

Tidskomplexitet

Konvertera mellan tider o.s.v.

Sorteringsalgoritmer

Algoritm	Typ	Best Case	Average Case	Worst Case	Övrigt
Merge sort	Sortering	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Implementeras med array
Insertion sort	Sortering	$O(n)$	$O(n^2)$	$O(n^2)$	Implementeras med array
Heap sort	Sortering	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Implementeras med array
Quick sort	Sortering	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	
Selection sort	Sortering	$O(n^2)$	$O(n^2)$	$O(n^2)$	Implementeras med array
Radix sort	Sortering			$O(wn)$	w är antalet siffror (total, ental o.s.v.)

Sökalgoritmer

Algoritm	Typ	Best Case	Average Case	Worst Case	Övrigt
Binary search	Sökning	$O(1)$	$O(\log n)$	$O(\log n)$	Implementeras med array. Måste vara sorterad
Linear search	Sökning	$O(1)$	$O(n)$	$O(n)$	Implementeras med array.
Breadth-first search	Sökning			$O(\text{antal noder} + \text{antal kopplingar})$	Implementeras med en graf.
Depth-first search	Sökning			$O(\text{antal noder} + \text{antal kopplingar})$	Implementeras med en graf.

Binär sökning

Utgå från en sorterad lista.

1. Välj det mellersta elementet. Jämför med det sökta värdet. Lika? Hittad.
2. Mindre? Gå till 1) med alla element vänster om det förra mellersta elementet.
3. Större? Gå till 1) med alla element höger om det förra mellersta elementet.
4. Upprepa till det "mellersta" elementet är längst till vänster eller höger

Pseudokod

Set L to 0 and R to $n - 1$. If $L > R$, the search terminates as unsuccessful. Set m (the position of the middle element) to the floor (the largest previous integer) of $(L + R)/2$. If $A_m < T$, set L to $m + 1$ and go to step 2. If $A_m > T$, set R to $m - 1$ and go to step 2. Now $A_m = T$, the search is done; return m.

Köer

FIFO.

Prioritetskö

Sorterad FIFO. Läggts till med linjär tid. Läggts in i ordning.

Träd

Balans

Ett träd är balanserat om höjdskillnaden är ± 1 .

Manuell balansering

Skriv talen i storleksordning. Välj medianen som rot. Därefter skrivs det första talet vänster om medianen som barn till medianen. Det andra skrivs som barn till det barnet o.s.v. Samma sak sker för höger sida.

Order

Pre order

Ringa in trädet. Följ vägen som ritas. När pennan är på **vänster** sida av ett tal skrivs talet.

Level order

Skriv var nivå för sig, vänster till höger.

Post order

Ringa in trädet. Följ vägen som ritas. När pennan är på **höger** sida av ett tal skrivs talet.

In order

Ringa in trädet. Följ vägen som ritas. När pennan är **under** ett tal skrivs talet.

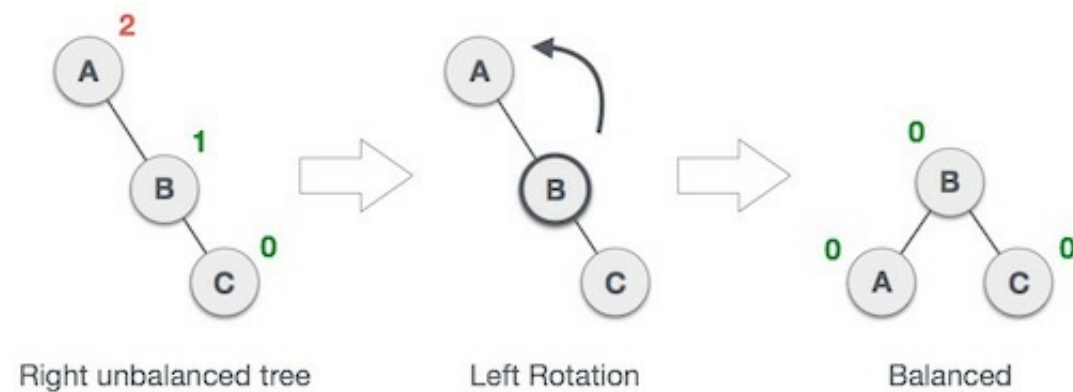
Trädtyper

BST - binärt sökträd

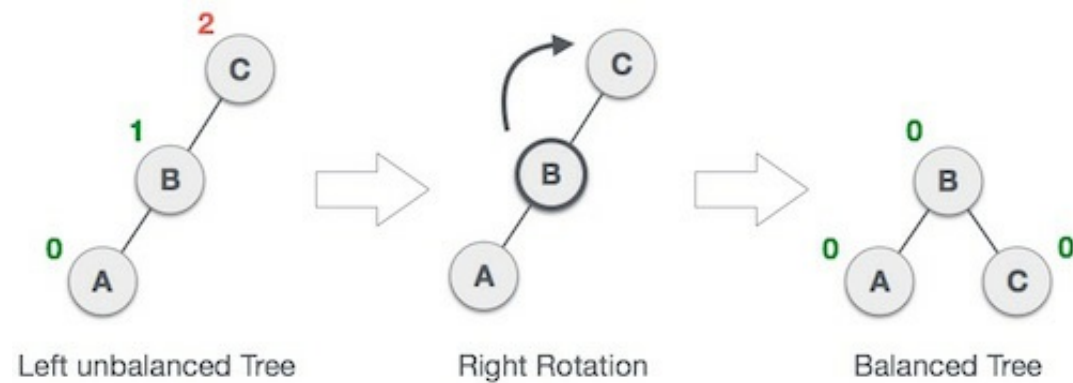
AVL - självbalanserande BST

$\text{BalanceFactor} = \text{height}(\text{left-sutree}) - \text{height}(\text{right-sutree})$ För att balansera sig själv kan trädet göra följande:

Vänster-rotation



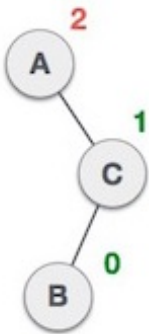
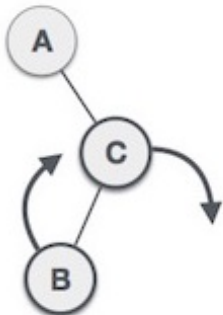
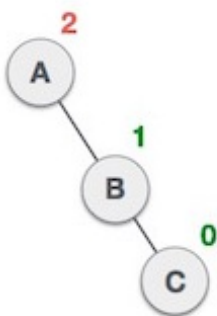
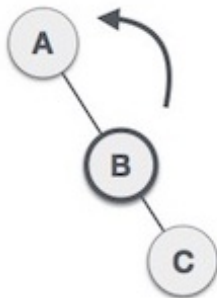
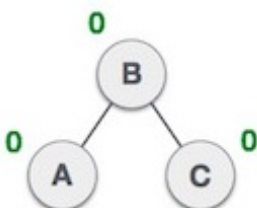
Höger-rotation



Vänster-höger-rotation

State	Action
	<p>A node has been inserted into the right subtree of the left subtree. This makes C an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.</p>
	<p>We first perform the left rotation on the left subtree of C. This makes A, the left subtree of B.</p>
	<p>Node C is still unbalanced, however now, it is because of the left-subtree of the left-subtree.</p>
	<p>We shall now right-rotate the tree, making B the new root node of this subtree. C now becomes the right subtree of its own left subtree.</p>
	<p>The tree is now balanced.</p>

Höger-vänster-rotation

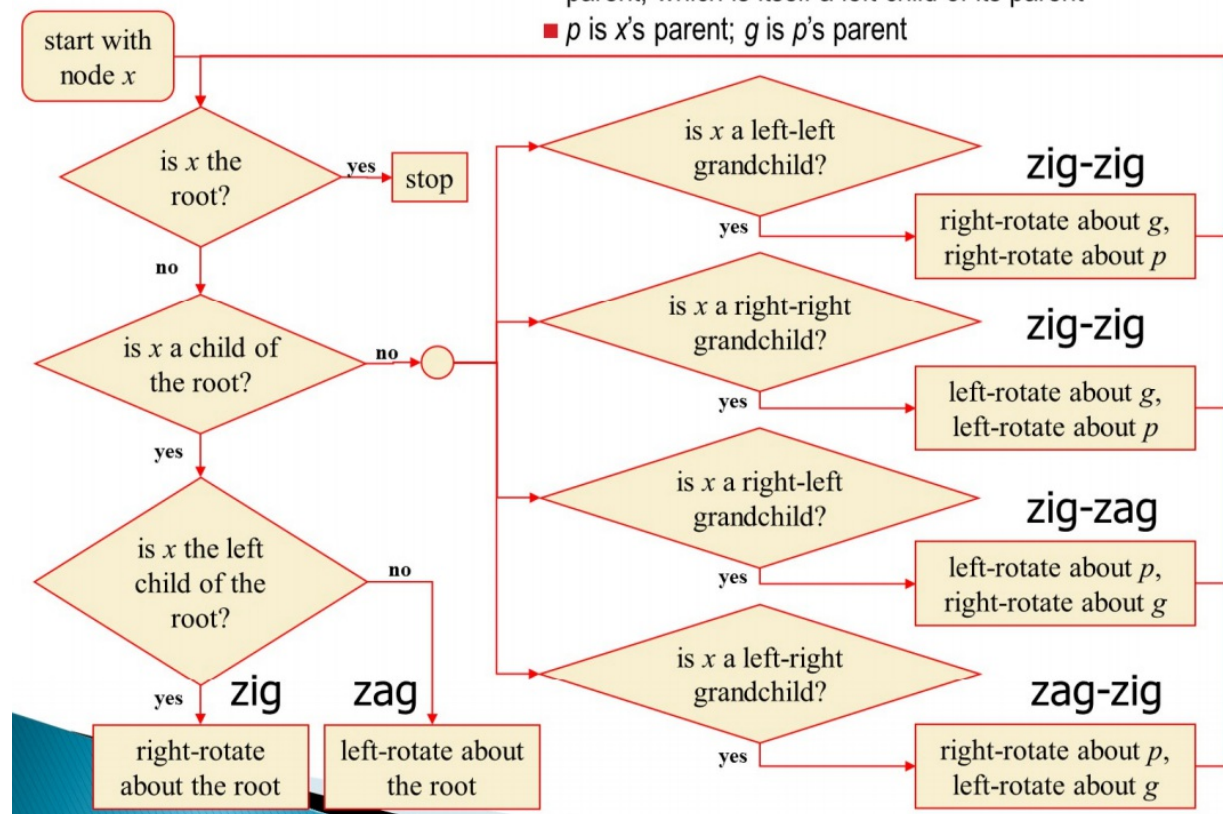
State	Action
	<p>A node has been inserted into the left subtree of the right subtree. This makes A, an unbalanced node with balance factor 2.</p>
	<p>First, we perform the right rotation along C node, making C the right subtree of its own left subtree B. Now, B becomes the right subtree of A.</p>
	<p>Node A is still unbalanced because of the right subtree of its right subtree and requires a left rotation.</p>
	<p>A left rotation is performed by making B the new root node of the subtree. A becomes the left subtree of its right subtree B.</p>
	<p>The tree is now balanced.</p>

Tidskomplexitet

Konstant tid - roterar alltid tre element

Splay träd

Splaying:



Tidskomplexitet

Konstant tid - roterar alltid tre element

Max-heap

Finns även som min-heap vilket är tvärt om.

Insert

Lägg till elementet i den nedersta nivån av heapen. Jämför elementet med dess förälder. Är föräldern större än barnet, stoppa. Är föräldern mindre än barnet, byt plats på elementen och jämför med nästa förälder.

Delete / extract root

Ersätt roten med det sista elementet på den sista nivån av trädet. Jämför den nya roten med dess barn. Är roten större än bägge barnen, stoppa. Om inte, byt plats på roten med ett av barnen och jämför den nya roten med dess barn.

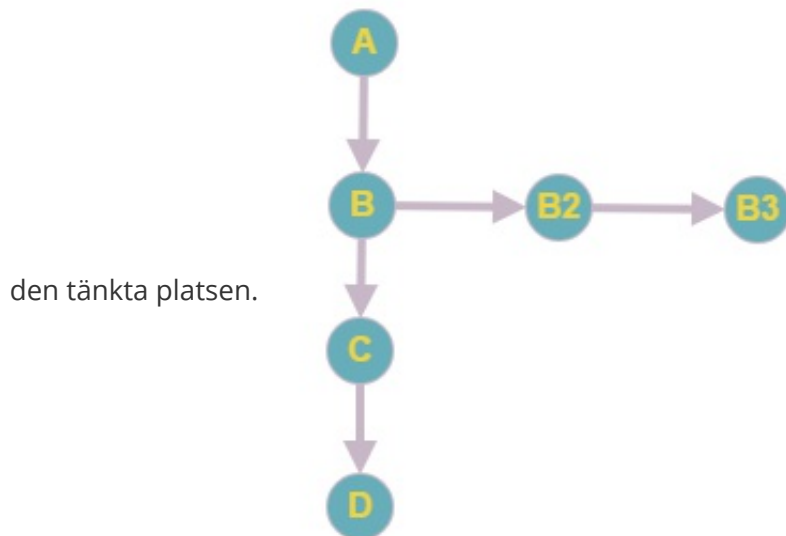
Bygg en max-heap:

1. Lägg in elementen i ett träd. Det första elementet blir en rot, det tredje och andra blir barn o.s.v. Vänster till höger.
2. Jämför den sista föräldern med dess barn och byt som nödvändigt.
3. Jämför den näst sista o.s.v. Nivå för nivå till dess att roten är kontrollerad med dess barn.

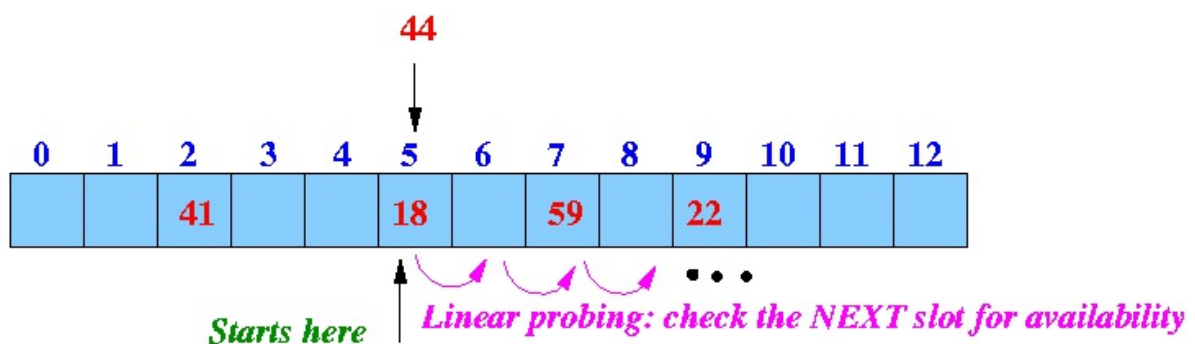
Hash table

Länkning i kedjor / seperate chaining

Om en kolission uppstår läggs helt enkelt det kolliderande elementet till till en länkad lista på



Linear probing (linjär probing)



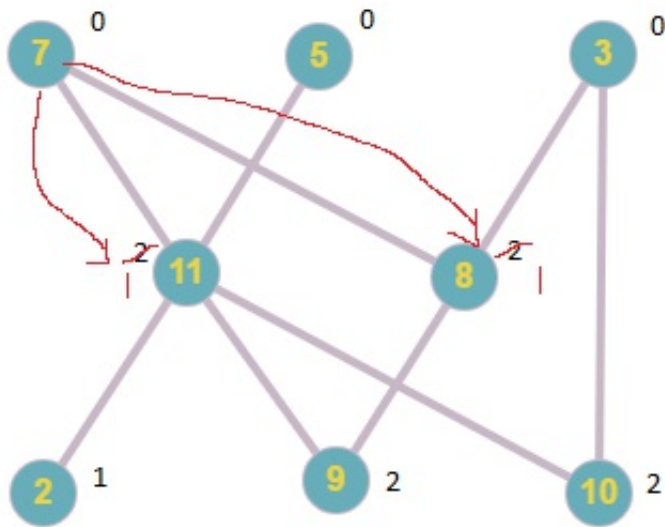
Skriv mer här

Grafer

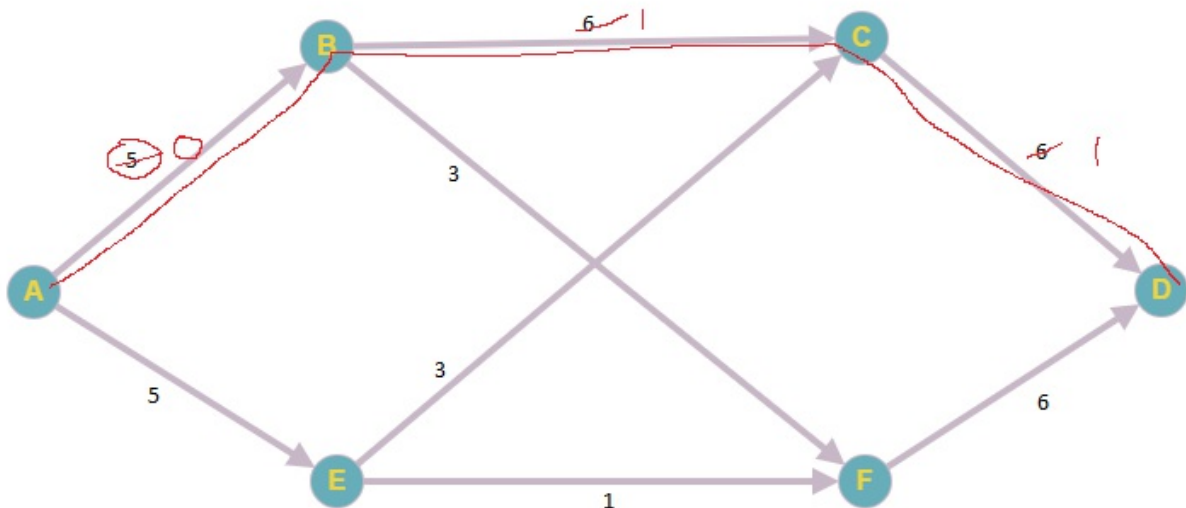
Topologisk sortering

Notera att detta enbart är möjligt om det inte finns någon cykel!

1. För alla noder, kalkylera "in degree" (antalet "inmatningar")
2. Placera alla noder som har en "in degree" av noll i en stacken
3. För alla noder i stacken: Subtrahera 1 från alla nodens barn. Om noden nu har 0 som "in degree" placera i stacken
4. Repetera tills alla element i stacken gått genom. Svaret är nu elementen i stacken.



Maximal flow



För att beräkna maximalt flöde görs följande: Välj en stig från start till slut. På denna stigen väljs den del som har lägst värde. Subtrahera alla antra tal på stigen med detta värde. Välj en ny stig. Notera att det nu är de nya värdena som måste följas. När inget mer sätt kvarstår att trycka flödet beräknas maximalt flöde genom att addera alla ursprungliga tal från start-noden och subtrahera summan med de nya talen vid samma nod.

BFS

Lägg till startnoden till en stack. Markera noden som besökt. Lägg till alla barn till startnoden till stacken. För varje element i stacken: lägg till alla barn om barnet inte redan är besökt. Markera från vilken nod den tillagda noden upptäcktes. När stacken är slut (inga noder är obesökta) så avslutas sökningen och man kan backtracka. Kolla varifrån slutmålet kom. Kolla varifrån den noden kom o.s.v. hela vägen till start.

DFS

Som BFS fast djupet först.

MST

1. Välj den koppling mellan noder med minst väg. Lägg noderna i samma disjunkta set.
2. Välj den koppling med näst minst väg som inte är i samma set. Lägg samman noderna.
3. Skriv ut den slutgiltiga vägen, noderna och vikterna som stöts på på vägen.

Kostnaden är summan av alla vikterna.

Dijkstra

Skriv mer här

Disjunkta mängder

En samling träd. Sätt alla element var för sig.

Find

Sök genom alla element

Union

Sätt samman träden

Find med compression

Alla element som besöks kopplas direkt till roten.

Union med rank

Koppla det minsta trädet till roten av det största trädet.

Tidskomplexitet

Find utan compression: $O(n)$ Union utan rank: $O(n)$ Find med compression: $O(1)$ Union med rank: $O(n \log n)$

Huffman kod

Konstruera ett träd med given frekvens

Sortera tecknena efter frekvens i fallande ordning. Det två sista tecknena kopplas samman med en nod. Denna nod skrivs med summan av de bägge frekvenserna. Denna resulterande nod läggs till i de sorterade tecknena - observera ordningen. De två tecknen som redan ritats stryks ur listan. Nästa två tal väljs. Repetera.

För att få fram vilken kod ett tecken ska ha börjas ur den ursprungliga noden. För varje steg till vänster skrivs 0, för varje till höger skrivs 1.

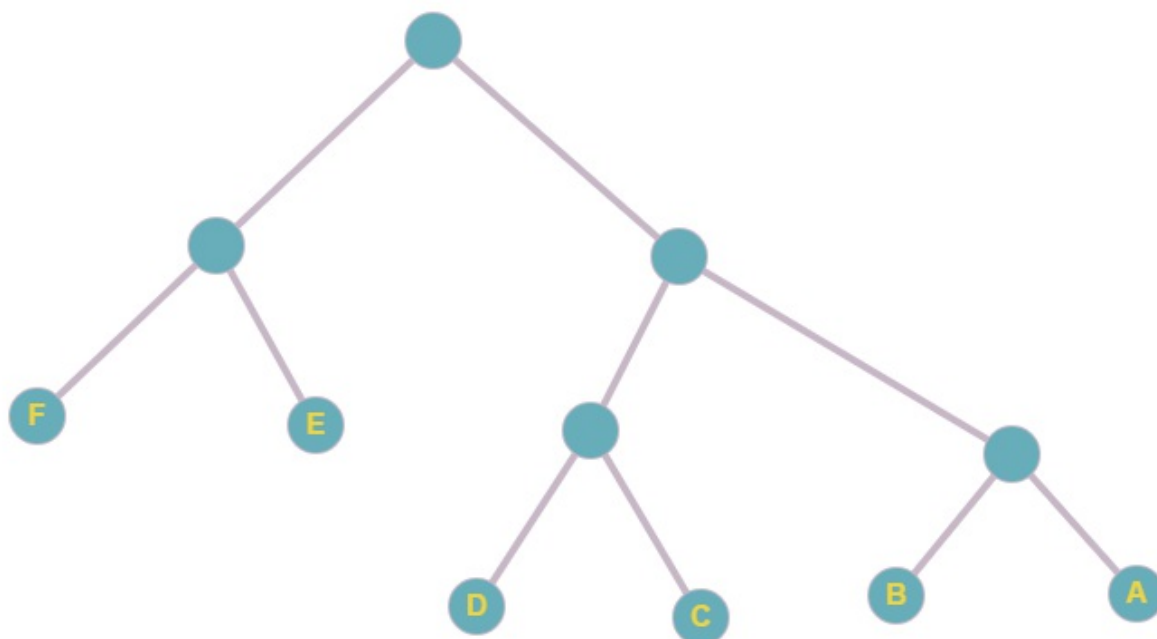
Konstruera ett träd med given kod

Sortera de kodade tecknena efter kodlängd stigande. Följ koden (vänster om 0, höger om 1). Rita noder där det behövs. Vid den sista noden (biten) skrivs tecknet.

Optimal framställning

1. Beräkna antal bitar som krävs för att skriva samtliga tecken.
2. Beräkna det genomsnittliga bitantalet med huffmankoden
3. Se om 2) är mindre än 1)

Exempel



Här får vi följande kod: Här får vi en genomsnittlig kodlängd på 2.7 bitar. För att koda sex tecken binärt krävs 3 bitar. Således är Huffmankoden optimal.

Sorteringsalgoritmer

Quick sort

Tidskomplexitet

Best Case	Average Case	Worst Case	Övrigt
$O(n \log n)$	$O(n \log n)$	$O(n^2)$	

1. Välj ett pivotelement. Kan exempelvis väljas som det element längst till vänster eller det i mitten.
2. Öppna det första elementet i listan. Jämför pivot med elementet. I fel ordning? Byt med

senast öppna kort.

- Öppna nästa elementet i listan. I rätt ordning? Låt vara öppet
- Öppna nästa element i listan. I fel ordning? Byt med senast öppna kort.
- Alla element vänster om pivot är nu mindre än pivot. Alla element höger om pivot är större än pivot.
- Gå till 1) för vänster och höger del av pivot.

Selection sort

Tidskomplexitet

Best Case	Average Case	Worst Case	Övrigt
$O(n^2)$	$O(n^2)$	$O(n^2)$	Implementeras med array

- Skapa en ny lista.
- Välj det minsta/största elementet i den osorterade listan.
- Lägg in elementet i den nya listan.
- Fortsätt så länge som den gamla listan har element

Insertion sort

Tidskomplexitet

Best Case	Average Case	Worst Case	Övrigt
$O(n)$	$O(n^2)$	$O(n^2)$	Implementeras med array

- Öppna det första elementet i listan.
- Öppna nästa elementet i listan. Lägg det nya elementet i rätt ordning.
- Upprepa till alla element är kollade

Heap sort

Best Case	Average Case	Worst Case	Övrigt
$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Implementeras med array

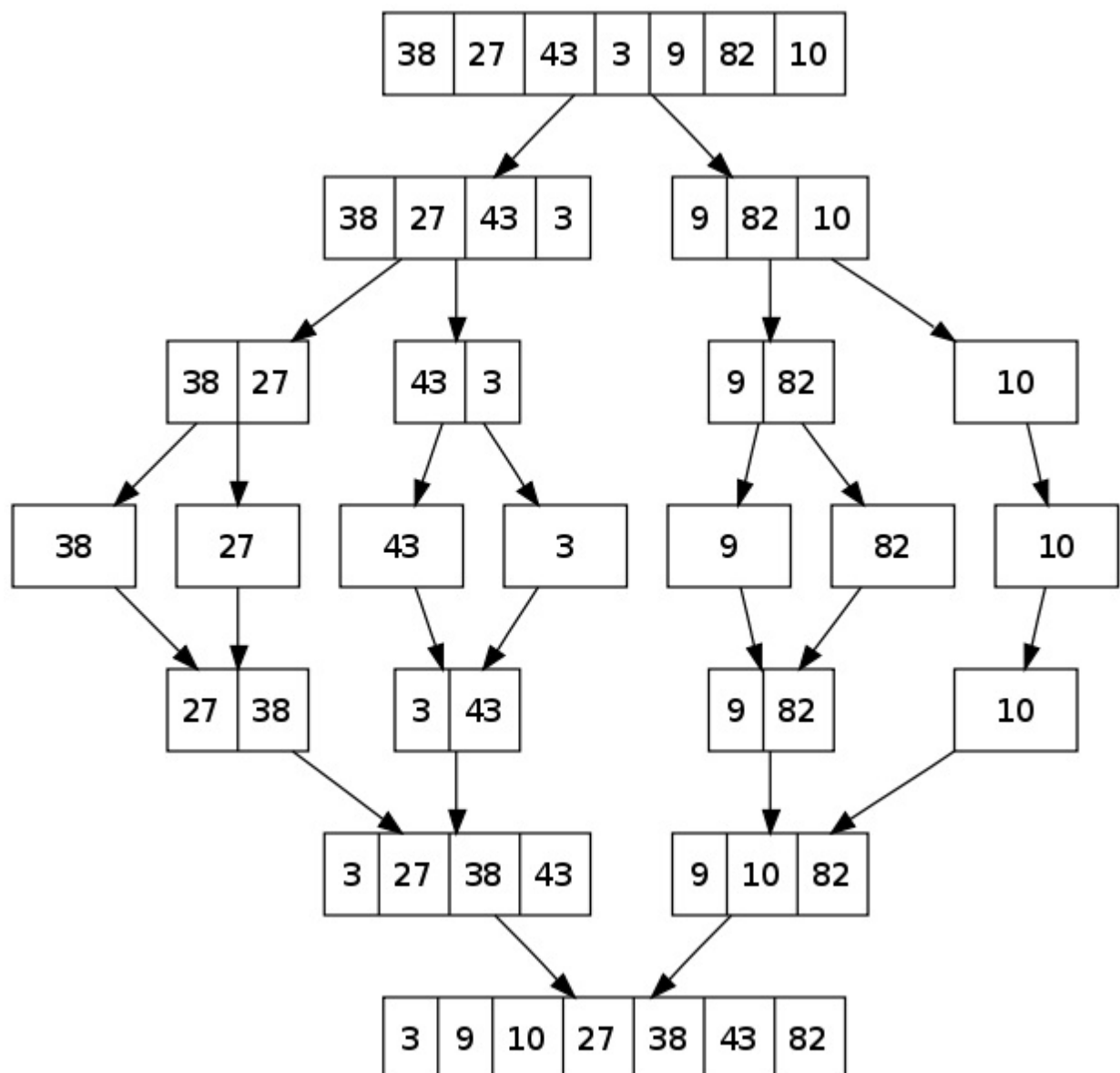
Bygg en max-heap:

- Lägg in elementen i ett träd. Det första elementet blir en rot, det tredje och andra blir barn o.s.v. Vänster till höger.
- Jämför den första föräldern med dess barn och byt som nödvändigt.
- Jämför den näst första o.s.v.
- Ersätt roten med det sista elementet och ta bort noden. Noden läggs i slutet på listan (som fylls på störst till minst)
- Gå till steg 2)
- Upprepa till enbart ett element är kvar.

Merge sort

Best Case	Average Case	Worst Case	Övrigt
$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Implementeras med array

1. Dela arrayen i mitten.
2. Dela varje array i mitten.
3. Upprepa 2) till dess att alla arrayer är enkla element.
4. "Mergea" alla element parvis. Lägg elementen i ordning
5. "Mergea" array parvis. Lägg elementen i ordning.
6. Upprepa 5) tills enbart en lista återstår.



(Shell sort)

(Bucket sort)

(Radix sort)