

# Orthogonal Range Searching in 2D using Ball Inheritance

Mads Ravn

Computer Science, Aarhus University

2015

- Giv en præsentation af den i specialet introducerede simplificerede datastruktur til range searching i 2d. (3)
- Beskriv ball-inheritance problemet og forklar sammenhængen til range searching. (2)
- Beskriv også det klassiske kd-træ (1)
- og fortæl om hvilke eksperimenter du har foretaget for at sammenligne performance af de to strukturer. Forklar hvad du så og om det var som forventet. (4)

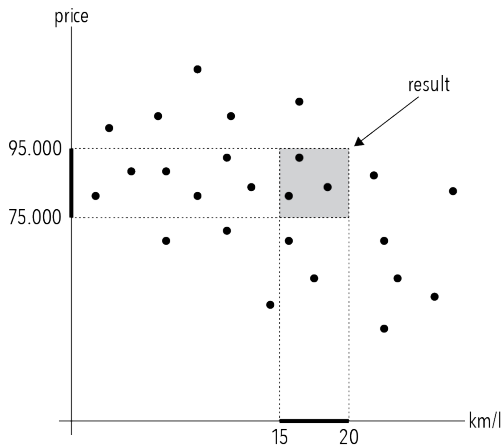
# Outline

- 1 Introduction
  - Orthogonal Range Searching i 2D
  - kd-tree
- 2 Ball Inheritance Search
  - Ball Inheritance Problem
  - Ball Inheritance Search Data Structure
- 3 Resultater
  - Resultater

# Outline

- 1 Introduction
  - Orthogonal Range Searching i 2D
  - kd-tree
- 2 Ball Inheritance Search
  - Ball Inheritance Problem
  - Ball Inheritance Search Data Structure
- 3 Resultater
  - Resultater

# Orthogonal Range Searching i 2D



Hvert punkt i 2D  
søgerum rep.  
objekt.  
En rektangulær  
søgeboks er  
defineret ved  
Opdeling af akser.

# Preliminaries

## Orthogonal range searching

- Svar effektivt på forespørgslen  $q = [x_1, x_2] \times [y_1, y_2]$ .
- $p \in [x_1, x_2] \times [y_1, y_2] \Leftrightarrow p_x \in [x_1, x_2] \wedge p_y \in [y_1, y_2]$

## Preliminaries

- $n$  punkter fra  $\mathbb{R}^2$ . Alle koordinater er unikke
- Rank space. Sorteret array. Vi finder  $\hat{y}_1$  og  $\hat{y}_2$  og så ved vi hvor mange elementer der er imellem dem.
- $n$  er en potens af 2
- static og output-sensitive

# Outline

- 1 Introduction
  - Orthogonal Range Searching i 2D
  - kd-tree
- 2 Ball Inheritance Search
  - Ball Inheritance Problem
  - Ball Inheritance Search Data Structure
- 3 Resultater
  - Resultater

# kd-træ

Jon L. Bentley. 1975.

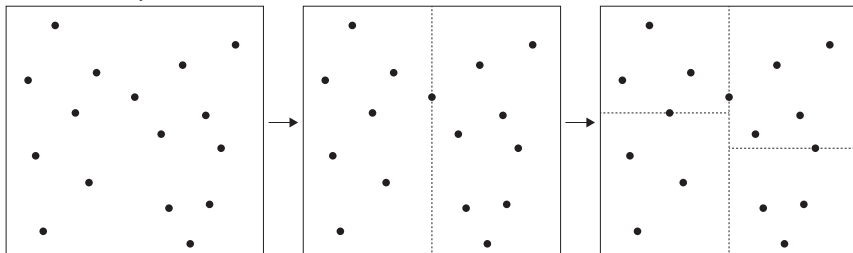
- $\mathcal{O}(n)$  plads
- $\mathcal{O}(\sqrt{n} + k)$  tid

Konstruktion: Givet  $n$  punkter,  $x$  eller  $y$  på skift. Et punkt per blad i træet.



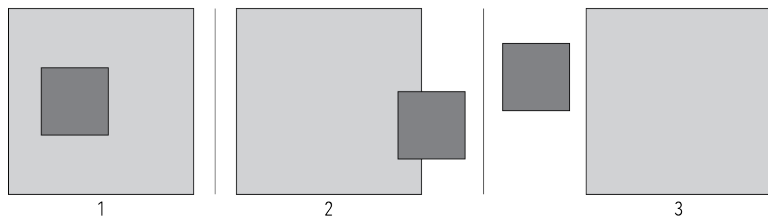
# Opbygning af kd-træ

Det  $\lceil \frac{n}{2} \rceil$ 'te element bliver valgt som median. Skille-linje. Knudens område er punkter i undertræ.



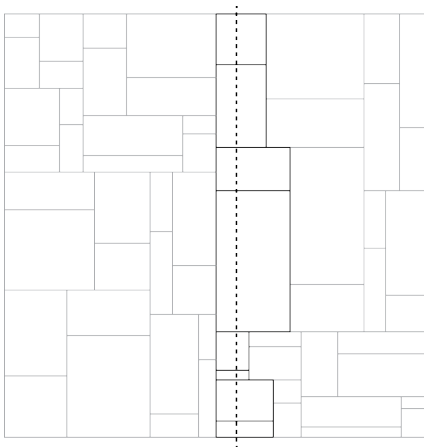
# Søgning i kd-træ

■ = node region  
 □ = search query



1 tager  $\mathcal{O}(k)$  tid, 3 tager  $\mathcal{O}(1)$  tid. Bound på 2.

# Søgning i kd-træ



$\mathcal{O}(\sqrt{n})$  regioner.  
 $\mathcal{O}(\sqrt{n} + k)$  tid.

# Outline

- 1 Introduction
  - Orthogonal Range Searching i 2D
  - kd-tree
- 2 Ball Inheritance Search
  - Ball Inheritance Problem
  - Ball Inheritance Search Data Structure
- 3 Resultater
  - Resultater

# Ball Inheritance

- Vi er givet et perfekt binært træ.

# Ball Inheritance

- Vi er givet et perfekt binært træ.
- Roden indeholder  $n$  punkter(bolde) som er blevet fordelt sådan at hvert blad lagrer et punkt. Boldene i roden er sorteret.

# Ball Inheritance

- Vi er givet et perfekt binært træ.
- Roden indeholder  $n$  punkter(bolde) som er blevet fordelt sådan at hvert blad lagrer et punkt. Boldene i roden er sorteret.
- Hver knude har en liste over hvilke bolde der går igennem knuden. Boldene i knudens liste har samme rækkefølge som boldene i forældre-knudens liste.

# Ball Inheritance

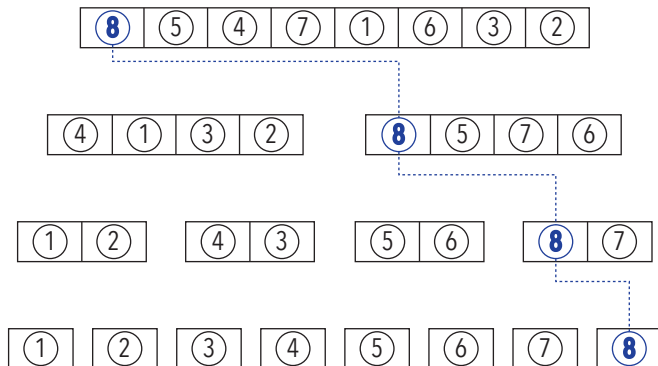
- Vi er givet et perfekt binært træ.
- Roden indeholder  $n$  punkter(bolde) som er blevet fordelt sådan at hvert blad lagrer et punkt. Boldene i roden er sorteret.
- Hver knude har en liste over hvilke bolde der går igennem knuden. Boldene i knudens liste har samme rækkefølge som boldene i forældre-knudens liste.
- Løs: Givet en knude og et index i knudens liste, hvilket blad ender denne bold ved?



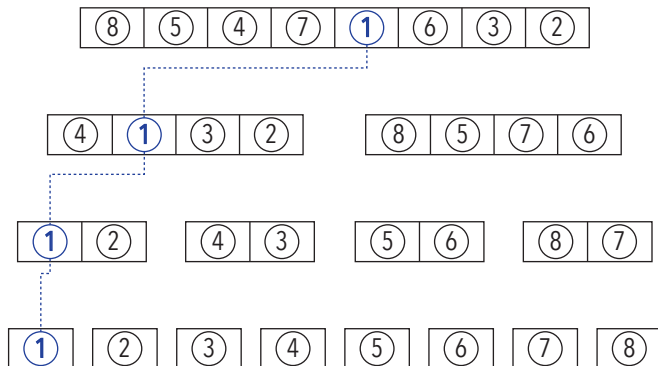
# Ball Inheritance

- Vi er givet et perfekt binært træ.
- Roden indeholder  $n$  punkter(bolde) som er blevet fordelt sådan at hvert blad lagrer et punkt. Boldene i roden er sorteret.
- Hver knude har en liste over hvilke bolde der går igennem knuden. Boldene i knudens liste har samme rækkefølge som boldene i forældre-knudens liste.
- Løs: Givet en knude og et index i knudens liste, hvilket blad ender denne bold ved?
- Vi ønsker at følge en bold ned til bladet for at “dekodere” punktets koordinater.

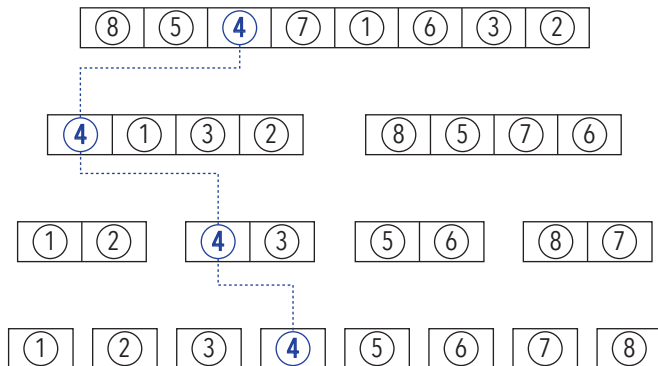
# Ball Inheritance



# Ball Inheritance



# Ball Inheritance

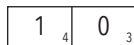
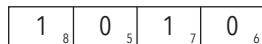
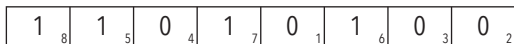


# Rank-select

Givet en bitvektor, så er rank-select query en constant-time query der finder boldens nye position i barnets bitvektor.

Man kan nu komme ned med  $\mathcal{O}(\lg n)$  tid.  $\mathcal{O}(1 \cdot \lg n)$ . Fylder  $\mathcal{O}(n)$  bits per level.

# Ball Inheritance



# Faster Queries

Færre skridt fra knude til blad. Knuder på niveau deleligt med  $B^i$  hopper  $B^i$  niveauer over. Det er  $\mathcal{O}(B^i)$  bits per bold per  $B^i$ te level.  $B = \Omega(\lg^\epsilon n)$ .

Det bruger

- $\mathcal{O}(\frac{n}{\epsilon}) = \mathcal{O}(n)$  plads
- $\mathcal{O}(\lg^\epsilon n)$  tid

hvor  $\epsilon > 0$  er en arbitrær lille konstant. Space-time tradeoff.

# Outline

- 1 Introduction
  - Orthogonal Range Searching i 2D
  - kd-tree
- 2 Ball Inheritance Search
  - Ball Inheritance Problem
  - Ball Inheritance Search Data Structure
- 3 Resultater
  - Resultater



# BISintro

Ball Inheritance Search (BIS) er en datastruktur som er en simplificering af den datastruktur der findes i **Orthogonal Range Searching on the RAM, Revisited**[1] af Chan et al.

$\mathcal{O}(\lg n + k \cdot \lg^\epsilon n)$  imod  $\mathcal{O}(\lg \lg n + (1 + k) \cdot \lg^\epsilon n)$ .

# Ball Inheritance Search

- $\mathcal{O}(n + \frac{n}{\epsilon})$  plads.
- $\mathcal{O}(\lg n + k \cdot \lg^\epsilon n)$  tid, hvor  $\epsilon > 0$  er en arbitrær lille konstant

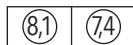
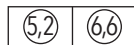
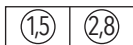
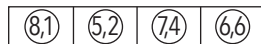
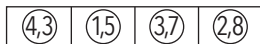
da vi har valgt  $B = \Omega(\lg^\epsilon n) = \lceil \frac{1}{2} \lg^{\frac{1}{3}} n \rceil$ .

# Ball Inheritance Search

Specifikt til BIS, så er punkterne **fordelt** efter  $x$  og **sorteret** efter  $y$ . Det betyder

- Undertræer i knuder i mellem  $\hat{x}_1$  og  $\hat{x}_2$  kun indeholder punkter i  $[x_1, x_2]$ .
- Bolde mellem  $\hat{y}_1$  og  $\hat{y}_2$  kun indeholder punkter i  $[y_1, y_2]$ . Ligesom fractional cascading.

# Ball Inheritance Search



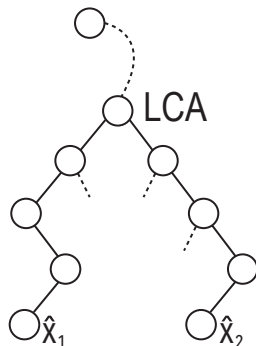
# Ball Inheritance Search

I denne datastruktur bruger vi ball-inheritance til

- Følge en bold ned når vi ved den ligger i vores søge-område. Dvs dekode fra bold-til-blad(punkt).
- Fra roden, opdatere et interval af hvilke bolder der ligger i  $[y_1, y_2]$ .

Nu har vi markeret de bolde der ligger i  $[y_1, y_2]$  på de knuder hvis undertræ kun indeholder punkter i  $[x_1, x_2]$ . Og det er netop de punkter der ligger i  $[x_1, x_2] \times [y_1, y_2]$ .

# Ball Inheritance Search



- Rank space opslag ved roden.
- Vedligehold  $[\hat{y}_1, \hat{y}_2]$  ned til LCA.
- Find fully contained knuder og deres  $[\hat{y}_1, \hat{y}_2]$  interval.
- Ball Inheritance fra knuder.
- $p \in [x_1, x_2] \times [y_1, y_2] \Leftrightarrow ..$

# Ball Inheritance Search

Vi har nu nogle knuder og lister over indeces i disse knuder. Det er præcis det problem ball inheritance løser. Vi kan nu bruge ball inheritance på alle disse knuder til at finde ud af hvilke blade der indeholder punkter i  $[y_1, y_2]$ .

Det giver en kørselstid på  $\mathcal{O}(\lg n + k \cdot \lg^\epsilon n)$  for at finde  $k$  punkter.

# plads

Denne datastruktur bruger  $\mathcal{O}(n)$  plads.

- Bit vectors.  $\mathcal{O}(n)$  bits per level.



# plads

Denne datastruktur bruger  $\mathcal{O}(n)$  plads.

- Bit vectors.  $\mathcal{O}(n)$  bits per level.
- Store hop  $\mathcal{O}(\lg \Sigma)$  bits per entry hvert  $\lg \Sigma$  level. ( $\lg \Sigma = B^i$ ).

# plads

Denne datastruktur bruger  $\mathcal{O}(n)$  plads.

- Bit vectors.  $\mathcal{O}(n)$  bits per level.
- Store hop  $\mathcal{O}(\lg \Sigma)$  bits per entry hvert  $\lg \Sigma$  level. ( $\lg \Sigma = B^i$ ).
- Egentlig punkter

# plads

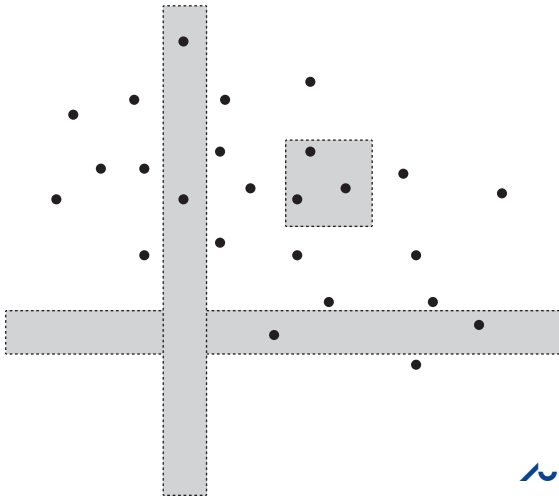
Denne datastruktur bruger  $\mathcal{O}(n)$  plads.

- Bit vectors.  $\mathcal{O}(n)$  bits per level.
- Store hop  $\mathcal{O}(\lg \Sigma)$  bits per entry hvert  $\lg \Sigma$  level. ( $\lg \Sigma = B^i$ ).
- Egentlig punkter
- Binær søgning

# Outline

- 1 Introduction
  - Orthogonal Range Searching i 2D
  - kd-tree
- 2 Ball Inheritance Search
  - Ball Inheritance Problem
  - Ball Inheritance Search Data Structure
- 3 Resultater
  - Resultater

# Setup



# Setup

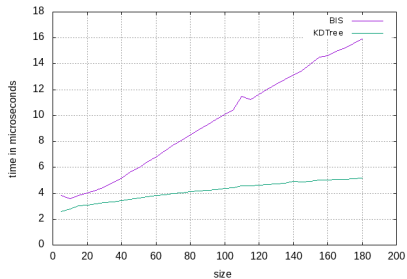
- Square area  $\sqrt{n} \cdot \sqrt{k} \times \sqrt{n} \cdot \sqrt{k}$  returnerer  $k$  punkter.
- Slices af størrelse  $k$  returnerer  $k$  punkter.  $[0, n] \times [y, y + k]$
- $\sqrt{n} + k = \lg n + k \cdot \lg^\epsilon n \Leftrightarrow k = \frac{\sqrt{n} - \lg n}{\lg^\epsilon n - 1}$

# Setup

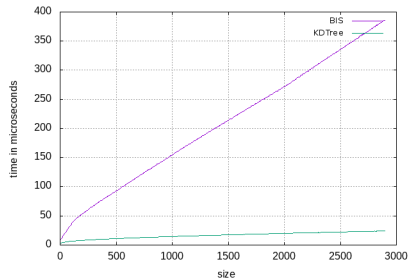
- $\mathcal{O}(\lg n + k \cdot \lg^\epsilon n)$  vs  $\mathcal{O}(\sqrt{n} + k)$ .  $y = ax + b$
- BIS er mere stabil når vi ændrer shape.  $\mathcal{O}(\sqrt{n})$  er problemet.
- Slice er godt for BIS og square er godt for kd-træ.
- kd-træ er hurtigere jo flere punkter, og jo mindre  $\mathcal{O}(\sqrt{n})$  er.

# Squared

KDTree vs BIS - squared with  $\lg n = 17$

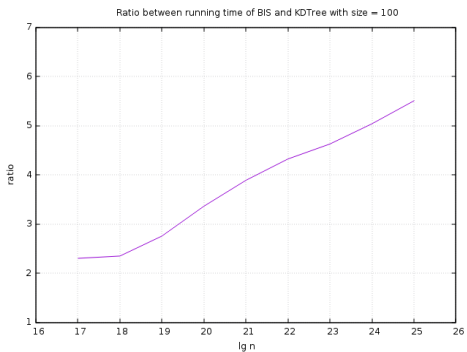


KDTree vs BIS - squared with  $\lg n = 25$



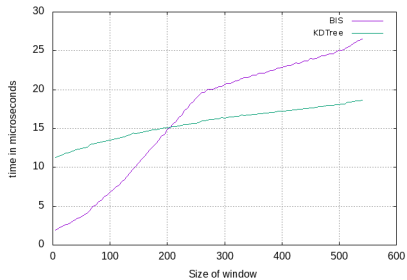


## Squared

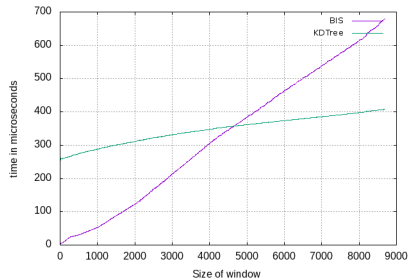


# Vertical

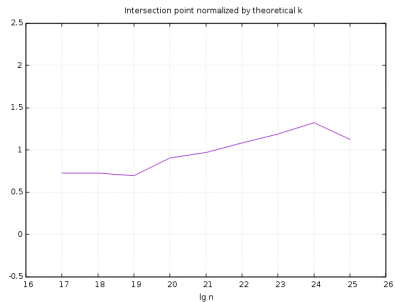
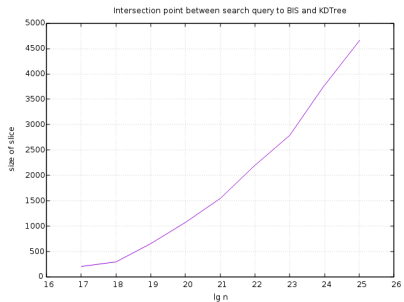
KDTree vs BIS - vertical 17



KDTree vs BIS - vertical 25

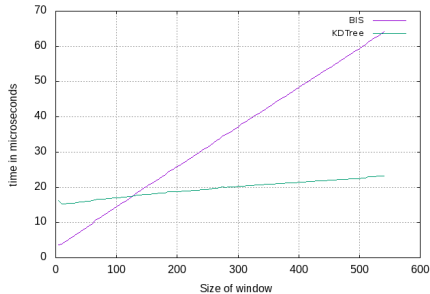


# Vertical

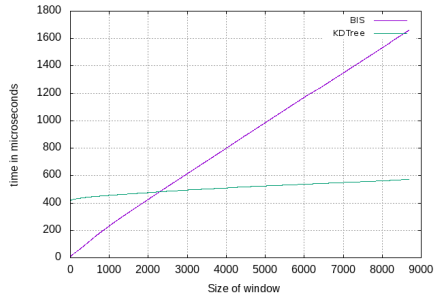


# Horizontal

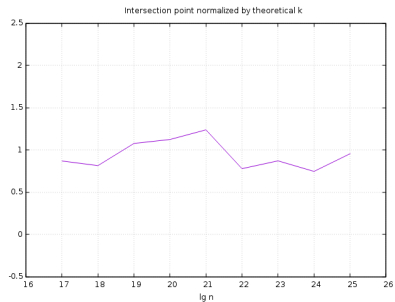
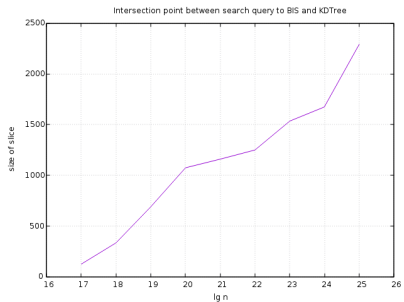
KDTree vs BIS - horizontal 17



KDTree vs BIS - horizontal 25

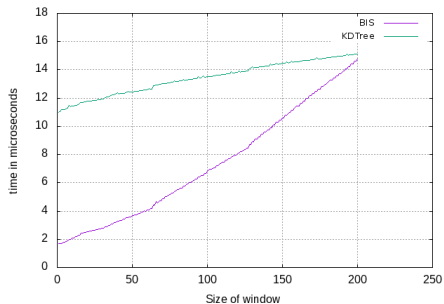


# Horizontal

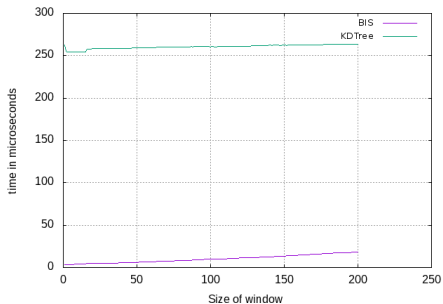


# Small k

KDTree vs BIS - vertical 17

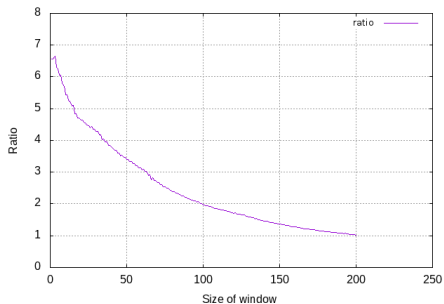


KDTree vs BIS - vertical 25

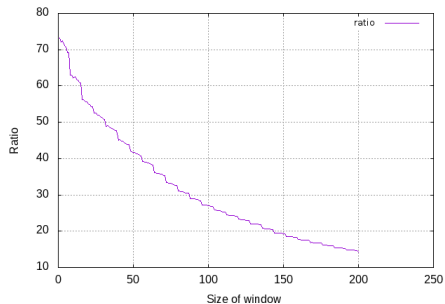


# Small k

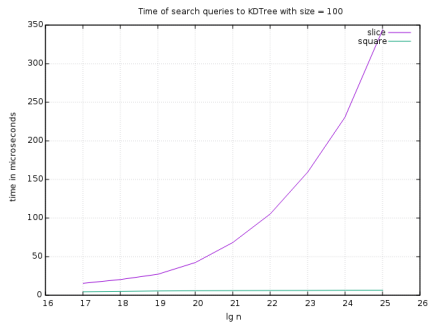
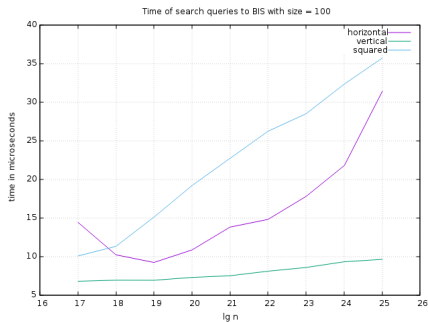
KDTree vs BIS - vertical 17



KDTree vs BIS - vertical 25

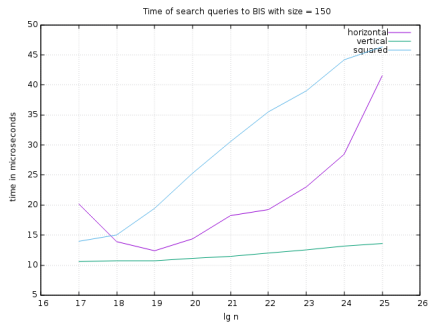
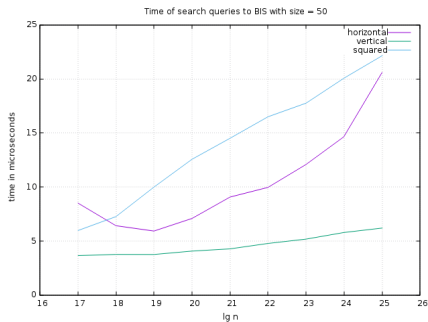


# Shapes

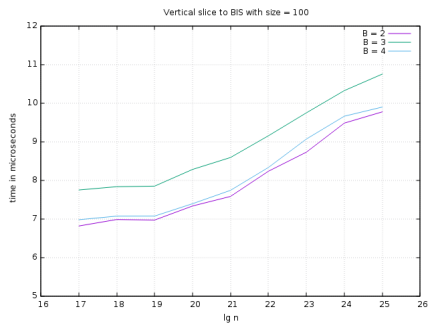
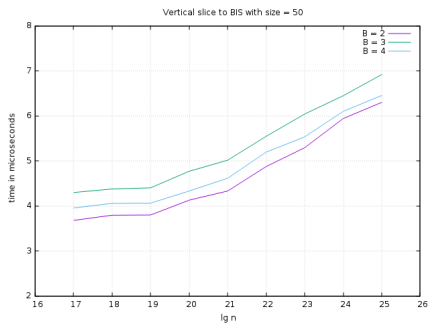




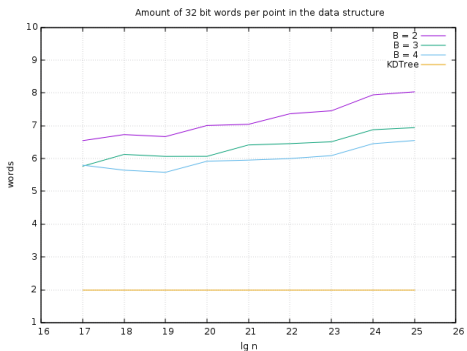
# Shapes



# Different B



## Sizes



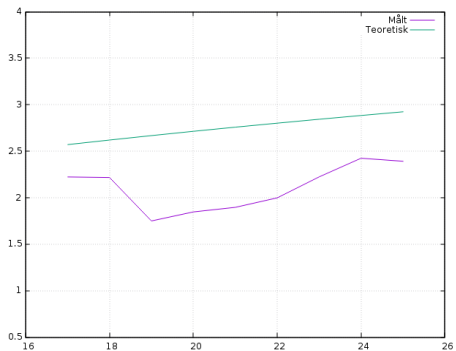
# Future work

- Bedre bit-fiddling
- Cache
- Concurrency

Færdig

Spørgsmål?

# praktisk log epsilon n



# Små hop

Hvert niveau gemmer  $n$  bits som indikerer om bolden er gået til højre eller venstre. Hvert 32 bit gemmer vi et 32 bit major checkpoint. Precomputed tabel med 16 bit tal som tæller antal 1-entries.  $\mathcal{O}(n)$  bits per level.

# Store hop

$\mathcal{O}(\lg \Sigma)$  per entry.  $\Sigma = 2^{B^i}$ . Så plads er  $\mathcal{O}(B^i)$  bits per entry.  
Det er

$$\sum_{i=1}^{\lg_B \lg n} \frac{\lg n}{B^i} \cdot \mathcal{O}(B^i) = \mathcal{O}(\lg n \cdot \lg_B \lg n)$$

for hele kæden. Vælg nu  $B = \Omega(\lg^\epsilon n)$ .

Vi har  $n$  punkter, hvilket giver  $\mathcal{O}(n \lg n \cdot \lg_B \lg n)$  bits. Det er  $\mathcal{O}(n \cdot \frac{\lg \lg n}{\epsilon \lg \lg n}) = \mathcal{O}(\frac{n}{\epsilon})$  ord.



# Store hop

Vi hopper højst  $B$  hop på  $B^i$  før vi rammer hop på  $B^{i+1}$ . Vi har at  $i = \lg_B \lg n$  er det største  $i$  sådan at  $B^i \leq \lg n$ .

$$\mathcal{O}(B \lg_B \lg n) = \mathcal{O}(\lg^\epsilon n)$$

$$B = \lg^{\epsilon/2} n = \Omega(\lg \lg n).$$

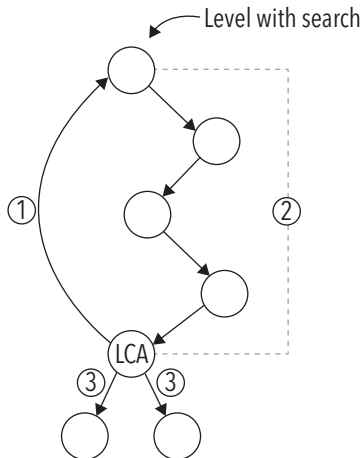
- Vi har  $B$  hop på  $B^0$  før vi rammer  $B^1$ .
- Vi har  $B$  hop på  $B^1$  før vi rammer  $B^2$ .
- Vi har  $B$  hop på  $B^2$  før vi rammer  $B^3$ .
- ...

Det er  $B \cdot i$  hvor  $i \leq \lg_B \lg n$ .

## OBIS

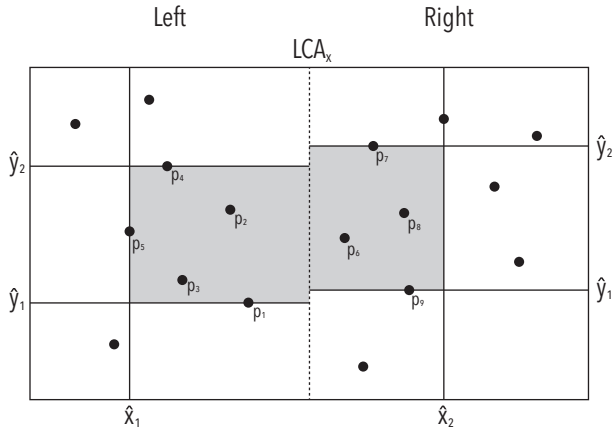
OBIS af Chan et al. Med  $\mathcal{O}(n)$  plads og  $\mathcal{O}(\lg \lg n + (1 + k) \cdot \lg^\epsilon n)$ .  
Bruger også Ball Inheritance til at finde de  $k$  punkter.

## OBIS



- Op til nærmeste level med pred-search
- Gå ned til LCA højst  $\lg \lg n$  levels nede.
- Gå ned og find resultater i begge børn af LCA.

## OBIS



$n$  punkter. Areal på  $n^2$ . Vi har nu at for hvert punkt er der  $\frac{A}{n^2}$  for at et punkt ligger i  $A$ . Dvs hvis vi har  $n$  punkter og et areal på  $\sqrt{n}\sqrt{k} \times \sqrt{nk}$ , må vi regne med at finde  $\frac{n \cdot k}{n}$  punkter.

## small k

