Prioritetskøer og Hobe

- Prioritetskøer
- Træer og hobe
- Repræsentation af hobe
- Algoritmer på hobe
- Hobkonstruktion
- Hobsortering

Prioritetskø

Prioritetskø: Vedligehold en dynamisk mængde S af elementer.

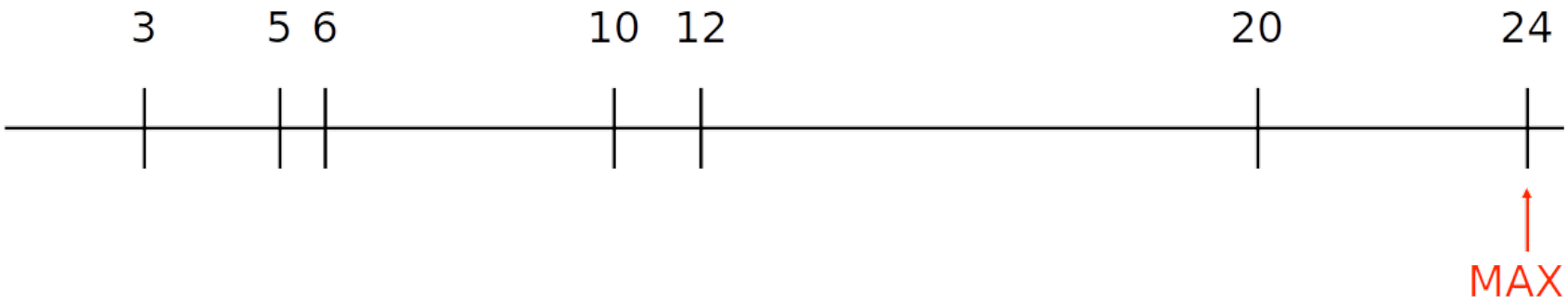
Hver element x er tilknyttet en nøgle $x.\texttt{key}$ og satellitdata $x.\texttt{data}$.

`Max()` returnér element med største nøgle

`ExtractMax()` returnér og fjern element med største nøgle

`IncreaseKey()` sæt $x.\texttt{key} = k$, hvor k er større end den gamle $x.\texttt{key}$.

`Insert(x)` Tilføj elementet x til mængden S.



Prioritetskøer

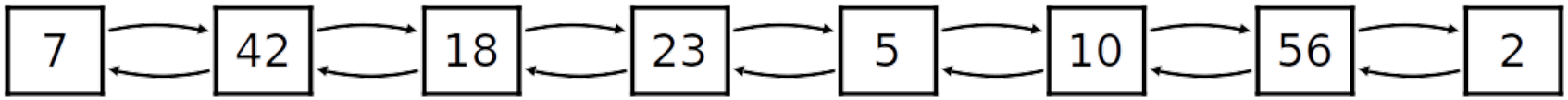
Anvendelser:

- Skedulering
- Korteste veje i vægtede grafer (Dijkstras algoritme)
- Mindste udspændende træ (Prims algoritme)
- Kompression (Huffmans algoritme)
- ...

Udfordring: hvordan kan vi lave datastrukturen med nuværende teknikker?

Prioritetskø

Løsning med hægtet liste 1: vedligehold S i en dobbelt-hægtet liste.



`Max()` lineær søgning efter element med størst nøgle.

`ExtractMax()` lineær søgning efter element med størst nøgle. Returnér og fjern elementet.

`IncreaseKey()` sæt x`.key` $= k$.

`Insert(x)` Tilføj elementet x i starten af listen.

Tid:

`Max` og `ExtractMax` i $O(n)$ tid (hvor $n = |S|$),

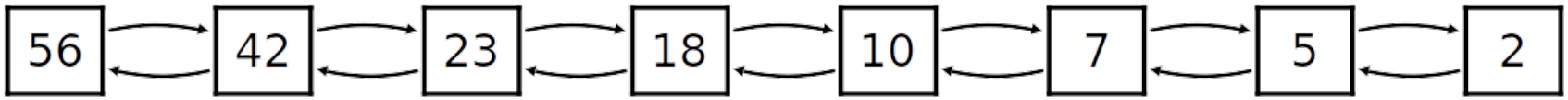
`IncreaseKey` og `Insert` i $O(1)$ tid.

Plads:

$O(n)$

Prioritetskø

Løsning med hægtet liste 2: vedligehold S i en sorteret dobbelt-hægtet liste.



`Max()` Returner listens første element.

`ExtractMax()` Returnér og fjern listens første element.

`IncreaseKey()` Sæt x`.key` $= k$. Lineær søgning for at finde ny plads til x.

`Insert(x)` Søg lineært efter den rette plads til x i listen.

Tid: (hvor $n = |S|$)

`Max` og `ExtractMax` i $O(1)$ tid,

`IncreaseKey` og `Insert`i $O(n)$ tid.

Plads:

$O(n)$

Prioritetskø

Datastruktur	MAX	EXTRACTMAX	INCREASEKEY	INSERT	Plads
hægtet liste	O(n)	O(n)	O(1)	O(1)	O(n)
sorteret hægtet liste	O(1)	O(1)	O(n)	O(n)	O(n)

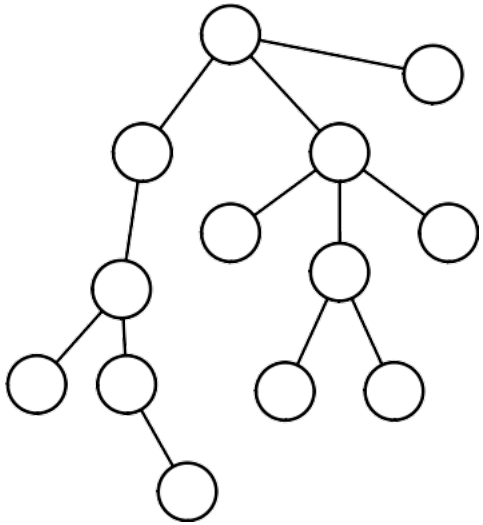
Udfordring: Kan vi gøre det bedre?

Kræver ny ide.

Prioritetskøer og Hobe

- Prioritetskøer
- Træer og hobe
- Repræsentation af hobe
- Algoritmer på hobe
- Hobkonstruktion
- Hobsortering

Rodfæstet træ



Rodfæstet træ:

- Knuder forbundet med kanter.
- Sammenhængende. Uden kredse.
- En knude er udvalgt til at være rod.
- Speciel type graf.

Terminologi:

- Børn, forælder, efterkommer, forfader, blade, interne knuder, sti, ...

Dybde og højde:

Lad v være en knude i træ T.

- Dybden af v = længden af sti fra v til roden
- Højden af v = længden af længste sti fra v til en bladefterkommer
- Dybden af T = højden af T = længden af længste sti fra rod til et blad

Binært træ

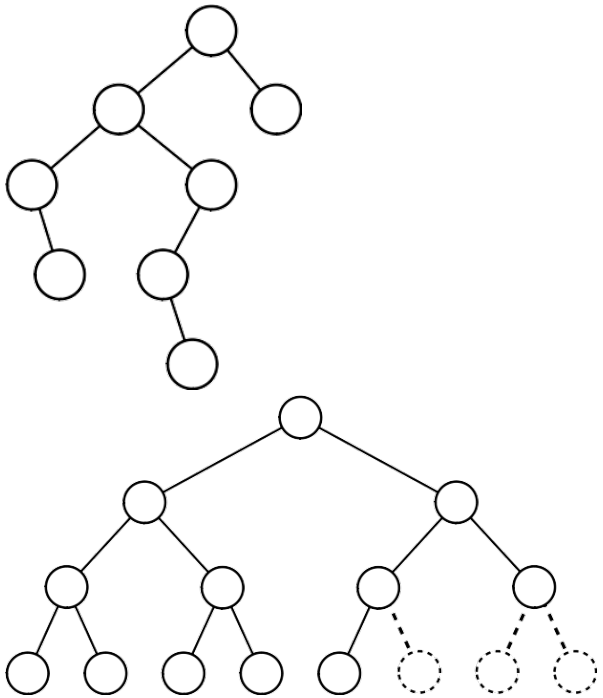
Binært træ:

- Rodfæstet.
- Enten tomt,
- Eller en knude med et venstre og højre barn, som begge er binære træer.

Perfekt binært træ: binært træ hvor alle interne knuder har præcis to børn, og alle blade har samme afstand fra roden.

Fuldstændigt binært træ: perfekt binært træ hvor 0 eller flere blade er fjernet fra højre mod venstre.

Lemma: Højden af et fuldstændigt binært træ med n knuder er $\Theta(\log n)$.



Hob

Hob: (heap) fuldstændigt binært træ, der overholder hob-orden.

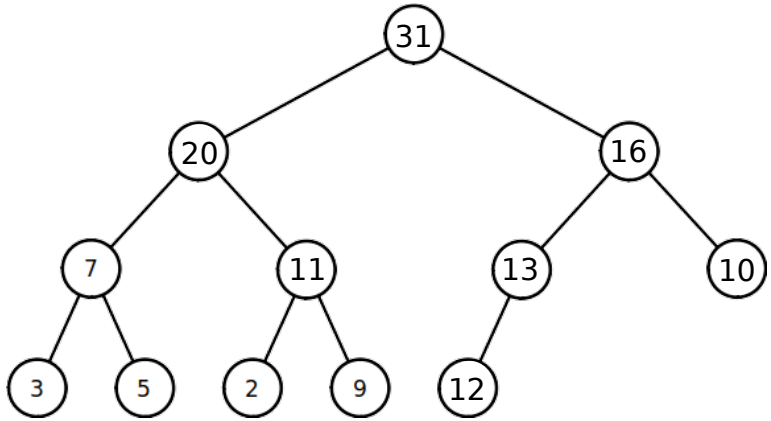
Hob-orden: (heap-order)

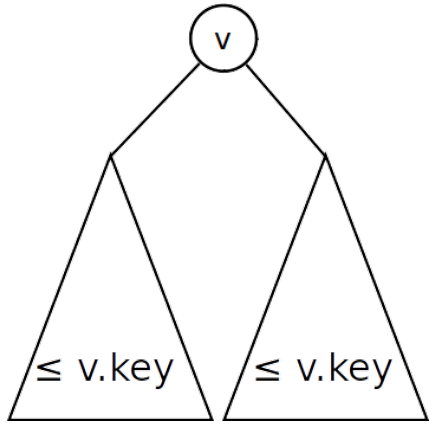
- Alle knuder indeholder en nøgle betegnet med `.key`.
- For alle knuder v:
  - alle nøgler i venstre deltræ og højre deltræ er $\leq v$`.key`.

Max-hob vs min-hob:

ovenstående er max-hob.

Udskift $\leq$ med $\geq$ for at få min-hob.





Prioritetskøer og Hobe

- Prioritetskøer
- Træer og hobe
- Repræsentation af hobe
- Algoritmer på hobe
- Hobkonstruktion
- Hobsortering

Hob

Datastruktur: Vi skal bruge følgende navigationsoperationer på en hob:

- Parent(x) returnér forældren af x
- Left(x) returnér venstre barn af x.
- Right(x) returnér højre barn af x.

Udfordring: Hvordan kan vi repræsentere en hob så vi kan understøtte navigation effektivt?

Hob – hægterepræsentation

Repræsentation med hægter:

hver knude består af

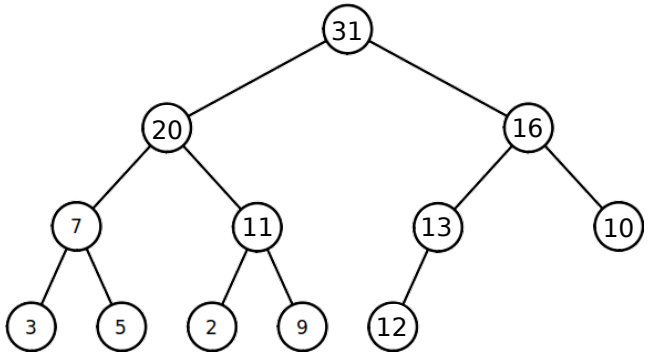
- v.key
- v.parent
- v.left
- v.right

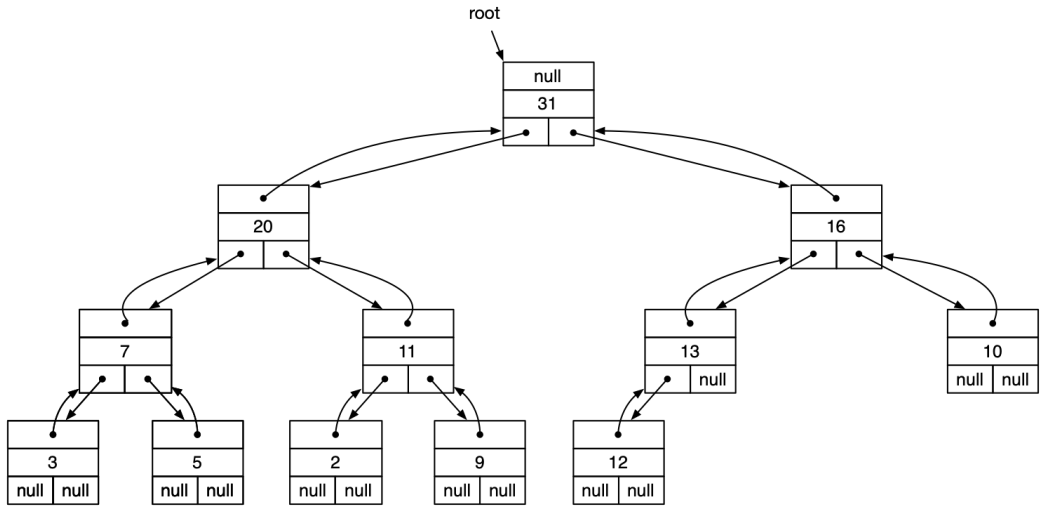
Parent(x), Left(x) , Right(x):

følge pointer/reference.

Tid: $O(1)$

Plads: $O(n)$





Hob – arrayrepræsentation

Repræsentation med array:

idé: giv hver knude et nummer

- Array `H[0..n]`
- `H[0]` bruges ikke.
- `H[1..n]` indeholder knuderne i niveauorden (level order).

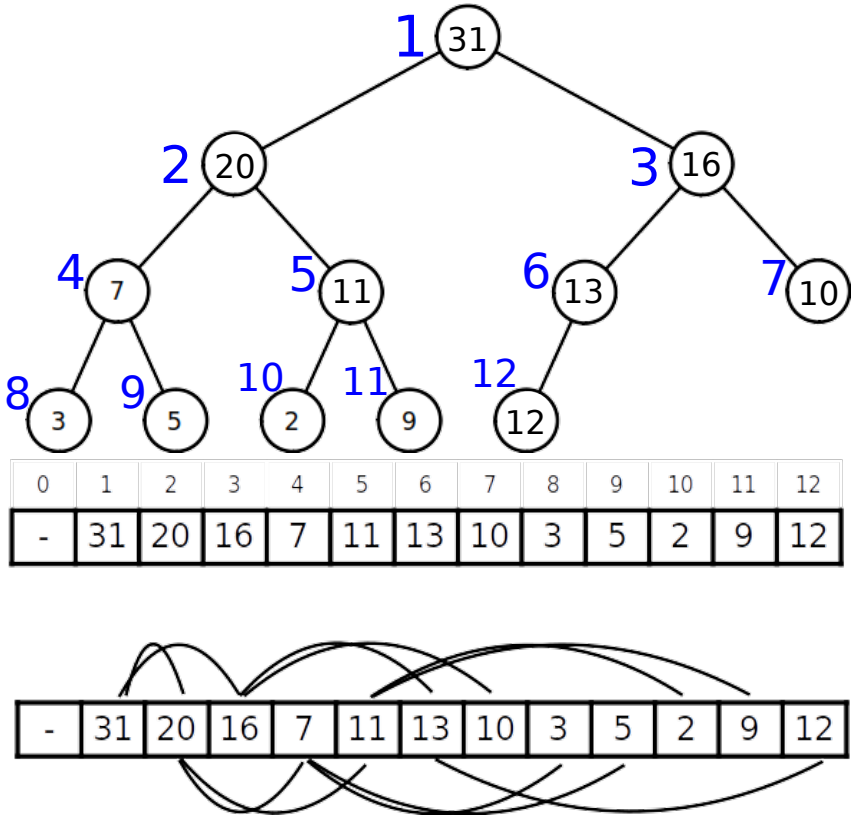
$$\text{Left}(x) \quad 2x$$

$$\text{Right}(x) \quad 2x+1$$

$$\text{Parent}(x) \quad \lfloor x/2 \rfloor$$

Tid: $O(1)$

Plads: $O(n)$



Prioritetskøer og Hobe

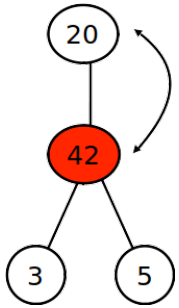
- Prioritetskøer
- Træer og hobe
- Repræsentation af hobe
- Algoritmer på hobe
- Hobkonstruktion
- Hobsortering

Algoritmer på hobe

Udfordring: hoborden er midlertidigt ikke overholdt.

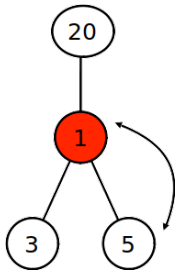
BubbleUp(x) Hvis hoborden er overtrådt i knude x fordi nøgle i x er større end nøgle i Parent(x).

- Ombyt x og Parent(x).
- Gentag med Parent(x) indtil hoborden er opfyldt.



BubbleDown(x) Hvis hoborden er overtrådt i knude x fordi nøglen i x er mindre end nøglen i Left(x) eller Right(x):

- Ombyt x med det barn b, der har den største nøgle,
- Gentag med b indtil hoborden er opfyldt.



Tid: Hvor hurtigt kører de?

- $\Theta(\log n)$ tid.

Hvordan kan vi bruge dem til implementation af prioritetskøoperationer?

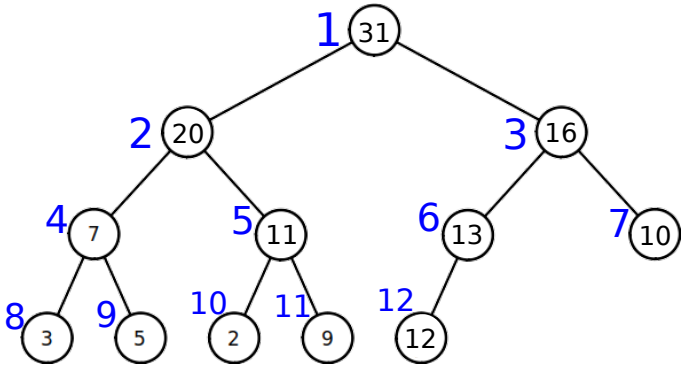
Operationer på prioritetskøer

```
MAX()
    returnér H[1]
```

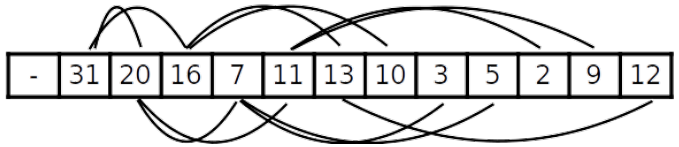
```
EXTRACTMAX()
    r = H[1]
    H[1] = H[n]
    n = n - 1

BUBBLEDOWN(1)
    returnér r
```

```
INSERT(x)
    n = n + 1
    H[n] = x
    BUBBLEUP(n)
```

```
INCREASEKEY(x,k)
    H[x] = k
    BUBBLEUP(x)
```



0	1	2	3	4	5	6	7	8	9	10	11	12
-	31	20	16	7	11	13	10	3	5	2	9	12



Tid:

- `Max` i $O(1)$ tid,
- `ExtractMax`, `IncreaseKey`, `Insert`, $\Theta(\log n)$ tid.

Prioritetskø

Datastruktur	MAX	EXTRACTMAX	INCREASEKEY	INSERT	Plads
hægtet liste	O(n)	O(n)	O(1)	O(1)	O(n)
sorteret hægtet liste	O(1)	O(1)	O(n)	O(n)	O(n)
hob	O(1)	O(log n)	O(log n)	O(log n)	O(n)

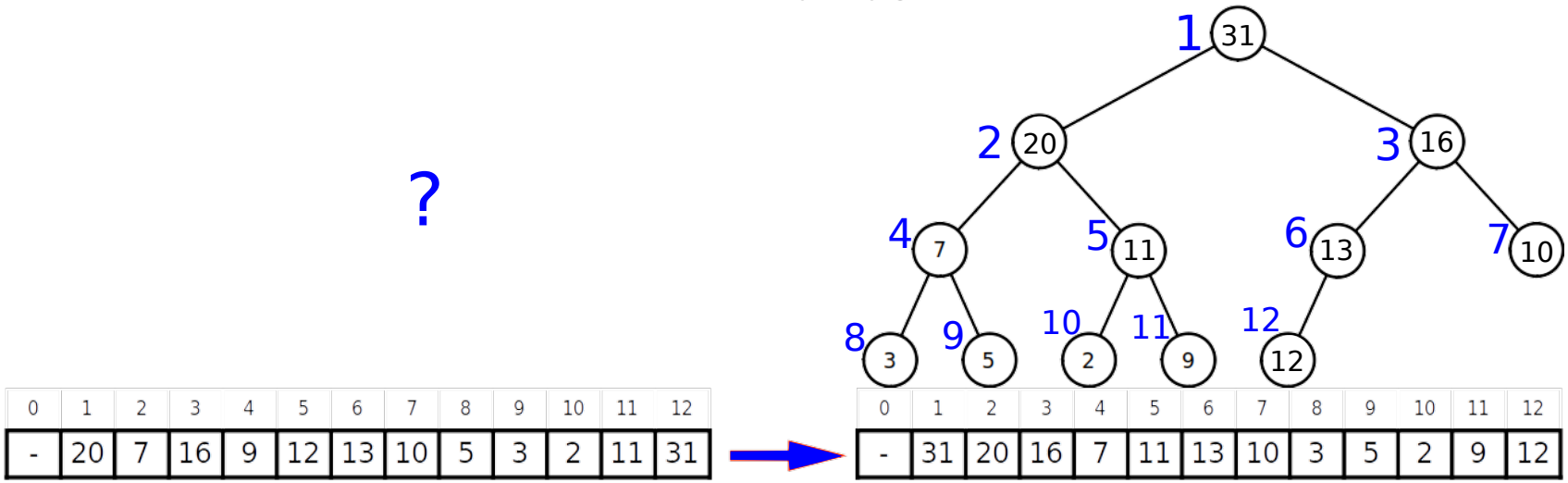
Prioritetskøer og Hobe

- Prioritetskøer
- Træer og hobe
- Repræsentation af hobe
- Algoritmer på hobe
- Hobkonstruktion
- Hobsortering

Hobkonstruktion

Hobkonstruktion: Givet n heltal i et array, byg en hob.

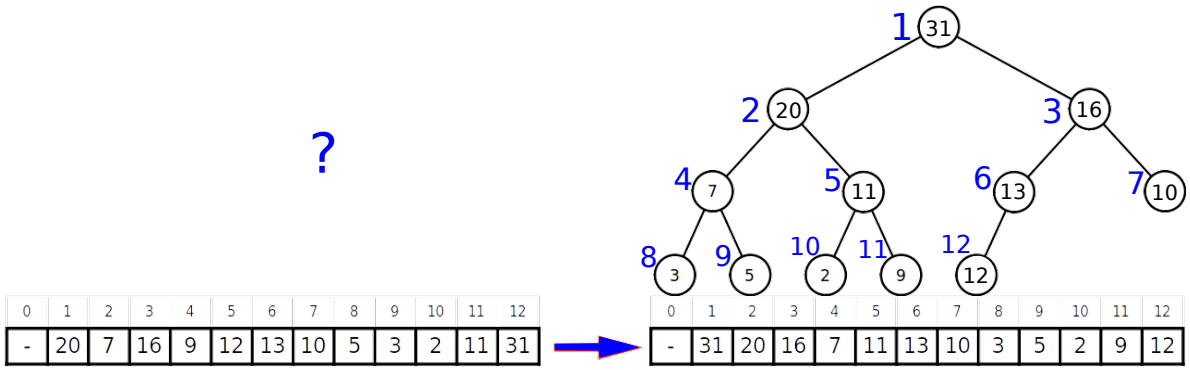
?



Hobkonstruktion – algoritme 1

Algoritme 1: oppefra og ned (fra venstre til højre)

For alle knuder i stigende niveauorden, kør `BubbleUp`.



Tid:

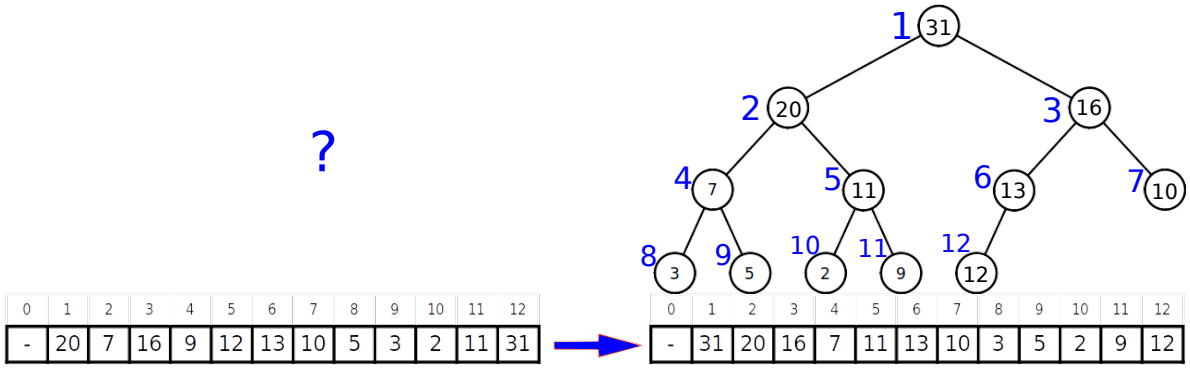
- Knude af dybde d: $O(d)$ tid på `BubbleUp`.
- ca. $n/2$ knuder af dybde $\log(n)$, ca. $n/4$ knuder af dybde $\log(n) - 1$, ca. $n/8$ knuder af dybde $\log(n) - 2$, ..., 2 knuder af dybde 1
- $n/2 \log n + n/4(\log(n) - 1) + n/8(\log(n) - 2) + \ldots + 2 \cdot 1 = \Theta(n \log n)$

Udfordring: kan vi gøre det bedre?

Hobkonstruktion – algoritme 2

Algoritme 1: nedefra og op (fra højre til venstre)

For alle knuder i faldende niveauorden, kør `BubbleDown`.



Tid:

- Knude af højde h: $O(h)$ tid på `BubbleDown`.
- ca. $n/4$ knuder af højde 1, ca. $n/8$ knuder af højde 2, ca. $n/16$ knuder af højde 3, ..., 1 knude af højde $\log n$,
- $n/4 \cdot 1 + n/8 \cdot 2 + n/16 \cdot 3 + \ldots + 1 \cdot \log n = \Theta(n)$,
- $\Theta(n)$ tid.

Prioritetskøer og Hobe

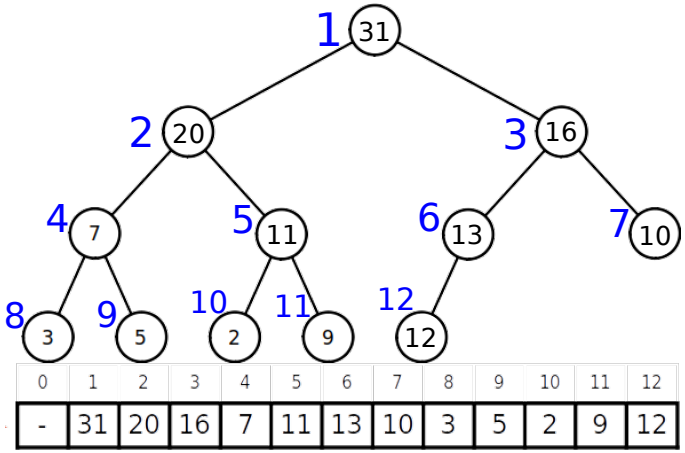
- Prioritetskøer
- Træer og hobe
- Repræsentation af hobe
- Algoritmer på hobe
- Hobkonstruktion
- Hobsortering

Hobsortering

Sortering: Hvordan kan man bruge en hob og prioritetskøoperationer til at sortere et array `H[0..n]` af n tal?

Algoritme:

- Konstruér en hob for H.
- Kald `ExtractMax` (n gange)
  - Indsæt resultaterne på plads n, $n-1$, $n-2$, osv.
- Returner arrray H.



Tid

- Hobkonstruktion i $\Theta(n)$ tid,
- n gange `ExtractMax` i samlet $\Theta(n \log n)$ tid,
- i alt: $\Theta(n \log n)$ tid.

Prioritetskøer og Hobe

- Prioritetskøer
- Træer og hobe
- Repræsentation af hobe
- Algoritmer på hobe
- Hobkonstruktion
- Hobsortering