (D,H)) som båda har 4. A har lägst höjd så vi ska välja den.
- Därefter är det olika vikt på alla så det är bara att slå ihop de lägsta hela tiden (4+5=9, 9+7=16). Slutresultatet kan ses i trädet till vänster

SORTERING

UMEÅ UNIVERSITET

SORTING ALGORITHMS

- Sorting ***changes the order between the objects*** in a structure according to a sort order (descending, ascending).
- Major characteristics of the algorithms are:
  - **Stability**

    Stable sort algorithms sort equal elements in the same order that they appear in the input.
  - **Computational complexity**

    Computation time (generally measured by the number of needed elementary operations)
- The input data should be stored in a data structure that allows random access

SORTING ALGORITMS

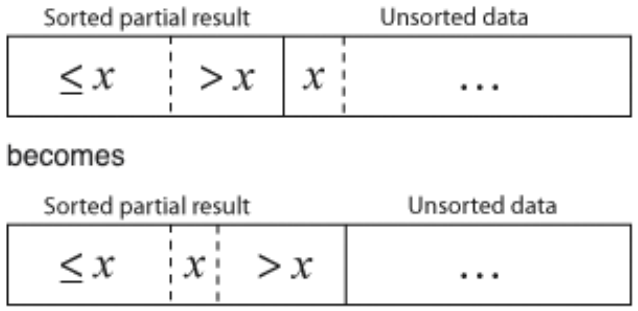
- **Selection** sort – *Urvals*
- **Insertion** sort – *Insticks*
- **Merge** sort – *Samsortering*
- **Exchange** sort – *Utbytes*
- **Bubble** sort – *Bubbelsortering*
- **Quicksort** – *Quicksort*
- etc.

PSEUDOCODE (BUBBLE SORT)

```
Algorithm bubbleSort (arr)
repeat
        swapped <-- false
        for j <-- low (arr) to high(arr) -1  do
                if arr [ j ] > arr [ j + 1 ] then
                        temp <-- arr [ j ]
                        arr [ j ] <-- arr [ j + 1 ]
                        arr [ j + 1 ] < -- temp
                        swapped <-- true
until not swapped
return arr
```

PSEUDOCODE (INSERTION SORT)

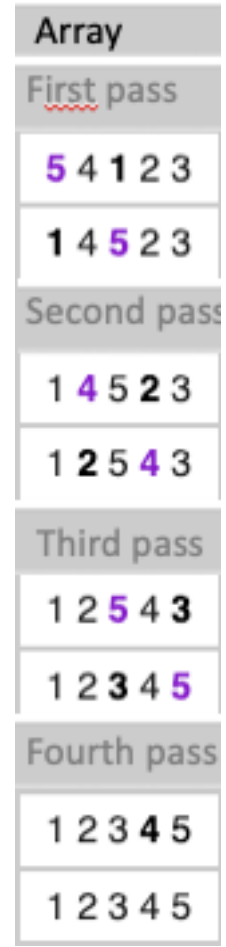
```
Algorithm insertionSort (arr):
   for j <-- 2 to length [arr]
            do key <-- art[j]
            i <-- j – 1
            while i > 0 and arr[i] > key
                  do   arr [i+1] <-- arr [i]
                        i <-- i-1
             arr [i+1] <-- key
```



Before	After
First pass	
5 2 4 3 1	**2 5** 4 3 1
Second pass	
2 **5 4** 3 1	2 **4 5** 3 1
2 4 5 3 1	**2 4** 5 3 1
Third pass	
2 4 **5 3** 1	2 4 **3 5** 1
2 **4 3** 5 1	2 **3 4** 5 1
2 3 4 5 1	**2 3** 4 5 1
Fourth pass	
2 3 4 **5 1**	2 3 4 **1 5**
2 3 **4 1** 5	2 3 **1 4** 5
2 **3 1** 4 5	2 **1 3** 4 5
2 1 3 4 5	**1 2** 3 4 5

PSEUDOCODE (SELECTION SORT)

```
Algorithm selectionSort (arr):
    for i <-- 0 to n-2
            min <--  i
            for j <--  i+1 to n-1
                    if arr [j] < arr [min]
                            min <--  j
            swap (arr [i], arr [min])
```



PSEUDOCODE (QUICKSORT SORT)

```
Algorithm quickSort  ( arr, l, r ):
        if  r - l  <--  1:
                return
        m  <--  partition  (arr, l ,r)
        quickSort  (arr, l, m)
        quicksort  (arr, m+1,  r)
```

```
Algorithm partition (arr, l , r ):
        pivot <-- arr[random(l ... r - 1)]
        i <-- l
        j <-- r
        while i ≤ j:
                while arr[i] < pivot
                        i ++
                while arr [j] > pivot
                        j --
                if i ≥ j
                        break
                swap(arr[i++], arr[j--])
        return j
```

PSEUDOCODE (QUICKSORT SORT)

- QuickSort like MergeSort is a **Divide and Conquer algorithm**.

- Complexity depends on the balance of the subarrays splitting:
  - The worst splitting is 1 to (N-1) => O(n2)
  - The best splitting is N/2 to N/2 => O (n*log n)
  - The average case => O (n*log n)

- **partition** function splits the array [l, r-1] according to the **pivot** element

LÖSNINGSFÖRSLAG PÅ ÄLDRE TENTOR

Slides från föregående år

UMEÅ UNIVERSITY

2013-03-26
UPPGIFT 8
GRAFER

UMEÅ UNIVERSITET

2013-03-26 UPPGIFT 8 GRAFER

- Personerna A, B, C, D, E och F bor tillsammans i ett hus. De vill koppla samman sina datorer, men använda så lite kabel som möjligt. Minimala kabelavståndet mellan två datorer ges av följande grannmatris.
- Rita en bild som visar hur A, B, C, D, E och F bör koppla samman sina datorer. Förklara också hur algoritmen du använt fungerar genom att rita bilder som visar hur algoritmen steg för steg konstruerar kabelnätet.

	A	B	C	D	E	F
A	0	2	1	5	7	5
B	2	0	5	10	6	9
C	1	5	0	5	6	4
D	5	10	5	0	8	2
E	7	6	6	8	0	4
F	5	9	4	2	4	0

VILKA GRAFALGORITMER KAN VI?

- Bredden först, djupet först
  - Traversering
- Floyds algoritm
  - Kortaste vägen från alla noder till alla andra noder (matrisorienterad)
- Dijkstras algoritm
  - Kortaste vägen från en nod till alla andra (graforienterad)
- Prims, Kruskals
  - Konstruera ett minsta uppspännande träd

2013-03-26 UPPGIFT 8 GRAFER

- Vi vill använda minsta möjliga kabel för att koppla ihop alla datorer -> **Minsta uppspännande träd**
- Två möjliga algoritmer:

Kruskal

- Utgå från en prioritetskö av alla bågar.
- I varje steg, plocka kortaste bågen från kön.
  - Fyra alternativ:
    - Bygg ett nytt träd.
    - Utöka ett träd.
    - Slå ihop två träd.
    - Ignorera bågen

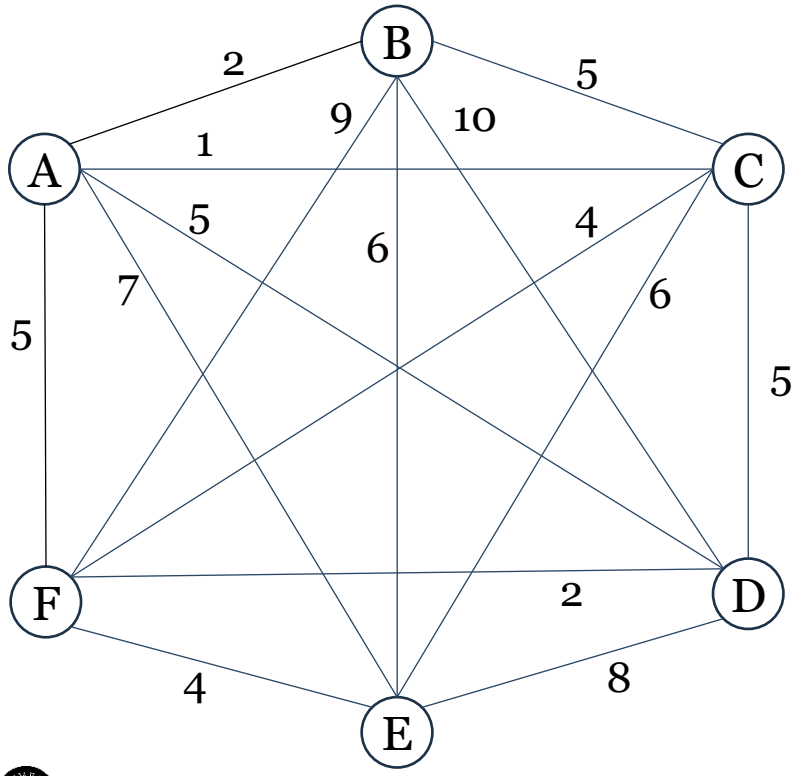
Prims

- Utgå från godtycklig startnod.
- Bygg upp ett större och större träd som till slut spänner upp grafen eller en sammanhängande komponent av den.
- I varje steg, bygg på trädet med en båge med minimal vikt.

2013-03-26 UPPGIFT 8
GRAFER

- Rita upp trädet

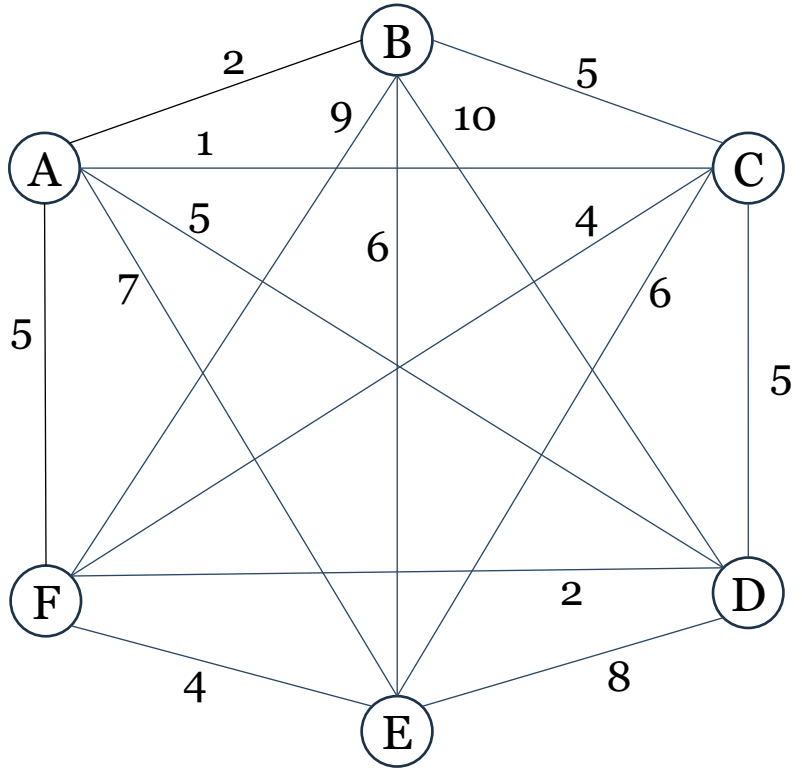
	A	B	C	D	E	F
A	0	2	1	5	7	5
B	2	0	5	10	6	9
C	1	5	0	5	6	4
D	5	10	5	0	8	2
E	7	6	6	8	0	4
F	5	9	4	2	4	0



2013-03-26 UPPGIFT 8
GRAFER
KRUSKALS ALGORITM

1. Skapa en prioritetskö *q* av alla bågar utifrån deras vikt

	A	B	C	D	E	F
A	0	2	1	5	7	5
B	2	0	5	10	6	9
C	1	5	0	5	6	4
D	5	10	5	0	8	2
E	7	6	6	8	0	4
F	5	9	4	2	4	0

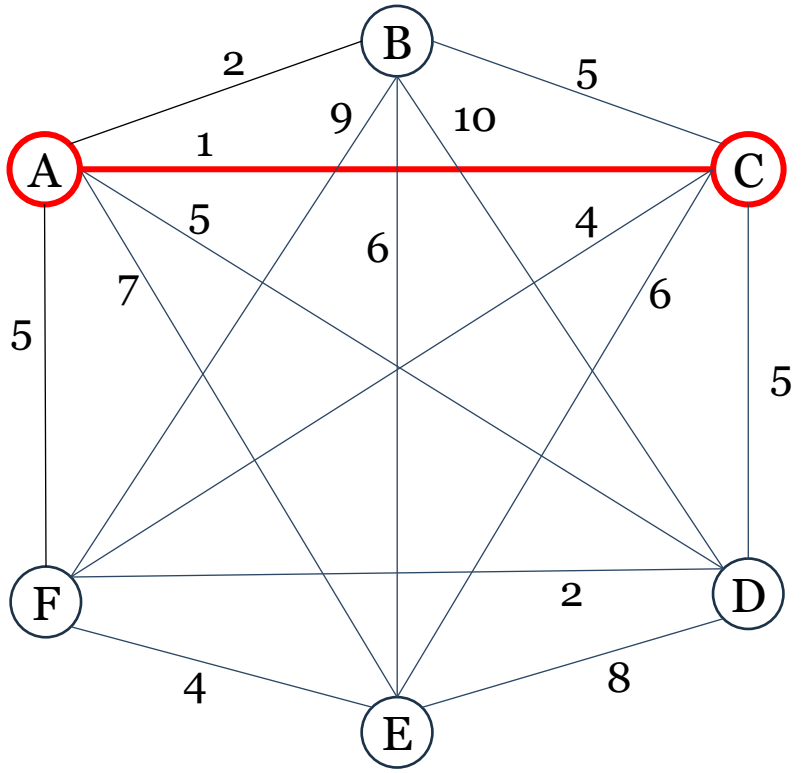


q={(A,C,1), (A,B,2), (D,F,2), (C,F,4), (E,F,4), (A,D,5), (A,F,5), (B,C,5), (C,D,5), (B,E,6), (C,E,6), (A,E,7), (D,E,8), (B,F,9), (B,D,10) }

2013-03-26 UPPGIFT 8 GRAFER

KRUSKALS ALGORITM

- Ta första bågen (A,C,1) ur kön
- Ingen av (A,C) är färgade:
  - Färglägg med ny färg.

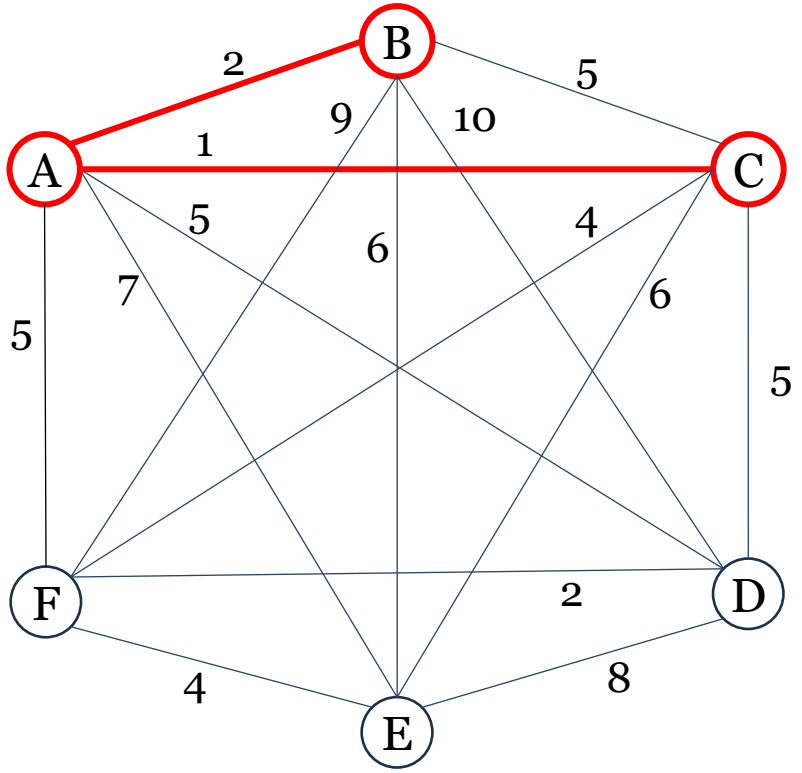


q={ (A,B,2), (D,F,2), (C,F,4), (E,F,4), (A,D,5), (A,F,5), (B,C,5), (C,D,5), (B,E,6), (C,E,6), (A,E,7), (D,E,8), (B,F,9), (B,D,10) }

2013-03-26 UPPGIFT 8 GRAFER

KRUSKALS ALGORITM

- Ta första bågen (A,B,2) ur kön
- A är färgad.
  - Färglägg med A:s färg.

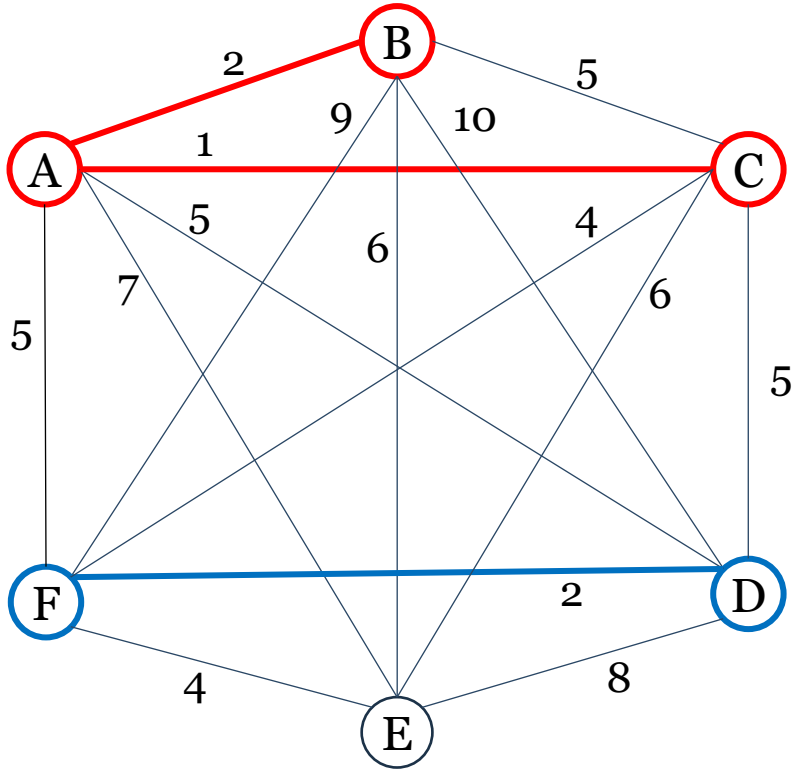


q={ (D,F,2), (C,F,4), (E,F,4), (A,D,5), (A,F,5), (B,C,5), (C,D,5), (B,E,6), (C,E,6), (A,E,7), (D,E,8), (B,F,9), (B,D,10) }

2013-03-26 UPPGIFT 8 GRAFER

KRUSKALS ALGORITM

- Ta första bågen (D,F,2) ur kön
- Ingen av (D,F) är färgade:
  - Färglägg med ny färg

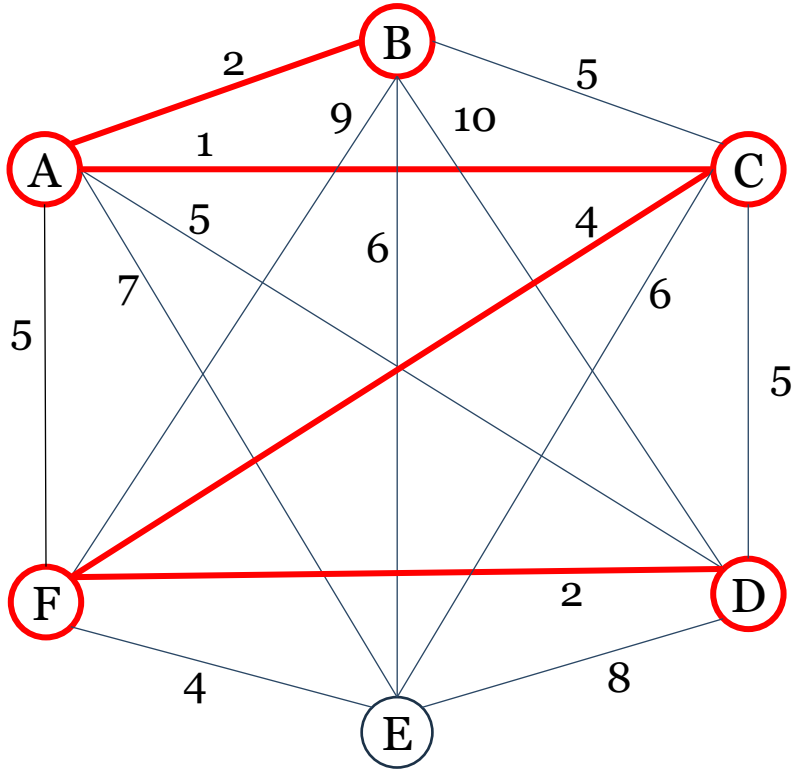


q={(C,F,4), (E,F,4), (A,D,5), (A,F,5), (B,C,5), (C,D,5), (B,E,6), (C,E,6), (A,E,7), (D,E,8), (B,F,9), (B,D,10) }

2013-03-26 UPPGIFT 8 GRAFER

KRUSKALS ALGORITM

- Ta första bågen (C,F,4) ur kön
- C och F färgade med olika färg.
  - Färglägg båda graferna med C:s färg

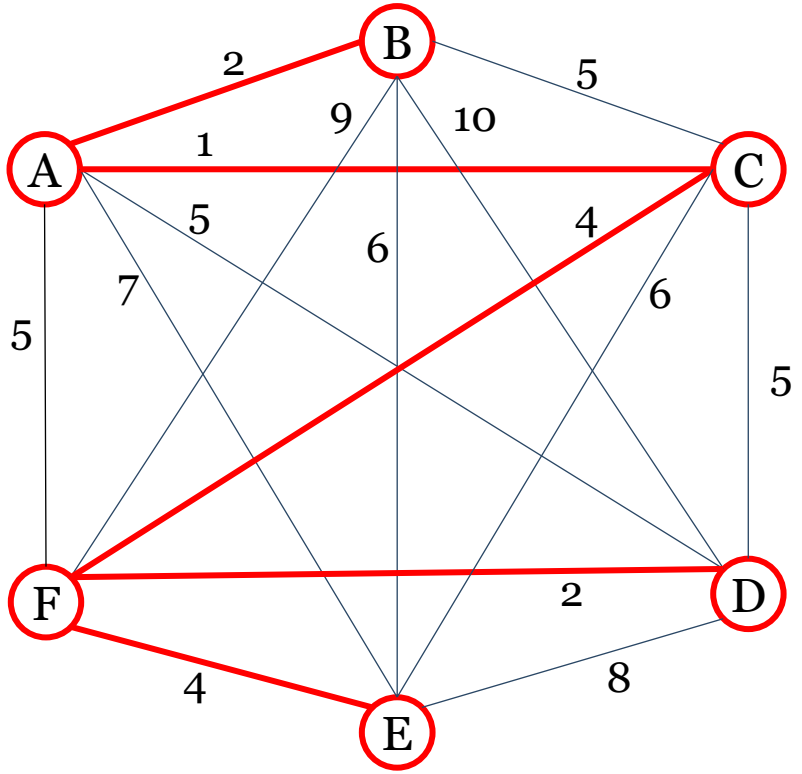


q={(E,F,4), (A,D,5), (A,F,5), (B,C,5), (C,D,5), (B,E,6), (C,E,6), (A,E,7), (D,E,8), (B,F,9), (B,D,10) }

2013-03-26 UPPGIFT 8 GRAFER

KRUSKALS ALGORITM

- Ta första bågen (E,F,4) ur kön
- F är färgad.
  - Färglägg med F:s färg.

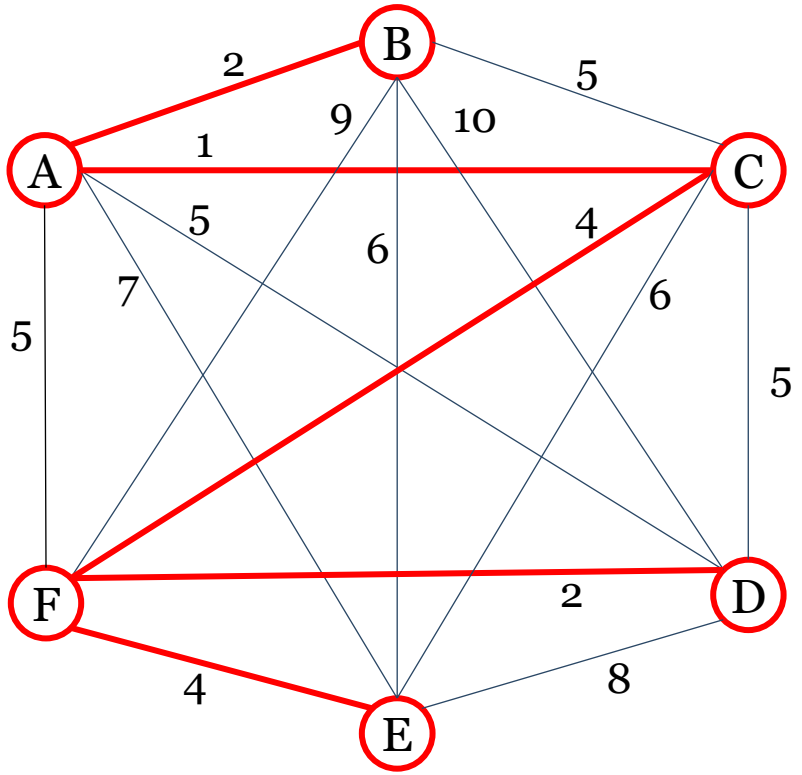


q={(A,D,5), (A,F,5), (B,C,5), (C,D,5), (B,E,6), (C,E,6), (A,E,7), (D,E,8), (B,F,9), (B,D,10) }

2013-03-26 UPPGIFT 8 GRAFER

KRUSKALS ALGORITM

- Ta första bågen (A,D,5) ur kön
- Båda färgade med samma färg.
  - Ignorera bågen.
- ….
- Alla noder är färgade nu, så resten kommmer se ut likadant (men algoritmen kommer ju köra igenom resten av kön förstås)

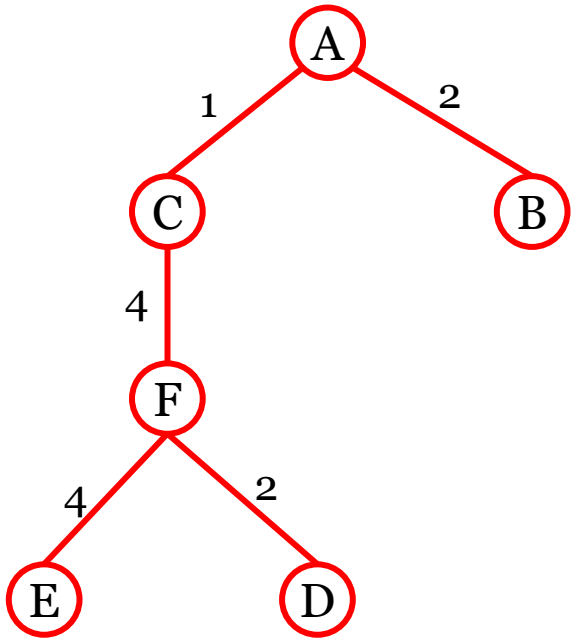


q={(A,F,5), (B,C,5), (C,D,5), (B,E,6),
(C,E,6), (A,E,7), (D,E,8), (B,F,9), (B,D,10) }

2013-03-26 UPPGIFT 8 GRAFER

KRUSKALS ALGORITM

- Resultatet: ett minsta uppspännande träd!



2020-10-23
UPPGIFT 11
HUFFMAN-KODNING

UMEÅ UNIVERSITET

2020-10-23 UPPGIFT 11
HUFFMAN-KODNING

- Du har analyserat en text och fått fram följande frekvenser:

Tecken	a	e	i	o	u	å
Antal förekomster	3	31	15	7	1	12

- Skapa ett huffmanträd där vikten på vänster grenar är 0 och höger är 1
- Sortera löven i storleksordning först, med största värdet till vänster och minsta till höger.
- Ange sedan för var och en av bokstäverna vilken kod de får.

2020-10-23 UPPGIFT 11
HUFFMAN-KODNING

- Sortera löven i storleksordning först, med största värdet till vänster och minsta till höger.

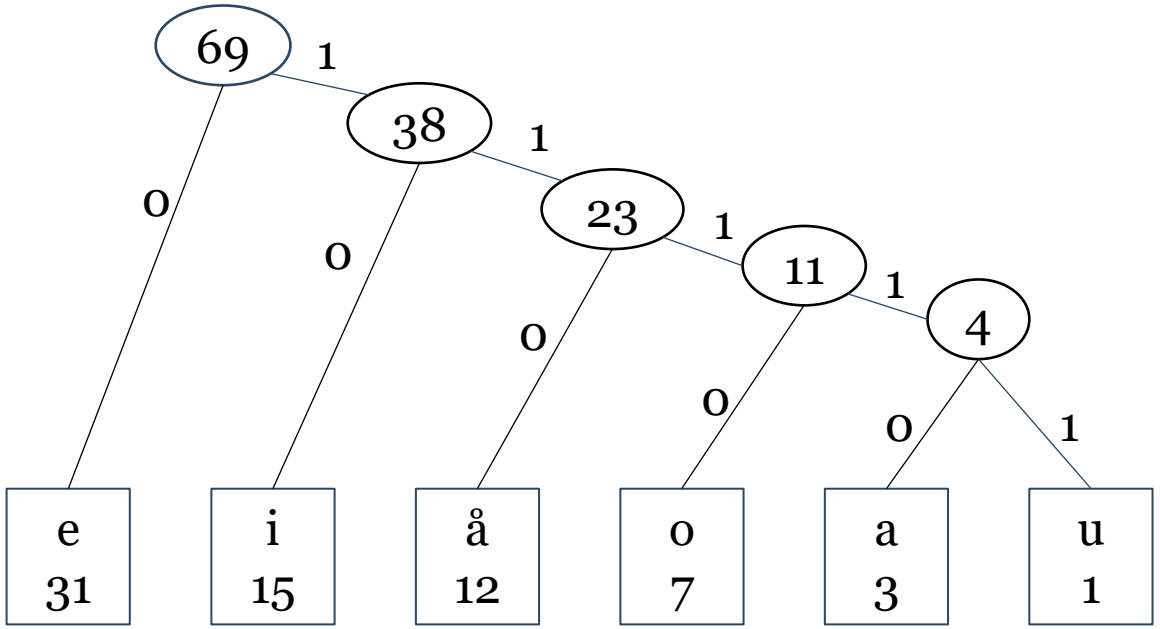
Tecken	a	e	i	o	u	å
Antal förekomster	3	31	15	7	1	12

e	i	å	o	a	u
31	15	12	7	3	1

2020-10-23 UPPGIFT 11
HUFFMAN-KODNING

- Skapa ett huffmanträd där vikten på vänster grenar är 0 och höger är 1
- Ange sedan för var och en av bokstäverna vilken kod de får

a = 1 1 1 1 0
e = 0
i = 1 0
o = 1 1 1 0
u = 1 1 1 1 1
å = 1 1 0



2018-03-08
UPPGIFT 2
ALGORITMER, KOMPLEXITET

UMEÅ UNIVERSITET

2018-03-08 UPPGIFT 2
ALGORITMER, KOMPLEXITET

Snittmängden s av två mängder s_1 och s_2 är en mängd som innehåller alla element e som ingår både i s_1 och i s_2.

a) Ge pseudokod för en algoritm som beräknar snittmängden där datatypen mängd är konstruerad som en (osorterad) riktad lista.

b) Ge pseudokod för en algoritm som beräknar snittmängden där datatypen mängd är konstruerad som en riktad lista där elementvärdena är sorterade i stigande ordning. Tänk på att s ska vara sorterad på samma sätt.

c) Förklara hur algoritmerna fungerar.

d) Vad har algoritmerna för relativ (förenklad) tidskomplexitet om du antar att mängderna innehåller ungefär lika många element n?

2018-03-08 UPPGIFT 2
ALGORITMER, KOMPLEXITET

a) Osorterad lista

```
s <- empty()              //output
p1 <- first(s1)           // p1 traverses s1
while not isend(p1, s1) do
   v1 <- inspect(p1, s1)  // v1 current value in s1
   found <- False         // True if same value found in s1
   p2 <- first(s2)        // p2 traverses s2
   while not isend(p2, s2) and not found do
       v2 <- inspect(p2, s2) // v2 current value in s2
       if v1 = v2 then
          found = True
       end
       p2 <- next(p2, s2) // advance in s2
   end
   if found then

        // s unsorted so insert first is ok
       s <- insert(v1, first(s), s)
   p1 <- next(p1, s1) // advance in s1
end
return s
```

2018-03-08 UPPGIFT 2
ALGORITMER, KOMPLEXITET

b) Sorterad lista

```
s <- empty()                  //output
p1 <- first(s1)               // p1 traverses s1
p2 <- first(s2)                 // p2 traverses s2
p <- first(s) // p where to insert new element in s (at end)
while not (isend(p1, s1) or isend(p2, s2))do
   v1 <- inspect(p1, s1)  // read current values
   v2 <- inspect(p2, s2)
   if v1 = v2 then
       (s, p) <- insert(v1, p, s) // insert last (at p) in s
       p <- next(p, s)  // make sure p is at end of s
       p1 <- next(p1, s1) // advance both
       p2 <- next(p2, s2)
   else if v1 < v2 then
       // value in s1 smaller, move forward in s1
       p1 <- next(p1, s1)
   else // value in s2 smaller, move forward in s2
       p2 <- next(p2, s2)

   end
end
return s
```

UMEÅ UNIVERSITET

2018-03-08 UPPGIFT 2
ALGORITMER, KOMPLEXITET

c) Förklaring

- osorterad: Loopa igenom s1. För varje element, plocka ut elementvärdet och loopa igenom s2. Om samma elementvärde finns i s2, stoppa in i s.
- sorterad: Loppa parallellt igenom s1 och s2. Om värdena på förstaelementen i bägge listorna är samma, stoppa in värdet i s. Avancera bägge positionerna. Om värdena olika, avancera positionen för det minsta värdet.

d) Komplexitet

- osorterad $O(n^2)$
- sorterad $O(n)$

2020-10-23
UPPGIFT 2
HEAP

UMEÅ UNIVERSITET

2020-10-23 UPPGIFT 2
HEAP

Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

2020-10-23 UPPGIFT 2
HEAP

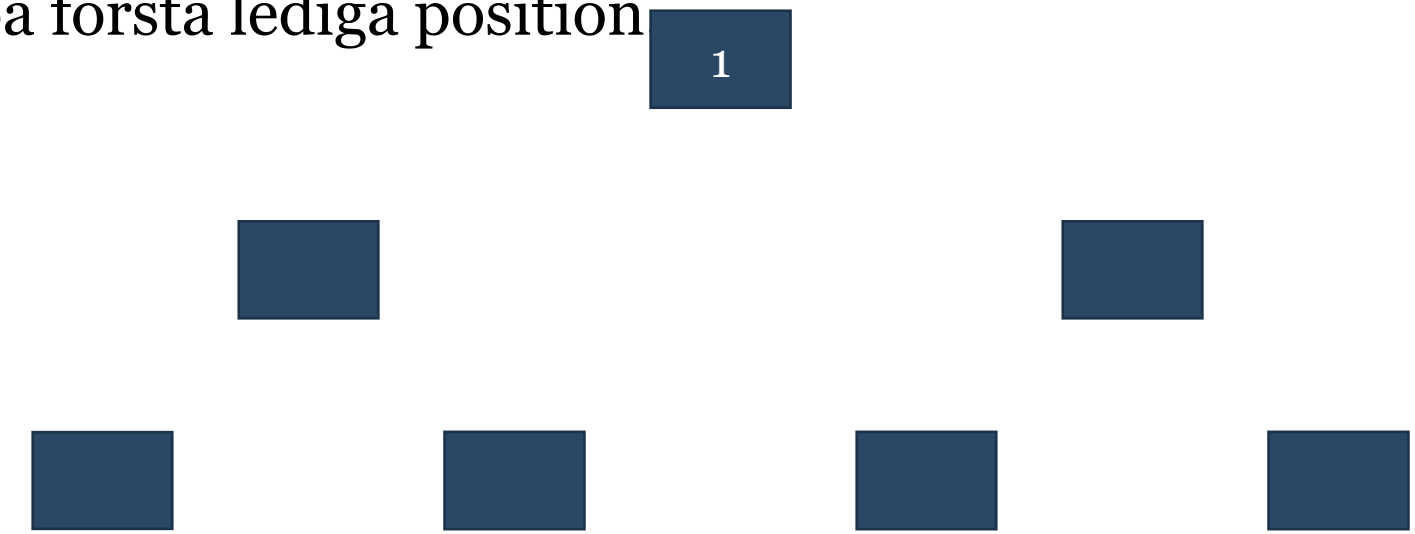
Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.

Börjar med en tom heap.

2020-10-23 UPPGIFT 2
HEAP

Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.

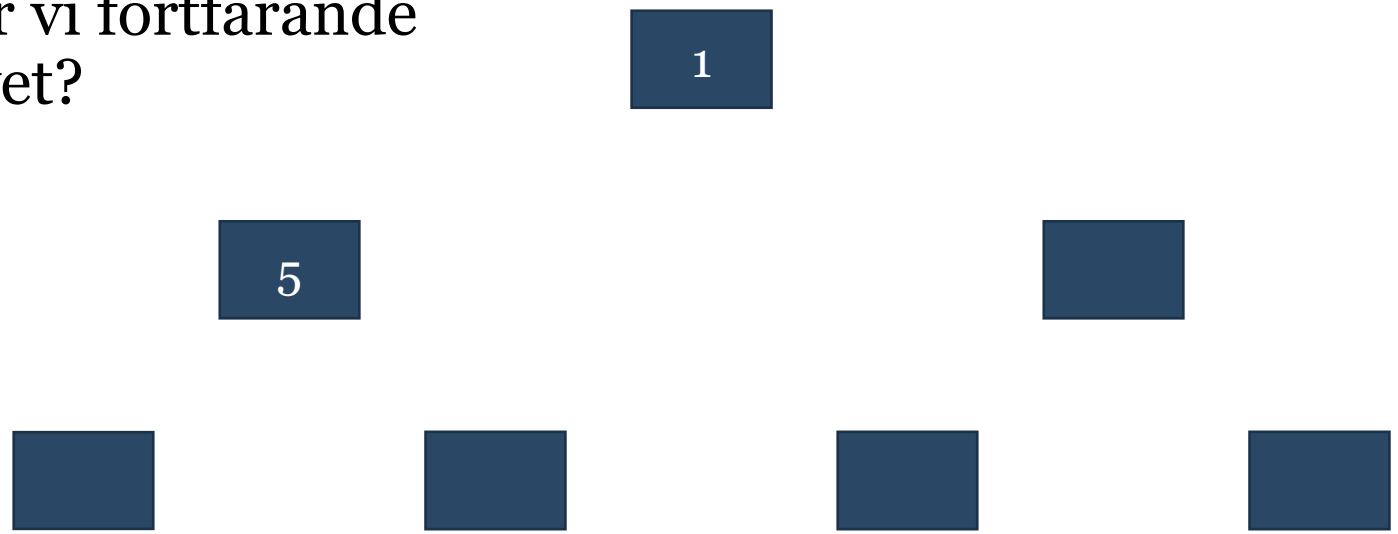
Sätt in på första lediga position



2020-10-23 UPPGIFT 2
HEAP

Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.

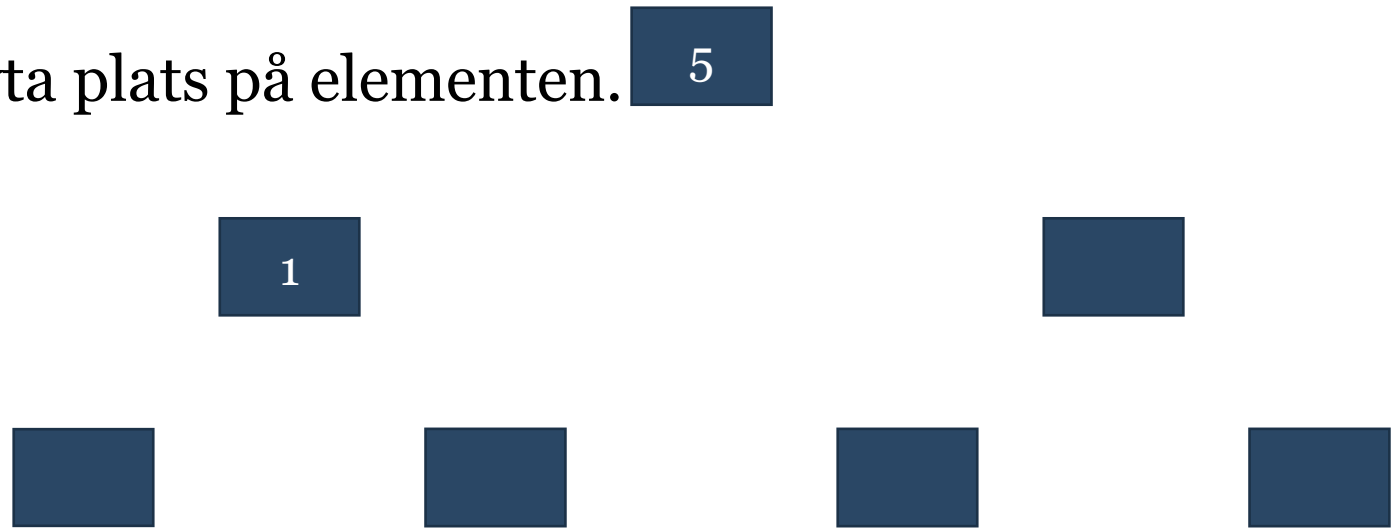
Uppfyller vi fortfarande hög-kravet?



2020-10-23 UPPGIFT 2
HEAP

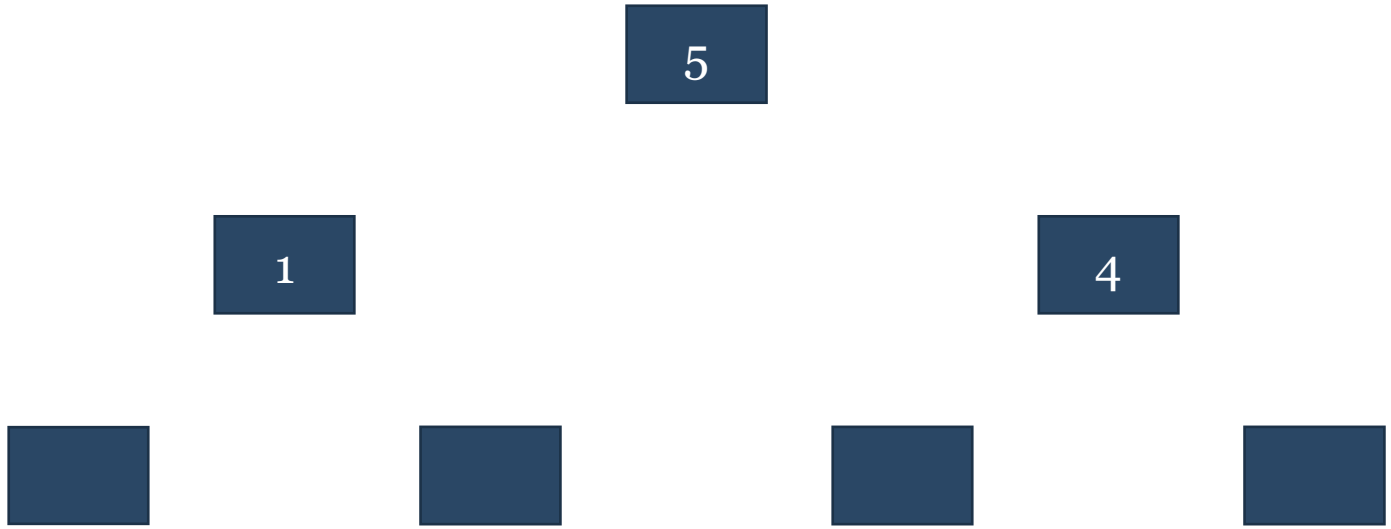
Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.

Nej!
Måste byta plats på elementen.



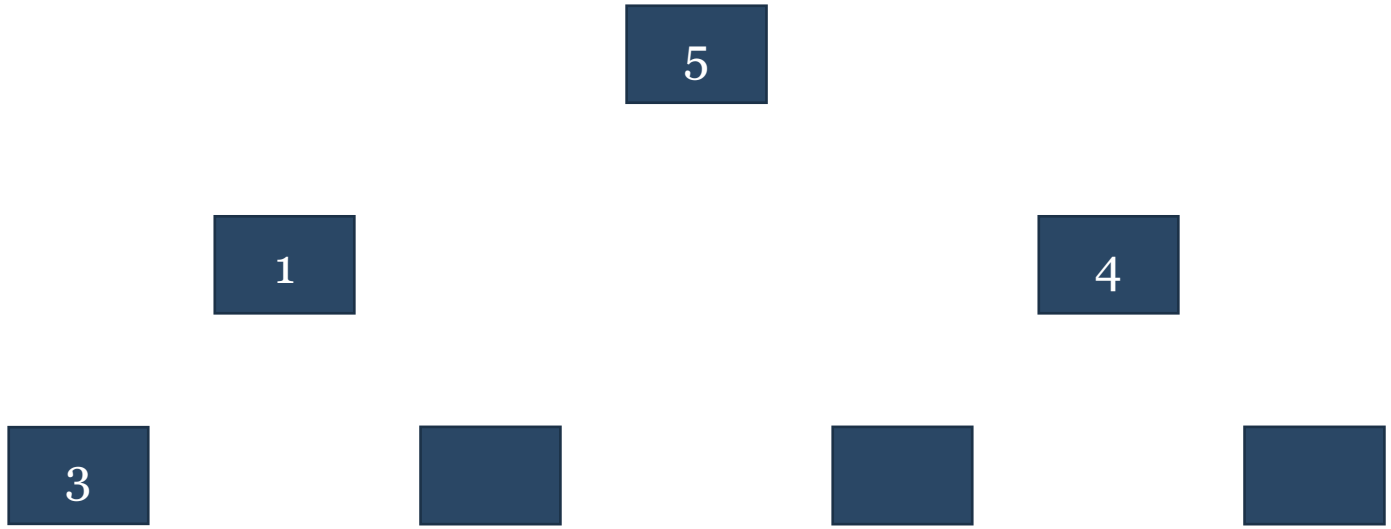
2020-10-23 UPPGIFT 2
HEAP

Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.



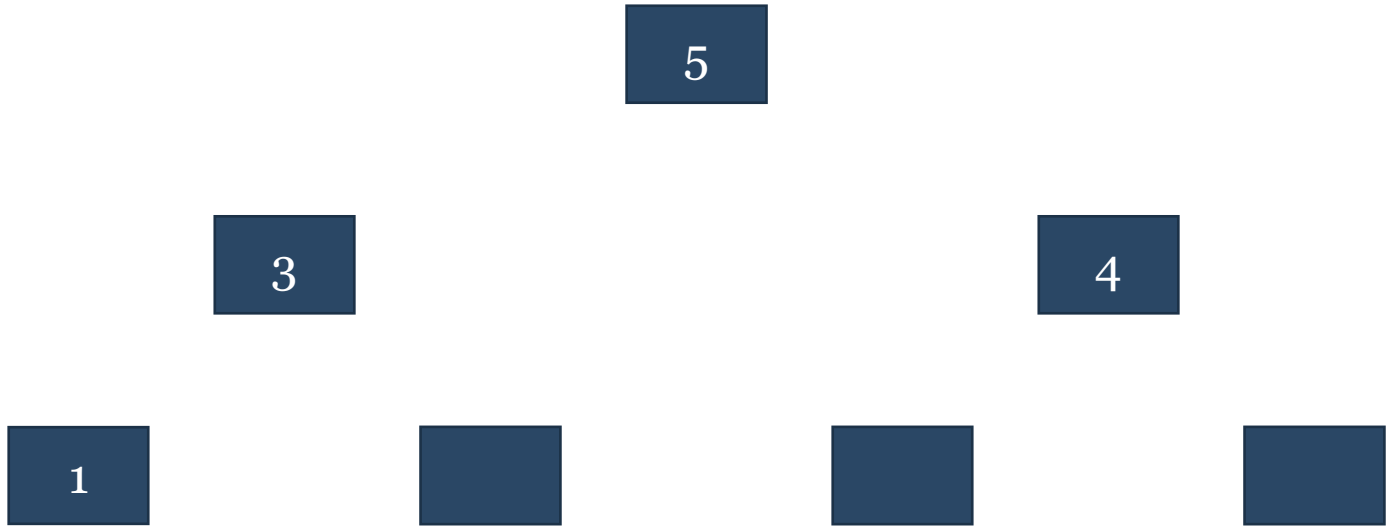
2020-10-23 UPPGIFT 2
HEAP

Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.



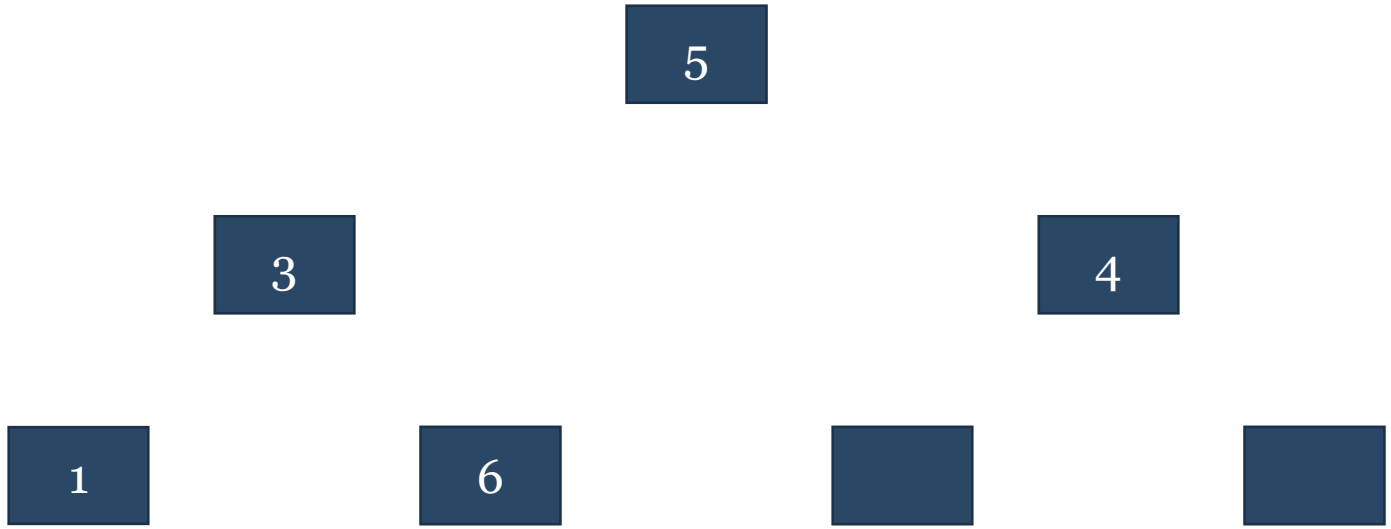
2020-10-23 UPPGIFT 2
HEAP

Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.



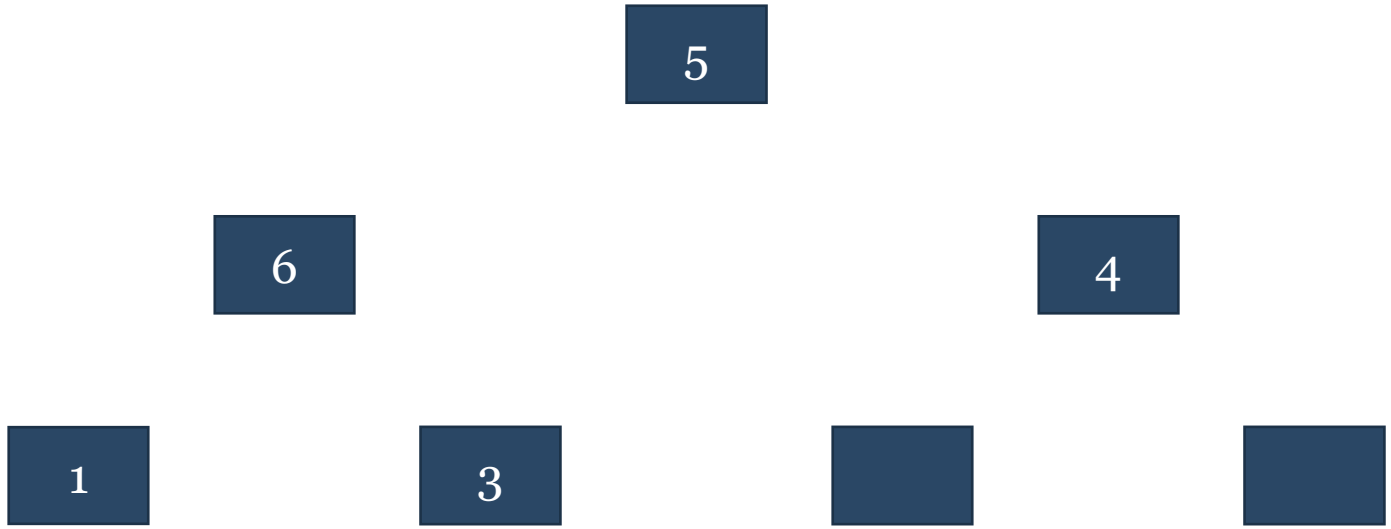
2020-10-23 UPPGIFT 2
HEAP

Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.



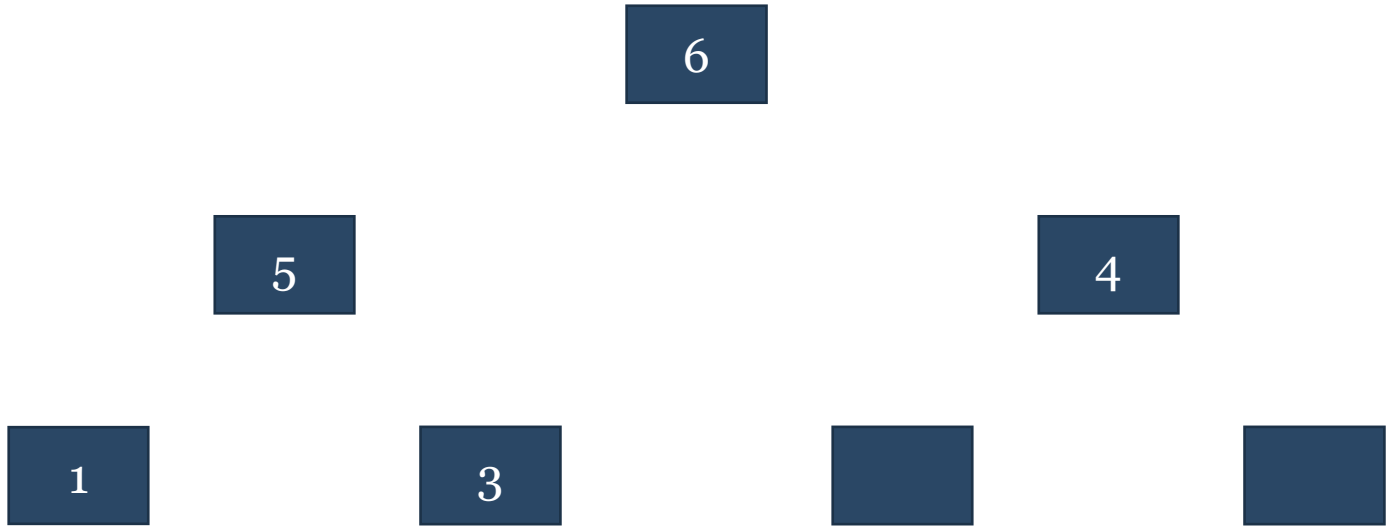
2020-10-23 UPPGIFT 2
HEAP

Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.



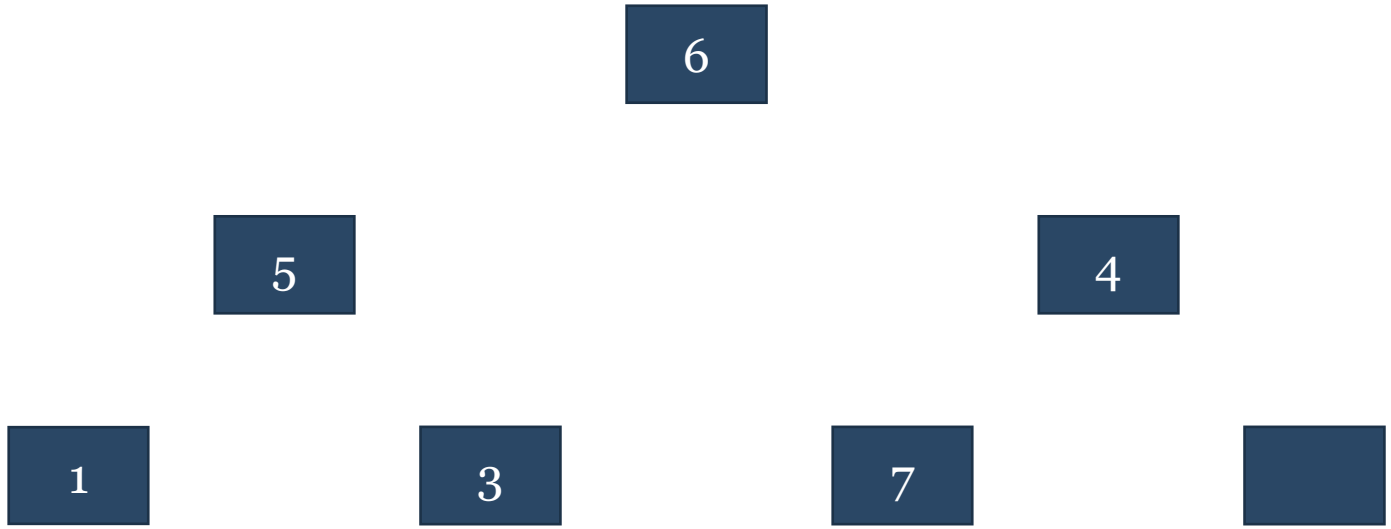
2020-10-23 UPPGIFT 2
HEAP

Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.



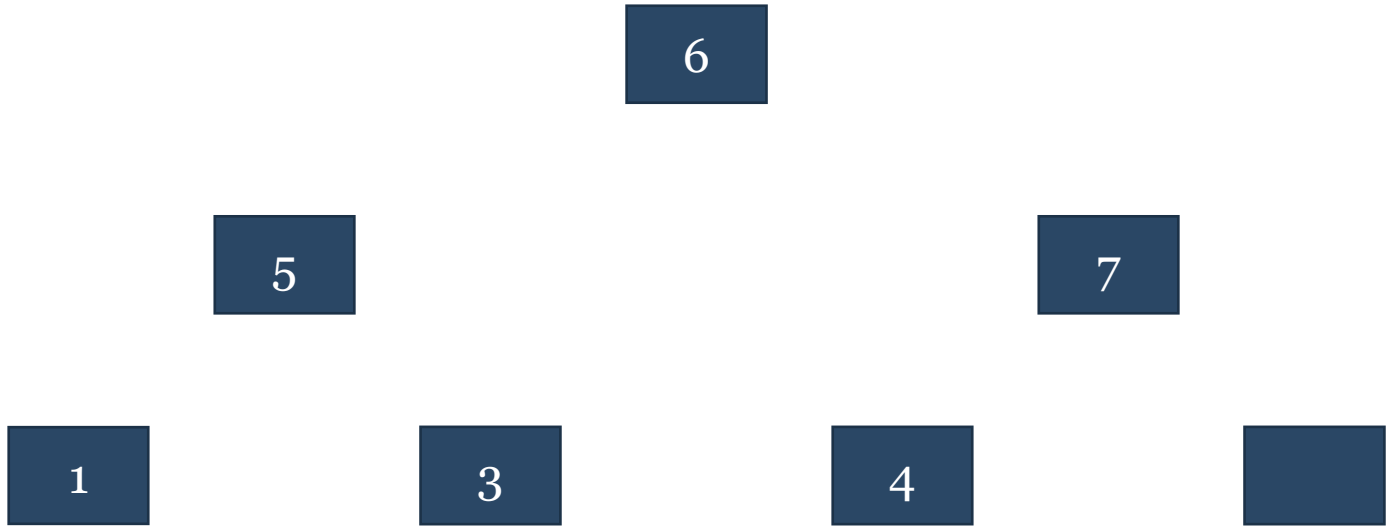
2020-10-23 UPPGIFT 2
HEAP

Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.



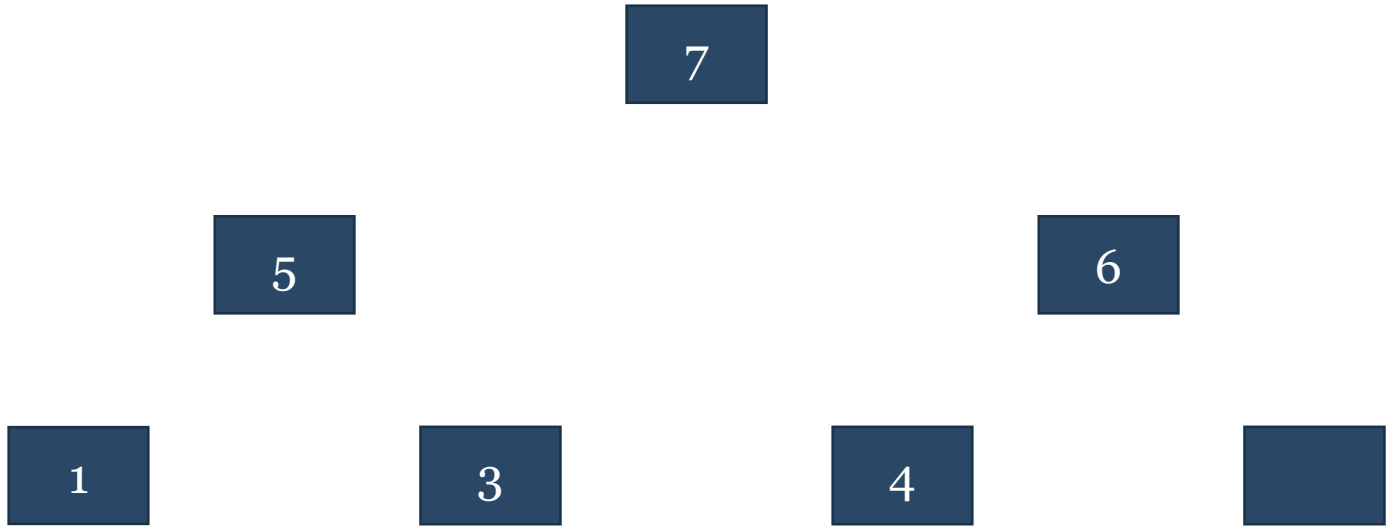
2020-10-23 UPPGIFT 2
HEAP

Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.



2020-10-23 UPPGIFT 2
HEAP

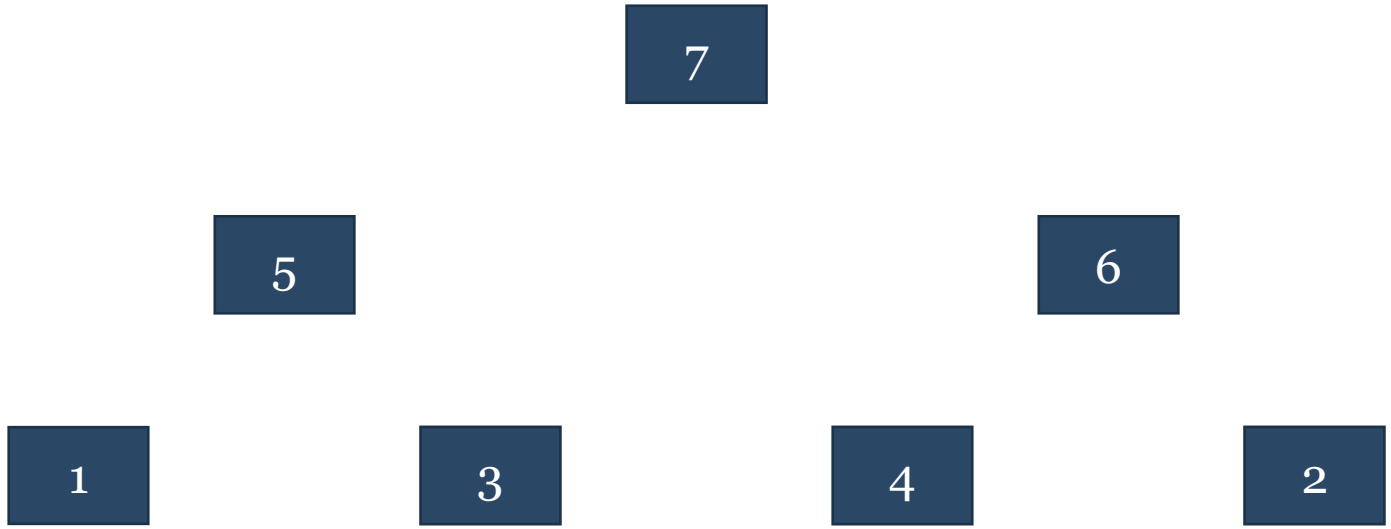
Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.



2020-10-23 UPPGIFT 2
HEAP

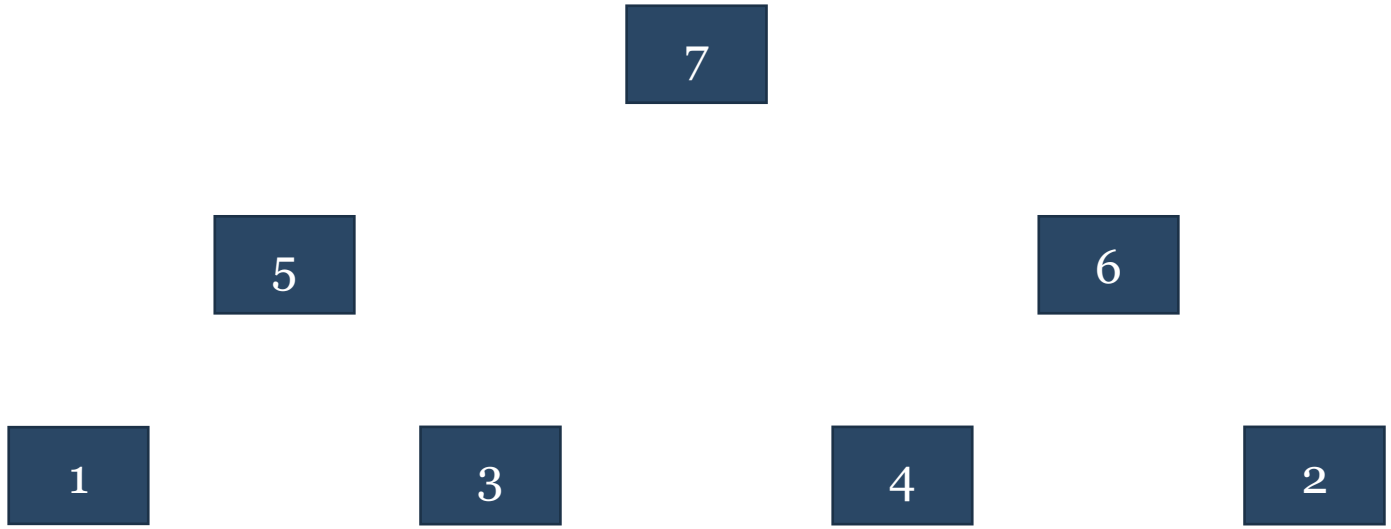
Stoppa in noderna med elementvärdena 1, 5, 4, 3, 6, 7 och 2 i en max-heap. Du skall stoppa in noderna i exakt denna ordning.

KLAR!



2020-10-23 UPPGIFT 2
HEAP

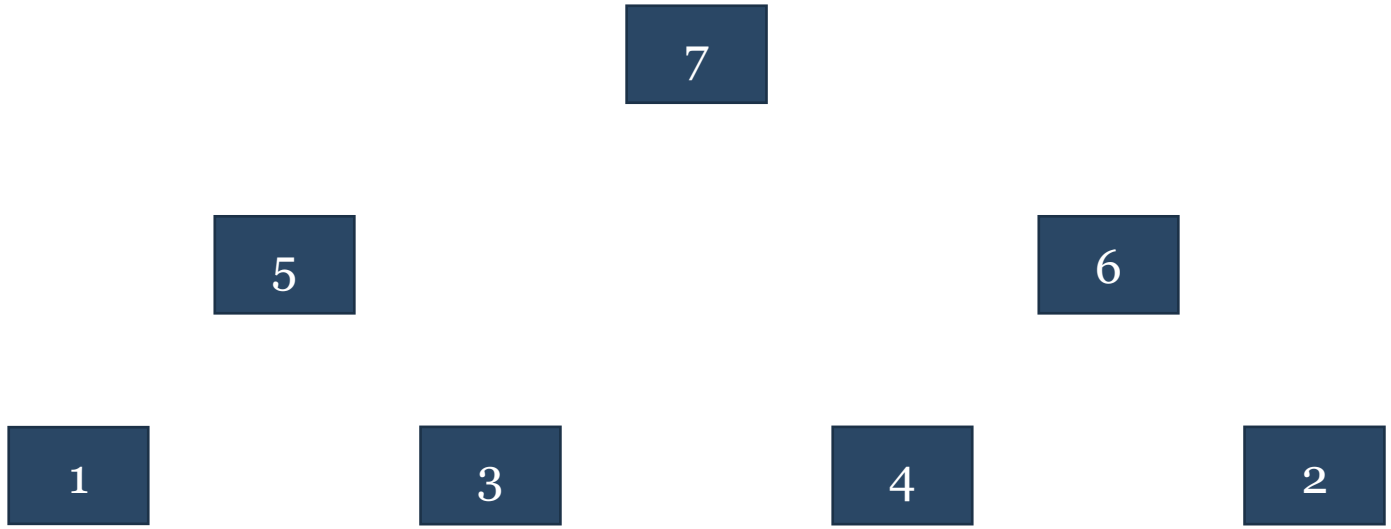
Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

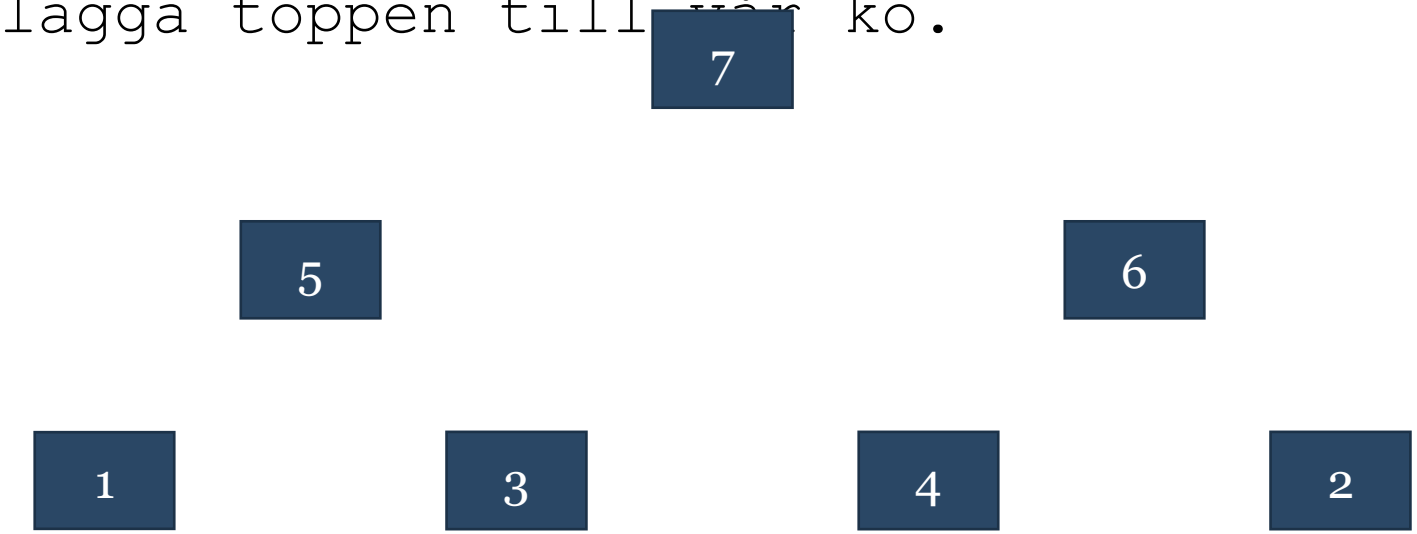
`KÖ: -` `Besökta: -`



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

```
KÖ: 7                          Besökta: -
Börja lägga toppen till vår kö.
```



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

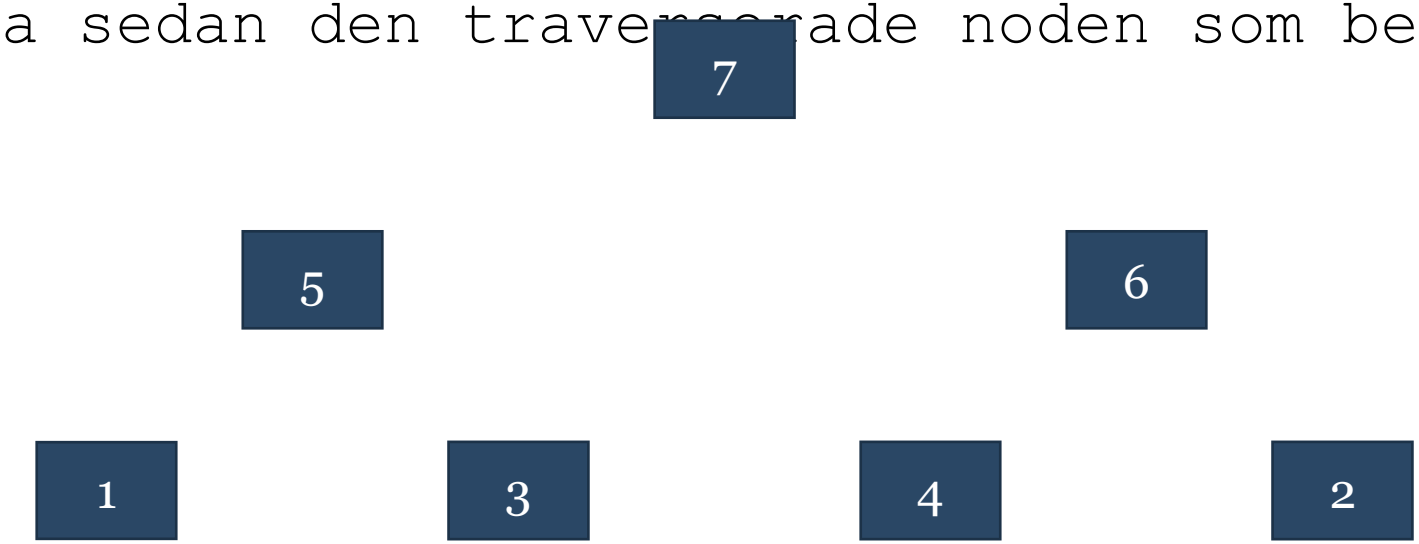
```
KÖ: (7) 5 6                    Besökta: -
Traversera första i kön och lägg till dess
eventuella barn sist.
```

2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

KÖ: 5 6 Besökta: 7

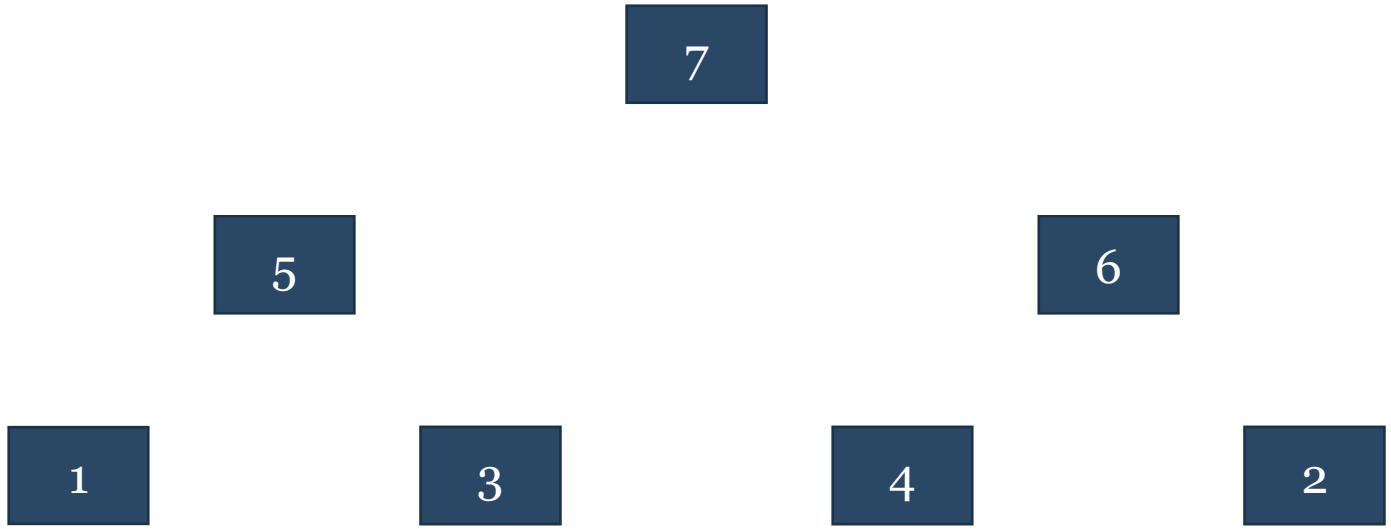
Markera sedan den traverserade noden som besökt.



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

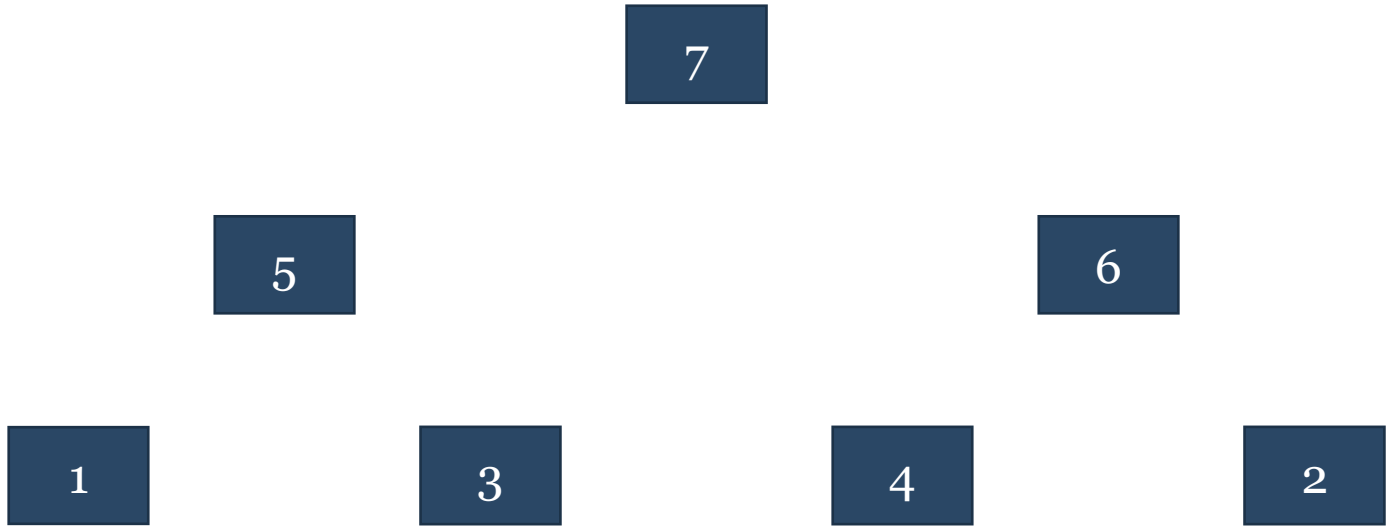
KÖ: (5) 6 Besökta: 7



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

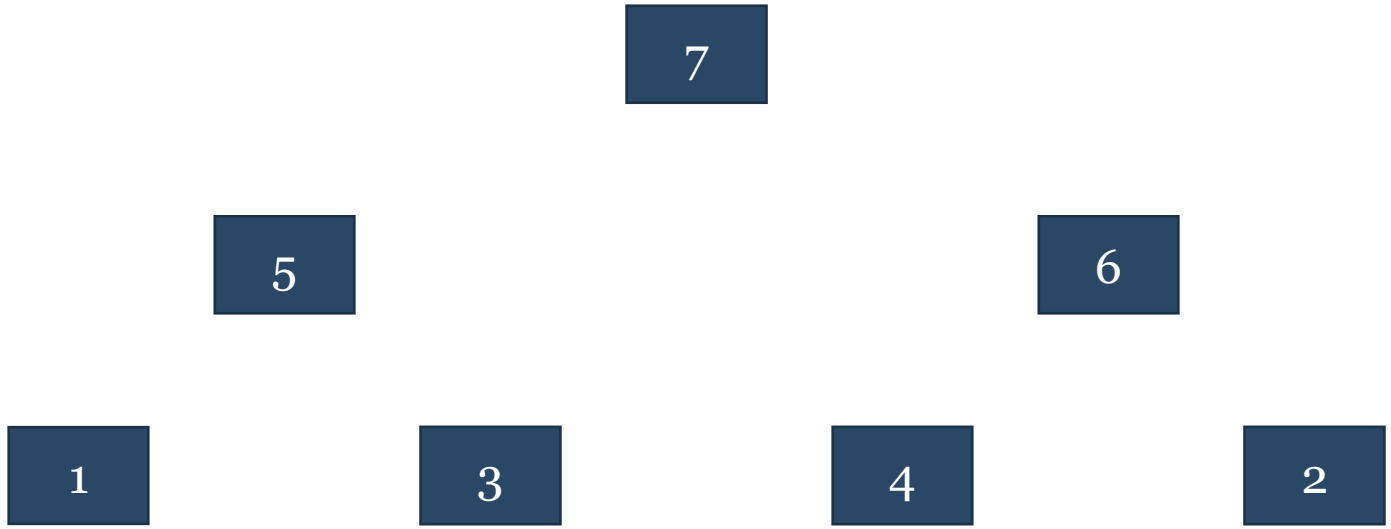
```
KÖ: (5) 6 1 3          Besökta: 7
```



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

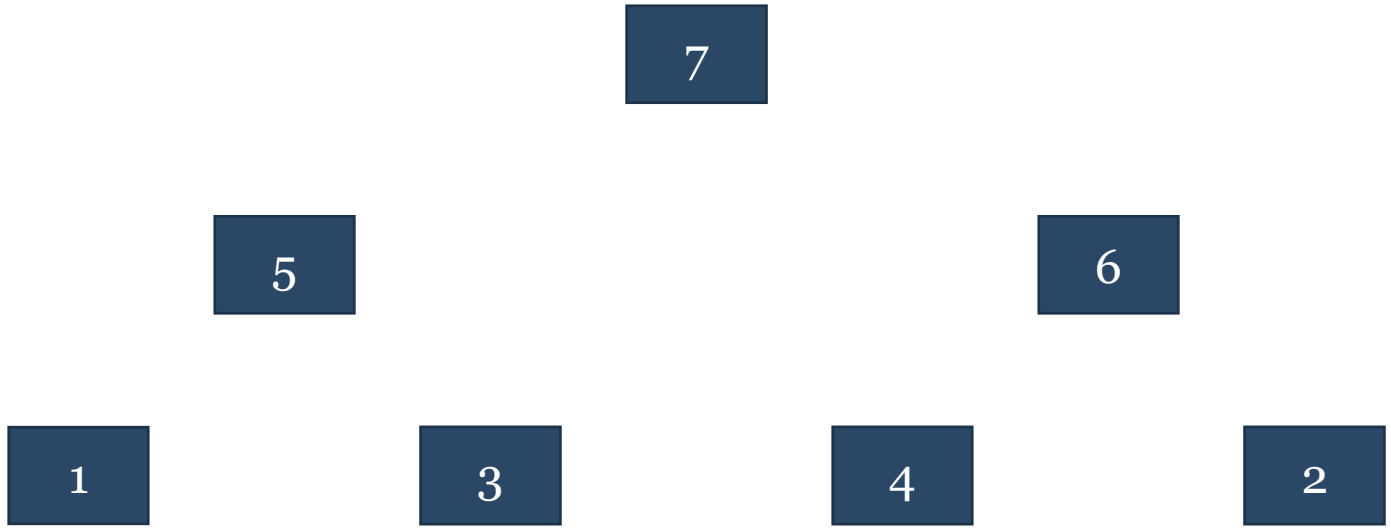
KÖ: 6 1 3 Besökta: 7 5



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

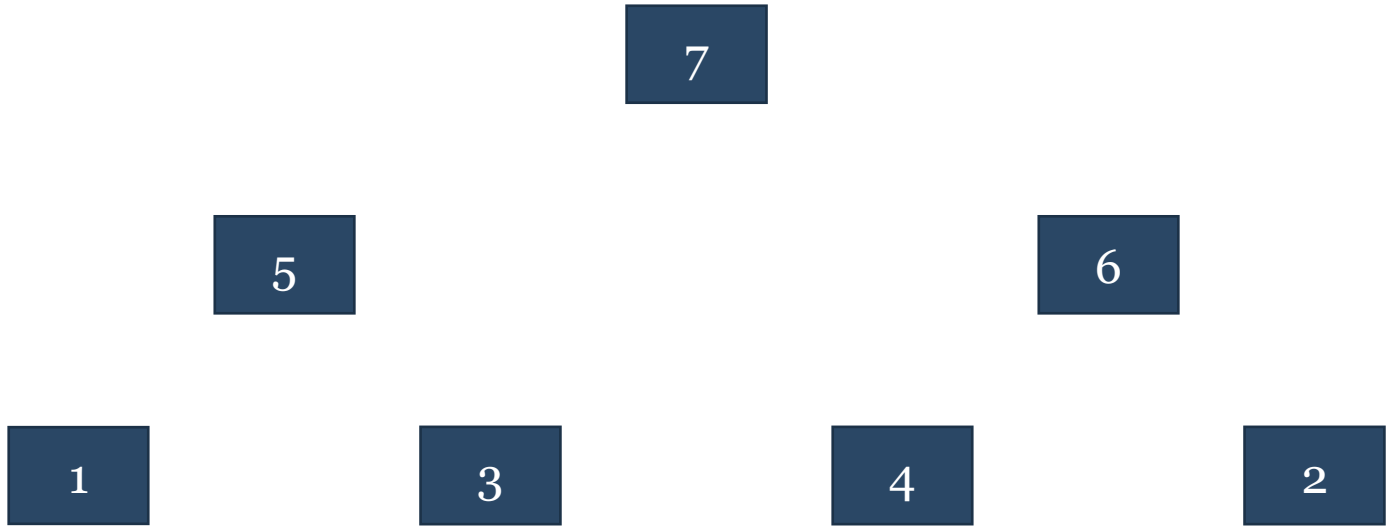
KÖ: (6) 1 3 Besökta: 7 5



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

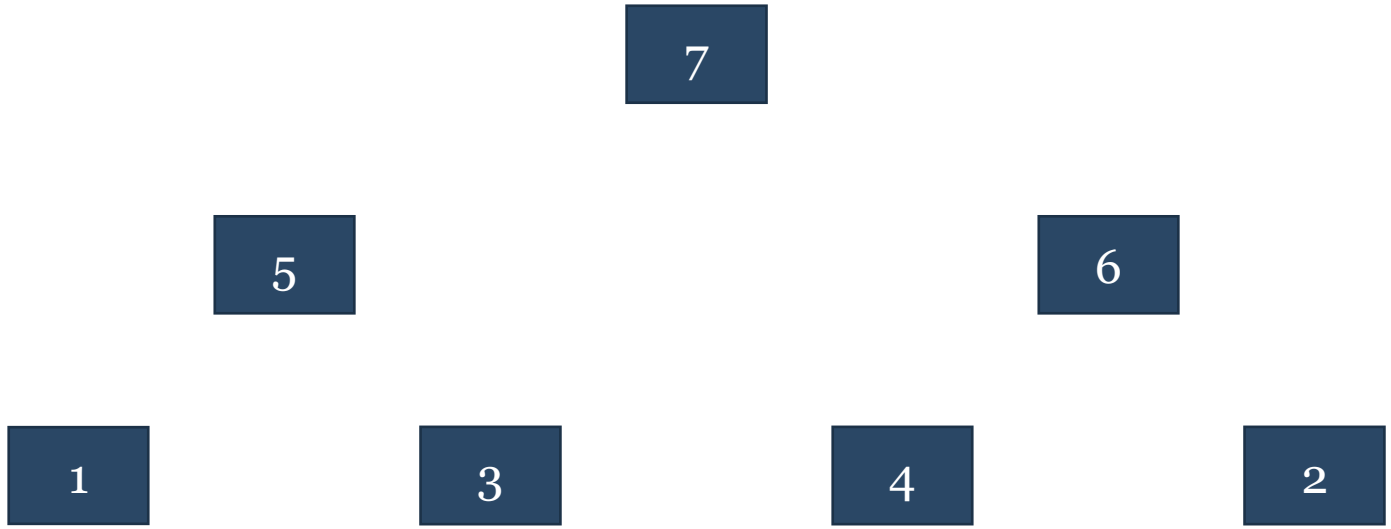
KÖ: (6) 1 3 4 2 Besökta: 7 5



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

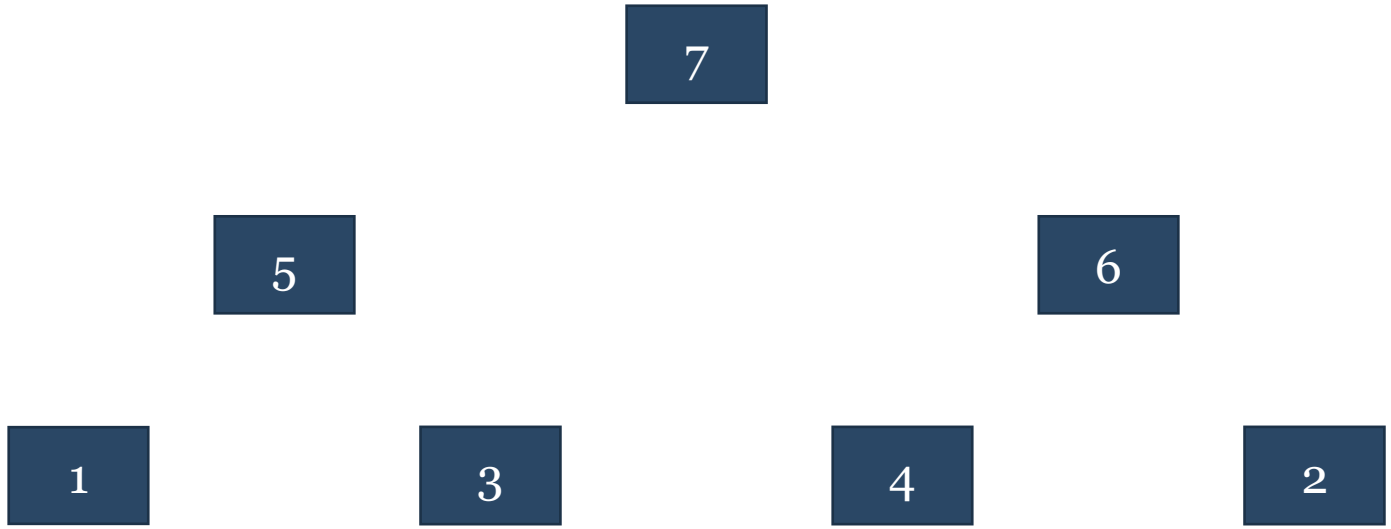
KÖ: 1 3 4 2 Besökta: 7 5 6



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

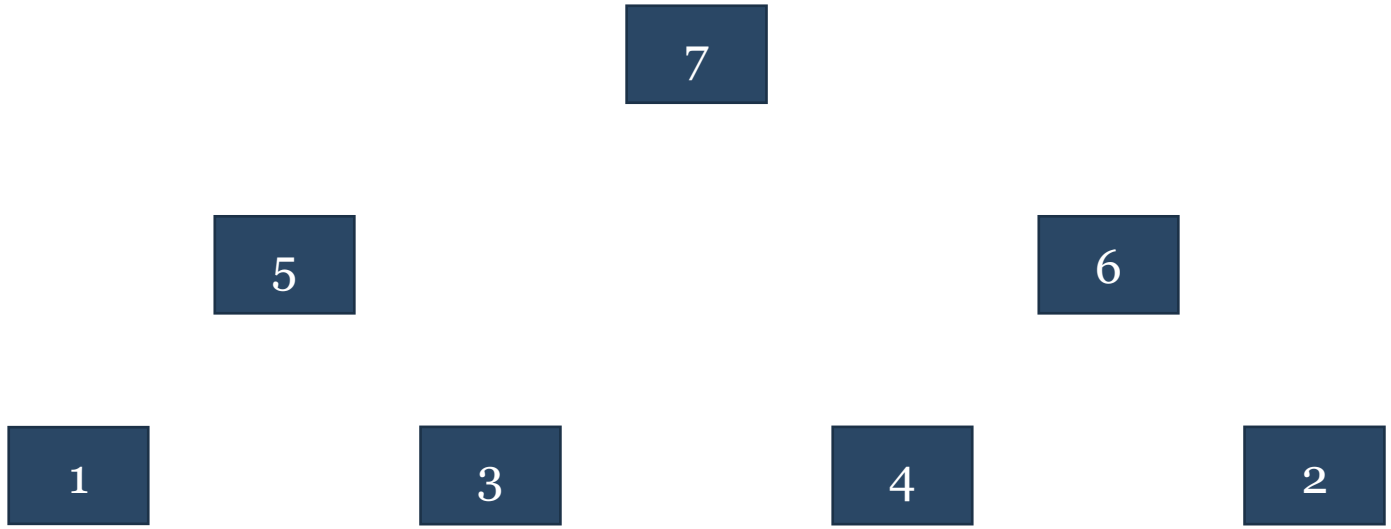
KÖ: (1) 3 4 2 Besökta: 7 5 6



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

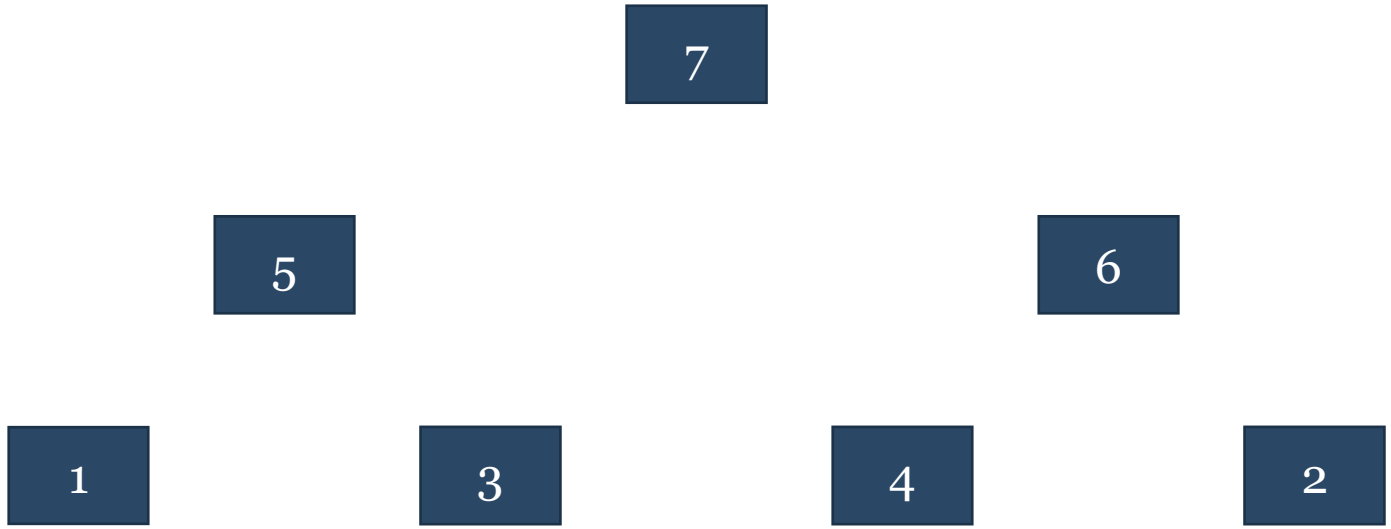
KÖ: 3 4 2 Besökta: 7 5 6 1



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

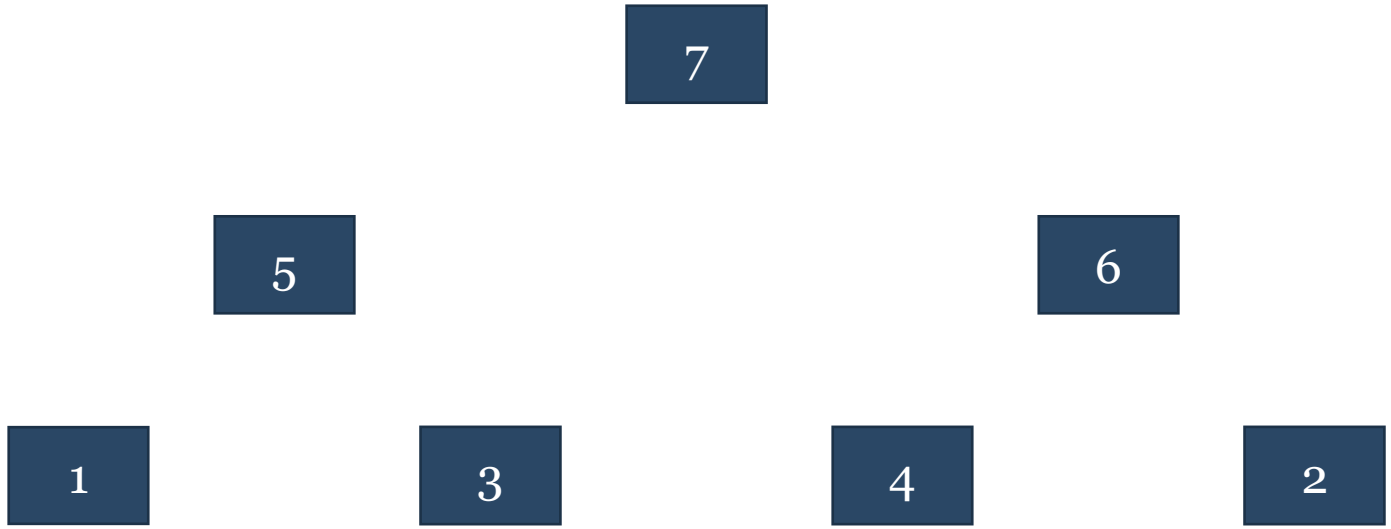
KÖ: (3) 4 2 Besökta: 7 5 6 1



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

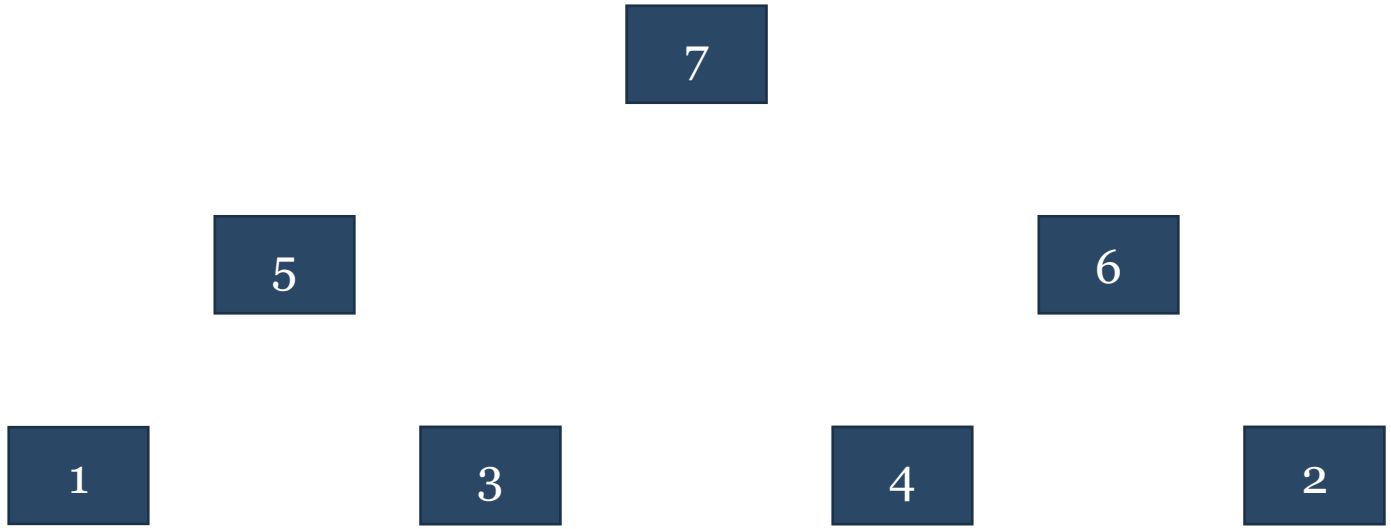
KÖ: 4 2 Besökta: 7 5 6 1 3



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

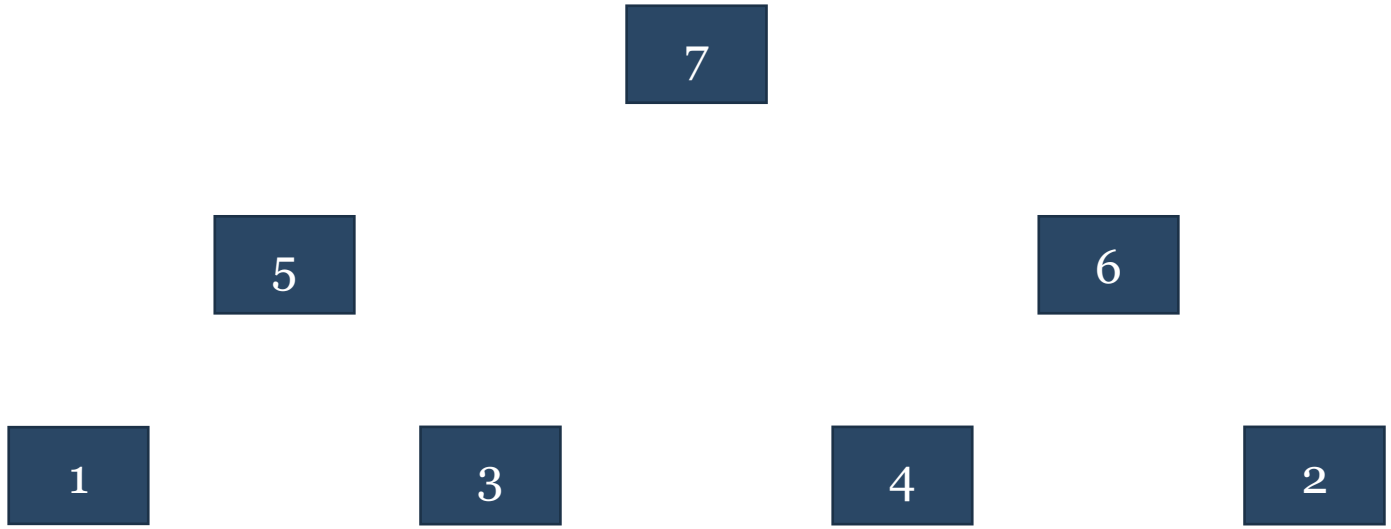
KÖ: (4) 2 Besökta: 7 5 6 1 3



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

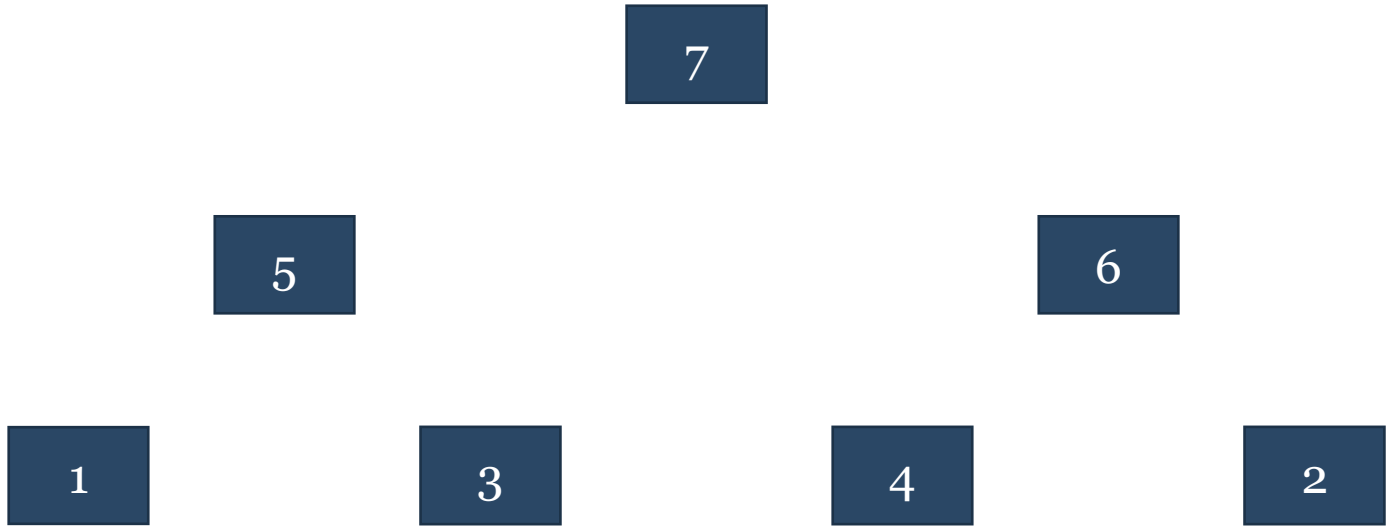
KÖ: 2 Besökta: 7 5 6 1 3 4



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

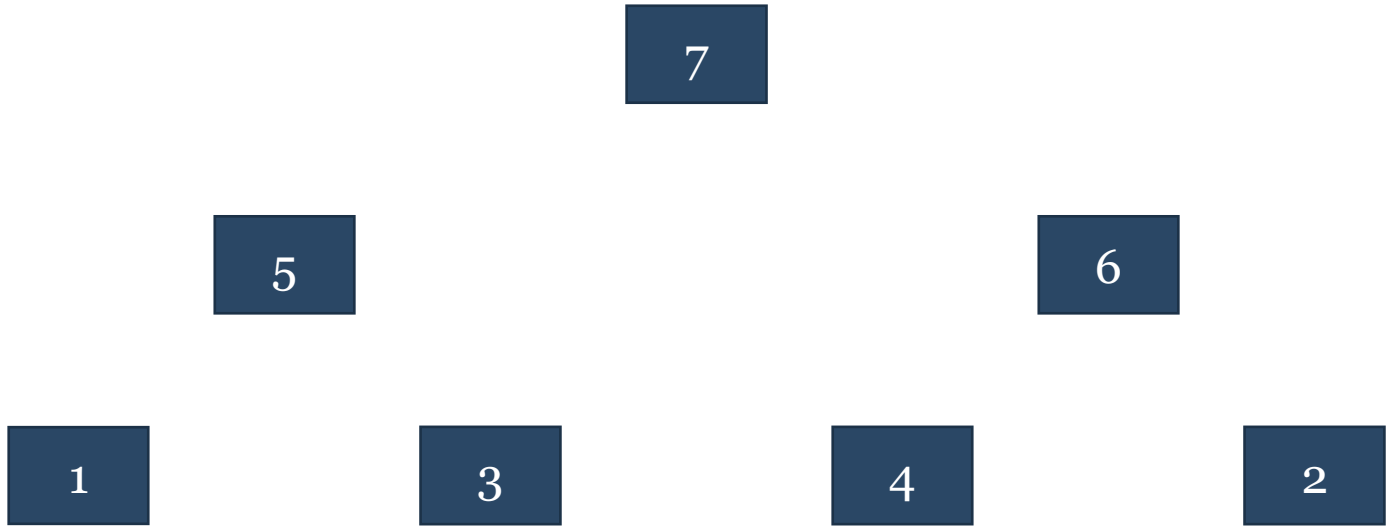
KÖ: (2) Besökta: 7 5 6 1 3 4



2020-10-23 UPPGIFT 2
HEAP

Gör sen en bredden-först traversering av din färdiga heap och skriv ned värdet på varje nod i den ordning de besöks:

```
KÖ: -                          Besökta: 7 5 6 1 3 4 2
Klar!
```



2020-10-23
UPPGIFT 14
TIDSKOMPLEXITET

UMEÅ UNIVERSITET

TIDSKOMPLEXITET (2+2+1 = 5 POÄNG)

1. Vad är syftet med asymptotisk analys av antalet operationer för en algoritm? Varför räcker det inte att studera en algoritm experimentellt för ett antal indata och dra slutsatser från det?
   - Svaret bör innehålla resonemang kring icke generella resultat och ej fullständiga datamängder och kanske bias i datamängderna
2. Utgå från begreppet *f(n) är O(g(n))* och visa att f(n) = $3n^2 + 40n - 13$ är $O(n^2)$.
   - Måste förklara definitionen av ordo och därmed visa att det räcker att beräkna c och n0.

$$c = \lim_{n\to\infty}\left(\frac{3n^2 + 40n - 13}{n^2} + 1\right) = \lim_{n\to\infty}(3 + 1) = 4$$

   - $3n^2 + 40n - 13 \leq 4n^2$ för alla n >= 40
   - Alltså är f(n) $O(n^2)$ med c = 4 och n0 = 40

TIDSKOMPLEXITET (2+2+1 = 5 POÄNG)

En algoritm har tidskomplexitet $O(n^2)$. Om man testkör den så ser man att det tar 5 sekunder att köra då n = 1 000 000. Ungefär hur lång tid kan vi anta att algoritmen tar att köra för 2 000 000 värden?

- Dubbelt så många data och kvadratisk algoritm tar 4 ggr så lång tid (2n)^2 = 4*n^2
- 20 sekunder.

2020-10-23
UPPGIFT 12

Datastrukturer användning/ implementation

UMEÅ UNIVERSITET

(1+1+1+2 = 5 POÄNG)

- Ge kortfattade svar på följande frågor:
- I implementationerna av både stack och tabell med hjälp av Array behöver man inte ’sudda ut’ borttagna element (med None t.ex.), varför inte ?
  - Man håller reda på sista positionen för inlagda värden så ingen risk att råka "ta fel".
- Hur kan det komma sig att boken och bilderna säger att Lista är en homogen datatyp fast man kan blanda både text och heltal i den implementation ni använde under kursen (DirectedList)?
  - Lista i boken är ADT, DirectedList är en implementerad datatyp.

(1+1+1+2 = 5 POÄNG)

- Varför är det viktigt att bara använda sig av de funktioner som finns definierade i interfacet när du använder en datatyp ?
  - Utbytbarhet (man kan vilja ändra den underliggande representationen).
- I sorteringsalgoritmen quicksort finns något som kallas pivot-element, vad är det och vad kan ett dåligt val av pivotelement ha för effekt på komplexiteten ?
  - Pivotelement används för att dela upp listan i två eller tre delar beroende på storlek på elementen.
  - Om man konstant väljer det minsta/största elementet i arrayen så delas det inte upp i lika stora dellistor utan alla på ena sidan $O(n^2)$ istället för $O(n \log n)$

2020-10-30
FRÅGA 13
GRAFER

UMEÅ UNIVERSITET

GRAFER (2 + 3 = 5 POÄNG)

1.I matrisrepresentationerna nedan motsvaras $A(i, j) = 1$ av att j är granne till i (mer specifikt att det går en båge från i till j). Vilken/vilka av följande matrisrepresentationer motsvarar en DAG? Motivera!

2.Prims algoritm och Dijsktras algoritm är båda grafalgoritmer. Förklara kortfattat syftet med algoritmerna (vilket typiskt grafproblem ska de lösa) och ge ett exempel på ett verkligt problem som rör trafiknätet (vägarna) i Umeå kommun som skulle kunna lösas med var och en av algoritmerna.

0 1 0 A = 0 0 0 1 0 0	0 1 0 B = 1 0 0 1 0 0	0 1 0 C = 0 1 0 1 0 0	0 1 0 D = 0 0 1 1 0 0

GRAFER (2 + 3 = 5 POÄNG)

1. I matrisrepresentationerna nedan motsvaras $A(i, j) = 1$ av att j är granne till i (mer specifikt att det går en båge från i till j). Vilken/vilka av följande matrisrepresentationer motsvarar en DAG? Motivera!
   - En DAG är en Directed Acyclic Graph, dvs en riktad graf utan cykler.
   - Om vi kallar noderna för a, b och c kan vi se att
     - Graf A har en kant a->b och en annan c->a, den saknar cykler och är en DAG
     - Graf B har vi kanter a-> b, b->a och c->a, dvs man kan gå i en cykel a-b-a, inte en DAG.
     - Graf C a->b, b->b, c->a har cykler, ej DAG
     - Graf D a->b, b->c, c->a har cykler, ej DAG

0 1 0 A = 0 0 0 1 0 0	0 1 0 B = 1 0 0 1 0 0	0 1 0 C = 0 1 0 1 0 0	0 1 0 D = 0 0 1 1 0 0

GRAFER (2 + 3 = 5 POÄNG)

Prims algoritm och Dijsktras algoritm är båda grafalgoritmer. Förklara kortfattat syftet med algoritmerna (vilket typiskt grafproblem ska de lösa) och ge ett exempel på ett verkligt problem som rör trafiknätet (vägarna) i Umeå kommun som skulle kunna lösas med var och en av algoritmerna.

- Prims ska finna ett minsta uppspännande träd. Exempel: Hur ska man ploga vägar så att man plogar minst sammanlagd sträcka men ändå förbinder alla skolor i stan med varandra.
- Dijsktras, kortaste vägen från en nod till alla andra noder i en graf. Exempel: Ska åka buss från Umedalen till Röbäck, vilka busslinjer ska jag ta för att komma fram så snabbt som möjligt?

2013-03-26
UPPGIFT 1
FÄLT OCH PSEUDOKOD

UMEÅ UNIVERSITET

UPPGIFT 1 –FÄLT OCH PSEUDOKOD (5P)

- Skriv en algoritm i pseudokod som, givet två fält (array) med heltal avgör om arrayerna innehåller samma värden. Ordningen eller antalet av varje värde ska inte spela någon roll. Algoritmen skall returnera sant om arrayerna innehåller samma värden, falskt annars. Använd er av operationerna i datatypen i er algoritm.

  Exempel:

```
{ 1, 2, 3, 4, 100, 120}{ 1, 4,100, 2,120, 3}-> sant
{ 2, 5, 8, -2, -2, 100, 102} { 2, 5, -2, 100, 102} -> falskt
```

- Lösningsidé:
  - Gör ett grundantagande att det inte finns hål i arrayen.
  - Gör en delalgoritm som kollar om ett element är medlem i en array eller ej. Använd den sen för att först kolla alla element i första arrayen finns med i den andra och sen tvärtom.

```
Algorithm isInArray(val, arr)
index <- low(arr)
found <- False
while not found and index <> high(arr) do
    if inspectValue(index, arr) = val then
        found <- True
    index <- index + 1 // Depending on
type of index...
return found
```

```
Algorithm isEqual(arr1, arr2)

index <- low(arr1)

equal <- True

while equal and index <> high(arr1) do
    val <- inspectValue(index, arr1)

    if not isInArray(val, arr2) then

       equal <- False

    end

    index <- index + 1 // Depending on type of index...

end

index <- low(arr2)

while equal and index <> high(arr2) do
    val <- inspectValue(index, arr2)

    if not isInArray(val, arr1) then

       equal <- False

    end

    index <- index + 1 // Depending on type of index...

end

return equal
```

2018-03-08
UPPGIFT 1
TERMINOLOGI

UMEÅ UNIVERSITET

TERMINOLOGI

- Förklara och ge exempel och motexempel på
- **Heterogen datatyp:** En sammansatt datatyp där elementen kan ha olika typer.
  - Exempel Post, cell
  - Motexempel Lista, fält, kö, stack, tabell
- **Ordnad datatyp:** En sammansatt datatyp där elementen är ordnade, dvs. det finns ett första element, ett nästa element, osv.
  - Ej att förväxla med sorterad datatyp
  - Exempel Lista, kö, fält, träd, . . .
  - Motexempel Mängd, tabell, graf

TERMINOLOGI

- Förklara och ge exempel på
- **Fyllnadsgrad:** Andelen upptagna platser i en statisk datatyp, vanligen en hashtabell.
- **Stabil sortering:** En sorteringsalgoritm som bibehåller ordningen på två in-element som är lika enligt sorteringsordningen.

2020-10-30
FRÅGA 16
TABELLER

UMEÅ UNIVERSITET

TABELLER (2 + 2 + 2 = 6 POÄNG)

- Data i en tabell lagras med hjälp av en nyckel (key). Vad är en nyckel, vilka krav kan man ställa på den datatyp som nyckeln har?
  - Nycklen ska vara unik i en tabell och kopplas till ett värde. Tex personnummer som nyckel och namn som värde. Datatypen måste avgöra likhet mellan två nycklar.
- Den potentiella uppsättningen nycklar är mycket stor. Ge en metod för att minska den uppsättningen. Hur fungerar den?
  - Hashtabeller, för detaljer se föreläsningen om dem.
- Vilket problem kan vi få pga att vi använder metoden i förra frågan och hur kan vi lösa problemet?
  - Kollisioner, kan lösas med sluten hashing (linjär och kvadratisk teknik) eller öppen hashing, se föreläsning för detaljer.

2018-03-26
UPPGIFT 4
GRAF

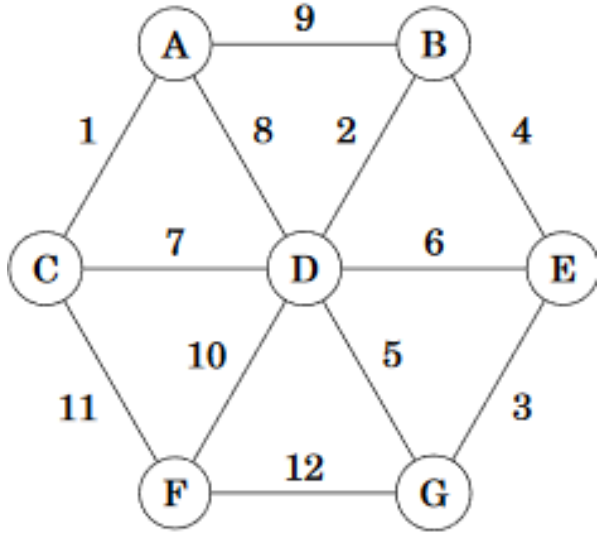
UMEÅ UNIVERSITET

UPPGIFT 4 – GRAF (1+4=5P)

Företaget du arbetar för har ett uppdrag där ni skall skriva en modul som skall användas för att konstruera ett nytt datornät där våra byar i glesbygden skall kopplas upp med hjälp av ett fibernät. Kraven är att alla byar som är med i projektet skall kunna nås från vilken annan by som är med i projektet direkt eller indirekt via någon annan by. Givetvis så är det en kostnad förknippad med att dra fiber mellan två byar som beror dels på avståndet men också på markförhållandena, etc. Beställarna (staten) vill ha den billigaste lösningen för fiberdragningen.

a) Detta är ett problem som kan härledas till typproblemet “Minimalt uppspännande träd”. Ange en algoritm som löser detta problem.

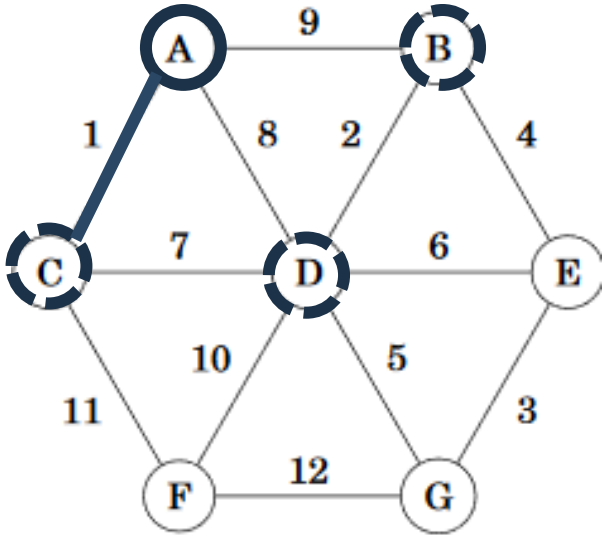
b) Visa hur din algoritm från a) fungerar på nedanstående nätverk. Siffrorna på bågarna i grafen är kostnaden för att dra en fiber mellan dess noder. Visa hur grafen ser utefter varje steg



PRIMS ALGORITM

1. Låt A bli rot i trädet.
2. Markera A som stängd.
3. För var och en av (de icke-stängda) grannarna:
   1. Markera den som öppen (om den inte är det).
   2. Stoppa in den aktuella noden n, grannen w och vikten d i en prioritetskö q. Är vikterna lika ska det nya elementet läggas in först i kön. (dvs relationen är ≤)
4. Upprepa:
   1. Ta första bågen (n,d,w) ur q.
   2. Om destinationsnoden d är öppen:
   3. Lägg till bågen till trädet.

   tills d öppen (lagt till en båge) eller q tom (klara).
5. Låt d bli den nya aktuella noden, stäng den och gå till 3.



q = {AB9}

q = {AC1, AB9}

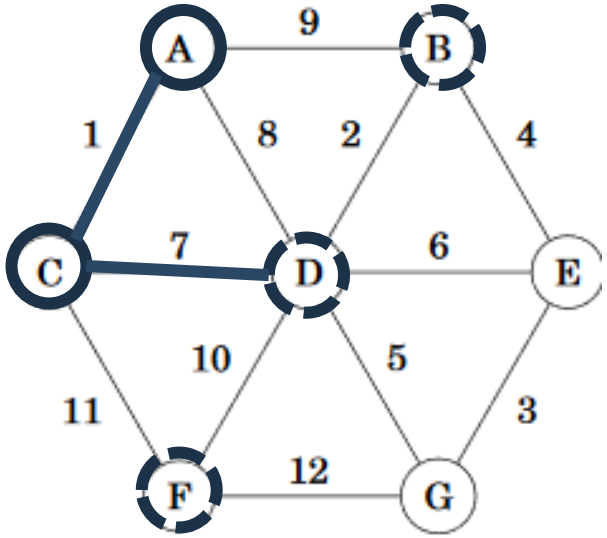
q = {AC1, AD8, AB9}

q = {AD8, AB9}

PRIMS ALGORITM

1. Låt A bli rot i trädet.
2. Markera A som stängd.
3. För var och en av (de icke-stängda) grannarna:
   1. Markera den som öppen (om den inte är det).
   2. Stoppa in den aktuella noden n, grannen w och vikten d i en prioritetskö q. Är vikterna lika ska det nya elementet läggas in först i kön. (dvs relationen är ≤)
4. Upprepa:
   1. Ta första bågen (n,d,w) ur q.
   2. Om destinationsnoden d är öppen:
   3. Lägg till bågen till trädet.

   tills d öppen (lagt till en båge) eller q tom (klara).
5. Låt d bli den nya aktuella noden, stäng den och gå till 3.



q = {AD8, AB9}

q = {CD7, AD8, AB9}

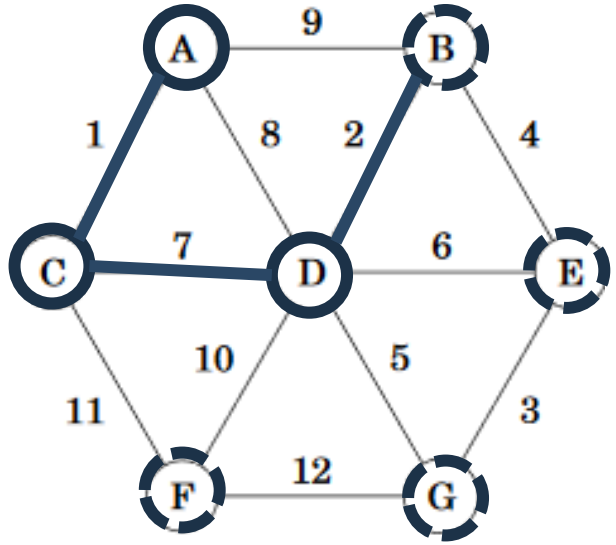
q = {CD7, AD8, AB9, CF11}

q = {AD8, AB9, CF11}

PRIMS ALGORITM

1. Låt A bli rot i trädet.
2. Markera A som stängd.
3. För var och en av (de icke-stängda) grannarna:
   1. Markera den som öppen (om den inte är det).
   2. Stoppa in den aktuella noden n, grannen w och vikten d i en prioritetskö q. Är vikterna lika ska det nya elementet läggas in först i kön. (dvs relationen är ≤)
4. Upprepa:
   1. Ta första bågen (n,d,w) ur q.
   2. Om destinationsnoden d är öppen:
   3. Lägg till bågen till trädet.

   tills d öppen (lagt till en båge) eller q tom (klara).
5. Låt d bli den nya aktuella noden, stäng den och gå till 3.



q = {AD8, AB9, CF11}

q = {DB2, AD8, AB9, CF11}

q = {DB2, DE6, AD8, AB9, CF11}

q = {DB2, DG5, DE6, AD8, AB9, CF11}

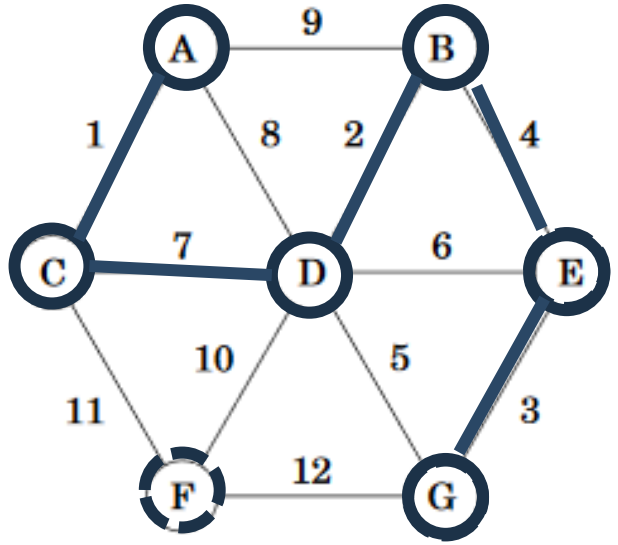
q = {DB2, DG5, DE6, AD8, AB9, DF10, CF11}

q = {DG5, DE6, AD8, AB9, DF10, CF11}

PRIMS ALGORITM

1. Låt A bli rot i trädet.
2. Markera A som stängd.
3. För var och en av (de icke-stängda) grannarna:
   1. Markera den som öppen (om den inte är det).
   2. Stoppa in den aktuella noden n, grannen w och vikten d i en prioritetskö q. Är vikterna lika ska det nya elementet läggas in först i kön. (dvs relationen är ≤)
4. Upprepa:
   1. Ta första bågen (n,d,w) ur q.
   2. Om destinationsnoden d är öppen:
   3. Lägg till bågen till trädet.

   tills d öppen (lagt till en båge) eller q tom (klara).
5. Låt d bli den nya aktuella noden, stäng den och gå till 3.



q = {DG5, DE6, AD8, AB9, DF10, CF11}

q = {BE4, DG5, DE6, AD8, AB9, DF10, CF11}

q = {DG5, DE6, AD8, AB9, DF10, CF11}

q = {EG3, DG5, DE6, AD8, AB9, DF10, CF11}

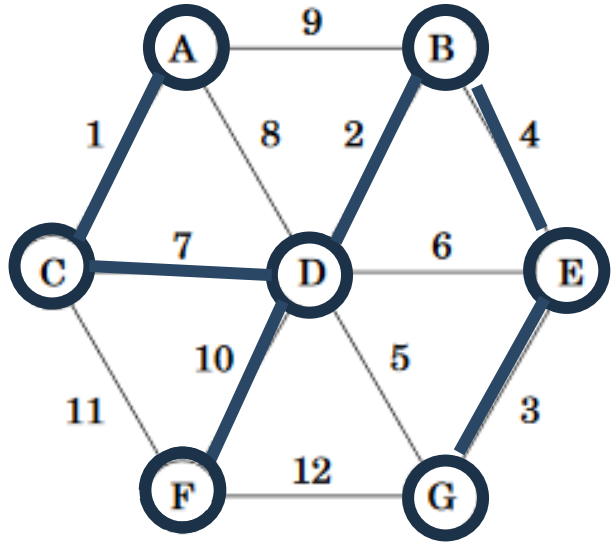
q = {DG5, DE6, AD8, AB9, DF10, CF11}

q = {DG5, DE6, AD8, AB9, DF10, CF11, GF12}

PRIMS ALGORITM

1. Låt A bli rot i trädet.
2. Markera A som stängd.
3. För var och en av (de icke-stängda) grannarna:
   1. Markera den som öppen (om den inte är det).
   2. Stoppa in den aktuella noden n, grannen w och vikten d i en prioritetskö q. Är vikterna lika ska det nya elementet läggas in först i kön. (dvs relationen är ≤)
4. Upprepa:
   1. Ta första bågen (n,d,w) ur q.
   2. Om destinationsnoden d är öppen:
   3. Lägg till bågen till trädet.

   tills d öppen (lagt till en båge) eller q tom (klara).
5. Låt d bli den nya aktuella noden, stäng den och gå till 3.



q = {DG5, DE6, AD8, AB9, DF10, CF11, GF12}

q = {DE6, AD8, AB9, DF10, CF11, GF12}

q = {AD8, AB9, DF10, CF11, GF12}

q = {AB9, DF10, CF11, GF12}

q = {DF10, CF11, GF12}

q = {CF11, GF12}

q = {GF12}

q = {}

MOTSVARANDE
2018-03-26
UPPGIFT 7
SORTERING

Vi använder istället den Max-Heap vi skapat i en tidigare uppgift

UMEÅ UNIVERSITET

2018-03-08 UPPGIFT 7 SORTERING (MODIFIERAD)

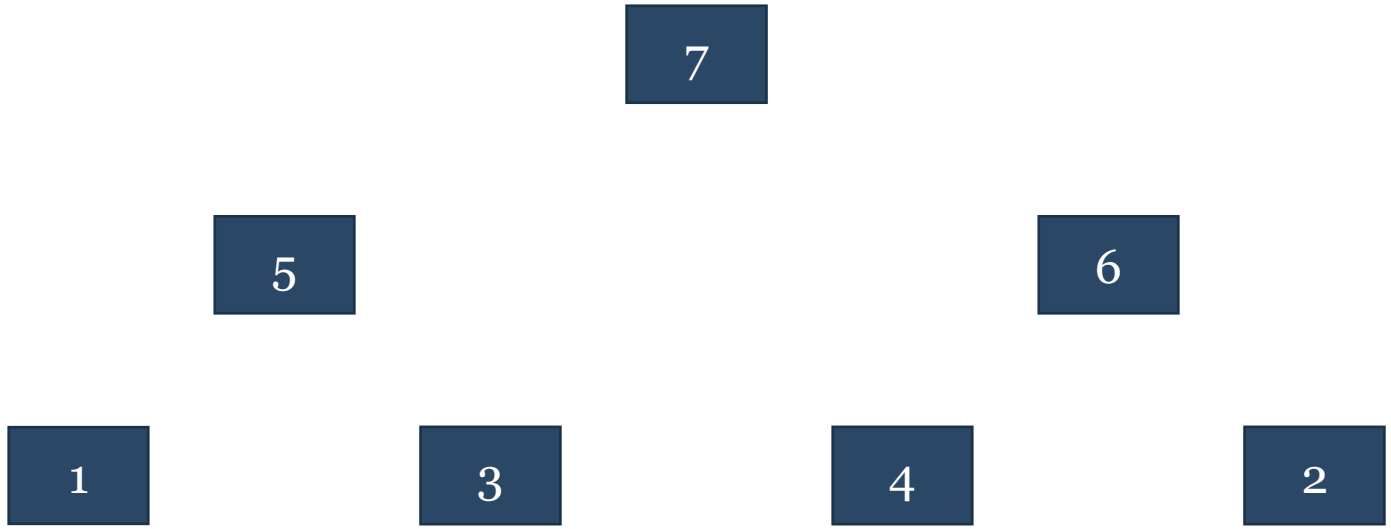
b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

c) Om man vill använd en Heap för att sortera värden, vilken komplexitet får sorteringen?

2018-03-08 UPPGIFT 7
SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

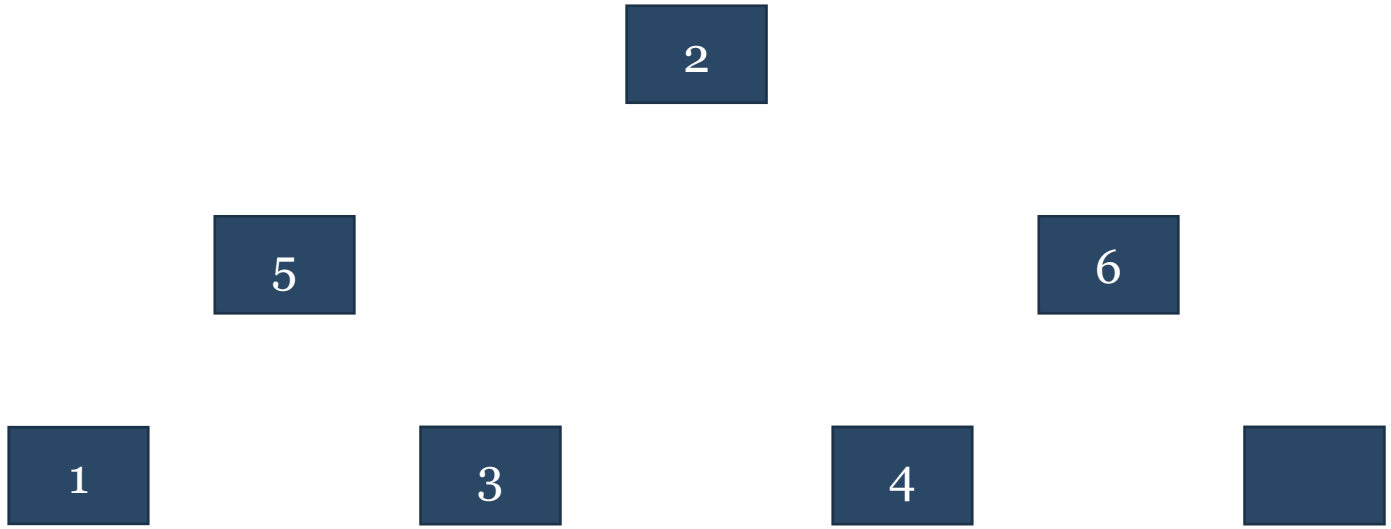
```
Utplockade element: -
Största värdet => Toppen på får hög!
```



2018-03-08 UPPGIFT 7
SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

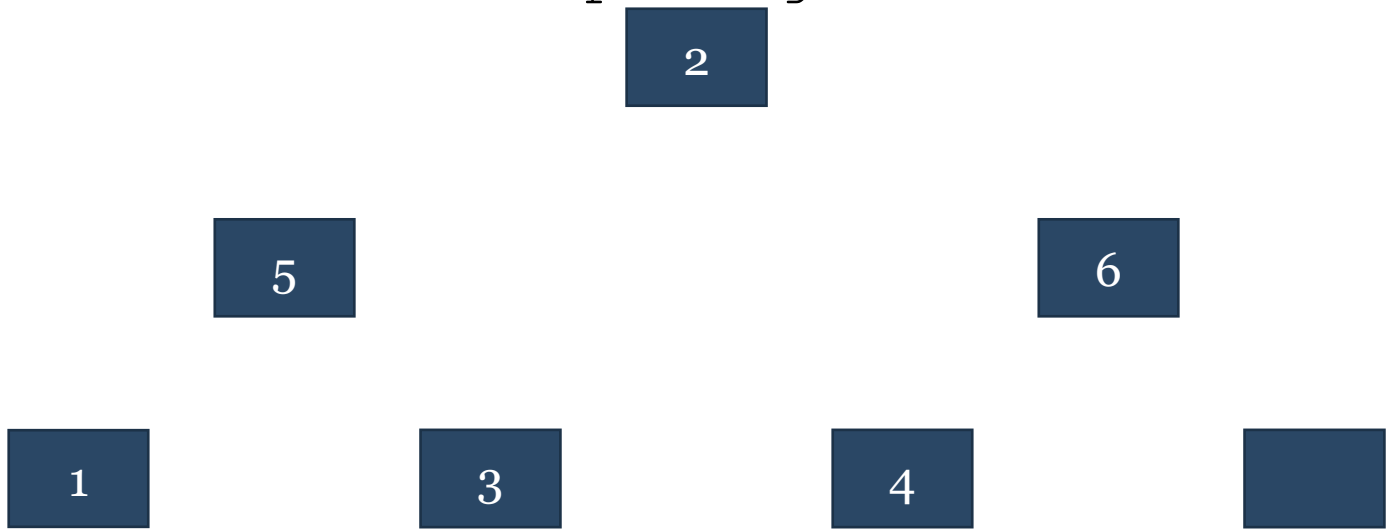
```
Utplockade element: 7
Lösning -> Ta sista elementet!
```



2018-03-08 UPPGIFT 7 SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

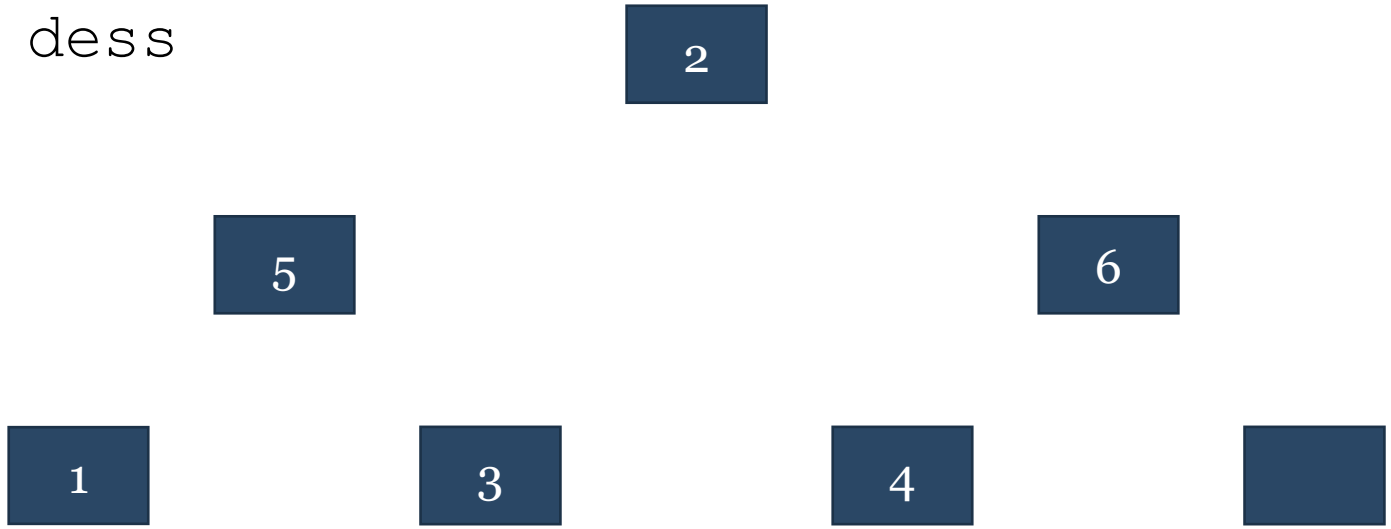
```
Utplockade element: 7
Problem -> Inte en heap längre!
```



2018-03-08 UPPGIFT 7 SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

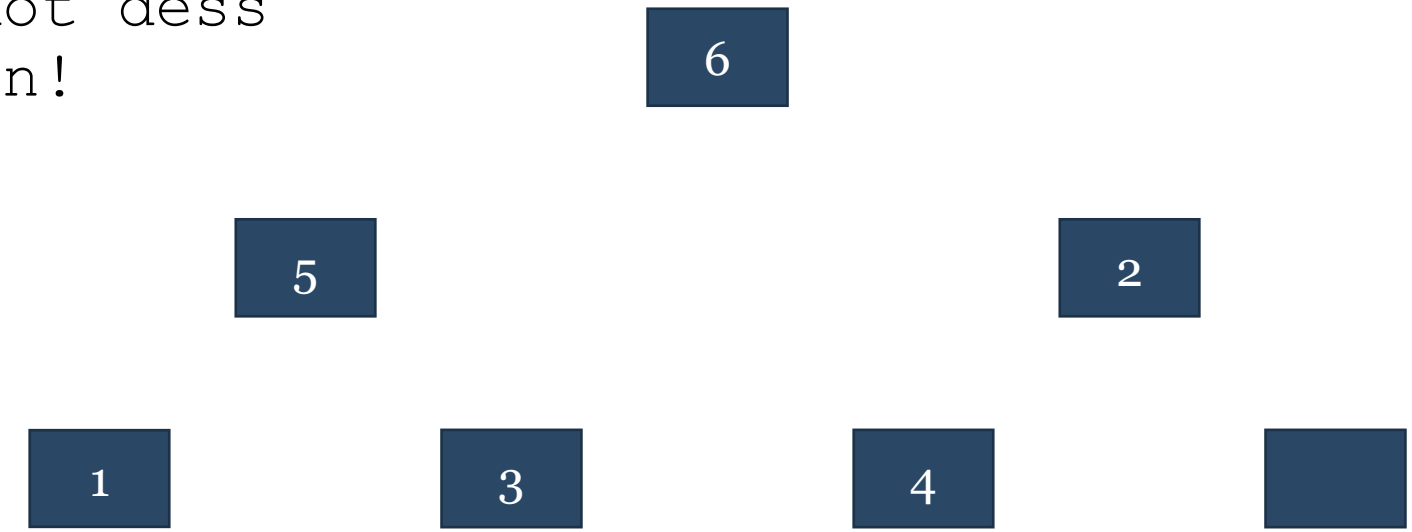
```
Utplockade element: 7
Lösning -> Åter ställ heapen med att byta den
felaktiga mot dess
största barn!
```



2018-03-08 UPPGIFT 7 SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

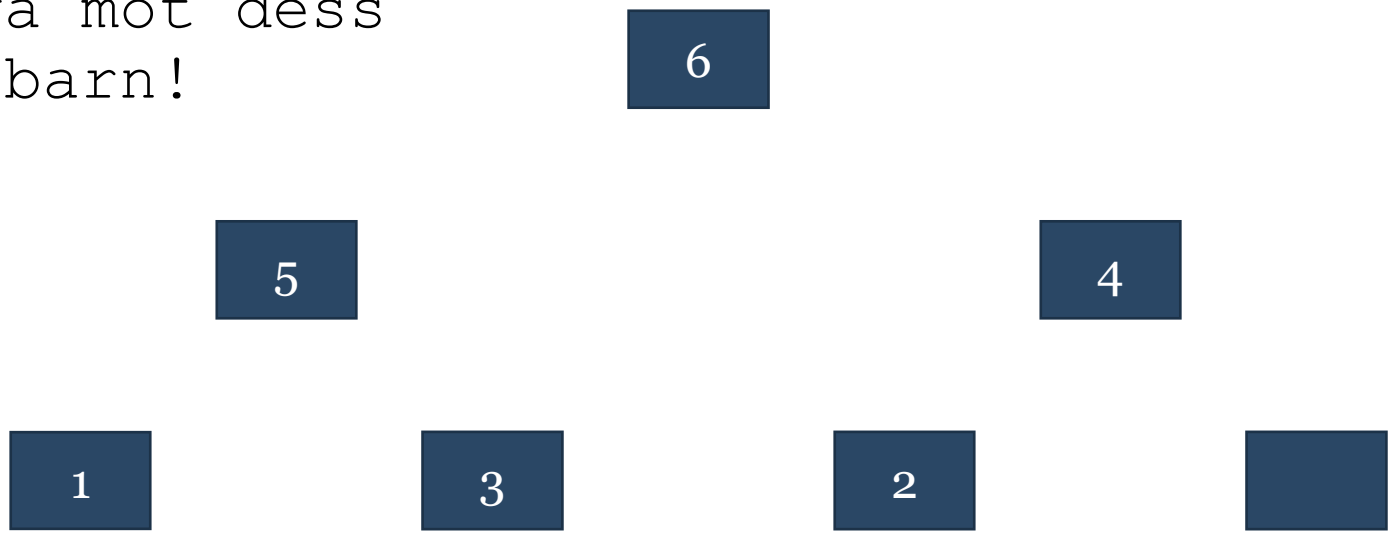
```
Utplockade element: 7
Lösning -> Åter ställ heapen med att byta den
felaktiga mot dess
största barn!
```



2018-03-08 UPPGIFT 7 SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

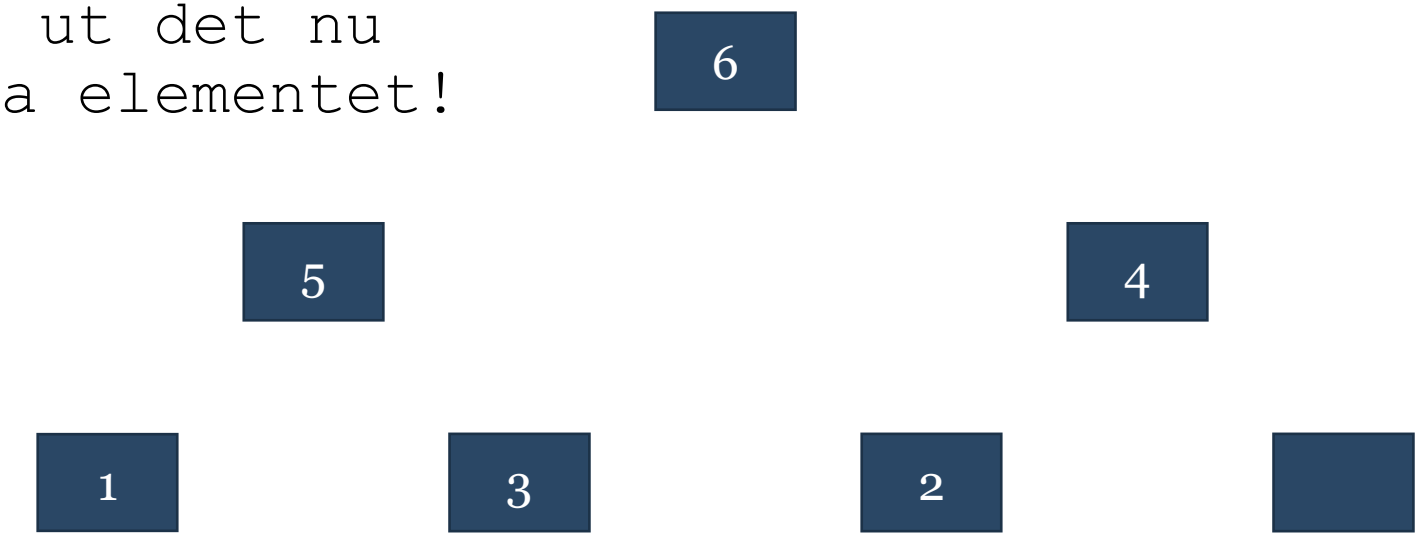
```
Utplockade element: 7
Lösning -> Åter ställ heapen med att byta den
felaktiga mot dess
största barn!
```



2018-03-08 UPPGIFT 7
SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

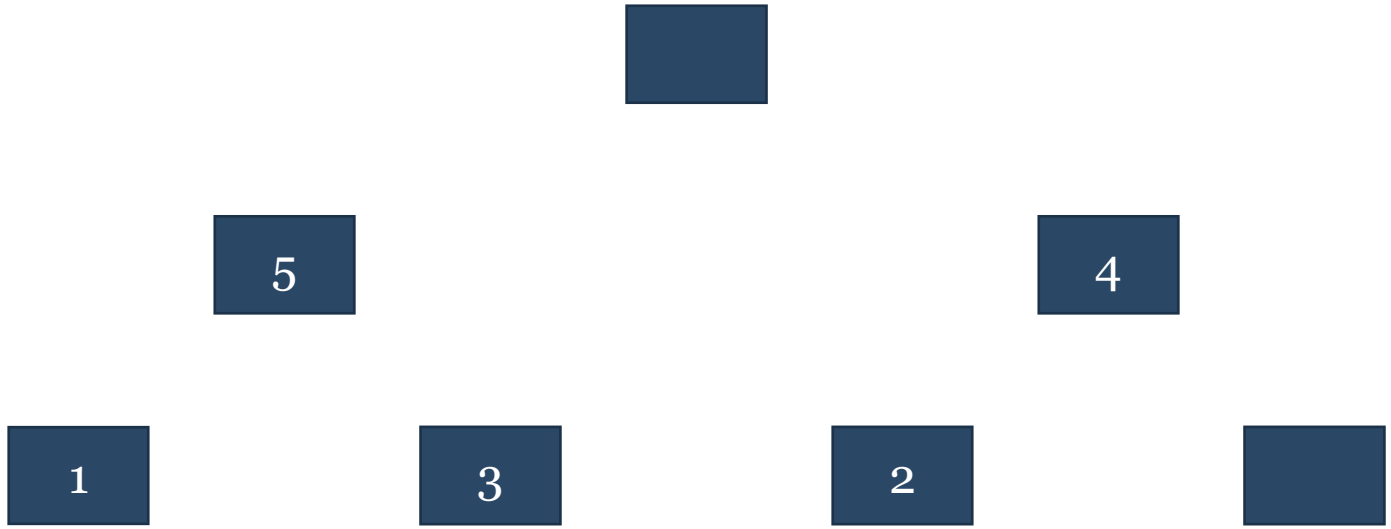
```
Utplockade element: 7
När heapen är återställd, så kan vi åter igen
plocka ut det nu
största elementet!
```



2018-03-08 UPPGIFT 7
SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

```
Utplockade element: 7 6
```

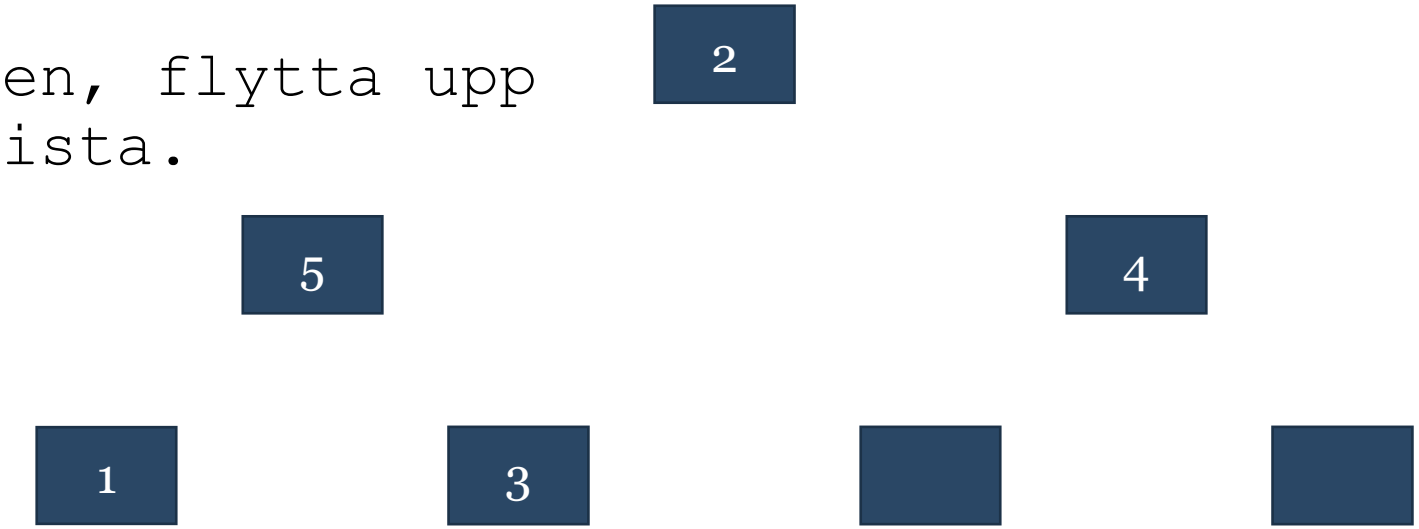


2018-03-08 UPPGIFT 7
SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

```
Utplockade element: 7 6

Återigen, flytta upp
från sista.
```

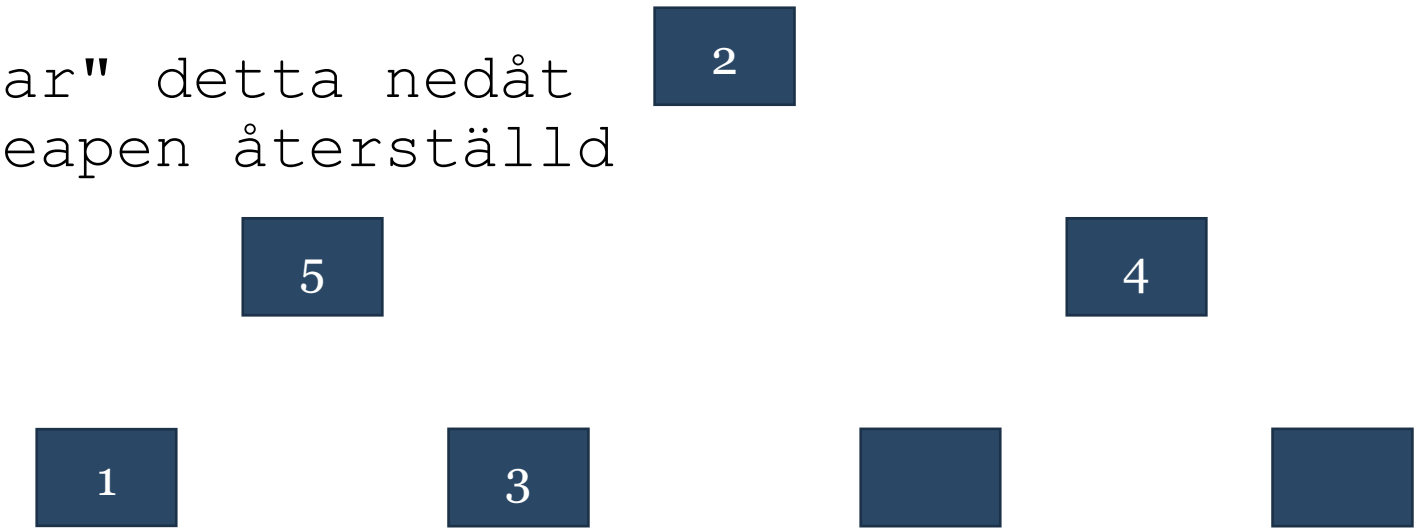


2018-03-08 UPPGIFT 7
SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

```
Utplockade element: 7 6

"Bubblar" detta nedåt
till heapen återställd
```

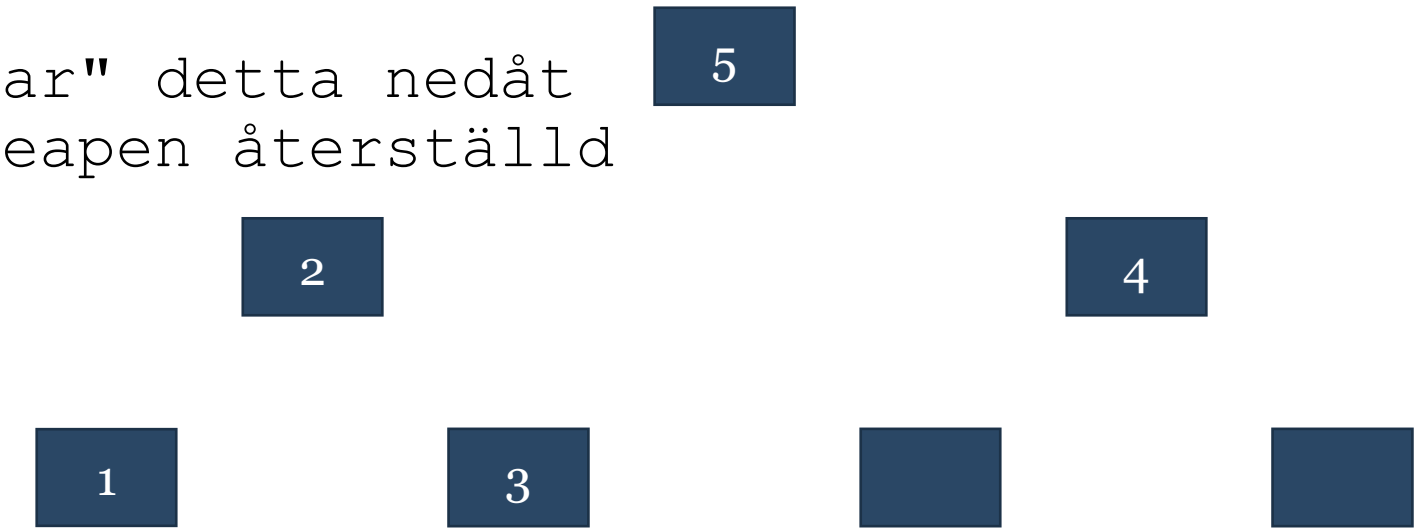


2018-03-08 UPPGIFT 7
SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

```
Utplockade element: 7 6

"Bubblar" detta nedåt
till heapen återställd
```

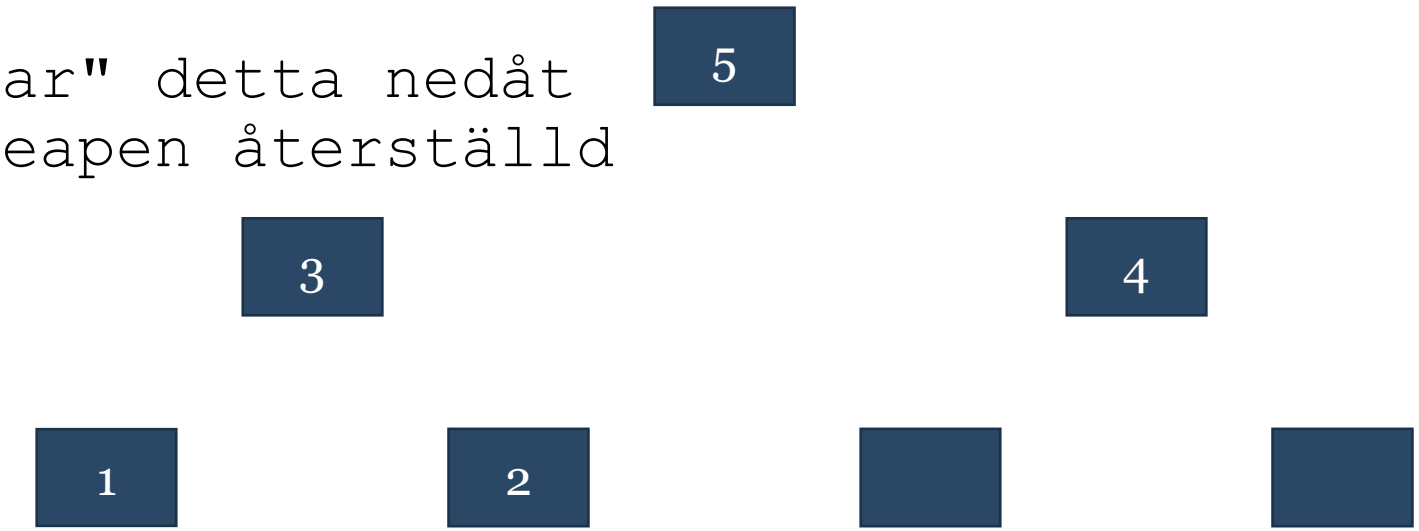


2018-03-08 UPPGIFT 7
SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

```
Utplockade element: 7 6

"Bubblar" detta nedåt
till heapen återställd
```

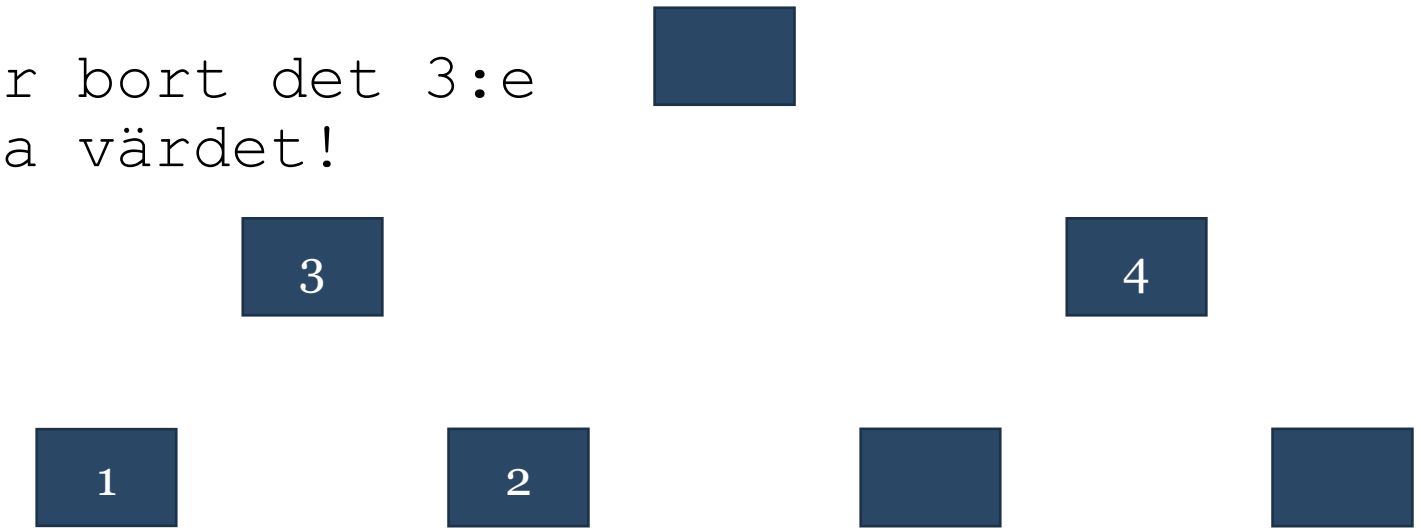


2018-03-08 UPPGIFT 7
SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

```
Utplockade element: 7 6 5

Plockar bort det 3:e
största värdet!
```

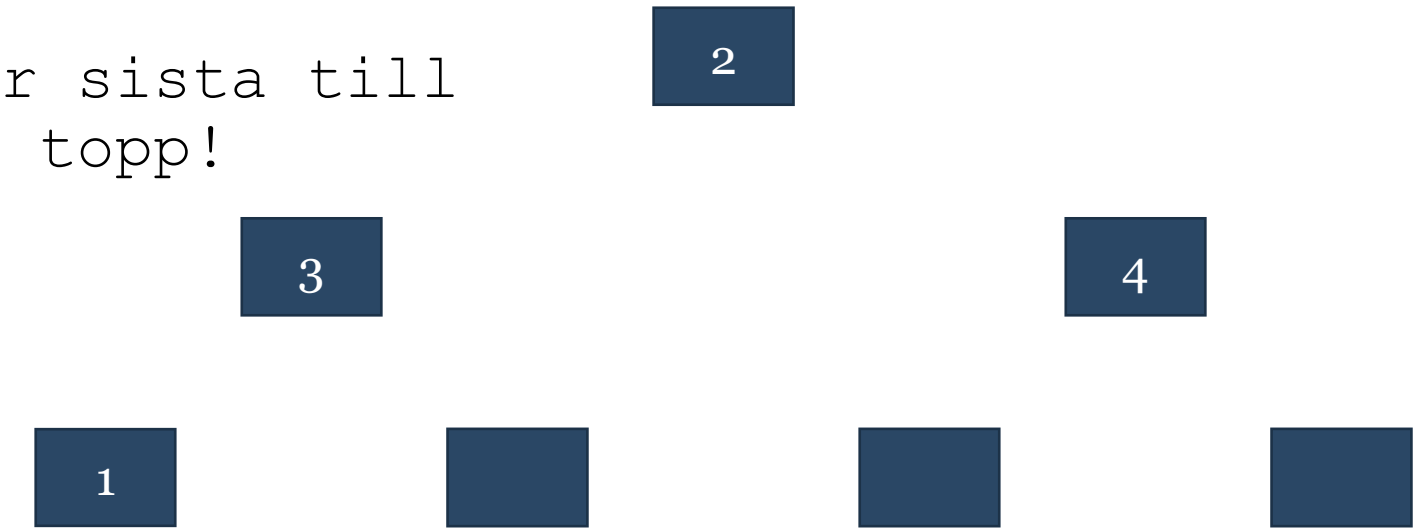


2018-03-08 UPPGIFT 7
SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

```
Utplockade element: 7 6 5

Flyttar sista till
högens topp!
```

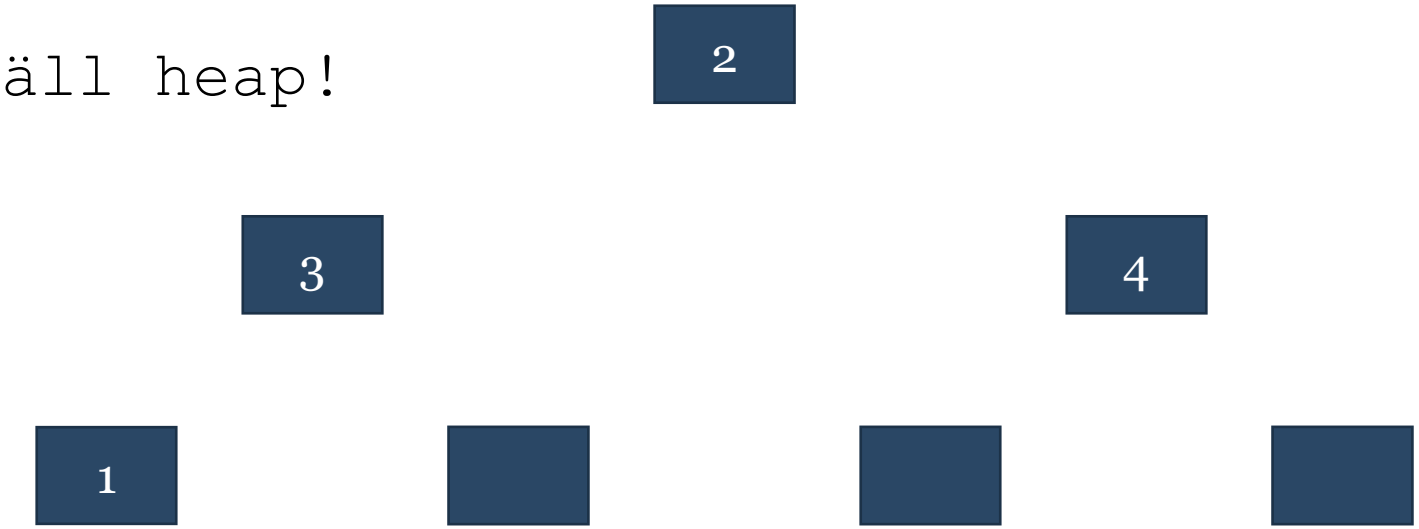


2018-03-08 UPPGIFT 7
SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

```
Utplockade element: 7 6 5

Återställ heap!
```

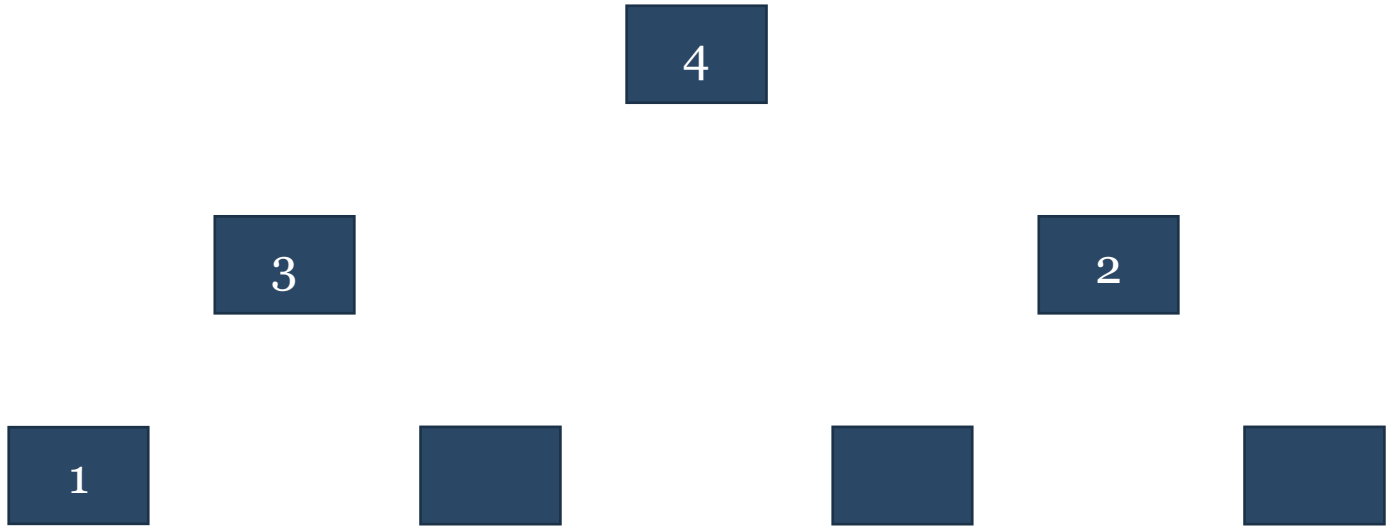


2018-03-08 UPPGIFT 7
SORTERING (MODIFIERAD)

b) Plocka ut de tre STÖRSTA elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

```
Utplockade element: 7 6 5

Klar!
```



2018-03-08 UPPGIFT 7 SORTERING (MODIFIERAD)

c) Om man vill använd en Heap för att sortera värden, vilken komplexitet får sorteringen?

Se till att vi ordnar värderna i en heap => Ordo(n log n)
Plocka ut största värdet, är alltid toppen => Ordo(1)
Återställa heapens struktur, dvs bubbla sista värdet nedåt =>
Maximalt Log n steg men detta ska göras för alla n elementen.

Alltså:
Svar: Ordo(n log n)

2018-03-08
UPPGIFT 7
SORTERING

UMEÅ UNIVERSITET

SORTERING

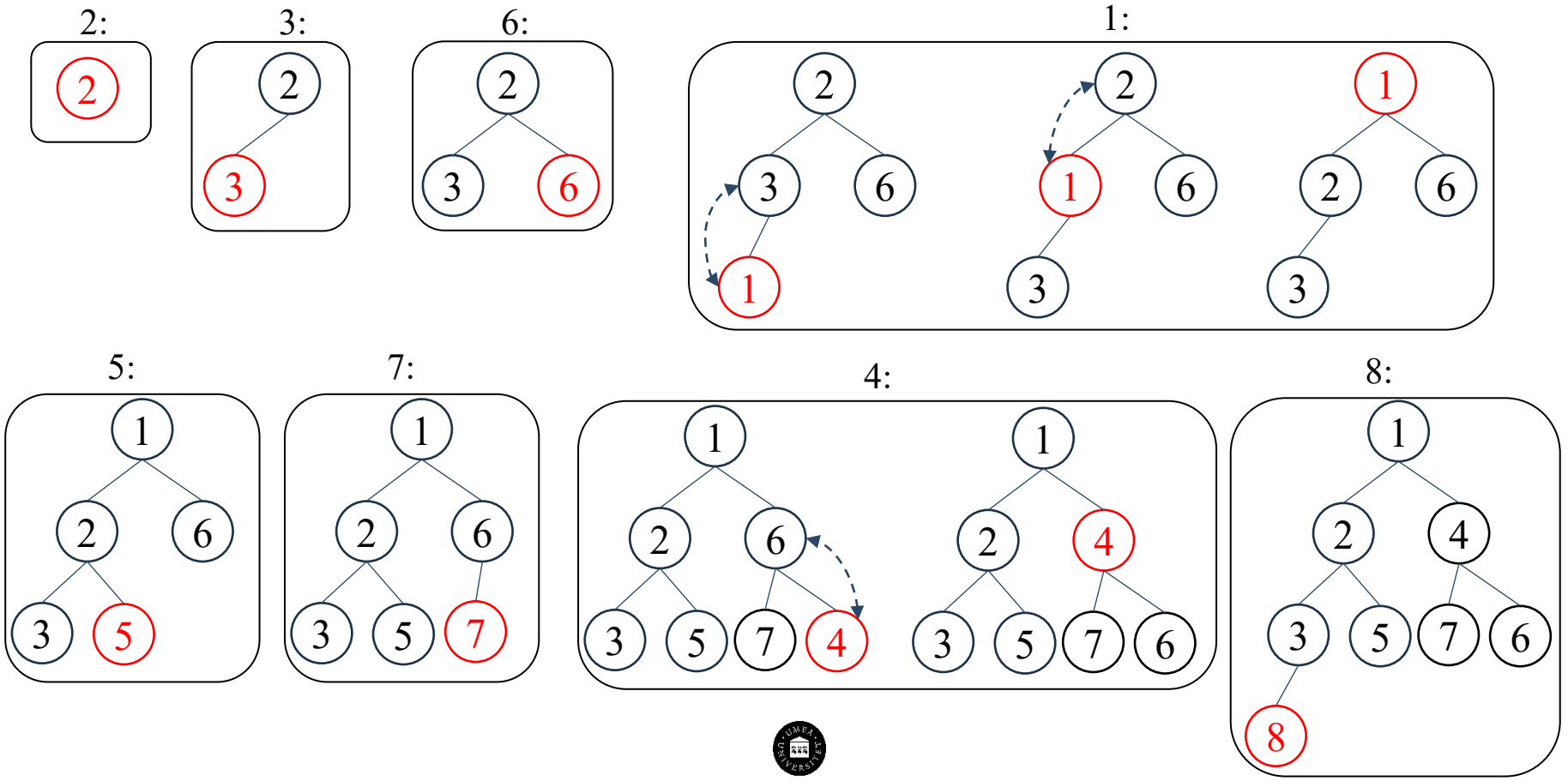
a) Visa hur en min-heap (förälderns etikett $<$ barnens etiketter) byggs upp när du sätter in värdena `2 3 6 1 5 7 4 8`. Visa trädet efter varje förändring.

b) Plocka ut de tre minsta elementen, ett i taget, och visa hur heap:en ser ut efter varje förändring.

c) Om man vill använd en Heap för att sortera värden, vilken komplexitet får sorteringen? Motivera ditt svar.

SORTERING

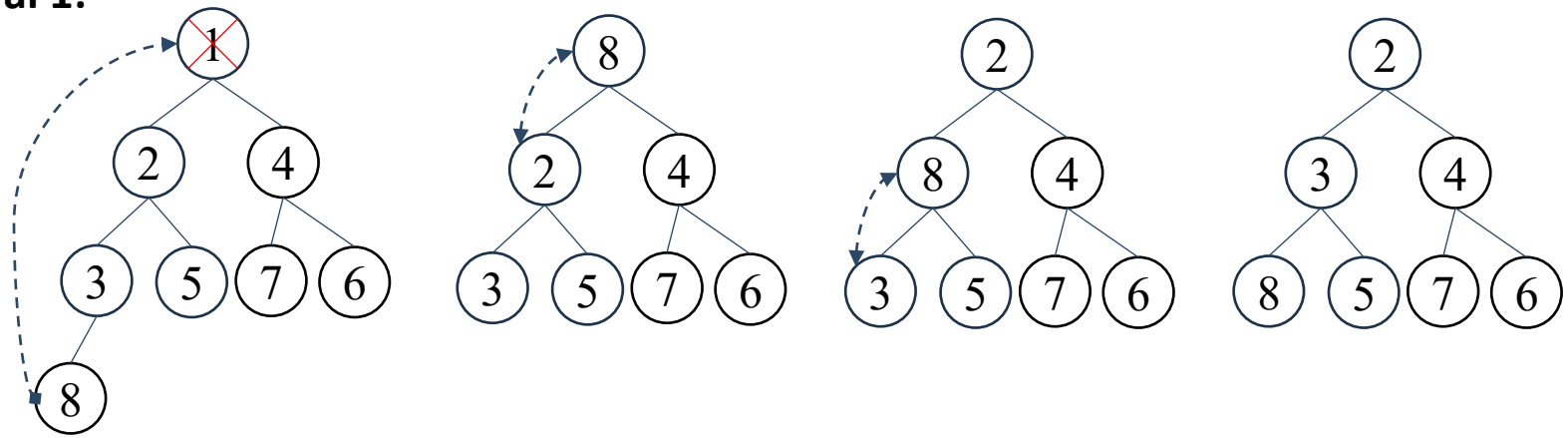
a) Sätt in 2 3 6 1 5 7 4 8 i en min-heap (<)



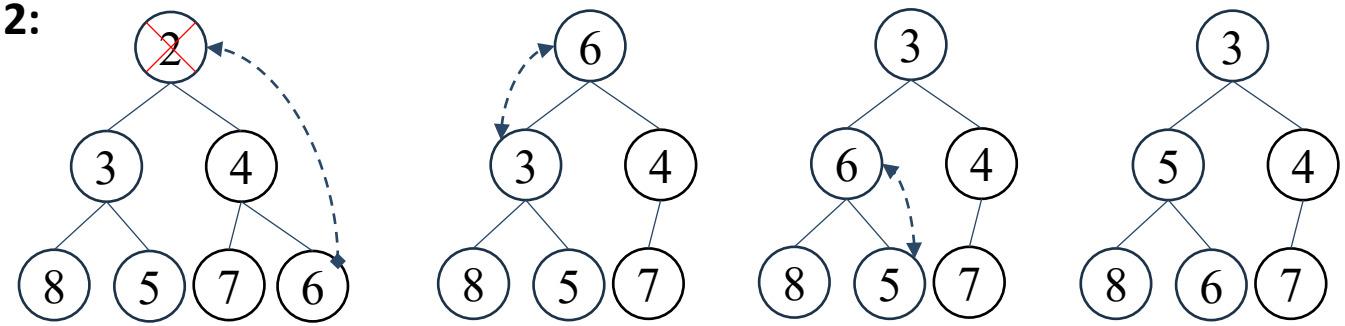
SORTERING

b) Ta ut 3 minsta talen.

Tal 1:



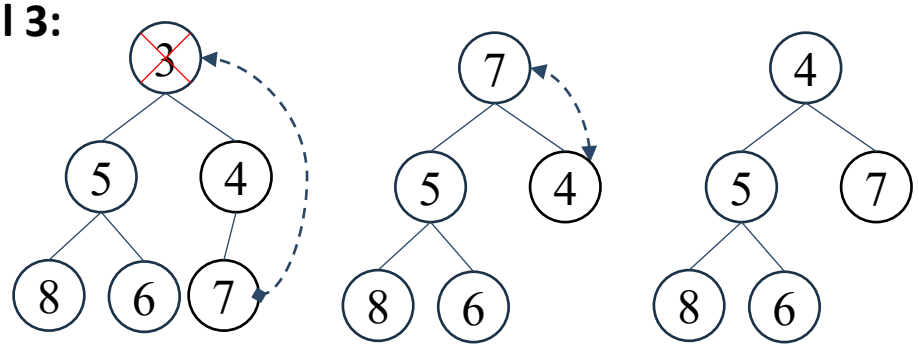
Tal 2:



SORTERING

b) Ta ut 3 minsta talen.

Tal 3:



c) Komplexitet för sorteringen:

- Varje insättning MAX log(n) byten
- Varje uttag: MAX log(n) byten
- **Totalt:** O(n log(n)) + O(n log(n)) = **O(n log(n))**

2020-10-30
FRÅGA 15
ALGORITMER, PSEUDOKOD

UMEÅ UNIVERSITET

ALGORITMER, PSEUDOKOD (6 POÄNG)

- Skriv en algoritm i pseudokod som kontrollerar om två listor är lika (dvs innehåller samma värden i samma ordning). Var noga med att använda de operationer för den abstrakta datatypen Lista som definierats under kursen.

```
Algorithm equal(l1, l2)
pos1 <- first(l1)
pos2 <- first(l2)
while pos1 <> end(l1) and pos2 <> end(l2) do
    val1 <- inspect(pos1, l1)
    val2 <- inspect(pos2, l2)
    if val1 <> val2 then
        return false
    pos1 <- next(pos1, l1)
    pos2 <- next(pos2, l2)
end
if not isempty(l1) or not isempty(l2) then
   return false
end

return true
```

2013-03-26
UPPGIFT 3
DATATYPER

UMEÅ UNIVERSITET

UPPGIFT 3 –DATATYPER (2 + 2 + 1 = 5P)

a) Förklara vad som menas med en prioritetskö.
   - En kö som inte är FIFO utan där man placeras på en plats i kön beroende på en prioritet (tänk akuten).

b) Ange någon datastruktur som kan användas för att implementera en prioritetsköså att alla operationer får maximalt tidskomplexitet O(log(n)). Motivera ditt svar.
   - Heap, förklara för varje operation i priokön hur man gör det i heapen och vilken komplexitet det blir.

c) För att traversera en graf bredden först så har man stor nytta av en annan datatyp. Vilken?
   - En vanlig kö.

2013-03-26
UPPGIFT 7
TRÄD

UMEÅ UNIVERSITET

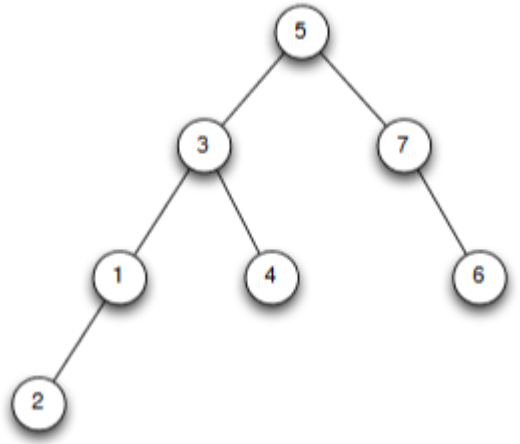
UPPGIFT 7 –TRÄD(2+2+2=6P)

a) Stoppa in noderna med elementvärdena 3,5,6,7,1,4,2 och 0 i ett binärt sökträd. Du skall stoppa in noderna i exakt denna ordning och visa hur trädet ser ut efter varje ny insatt nod.

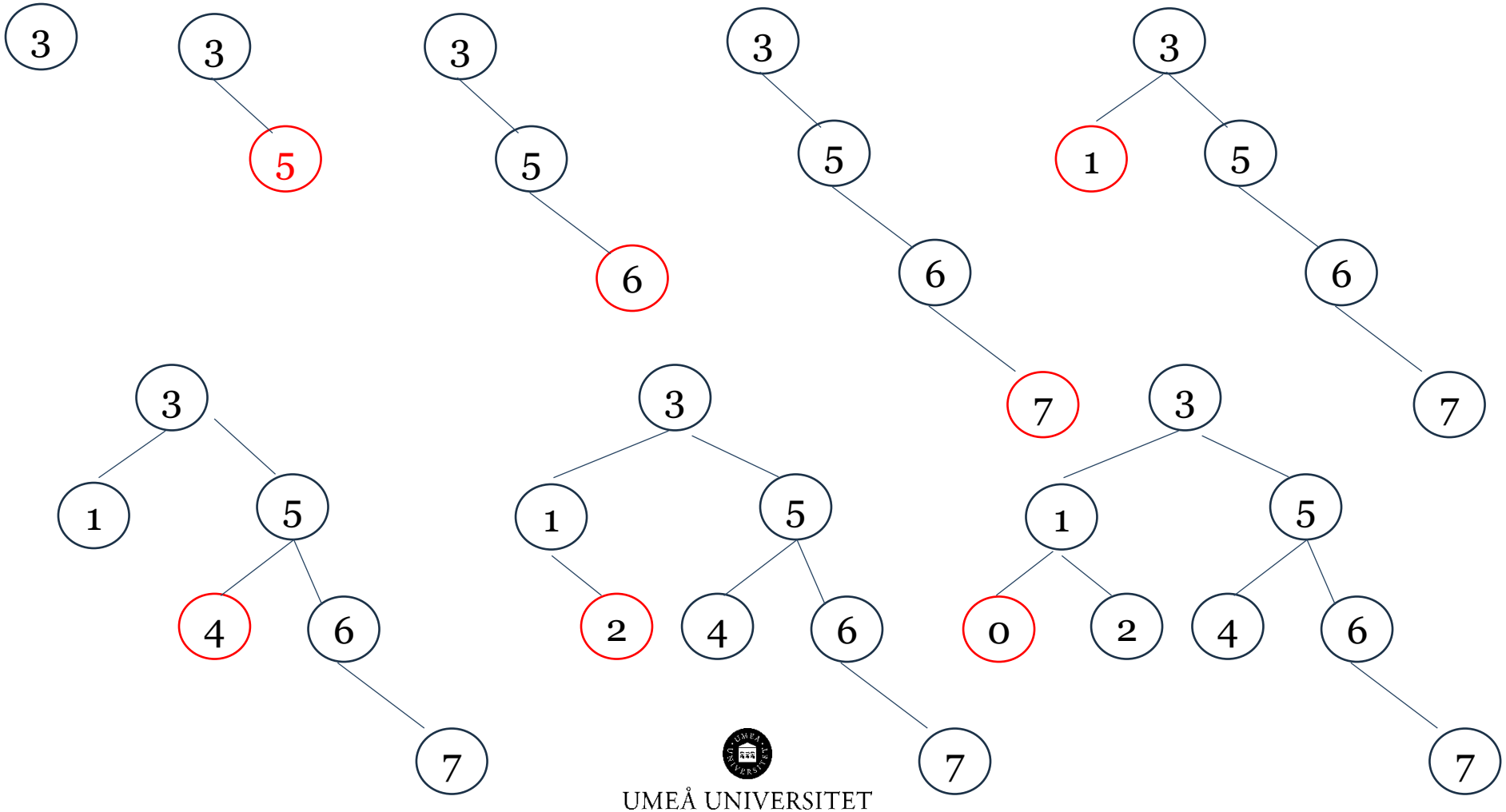
b) Utför en sökning efter talet 2, visa hur sökningen går till och tala om hur många noder som du besökt innan du fann talet 2.

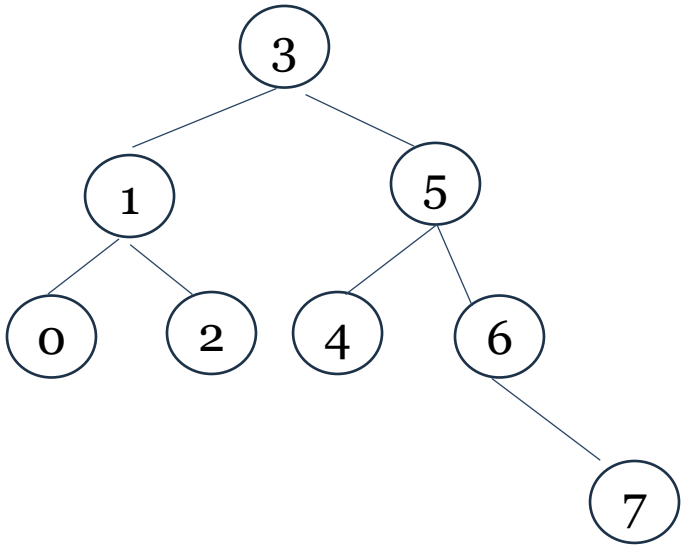
c) Givet trädet nedan ange ordningen som noderna besöks givet de olika traverseringsstrategierna nedan

   i. Djupet först inorder
   ii. Djupet först postorder



a) Stoppa in noderna med elementvärdena 3,5,6,7,1,4,2och 0 i ett binärt sökträd. Du skall stoppa in noderna i exakt denna ordning och visa hur trädet ser ut efter varje ny insatt nod.



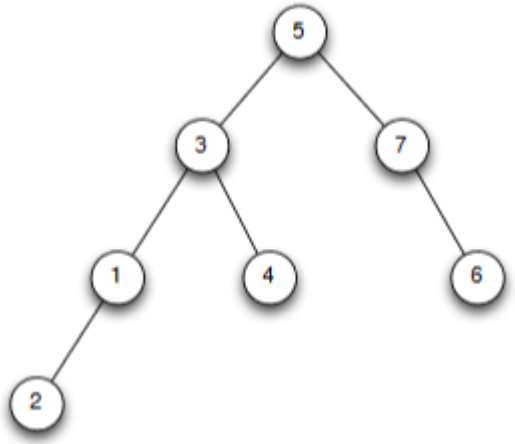


a) Utför en sökning efter talet 2, visa hur sökningen går till och tala om hur många noder som du besökt innan du fann talet 2.

Börja i roten, 2<3 sök i vänster delträd. Besök 1, 2>1, sök i höger delträd. Besök 2 Funnen!

Givet trädet nedan ange ordningen som noderna besöks givet de olika traverseringsstrate till höger.

i. Djupet först inorder 2, 1, 3, 4, 5, 7, 6
ii. Djupet först postorder 2, 1, 4, 3, 6, 7, 5

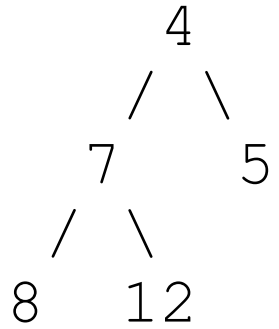


2020-10-30
UPPGIFT 1
TRÄD

UMEÅ UNIVERSITET

TRÄD (2 POÄNG)

- Vilken traverseringsstrategi har använts om noderna i trädet (se bild nedan) besöks i ordningarna enligt nedan:
- Besöksordning: 4 7 5 8 12
- Besöksordning: 8 7 12 4 5



a) Bredden först
b) Djupet först - inorder

TENTA 2013-03-26
UPPGIFT 6
KOMPLEXITETSANALYS

UMEÅ UNIVERSITET

2013-03-26 UPPGIFT 6 – KOMPLEXITETSANALYS 2+2=4P

- Antag att variabeln m är en tredimensionell matris med NxNx7 flyttal: `m[n][n][7]`

a) Vi vill summera talen i matrisen m, med den här algoritmen:

```
summa <- 0;
for i <- 0 to N-1 do
    for j <- 0 to N-1 do
        for k <-0 to 6 do
            summa <- m[i][j][k]+ summa
        end
    end
end
```

Vilken tidskomplexitet har algoritmen?

b) Vi provkör programmet med N= 5000. och ser att summeringen av matrisen *m* tar 5 sekunder. Hur lång tid kan vi räkna med att den summeringen tar om N= 10000?

2013-03-26 UPPGIFT 6 – KOMPLEXITETSANALYS 2+2=4P

a)
```
summa <- 0;
for i <- 0 to N-1 do
    for j <- 0 to N-1 do
        for k <-0 to 6 do
            summa <- m[i][j][k]+ summa
        end
    end
end
```
Yttersta loopen kör N ggr, nästa kör N ggr, sista kör 7 ggr = N*N*7 det blir $O(N^2)$

b) N= 5000 tar 5 sekunder. Hur lång tar N= 10000?

Vi noterar att N=10000 = 2*5000 så hur mycket tid tar en kvadratisk komplexitet på dubbelt så stora data?

$(2N)^2 = 4* N^2$

så det borde ta 4*5 = 20 sekunder.

TENTA 2013-03-26
UPPGIFT 4
HASHTABELL

UMEÅ UNIVERSITET

2013-03-26 UPPGIFT 4
HASHTABELL

1p

a) Vad menas med linjär teknik för hantering av kollisioner i en sluten hashtabell?

- Om man får en kollision på positionen i, så prövar man i+1, i+2, i+3, etc tills en ledig position påträffas.

1p

b) Vilka problem kan uppstå när man använder linjär teknik?

- Ett stort problem kan bli den sk. Klustringen som kan uppkomma, dvs att delar av hashtabellen fylls igen och kan leda till en lång kedja med kollisioner innan tom plats påträffas och söktiderna blir mer ordo(n) än den ordo(1) som man strävar efter.

2013-03-26 UPPGIFT 4
HASHTABELL

2p

c) Antag att man har satt in element vars nycklar är heltal i en hashtabell som representerats av en vektor med storlek 11 och hash-funktion h(x)=x % 11. I denna har man satt in 4 element och vektorn ser då ut på följande sätt:

0	1	2	3	4	5	6	7	8	9	10
-	-	13	36	15	5	-	-	-	-	-

- Visa hur fältet ser ut efter insättning av värdena 26 och 12 om linjär teknik för att hantera kollisioner används.

2013-03-26 UPPGIFT 4 HASHTABELL

- i = 26 % 11 = 4 dvs vi börjar på position 4 och provar oss fram:

0	1	2	3	4	5	6	7	8	9	10
-	-	13	36	15	5	-	-	-	-	-

- i +1 = 5

0	1	2	3	4	5	6	7	8	9	10
-	-	13	36	15	5	-	-	-	-	-

- i +2 = 6

0	1	2	3	4	5	6	7	8	9	10
-	-	13	36	15	5	26	-	-	-	-

2013-03-26 UPPGIFT 4
HASHTABELL

i = 12 % 11 = 1 dvs vi börjar på position 1 och provar oss fram:

0	1	2	3	4	5	6	7	8	9	10
-	12	13	36	15	5	26	-	-	-	-

Här hittar vi en ledig plats direkt!

2013-03-26
UPPGIFT 2
SORTERING

UMEÅ UNIVERSITET

2013-03-26 UPPGIFT 2
SORTERING

2p

a) Beskriv kortfattat sorteringsalgoritmerna insertionsort och quicksort med ord eller pseudokod. (Skriv bara några få meningar eller ”pseudokodsrader” per algoritm)

2013-03-26 UPPGIFT 2
SORTERING

I: Insertionsort

- Utgår ifrån en tom "mängd" (lista) sorterade element
- För varje element vi vill sortera
  - Gå igenom den sorterade mängden ett element i taget
    - Om vi hittar ett större element
      - Stoppa in det nya elementet innan detta element
    - Annars
      - Undersök nästa element i den sorterade mängden
- Upprepa tills alla element vi vill sortera är på rätt plats

2013-03-26 UPPGIFT 2
SORTERING

II: Quicksort

- Utgå ifrån ett fält med osorterade värden
  - Om fältet innehåller ett element, avbryt
- Annars
  - Välj ut ett pivot-värde
  - Gå igenom detta fält från bägge hållen med var sitt index
    - Om elementet är på fel sida om detta pivot-element så ska det flyttas till rätt sida (dvs vänster sida < Pivot, höger sida > Pivot)
    - Annars flytta fram index en position
  - När dessa index möts
    - Rekursivt anropa denna funktion för respektive osorterade del-lista.

2013-03-26 UPPGIFT 2
SORTERING

2p

b) Vad har respektive algoritm för medelfallskomplexitet?

I: Insertionsort

I denna algoritm så har vi n osorterade element som vi vill sätta in i en sorterade mängd som som i slutändan kommer innehålla alla dessa n elementen.
Även om vi inte behöver göra alla n^2 operationer för detta så utan i snitt mindre än hälften, så hamnar dessa under de konstanter som Ordo-notationen inte bryr sig om

Svar: Komplexiteten blir Ordo(n^2)

2013-03-26 UPPGIFT 2
SORTERING

2p

b) Vad har respektive algoritm för medelfallskomplexitet?

II: Quicksort

För varje nivå kommer vi gå igenom n element, men eftersom vi rekursivt delar listan i två varje gång så kommer detta i normala fall bara behöver göras $\text{Log}_2(n)$ gånger dvs "nivåer", alltså blir det totalt n * Log n operationer.

Svar: Komplexiteten blir Ordo(nLog(n))

2018-03-08
UPPGIFT 6
LISTOR

UMEÅ UNIVERSITET

UPPGIFT 6 — LISTOR (3+4=7P)

Rita en riktad lista som någonstans på mitten i direkt följd har tre element med värdena Stefan, Anna och Erik. Rita därefter figurer som steg för steg visar hur man:

a) sätter in ett nytt element med värdet Siv på platsen efter p, där p refererar till elementet med värdet Anna,

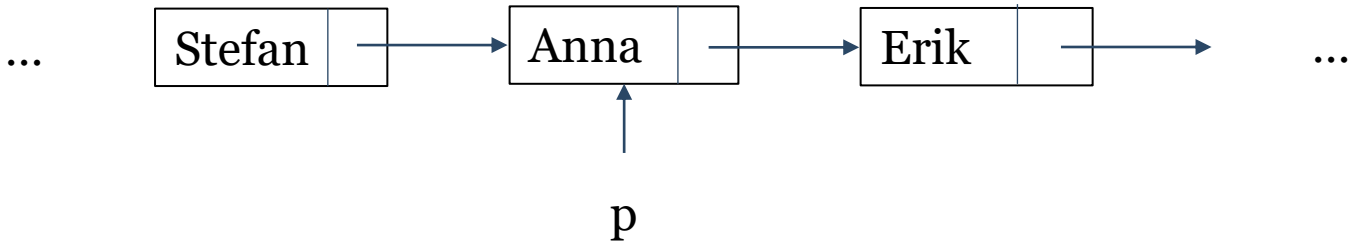
b) ur den resulterande listan med bl.a. Stefan, Anna, Siv och Erik i direkt följd, tar bort elementet med värdet Anna. Du får utgå från att p före borttagandet fortfarande refererar till elementet med värdet Anna.

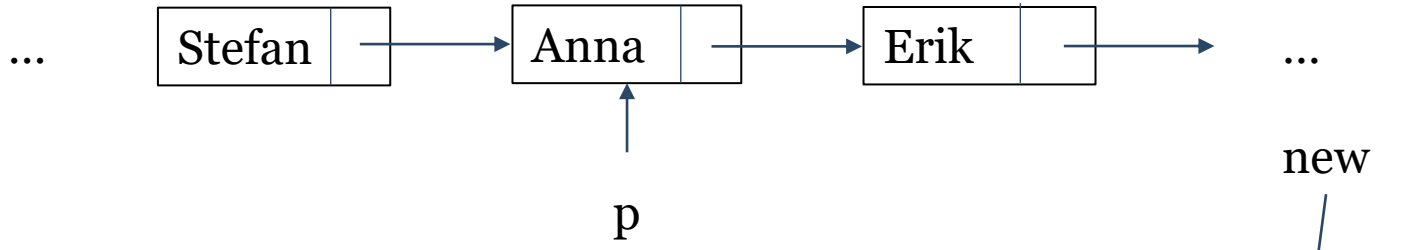
Skriv dessutom pseudokod som visar vad du gör och i vilken ordning. Var speciellt noggrann med att visa hur du undviker att tappa bort någon referens till något element. Redogör för ev. antaganden och beteckningar som gäller för din lösning.

OBS! Visar först för riktad lista men har även lösning för dubbellänkad.

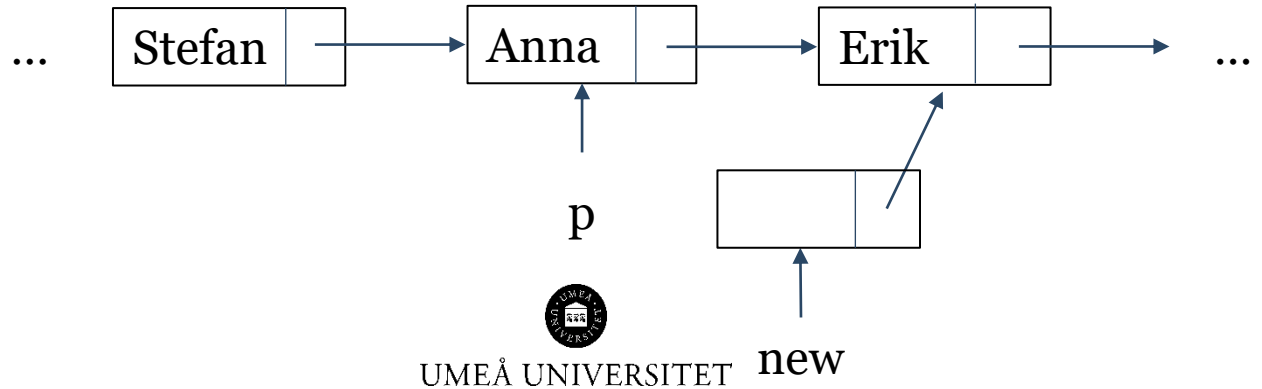
2018-03-08 UPPGIFT 6 LISTOR

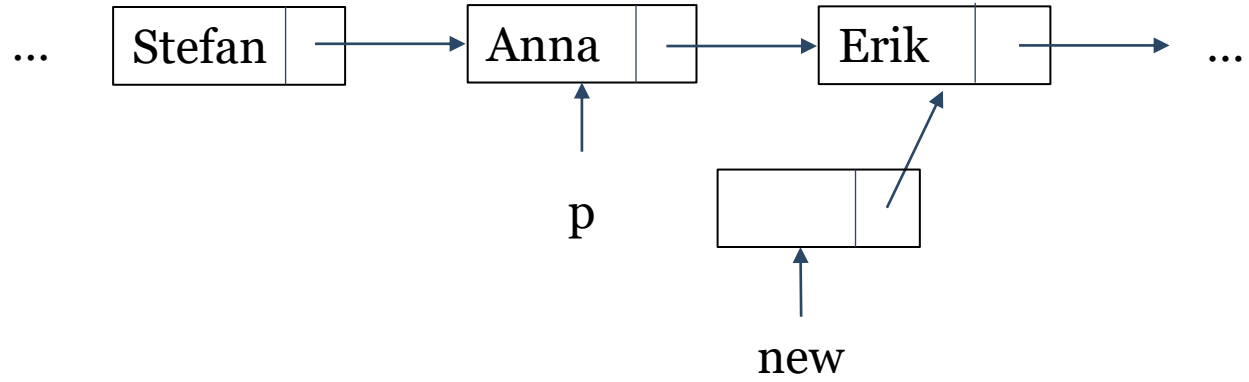
- Tanke: Vi ska sätta in före elementet på position p. Eftersom listan är riktad har vi svårt finna positionen för p så vi gör ett litet trick. Vi skapar ett nytt element och placerar det efter p och kopierar sedan värdet från p till det och det nya värdet in på p:s position.





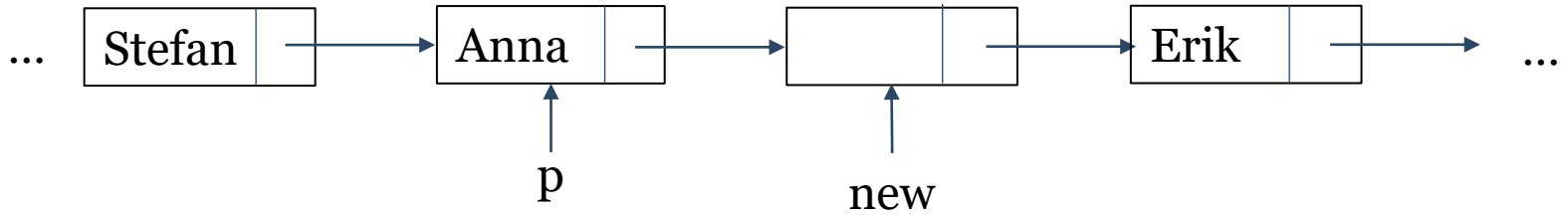
1. Skapa ett nytt element och låt new referera till det.
   ```
   new <- create() //threeCell_create
   ```
2. Placera det nya element efter elementet som p refererar till genom att:
   1. Låt new:s next referera till elementet med värde Erik
      ```
      new.next <- p.next
      // kan också skrivas
      new <- setNextLink(next(p), new)
      // eller (om L är listan vi arbetar i)
      new->next <- next(p, L)
      ```





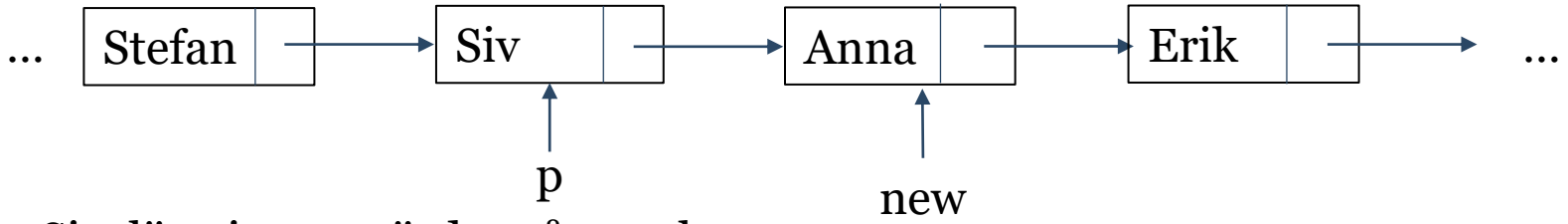
2. Låt p:s next referera till new-elementet

```
p.next <- new
```



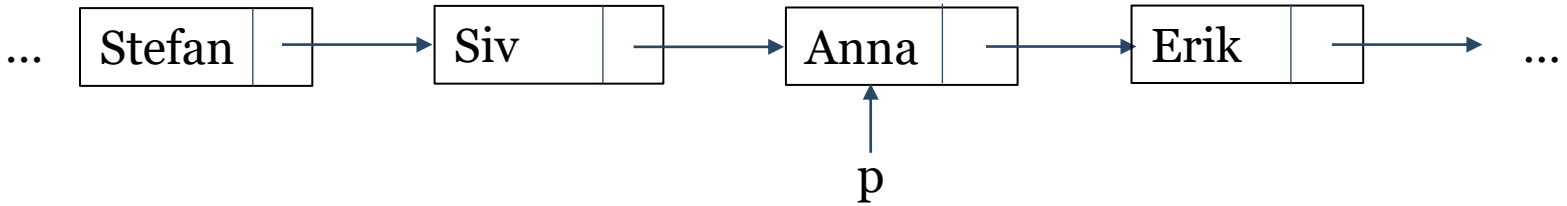
3. Kopiera p:s värde till new.

```
new.value <- p.value
```



4. Sist lägg in nya värdet på p:s plats

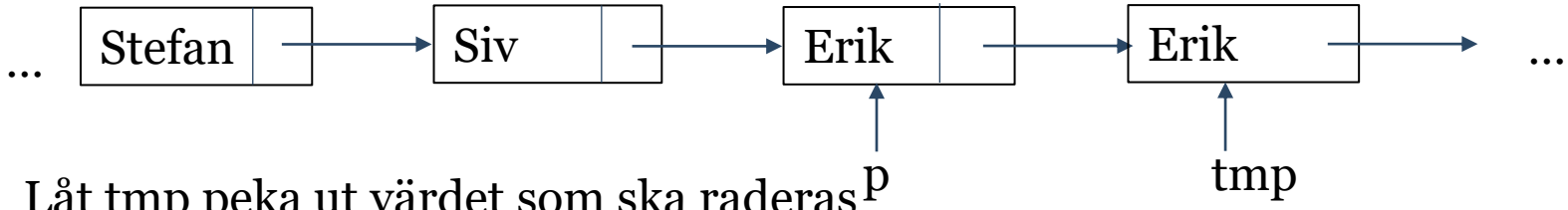
```
p.value <- ”Siv”
```



b) Nu skulle vi ta bort elementet som p referar till. Efteråt ska p peka på Erik. Vi Gör samma trick igen. Det är enklare ta bort Erik-elementet så vi kopierar värdet från det till p och tar sen bort elementet efter.

1. Låt p:s värde bli värdet på elementet efter.

```
p.value <- p.next.value
```

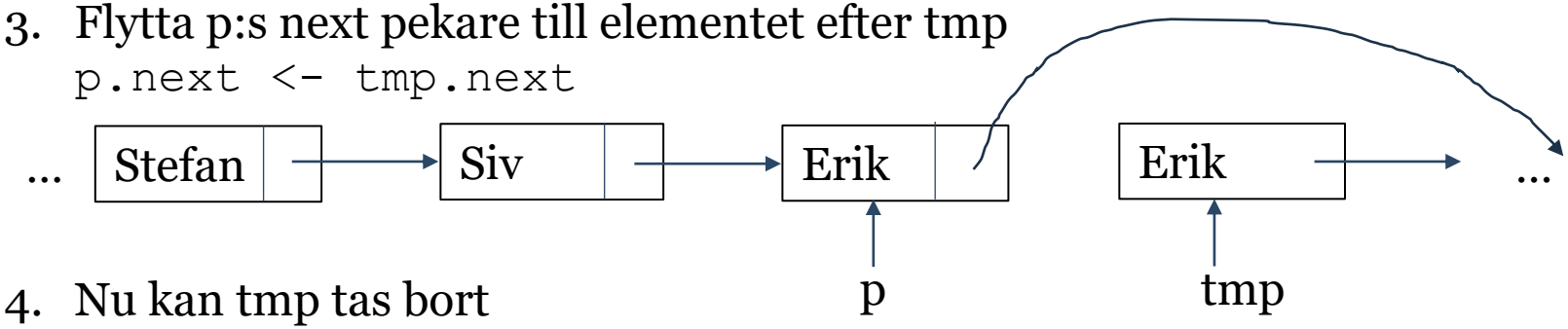


2. Låt tmp peka ut värdet som ska raderas

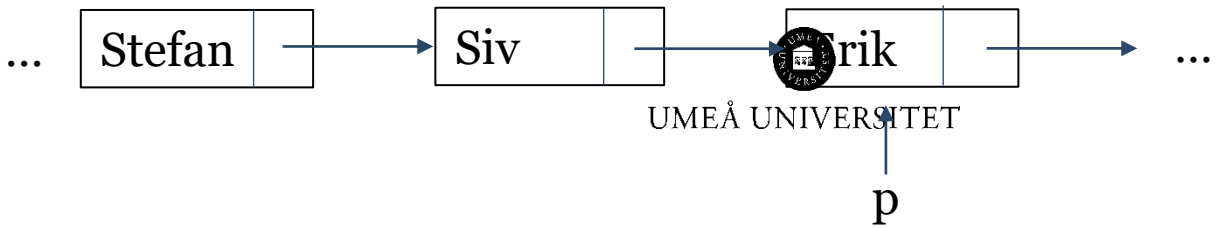
```
tmp <- p.next
```

3. Flytta p:s next pekare till elementet efter tmp

```
p.next <- tmp.next
```

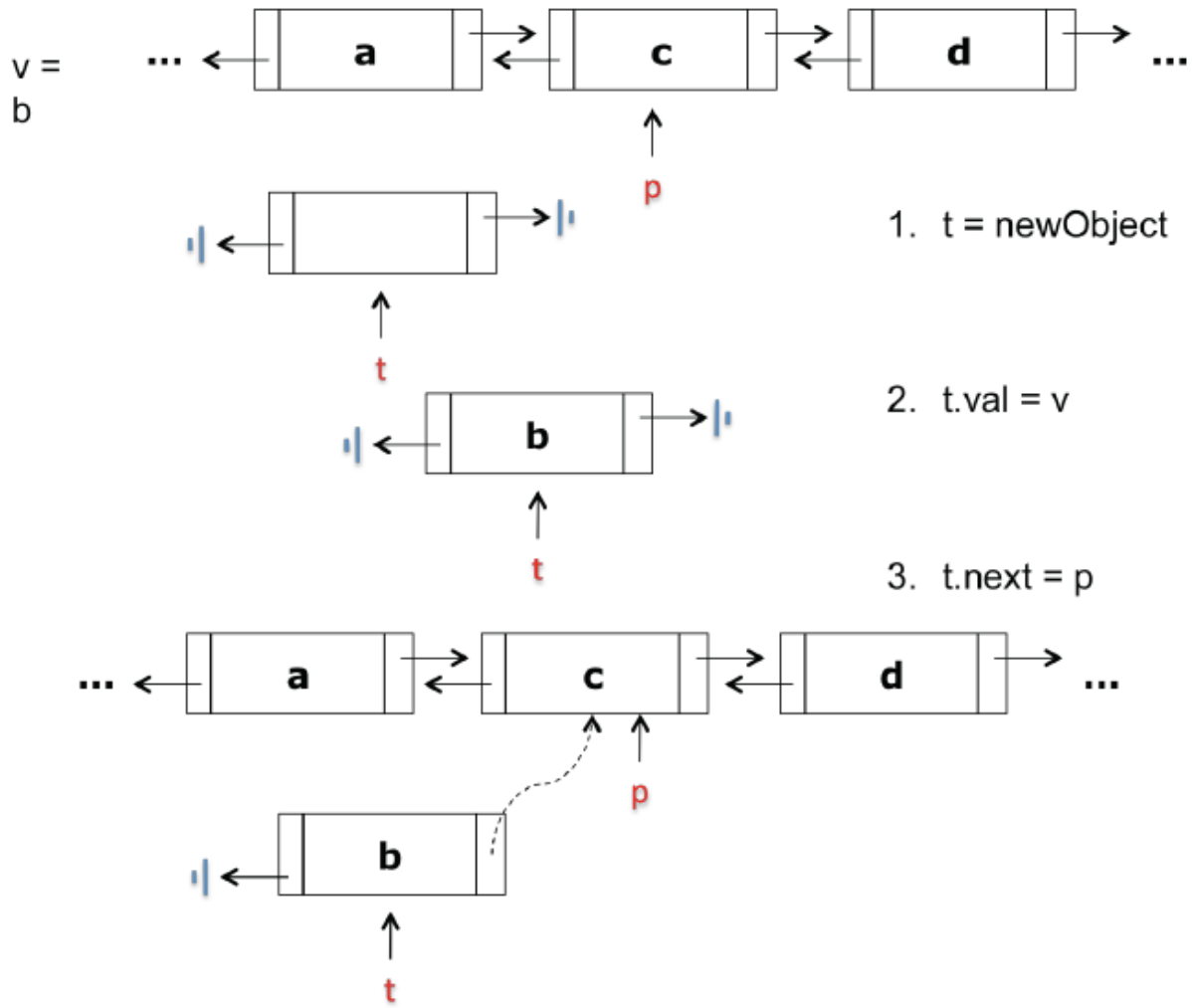


4. Nu kan tmp tas bort



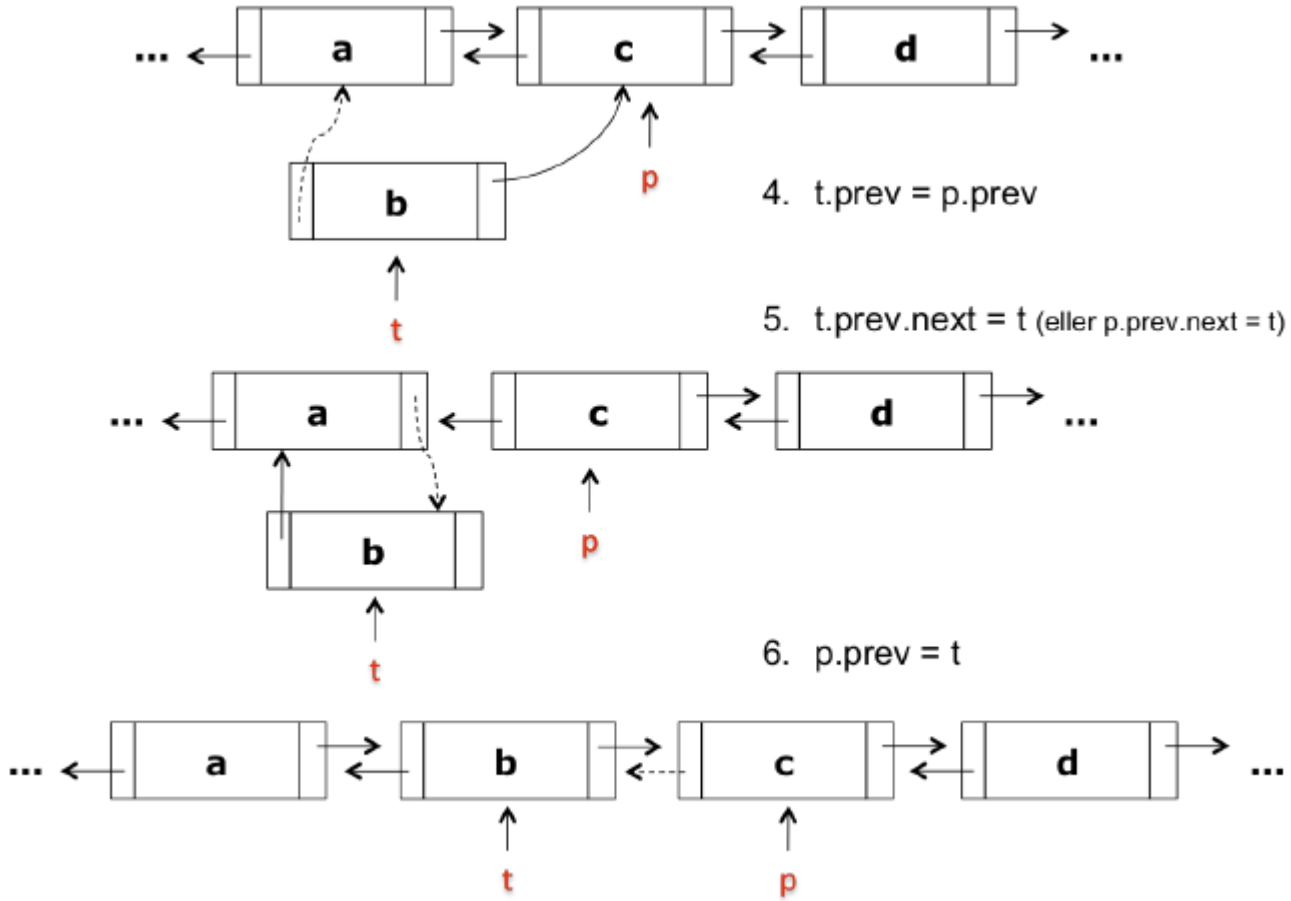
2018-03-08 UPPGIFT 6
DUBBELLÄNKADE LISTOR

a) Sätt in element *b* på platsen före *p*

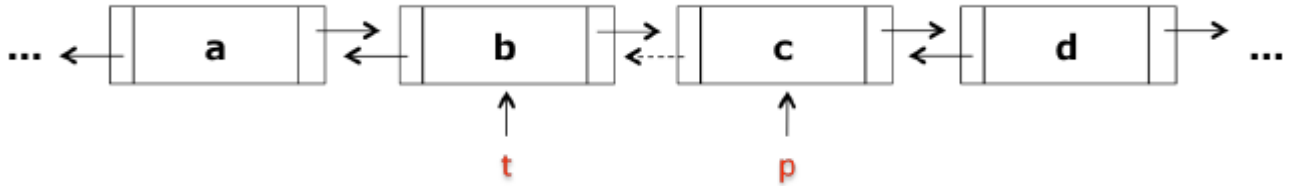


2018-03-08 UPPGIFT 6
DUBBELLÄNKADE LISTOR

a) Sätt in element *b* på platsen före *p*



2018-03-08 UPPGIFT 6
DUBBELLÄNKADE LISTOR



b) Ta bort element *c* ur listan

- Gör samma sak fast tvärtom ☺