Inhoud

[4.1 Terminologie m.b.t bomen 2](#_Toc87289697)

[4.2 Datastructuren voor bomen. 3](#_Toc87289698)

[4.2.1 Array-van-kinderen voorstelling 3](#_Toc87289699)

[4.2.2 Eerste-kind-volgende-broer voorstelling 4](#_Toc87289700)

[4.3 Recursie op bomen 4](#_Toc87289701)

[4.3.1 Alle toppen van de boom bezoeken 4](#_Toc87289702)

[4.3.2 Eenvoudige berekeningen op bomen 5](#_Toc87289703)

[4.4 Binaire bomen 6](#_Toc87289704)

[4.4.1 Definitie en eigenschappen 6](#_Toc87289705)

[4.4.3 Alle toppen van een binaire boom bezoeken. 8](#_Toc87289706)

[4.5 Binaire zoekbomen 9](#_Toc87289707)

[4.5.1 Opzoeken van een sleutel in een binaire zoekboom 10](#_Toc87289708)

[4.5.2 Toevoegen van een sleutel aan een binaire zoekboom 10](#_Toc87289709)

[4.5.3 verwijderen van een sleutel uit een binaire zoekboom 11](#_Toc87289710)

[4.5.4 Tijdscomplexiteit van de bewerkingen 12](#_Toc87289711)

[4.6 Binaire hopen 13](#_Toc87289712)

[4.6.1 Prioriteitswachtrij. 13](#_Toc87289713)

[4.6.2 Implementatie als binaire hoop 13](#_Toc87289714)

[4.6.3 Implementatie 14](#_Toc87289715)

[4.6.4 Opzoeken van het element met de kleinste sleutel 14](#_Toc87289716)

[4.6.5 Toevoegen van een element 15](#_Toc87289717)

[4.6.6 Verwijderen van het element met de kleinste sleutel 16](#_Toc87289718)

[4.6.7 Tijdscomplexiteit van de bewerkingen 16](#_Toc87289719)

Hoofdstuk 4 Bomen

we starten met de definitie van gewortelde bomen samen met hun basisbegrippen. We bespreken kort twee data structuren om zo’n bomen voor te stellen in het computergeheugen. Aangezien gewortelde bomen op een natuurlijke wijze recursief gedefinieerd zijn volgt hieruit dat een aantal bewerkingen en berekeningen ook gemakkelijk recursief geïmplementeerd worden. Binaire zoekbomen laten toe om snel elementen op te zoeken, toe te voegen en te verwijderen.

# 4.1 Terminologie m.b.t bomen

Er zijn veel situaties waarin informatie geordend is volgens een of andere hiërarchische structuur. De abstractie die zulke situaties modelleert is een boom, een fundamenteel begrip in de informatica.

Bomen hebben een recursieve structuur, dus er is een recursieve definitie aan toegewezen.

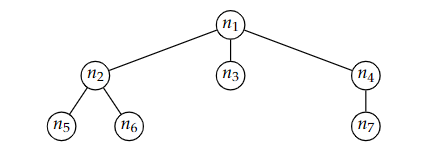
**Definitie:** Een gewortelde boom T is een verzameling van toppen die aan de volgende eigenschappen voldoet:

1. Er is een speciale top t die de wortel van de boom wordt genoemd.
2. De andere toppen zijn verdeeld in m >= 0 disjuncte verzamelingen T1,,,,Tm die op hun beurt elk weer een gewortelde boom zijn.

We zeggen dat bomen T1 tem Tm deelbomen zijn van T, de wortels van de deelbomen worden de kinderen van de wortel t genoemd. Uit de recursieve definitie van een gewortelde boom volgt dat elke top in de boom uiteindelijk de wortel is van een deelboom die bevat is in de boom.

* Het aantal kinderen van een top wordt de graad van die top genoemd.
* Een blad is een top met graad nul.
* Een top die geen blad is wordt een intern genoemd
* De graad van een boom wordt gedefinieerd als het maximum van de graden van zijn toppen.

Afbeelding met tafel

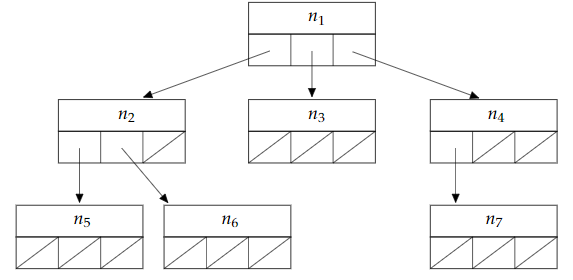
Automatisch gegenereerde beschrijving

## 4.2 Datastructuren voor bomen.

Er zijn verschillende manieren om bomen voor te stellen in het geheugen van een computer. We bespreken er twee.

## 4.2.1 Array-van-kinderen voorstelling

Een eenvoudige manier om een boom voor te stellen is door rechtstreeks de vader-kind relatie te implementeren. Dit betekent dat we een structuur top definiëren die een veld heeft om de data van de top bij te houden, alsook een array van referenties naar de kinderen van die top. Wanneer een kind niet bestaat wordt dit voorgesteld met referentie null.

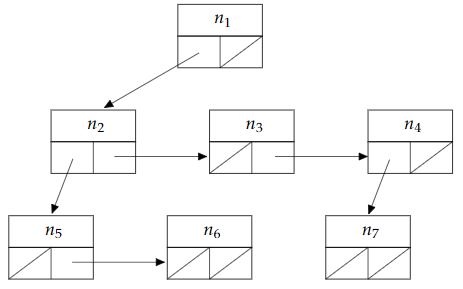


We zien in de figuur dat er veel referenties niet gebruikt worden. We hebben 21 referenties voorzien waarvan er maar 6 gebruikt worden. Het aantal referenties is juist 1 minder dan het aantal toppen van de boom. Het nadeel van een array-van-kinderen voorstelling is dus het inefficiënt gebruik van geheugen, zeker voor wanneer de graad van de boom zo groot is.

## 4.2.2 Eerste-kind-volgende-broer voorstelling

In plaats van in elke top referenties naar al zijn kinderen op te slaan, houden we altijd juist twee referenties bij:

* Een referentie naar zijn eerste kind
* Naar de volgende broer



We leiden af dat er heel wat minder null referenties zijn dan bij de andere voorstelling. We hebben 14 referenties waar er 6 effectief van gebruikt worden. Zo is de referentie naar het eerste kind null als en slechts als de top een blad is. Wanneer dat niet het geval is, dan zijn de kinderen van de top geschakeld in een lineaire lijst structuur.

Dit wil zeggen dat we hier geen rechtstreekse toegang hebben tot het derde kind, we moeten dus eerst de referentie volgen van het voorafgaande: 1-2-3. Dit wilt zeggen dat het efficiënter is qua geheugen maar meer tijd in beslag zal nemen.

# 4.3 Recursie op bomen

## 4.3.1 Alle toppen van de boom bezoeken

Er zijn meerdere mogelijkheden om de toppen van de boom te bezoeken, we bespreken er twee:

* Eerste mogelijkheid: eerst de wortel van de boom te bezoeken en daarna de toppen van zijn deelbomen te doorlopen.
* Tweede mogelijkheid: recursief de toppen van de deelbomen te doorlopen en daarna pas de wortel van de boom te bezoeken.

Deze mogelijkheden worden PREORDE en POSTORDE genoemd.

Om een boom te doorlopen in preorde gaat men als volgt tewerk:

1. Bezoek de wortel van de boom
2. Doorloop de deel bomen van de wortel in preorde

Om een boom te doorlopen in postorde gaat men als volgt tewerk:

1. Doorloop de deelbomen van de wortel in postorde
2. Bezoek de wortel van de boom

Pseudo code:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

## 4.3.2 Eenvoudige berekeningen op bomen

Veronderstel dat we het aantal toppen van een boom wensen te berekenen, maar dat de voorstelling van een boom dit niet rechtstreeks toelaat. Uit de definitie van een boom kunnen we zien dat het aantal toppen #T in de boom gelijk is aan:

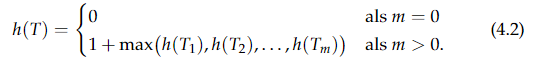


De formule drukt uit dat het aantal toppen in een boom gelijk is aan de som van het aantal toppen in elk van de deelbomen van de wortel, vermeerderd met 1 (we mogen de wortel zelf niet vergeten).

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Een andere berekening die eenvoudig geïmplementeerd kan worden is het berekenen van de hoogte van de boom. De hoogte h(T) van een boom is in het algemeen een meer dan het maximum van de hoogtes van zijn deelbomen.



# 4.4 Binaire bomen

## 4.4.1 Definitie en eigenschappen

we beschouwen nu bomen die zeer vaak voorkomen in de praktijk, nl binaire bomen. We geven opnieuw een definitie:

Afbeelding met horloge

Automatisch gegenereerde beschrijvingEen binaire boom is een verzameling toppen die:

1. Ofwel leeg is
2. Ofwel bestaat uit een wortel en twee disjuncte verzamelingen die op hun beurt ook een binaire boom zijn (Tl Tr). We noemen die twee delen respectievelijk de linker en rechterdeelboom van de wortel.

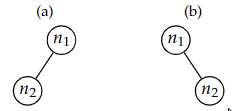
Afbeelding met tekst

Automatisch gegenereerde beschrijving

De verzameling Tl en Tr zijn op hun beurt weer binaire bomen. De wortel van Tr is de top n3. De linkerboom van n3 is leeg, terwijl de rechterdeelboom bestaat uit toppen n6 en n9, lege bomen worden eenvoudigweg niet getekend.

Verschillen gewone bomen en binaire bomen:

* Een binaire boom kan geen enkele top bevatten.
* Bij de binaire boom wordt er altijd ordening opgelegd bij de deelbomen.

Deze twee bomen zijn identiek als we ze beschouwen als gewone gewortelde bomen. Maar ze zijn verschillend als binaire bomen, bij de ene is de rechterdeelboom leeg, bij de ander de linker.

In een binaire boom is het makkelijk een bovengrens te geven voor het aantal toppen met een bepaalde diepte K, de diepte van een top in een binaire boom wordt op dezelfde manier gedefinieerd als in een gewone gewortelde boom.

**Eigenschap:** in een binaire boom is het aantal toppen met diepte k hoogstens 2^k

**Bewijs:** in een niet lege binaire boom is er steeds juist een top met diepte 0, nl de wortel. Deze wortel heeft hoogstens twee niet-lege deelbomen. Dus zijn er hoogstens 2 toppen met diepte 1. Elk van die toppen met hoogte 1 heeft opnieuw hoogstens 2 niet lege deelbomen. Dus het aantal toppen met diepte 2 is hoogstens 2x2=4. Zo verdergaand zien we dat het aantal toppen met diepte k hoogstens 2^k is.

Stel dat we weten dat de diepte van een binaire boom T gelijk is aan d. dan kunnen we met volgende eigenschap een bovengrens geven voor het aantal toppen dat die boom kan bevatten.



**Bewijs:** we tellen het aantal toppen van een boom T door het aantal toppen met een bepaalde diepte k te tellen en dit voor alle mogelijke dieptes.

Afbeelding met tekst

Automatisch gegenereerde beschrijvingDit bewijst de bovengrens voor T.

Een ondergrens voor het aantal toppen in een binaire boom met diepte d is ook gemakkelijk te bepalen. De boom zal het kleinste aantal toppen hebben wanneer er juist 1 top is op elke diepte, het minimum aantal toppen in een binaire boom is dus d+1.

Afbeelding met horloge

Automatisch gegenereerde beschrijvingVoorbeeld: een binaire boom met diepte 3 die het maximum aantal toppen bevat vs een binaire boom die het minimum aantal toppen bevat.

## 4.4.3 Alle toppen van een binaire boom bezoeken.

We hebben reeds gezien hoe we alle toppen van een gewortelde boom kunnen bezoeken, door de boom te doorlopen in ofwel pre- of postorde. Als we de binaire boom willen doorlopen in pre- of postorde dan is het bijna altijd zo dat de linkerdeelboom steeds eerst doorlopen wordt voor de rechterdeelboom.

**Stappenplan preorde:**

1. Bezoek de wortel van de boom
2. Als de linkerdeelboom niet leeg is, doorloop de linkerdeelboom dan recursief in preorde
3. Als de rechterdeelboom niet leeg is, doorloop de linkerdeelboom dan recursief in preorde

**Stappenplan postorde:**

1. Als de linkerdeelboom niet leeg is, doorloop de linkerdeelboom dan recursief in postorde
2. Als de rechterdeelboom niet leeg is, doorloop de linkerdeelboom dan recursief in postorde
3. Bezoek de wortel van de boom

Omdat elke top nu steeds twee deelbomen heeft, die eventueel leeg kunnen zijn, is er nog een derde manier om door een binaire boom te lopen.

* We kunnen de wortel bezoeken tussen het doorlopen van de linker- en rechterdeelboom.
* Deze manier heet de inorde.

**Stappenplan inorde:**

1. Als de linkerdeelboom niet leeg is, doorloop de linkerdeelboom dan recursief in inorde.
2. Bezoek de wortel van boom.
3. Als de rechterdeelboom niet leeg is, doorloop de rechterdeelboom dan recursief in inorde.

**Pseudo code:**

Afbeelding met tekst

Automatisch gegenereerde beschrijving

# 4.5 Binaire zoekbomen

Een vaak voorkomende activiteit in een computerprogramma is het bijhouden van een verzameling waarden waarbij we wensen:

* Gegevens toe te voegen aan de verzameling
* Gegevens te verwijderen uit de verzameling
* Controleren of een element aanwezig is in de verzameling

Er zijn verschillende gegevensstructuren die gebruikt worden om deze functionaliteiten te implementeren, zoals lineair gelinkte lijsten en hashtabellen.

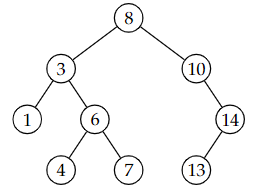
De binaire zoekboom kan deze bewerkingen ook implementeren, een noodzakelijk voorwaarde is:

* De labels moeten een totaal geordende verzameling vormen
  + Dwz: voor elk paar mogelijk labels x en y geldt:
    - 
* Dit wilt zeggen, dat wanneer x en y verschillend zijn, we kunnen zeggen welk element het kleinste is. Er zijn geen onvergelijkbare elementen.

**Definitie**: een binaire zoekboom is een gelabelde binaire boom die aan de binaire zoekboomeigenschap voldoet.

**Binaire zoekboomeigenschap:** voor elke top x van de binaire zoekboom geldt dat alle toppen in de linkerdeelboom van x een label hebben dat kleiner is dan het label van x, terwijl voor alle toppen in de rechterdeelboom van x geldt dat hun label groter is dan het label van x.

Afbeelding met tafel

Automatisch gegenereerde beschrijving

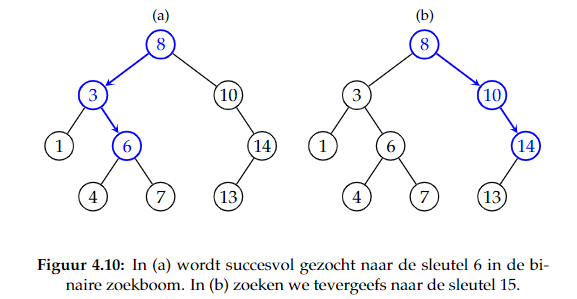
Merk op dat deze rij gesorteerd is van klein naar groot. Een binaire zoek-boom houdt zijn waarden dus ook gesorteerd bij, al is dat op het eerste zicht misschien niet zo duidelijk als bij een gesorteerde array.

## 4.5.1 Opzoeken van een sleutel in een binaire zoekboom

Wanneer we een sleutel x willen opzoeken, kunnen we gebruikmaken van de zoekboomeigenschap om uit te maken of x aanwezig is of niet.

We hebben zoekboom T

* Als T leeg is, is x niet aanwezig
* Als x kleiner is dan het label van wortel T, dan bevindt x zich in de linkerdeelboom.
* Als x groter is dan het label van wortel T, dan bevindt x zich in de rechterdeelboom.
* Als x gelijk is aan het label van de wortel T, bevindt x zich in de wortel van de boom



## 4.5.2 Toevoegen van een sleutel aan een binaire zoekboom

Wanneer we een sleutel x toevoegen aan een binaire zoekboom T, dan is het van belang dat de nieuwe boom T1 eveneens een binaire zoekboom is.

We kunnen het toevoegen van een sleutel x aan een binaire zoekboom op volgende manier formuleren:

* Vergelijk x met het label van de wortel
  + Wanneer het kleiner is, voeg x toe aan de linkerdeelboom
  + Wanneer x groter is, voeg x toe aan de rechterdeelboom

Afbeelding met horloge

Automatisch gegenereerde beschrijving

## 4.5.3 verwijderen van een sleutel uit een binaire zoekboom

Het verwijderen van een sleutel x start met het opzoeken van deze sleutel in de boom. Er kunnen zich 3 gevallen voordoen:

1. De sleutel x bevindt zich in een blad
2. De sleutel x bevindt zich in een top met een kind
3. De sleutel x bevindt zich in een top met twee kinderen

Afbeelding met horloge

Automatisch gegenereerde beschrijving**De sleutel x bevindt zich in een blad:**

Wanneer x zich in een blad bevindt, kunnen we dit eenvoudigweg uit dit blad verwijderen.

**De sleutel bevindt zich in een top met 1 kind:**

Wanneer de top n die x bevat slecht een kind heeft, dan kunnen we n vervangen door dit ene kind. Met andere woorden: de nieuwe linker of rechterkind van de ouder van n is zijn vroegere kleinkind.

**De sleutel bevindt zich in een top met 2 kinderen:**

Afbeelding met horloge, kettinkje

Automatisch gegenereerde beschrijvingWe kunnen x vervangen door het kleinste element y van de deelboom. Dit element is de opvolger van x, in de volgende stap verwijderen we element y uit de deelboom.

Voorbeeld: we willen sleutel 3 verwijderen uit de boom en zien dat 3, 2 kinderen heeft we gaan opzoek naar het kleinste element groter dan sleutel 3.

## 4.5.4 Tijdscomplexiteit van de bewerkingen

Het is duidelijk dat een binaire boom in het slechtste geval een uitvoeringstijd heeft die evenredig is met de diepte van de boom.

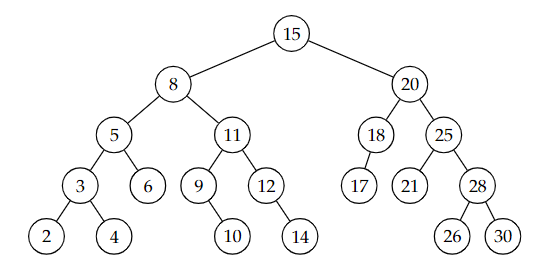
Wanneer de boom n toppen bevat, weten we dat:



Door van beide leden de logaritme met grondgetal twee te nemen zien we dat voor een totaal gebalanceerde boom geldt dat O(d) =O(lg(n)). Voor een totaal gebalanceerde boom hebben alle bewerkingen dus een tijdscomplexiteit O(lg(n),met n het aantal toppen in de zoekboom.

Echter zien we ook dat voor een totaal ongebalanceerde boom d+1≤n zodat O(d) =O(n).

Men kan aantonen dat in het gemiddeld geval (wanneer elementen random worden toegevoegd aan een boom) de resulterende boom een diepte zal hebben die O(lg(n)) is.



# 4.6 Binaire hopen

## 4.6.1 Prioriteitswachtrij.

Een abstract datatype dat vaak gebruikt wordt in toepassingen is een prioriteitswachtrij. Dit is een uitbreiding van de gewone FIFO wachtrij.

De elementen van de wachtrij bestaan typisch uit twee delen:

* Een sleutel
  + Geeft de prioriteit aan
* Een waarde
  + Bevat de waarde

In een prioriteitswachtrij kan men:

* Opzoeken
* Verwijderen
* Toevoegen

**Binaire hoop:**

Afbeelding met horloge, schaar

Automatisch gegenereerde beschrijving

## 4.6.2 Implementatie als binaire hoop

In essentie is een binaire hoop een complete binaire boom die aan een extra ordeningseigenschap voldoet.

**Definitie:** een complete binaire boom is een binaire boom van diepte d waarbij het aantal toppen met diepte k < d maximaal (dus 2^k) is.

Afbeelding met tafel

Automatisch gegenereerde beschrijving**Definitie**: de ordeningseigenschap voor binaire hopen zegt dat de sleutel van elke top hoogstens gelijk is aan de kleinste sleutel van zijn kinderen.

## 4.6.3 Implementatie

Het is mogelijk om een binaire hoop te implementeren als een binaire boom, waarbij men voor elke top bijhoudt wat zijn linker- en rechterkinderen zijn.

Veronderstel dat we de toppen van een complete binaire boom gaan nummeren op volgende manier:

* De wortel van de boom heeft rangnummer 1.
* Afbeelding met horloge

  Automatisch gegenereerde beschrijvingDe toppen op diepte 1 worden consecutief genummerd van links naar rechts (en krijgen rangnummers 2 en 3)
* Daarna worden de toppen op diepte 2 genummerd (startend vanaf 4)
* …

Omgekeerd telt wanneer een top rangnummer i heeft ( en deze niet de wortel van de boom is), dan heeft zijn ouder rangnummer floor(i/2). We stellen vast dat:

Afbeelding met tekst

Automatisch gegenereerde beschrijving

* We kunnen een binaire hoop opslaan in een array, waarbij de ouder-kind relaties afgeleid kunnen worden van de eigenschap:
  + Wanneer een top rangnummer i heeft dan hebben zijn linker- rechterkind respectievelijk rangnummer 2i en 2i+1

Afbeelding met tafel

Automatisch gegenereerde beschrijving

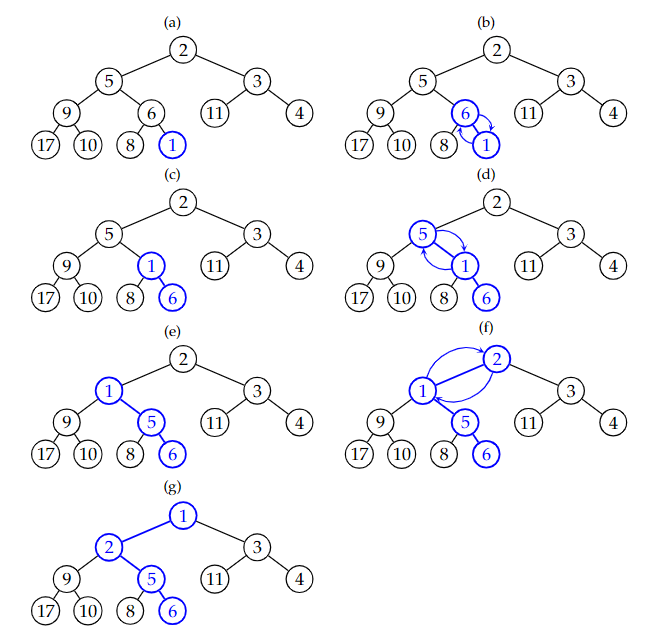
## 4.6.4 Opzoeken van het element met de kleinste sleutel

Uit de ordeningseigenschap voor binaire hopen volgt dat het element met de kleinste sleutel steeds de wortel van de boom is. Het opzoeken ervan is dus een triviale bewerking, die de binaire hoop ongewijzigd laat. Dit wordt uitgevoerd in constante tijd.

## 4.6.5 Toevoegen van een element

Om een nieuw element toe te voegen, ga je als volgt tewerk:

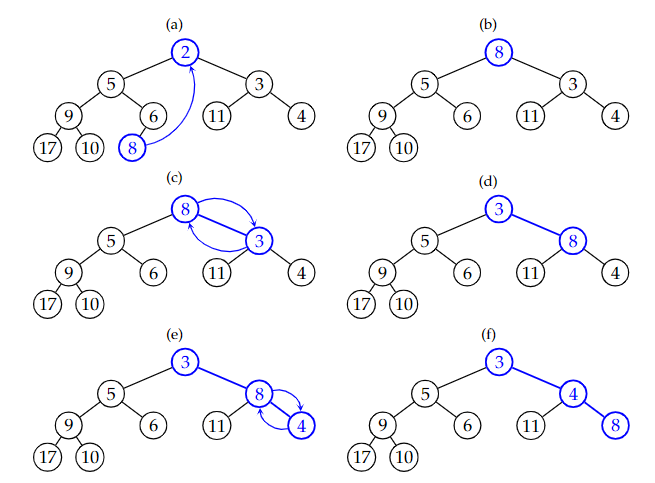
1. Creëer een nieuw element
2. Voeg dit element toe op de eerste beschikbare plaats. Dit betekent dus als een nieuw blad, met rangnummer i, waarbij we er steeds voor zorgen dat het diepste niveau gevuld is van links naar rechts.
3. Op dit moment is het in het algemeen zo dat de ordeningseigenschap voor binaire hopen nu kan geschonden zijn tussen i en parent(i). indien dit niet zo is, verwissel ze dan. Dit hersteld de ordeningseigenschap. Herhaal dit tot dat de binaire hoop hersteld is.



## 4.6.6 Verwijderen van het element met de kleinste sleutel

Wanneer we het element met de kleinste sleutel verwijderen, dan krijgen we een “gat” aan de wortel van de boom.

* We vullen dit op door het meest rechtste blad op het diepste niveau erin te “duwen”. En het meest rechtste blad op het diepste niveau te verwijderen.
* Op dit moment is wellicht de ordeningseigenschap geschonden tussen de wortel en zijn kinderen.
  + Herstel dit door de wortel om te wisselen met zijn kleinste kind.



## 4.6.7 Tijdscomplexiteit van de bewerkingen

Een binaire hoop met n toppen heeft diepte O(lg(n)). Dit betekent onmiddellijk dat de tijdscomplexiteit van de bewerkingen toevoegen van een element en verwijderen van het kleinste element in het slechtse geval O(lg(n)). Is.

Het ophalen van het kleinste element is uiteraard een bewerking die in een constante tijd kan uitgevoerd worden: T(n) =O(1).