Orthogonal Range Searching in 2D using Ball Inheritance

Mads Ravn

Computer Science, Aarhus University

2015

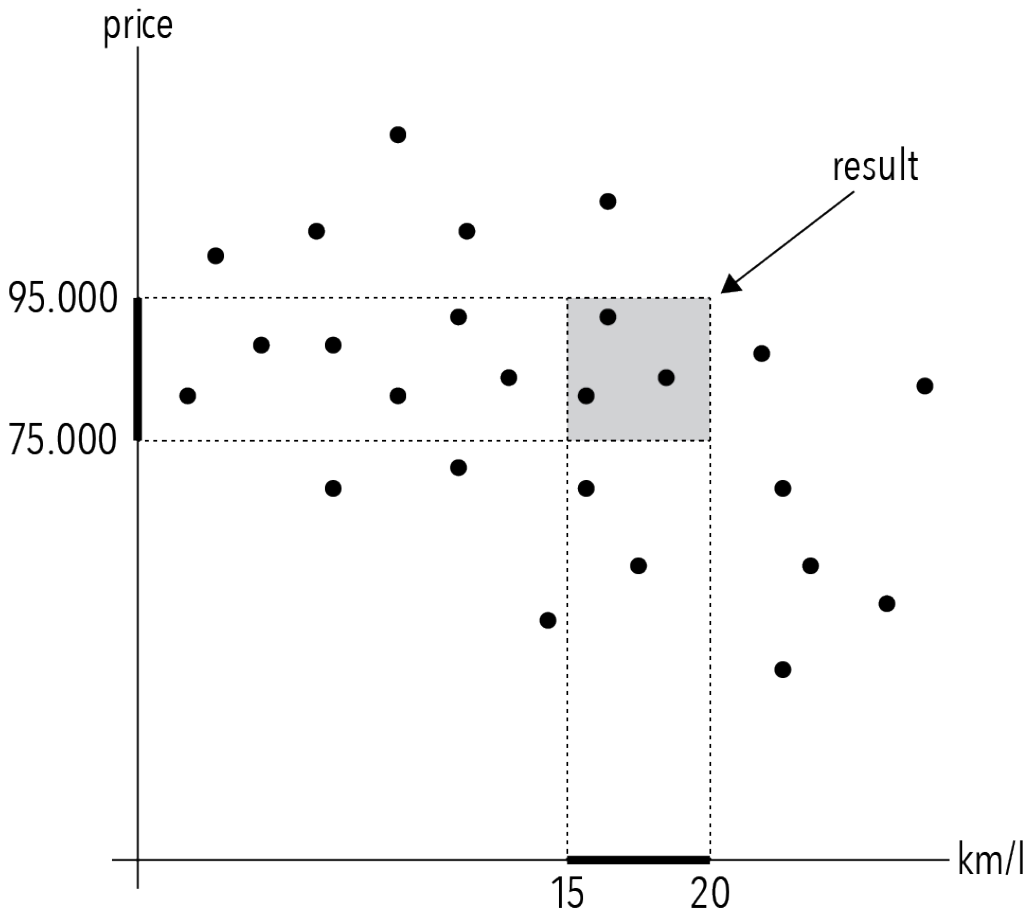
AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

- 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

Outline

Orthogonal Range Searching



AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Preleminaries

Orthogonal range searching

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

Preleminaries

- n punkter fra $\mathbb{R}^2$. Alle koordinater er unikke
- Rank space
- n er en potens af 2
- static og output-sensitive

Outline

kd-træ

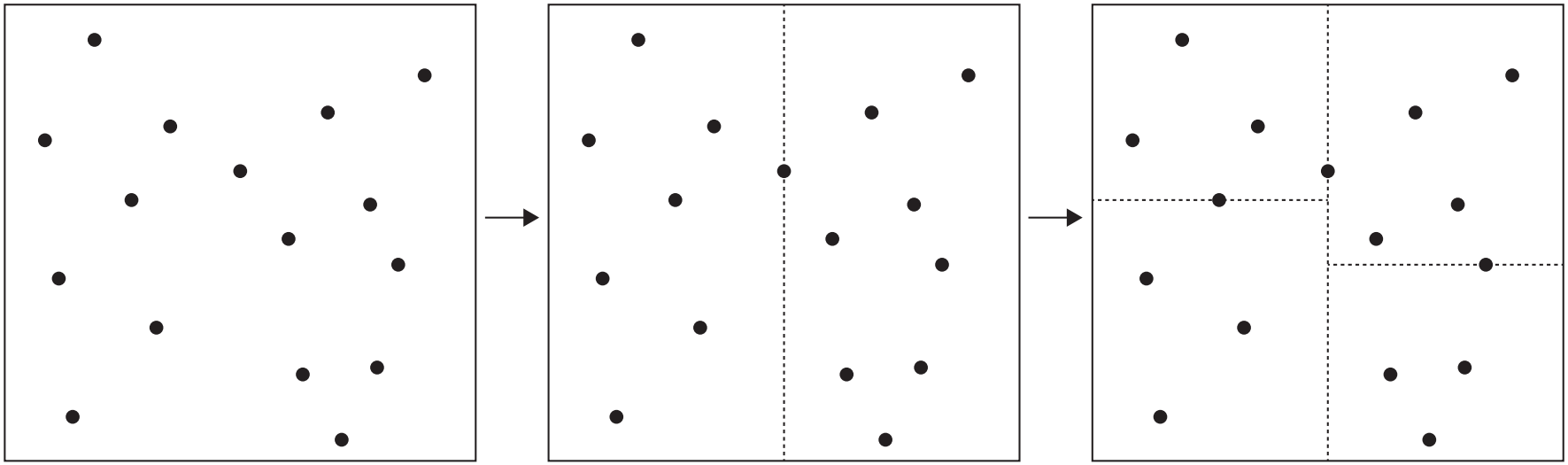
Jon L. Bentley. Multidimensional binary search trees used for associative searching. 1975.

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

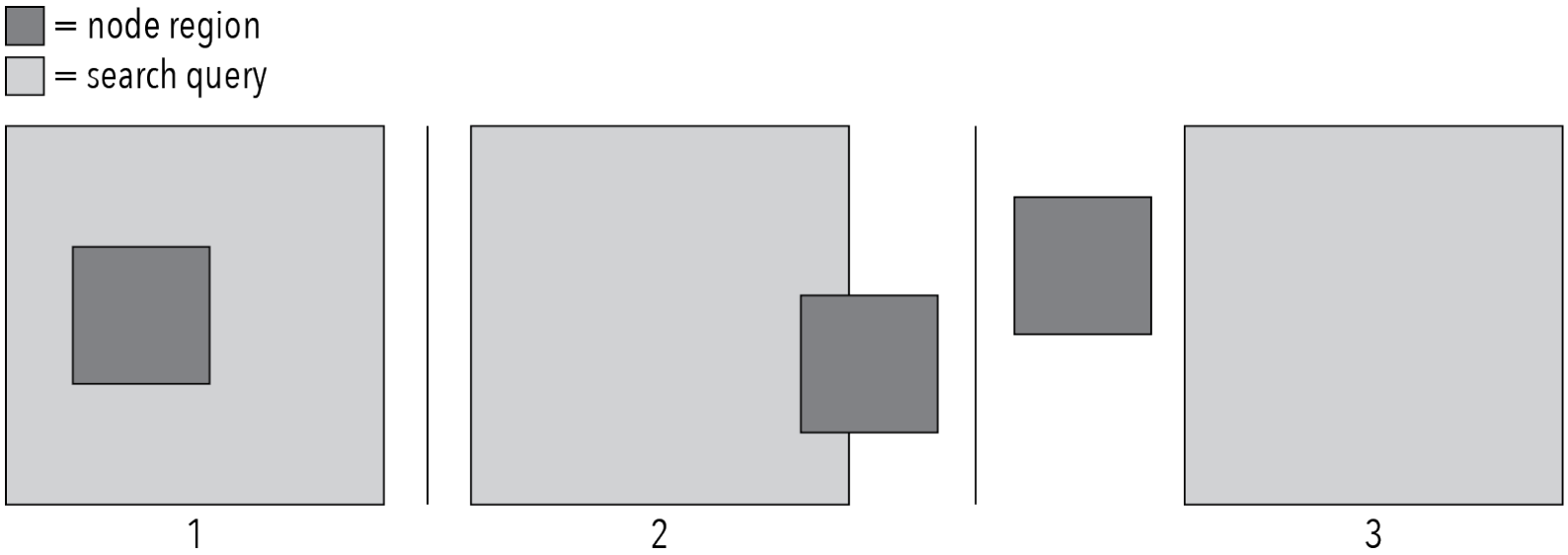
Givet n punkter: Punkterne bliver sorteret efter x eller y på skift. Median bliver fundet og punkterne mindre end medianen bliver givet til venstre barn og punkterne højere end medianen bliver givet til højre barn. Et punkt per blad i træet.

Opbygning af kd-træ

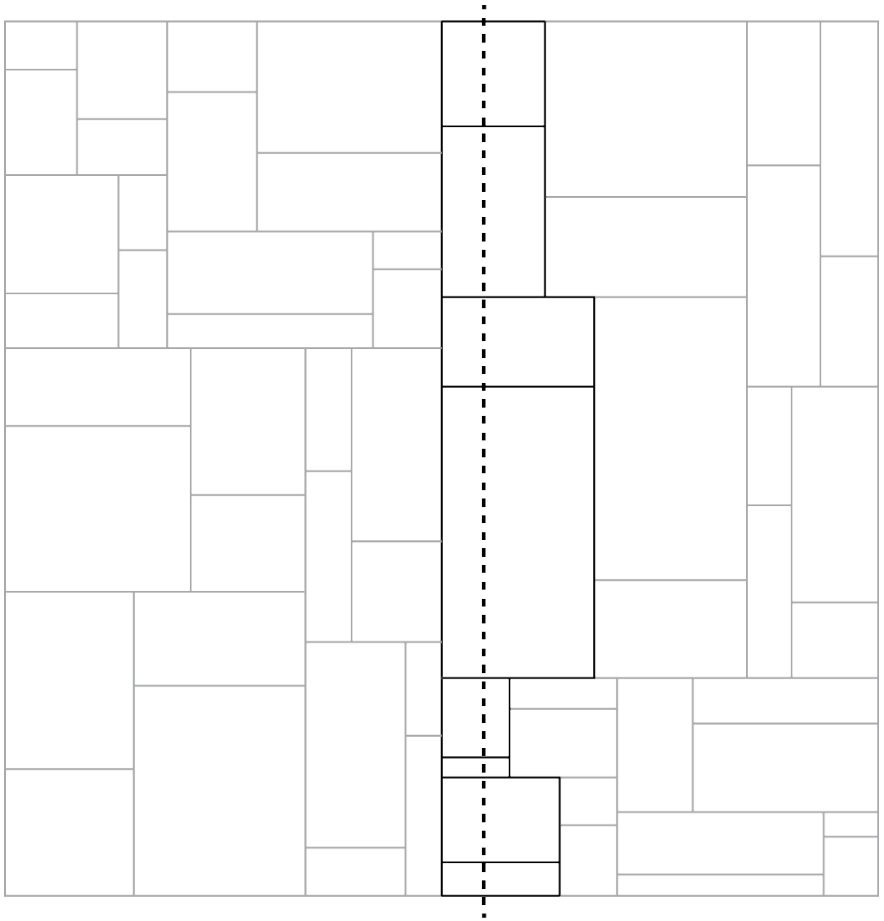
Det $\lceil \frac{n}{2} \rceil$'te element bliver valgt som median. Dette element fungerer som en skille-linje mellem de to punkt-mængder.
Medianen bliver låst fast
på denne plads i arrayet. Det er nu skillelinjen mellem de to regioner.



Søgning i kd-træ



Søgning i kd-træ



Outline

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 bolder der går igennem den. Boldene i knudens liste har samme rækkefølge som boldene i forældre-knudens liste.

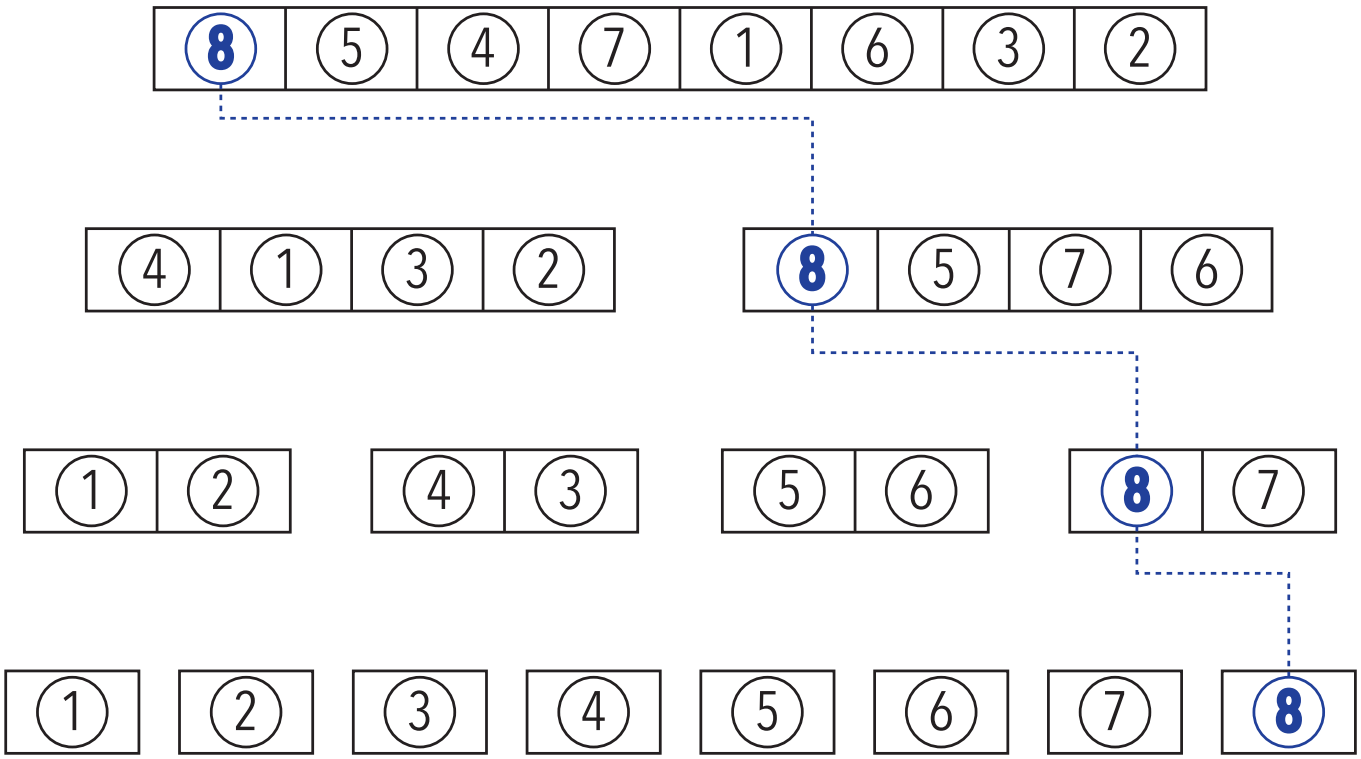
Ball Inheritance

- 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 bolder der går igennem den. 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?

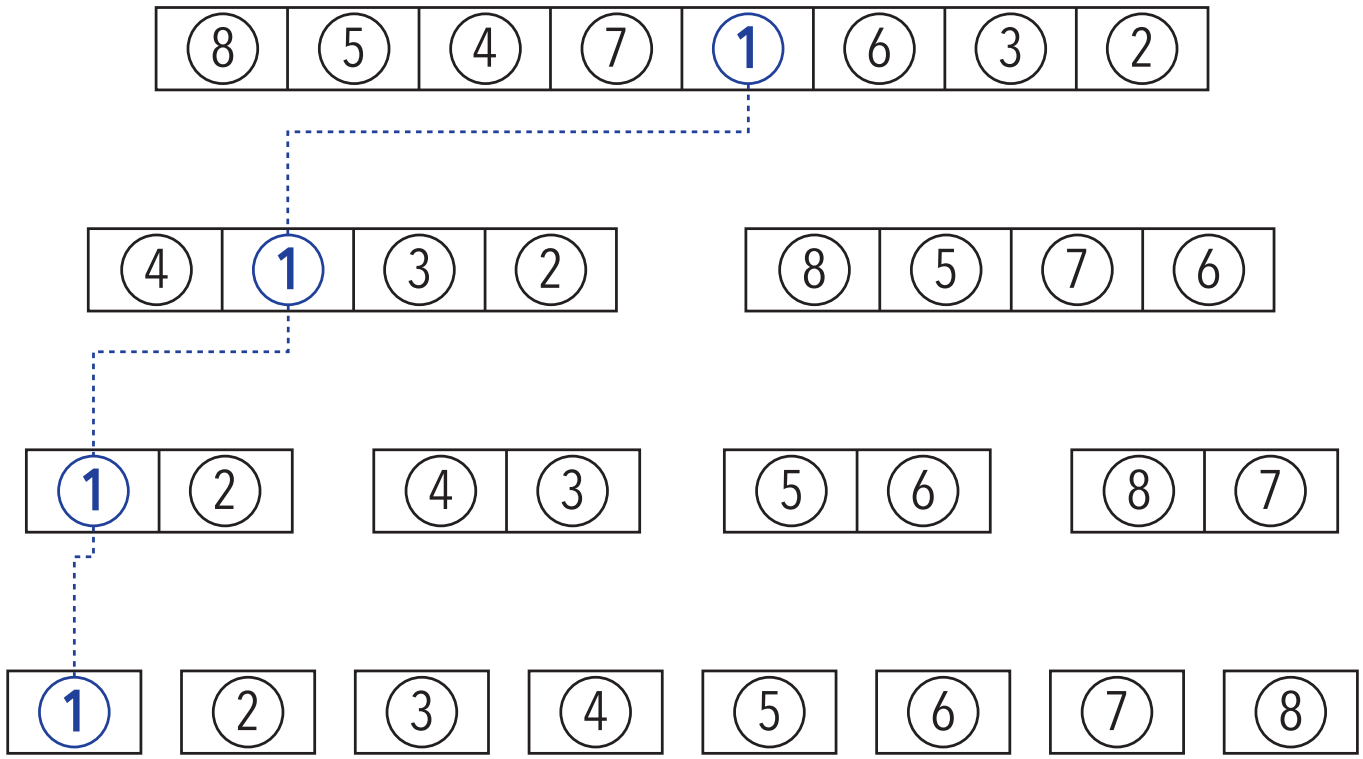
Ball Inheritance

- 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 bolder der går igennem den. 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 kan nu følge en bold fra en knude til et blad med $\mathcal{O}(\lg n)$ skridt.

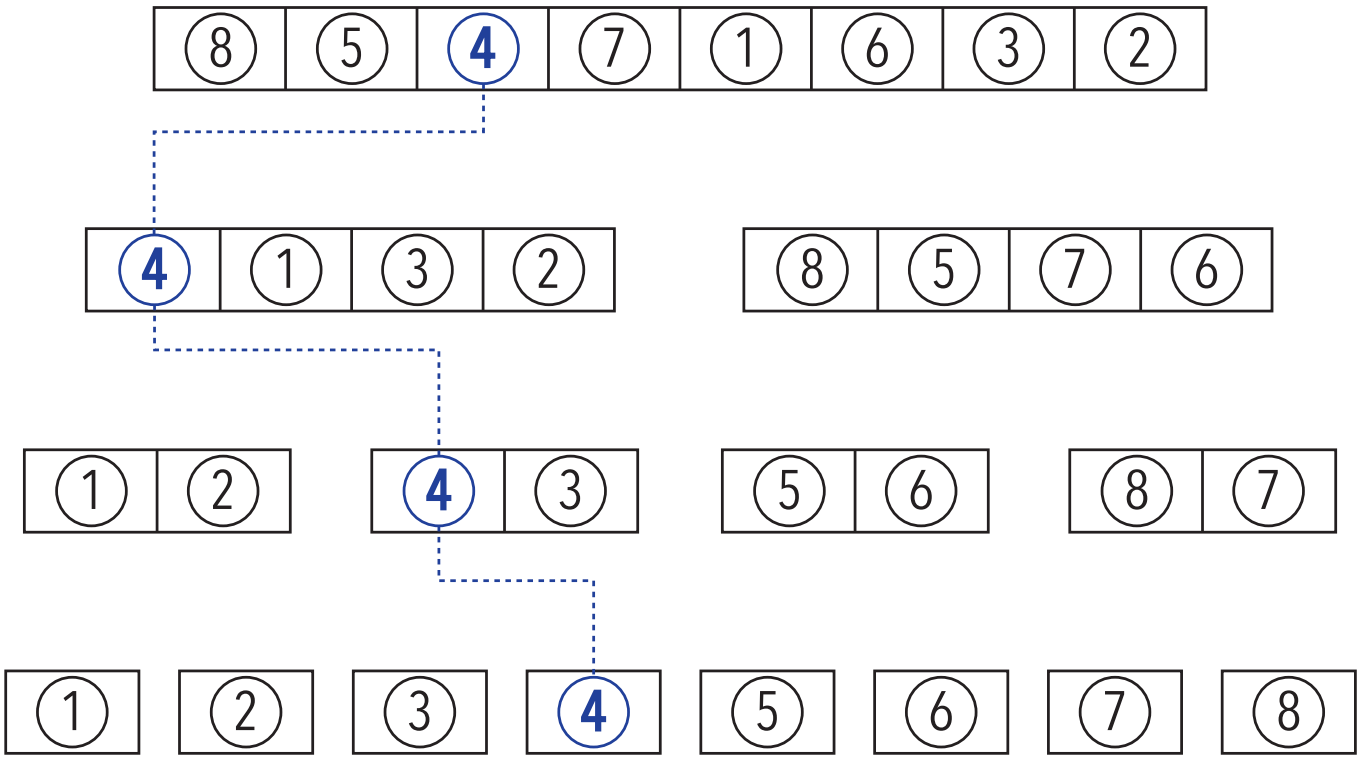
ball inheritance



ball inheritance



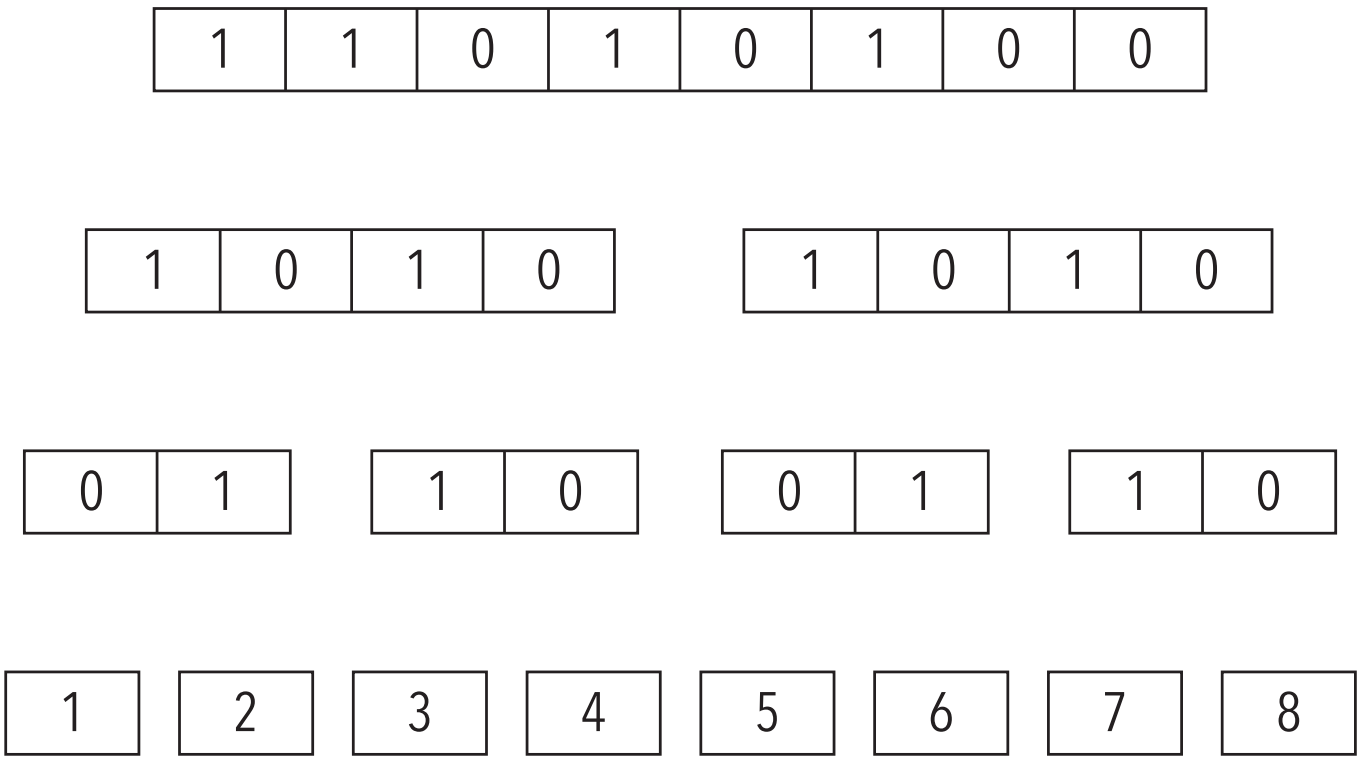
ball inheritance



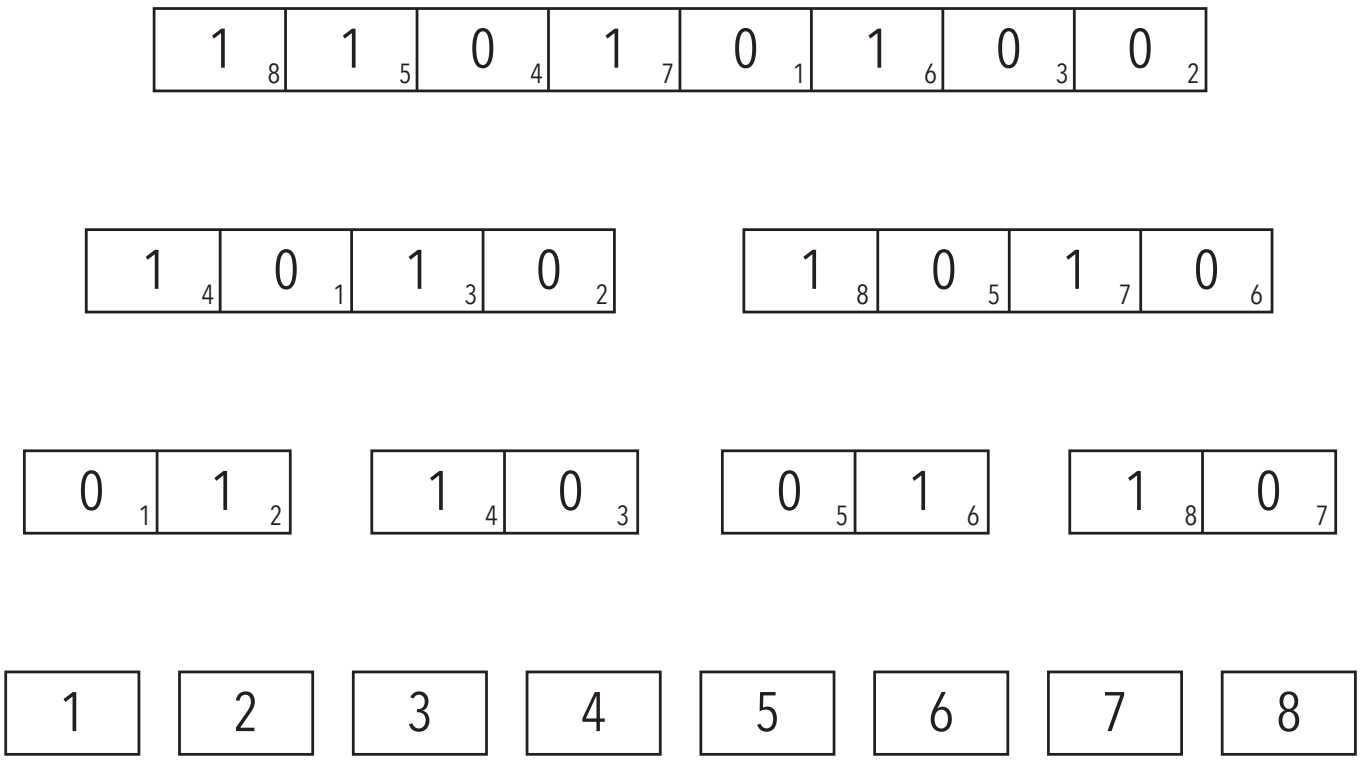
RANK SELECT

Rank-Select query er en constant-time query der kan afgøre hvor mange bolde før den i'te bold der går til et givet barn.
Teoretisk $rank_v(k) = \Sigma_{i \leq k} A_v[i]$. Til højde er det $rank_v(i)$ og til venstre er det $i - rank_v(i)$.
Husk RANK-SELECT struktur. Hvordan fungerer den? Med $\lg n$ og $\lg \Sigma$.

ball inheritance



ball inheritance



Faster Queries

Vi ønsker at gøre antallet skridt fra en knude til et blad mindre. Vi udvider alfabetet på udvalgte niveauer. 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. Vis koncept, tid og plads her

Outline

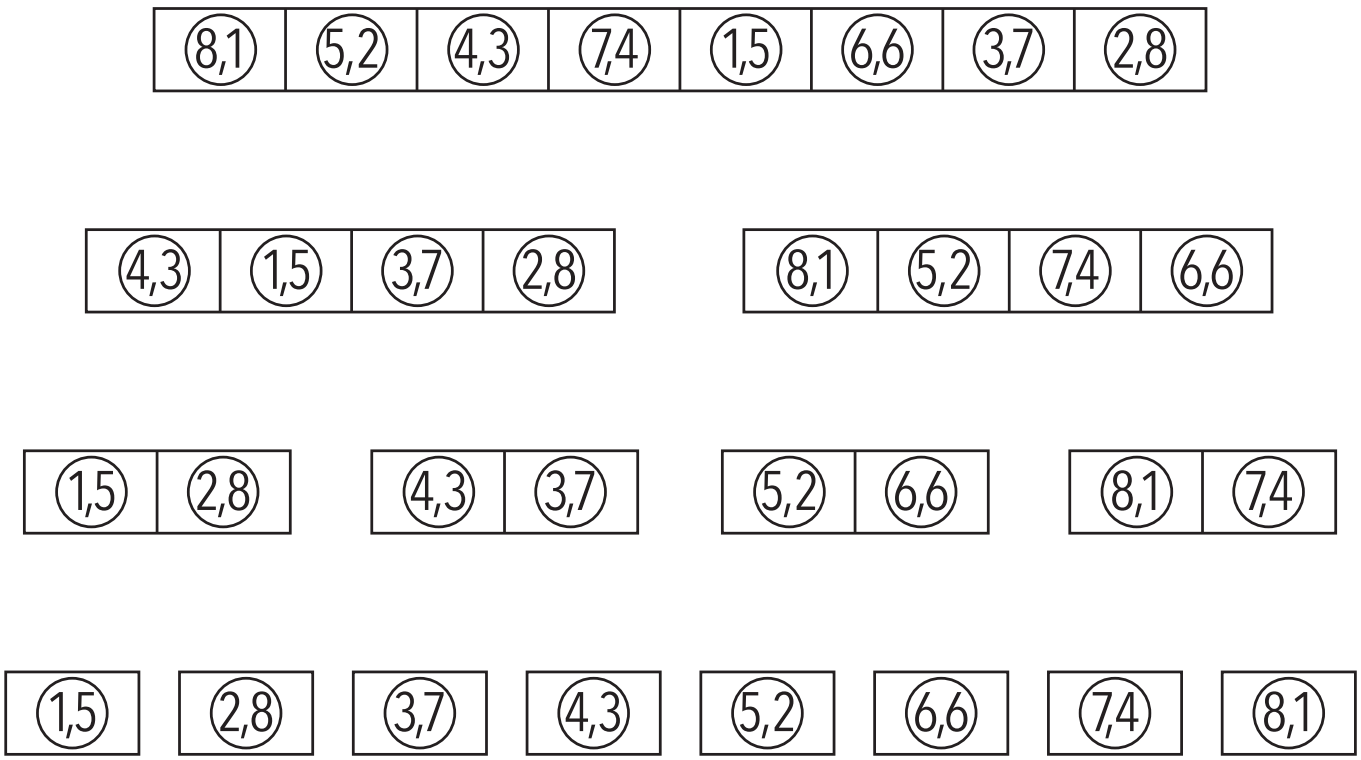
BISintro

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

Ball Inheritance Search

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

ball inheritance



AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

x-range

- Vi oversætter vores query til rank space
  $[x_1, x_2] \times [y_1, y_2] \Rightarrow [\hat{x}_1, \hat{x}_2] \times [\hat{y}_1, \hat{y}_2]$.

x-range

- Vi oversætter vores query til rank space $[x_1, x_2] \times [y_1, y_2] \Rightarrow [\hat{x}_1, \hat{x}_2] \times [\hat{y}_1, \hat{y}_2]$.
- Vi går ned til least common ancestor af $\hat{x}_1$ og $\hat{x}_2$ og herfra ned til $\hat{x}_1$ og $\hat{x}_2$. På den måde finder vi knuder der kun indeholder punkter i $[x_1, x_2]$.

y-range

- Vi har opdateret $[\hat{y}_1, \hat{y}_2]$ fra roden til både $\hat{x}_1$ og $\hat{x}_2$. Dvs vi ved hvilke bolde i hver knude vi fandt før der indeholder punkter i $[y_1, y_2]$.

y-range

- Vi har opdateret $[\hat{y}_1, \hat{y}_2]$ fra roden til både $\hat{x}_1$ og $\hat{x}_2$. Dvs vi ved hvilke bolde i hver knude vi fandt før der indeholder punkter i $[y_1, y_2]$.
- $[\hat{y}_1, \hat{y}_2]$ betegner det interval af y-koordinater der ligger mellem y_1 og y_2 i knuden v.

y-range

- Vi har opdateret $[\hat{y}_1, \hat{y}_2]$ fra roden til både $\hat{x}_1$ og $\hat{x}_2$. Dvs vi ved hvilke bolde i hver knude vi fandt før der indeholder punkter i $[y_1, y_2]$.
- $[\hat{y}_1, \hat{y}_2]$ betegner det interval af y-koordinater der ligger mellem y_1 og y_2 i knuden v.
- 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 $[y_1, y_2]$.

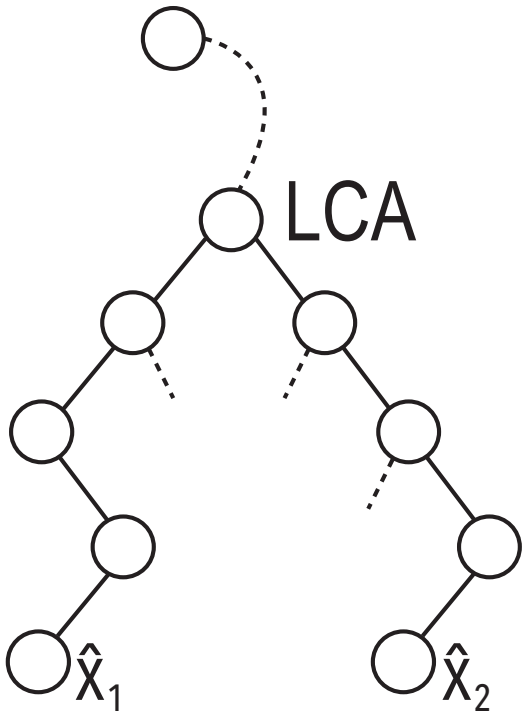
ballinheritance

Vi har nu at hver knude der er fully contained laver ball inheritance på det y-range den får givet. Det tager $\mathcal{O}(k \cdot \lg^{\epsilon} n)$ tid. Det tager $\mathcal{O}(\lg n)$ at lave binær søgning og at gå fra roden til $\hat{x}_1$ og $\hat{x}_2$.

ballinheritance

Vi har nu at hver knude der er fully contained laver ball inheritance på det y-range den får givet. Det tager $\mathcal{O}(k \cdot \lg^{\epsilon} n)$ tid. Det tager $\mathcal{O}(\lg n)$ at lave binær søgning og at gå fra roden til $\hat{x}_1$ og $\hat{x}_2$. Det giver en kørselstid på $\mathcal{O}(\lg n + k \cdot \lg^{\epsilon} n)$ for at finde k punkter.

ball inheritance



- 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.

AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

plads

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

- Bit vectors (Kommer vi til)

plads

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

- Bit vectors (Kommer vi til)
- Store hop (Kommer vi til)

plads

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

- Bit vectors (Kommer vi til)
- Store hop (Kommer vi til)
- Egentlig punkter

plads

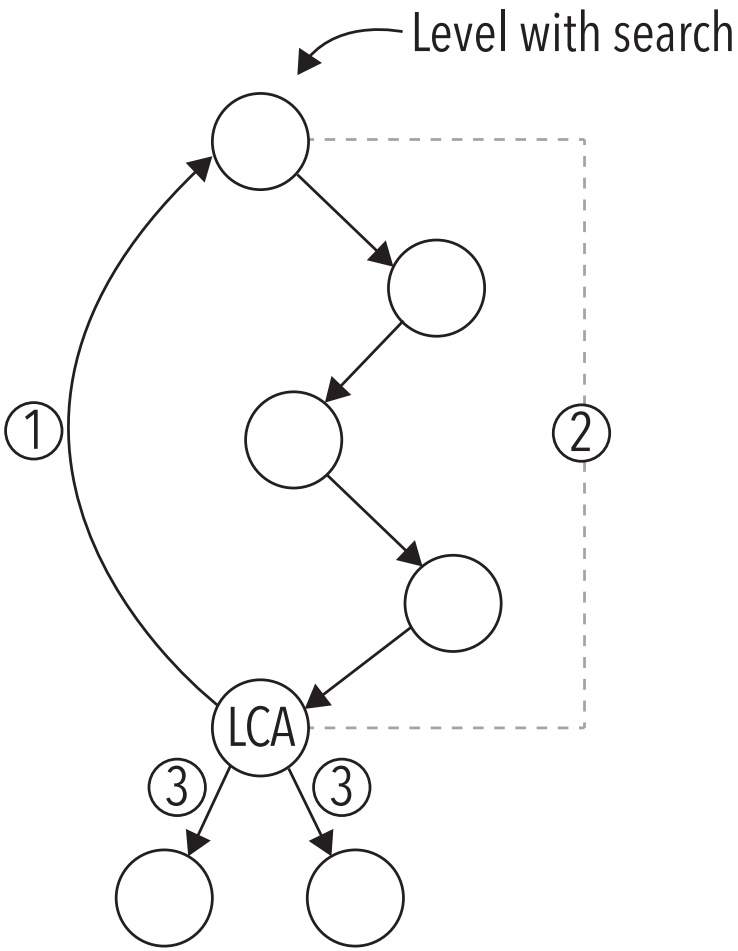
Denne datastruktur bruger $\mathcal{O}(n)$ plads.

- Bit vectors (Kommer vi til)
- Store hop (Kommer vi til)
- Egentlig punkter
- Binær søgning

OBIS

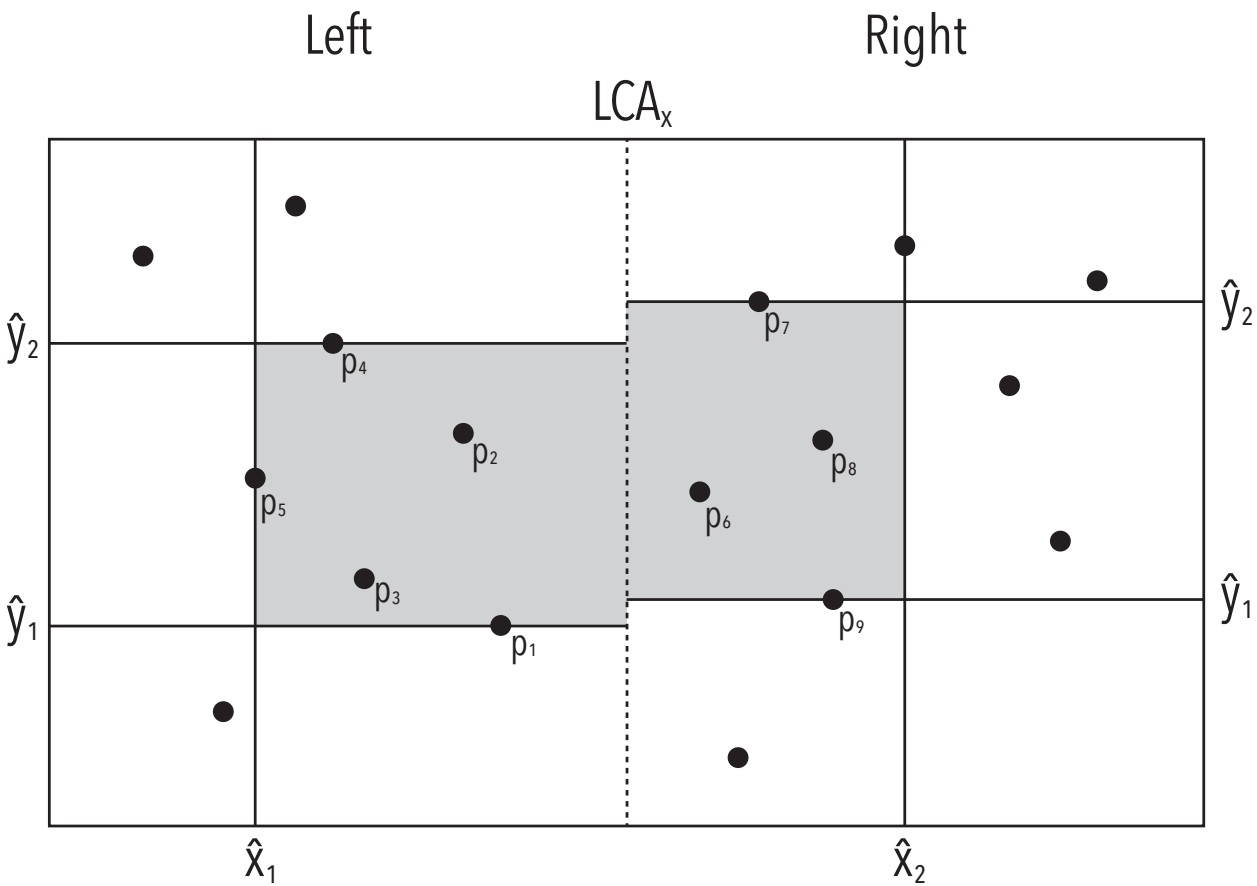
OBIS af Chan et al. Med $\mathcal{O}(n)$ plads og $\mathcal{O}(\lg \lg n + k \cdot \lg^{\epsilon} n)$.
Bruger også Ball Inheritance til at finde de k punkter.

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



AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

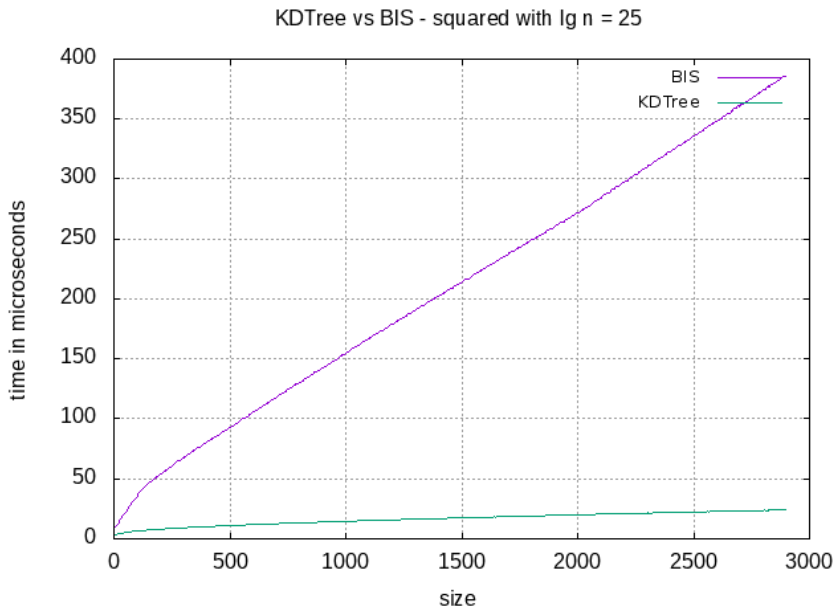
Outline

setup

- Square area $\sqrt{n} \cdot \sqrt{k} \times \sqrt{n} \cdot \sqrt{k}$ returnerer k punkter.
- Slices af størrelse k returnerer k punkter. $[0, n] \times [y, y + k]$
- Hvad forventer vi af $\mathcal{O}(\sqrt{n} + k)$ vs $\mathcal{O}(\lg n + k \cdot \lg^{\epsilon} n)$?
- $\sqrt{n} + k = \lg n + k \cdot \lg^{\epsilon} n \Leftrightarrow k = \frac{\sqrt{n} - \lg n}{\lg^{\epsilon} n - 1}$

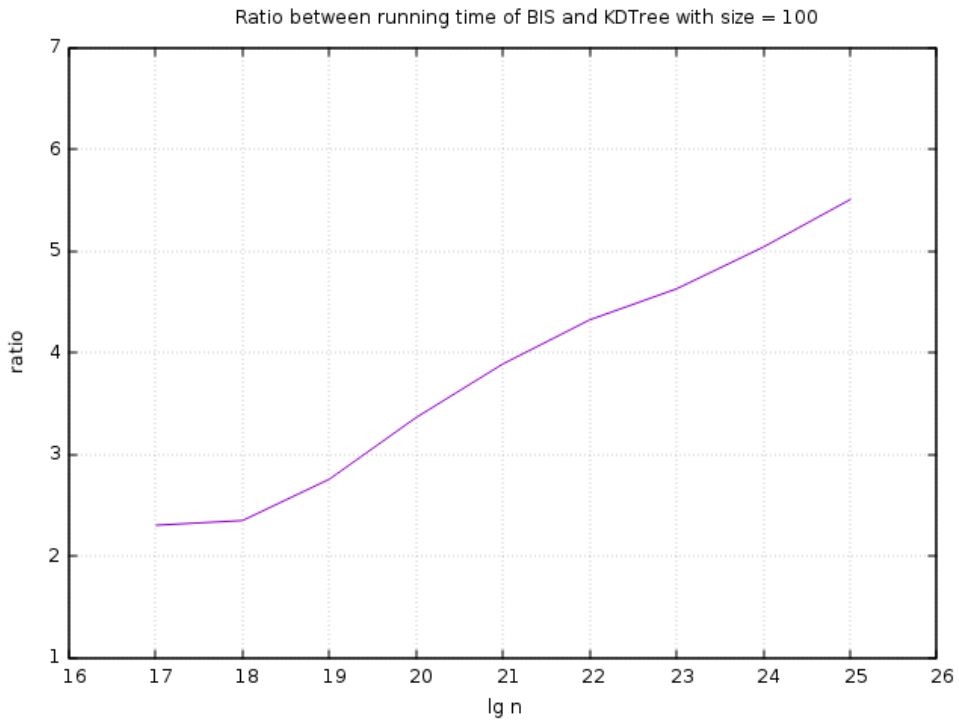
- Hvad vil vi gerne vise af resultater?
- Hvad forventer vi at se af tendenser og hvorfor?
- Mindre ændring mellem shapes når det er $\lg^{\epsilon} n$ i stedet for $\mathcal{O}(\sqrt{n})$.
- Hvert eksperiment er kørt mange gange. 10 forskellige data-set, mange for

Squared



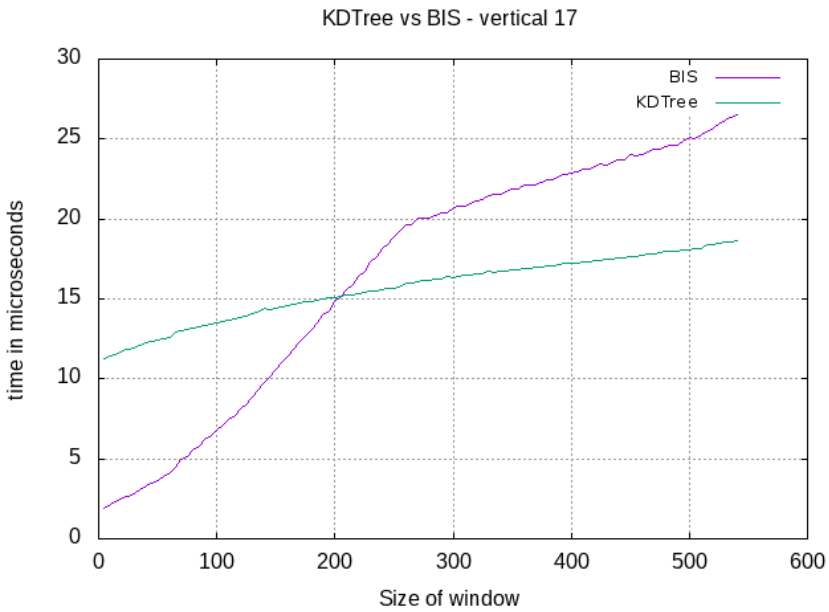
AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

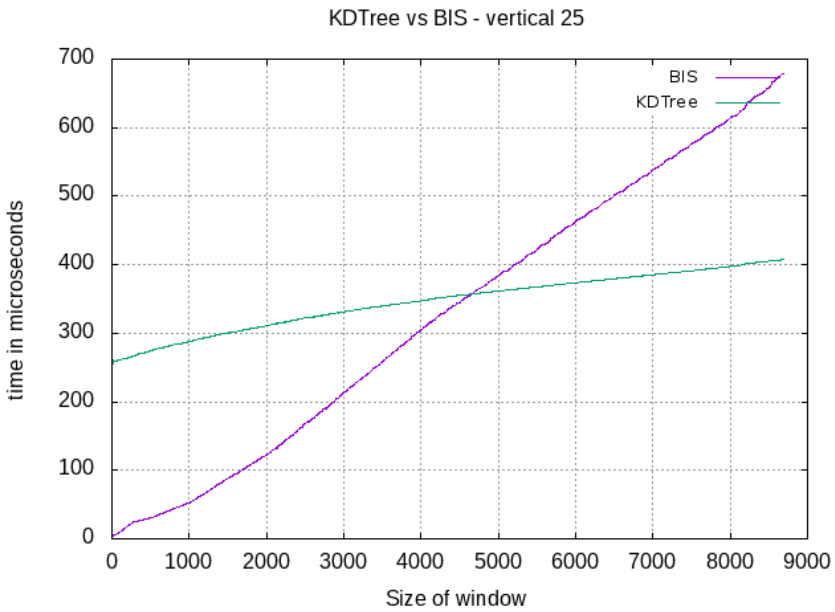
Squared



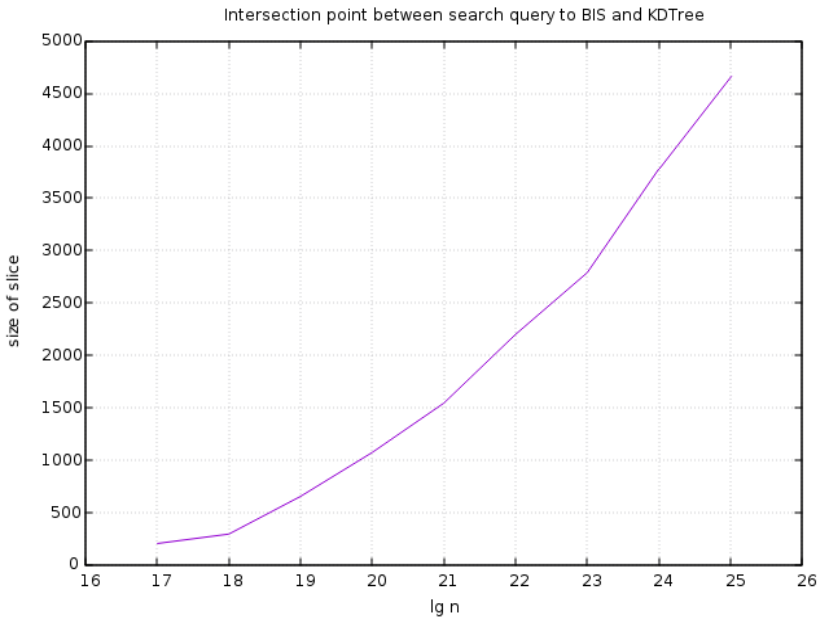
AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

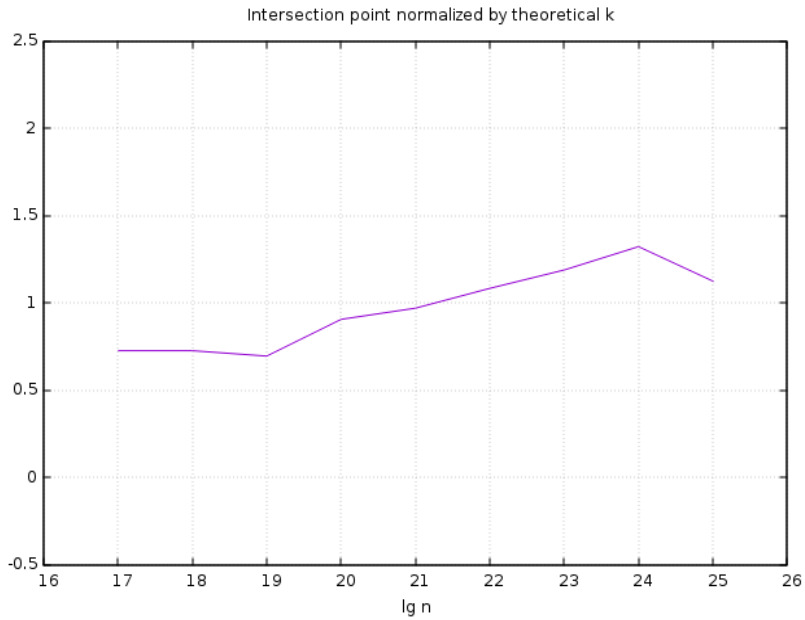
vertical


KDTree vs BIS - vertical 17
time in microseconds
Size of window
BIS
KDTree
0
5
10
15
20
25
30
0
100
200
300
400
500
600


KDTree vs BIS - vertical 25
time in microseconds
Size of window
BIS
KDTree
0
100
200
300
400
500
600
700
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

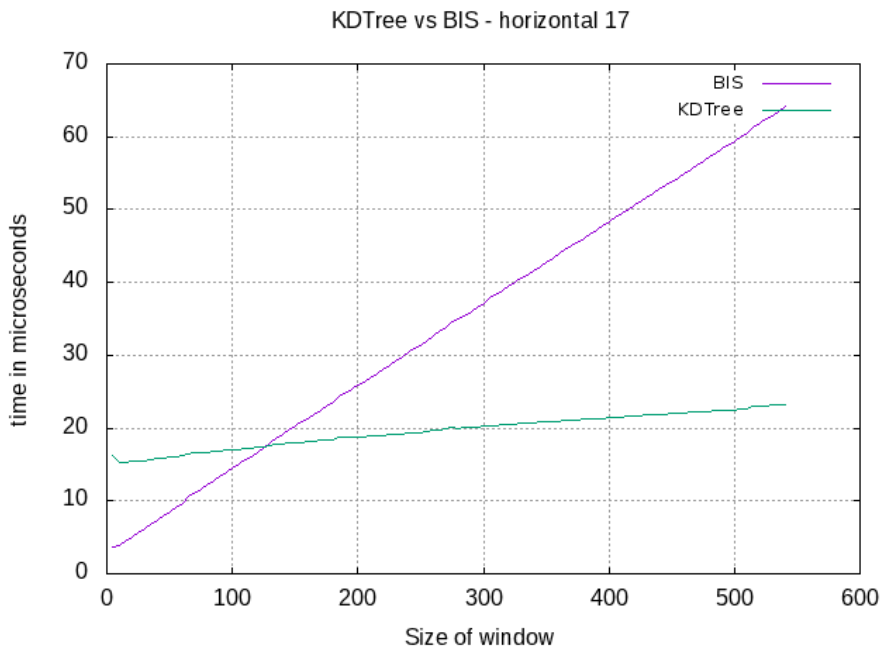
vertical

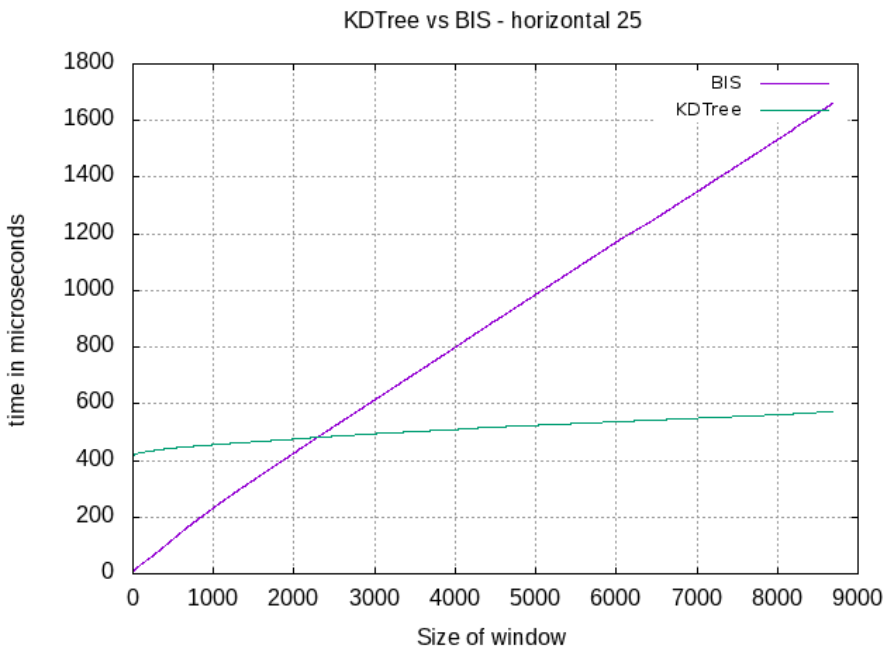




AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

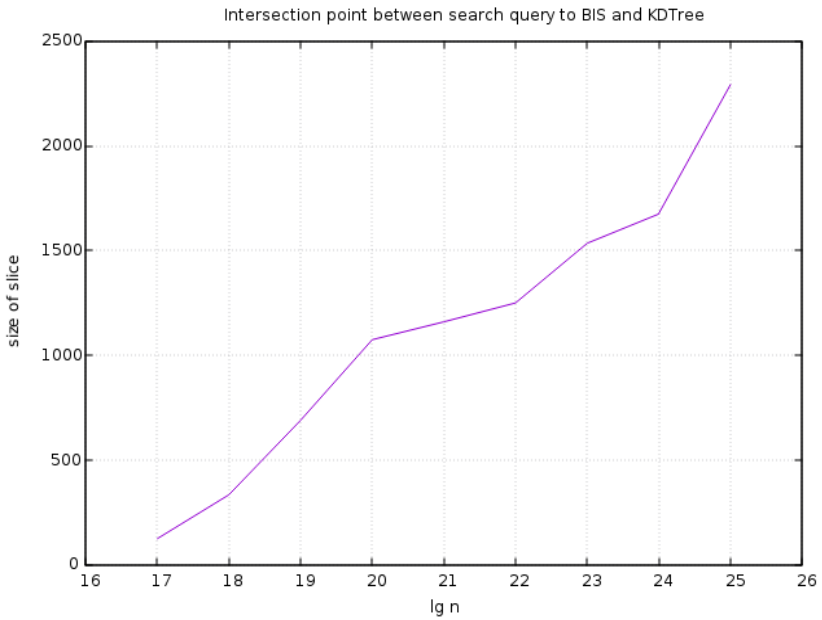
horizontal

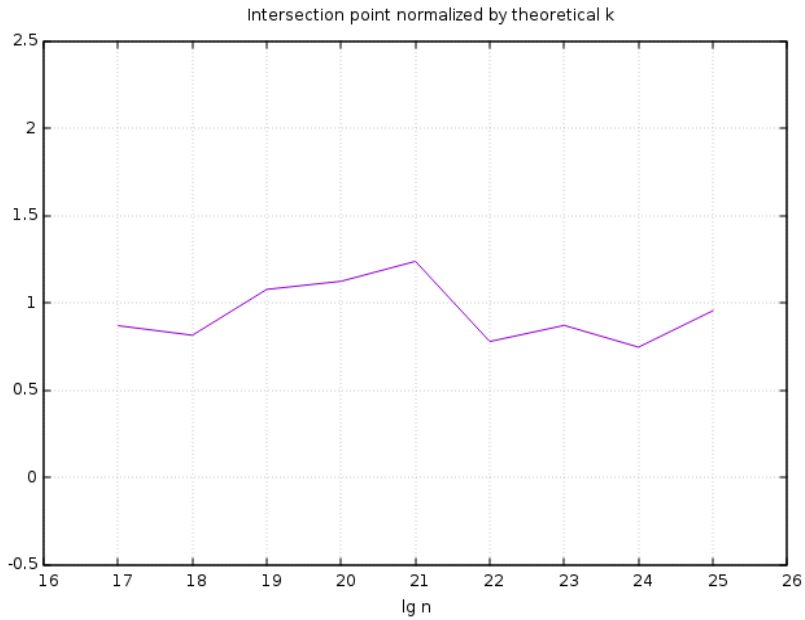




AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

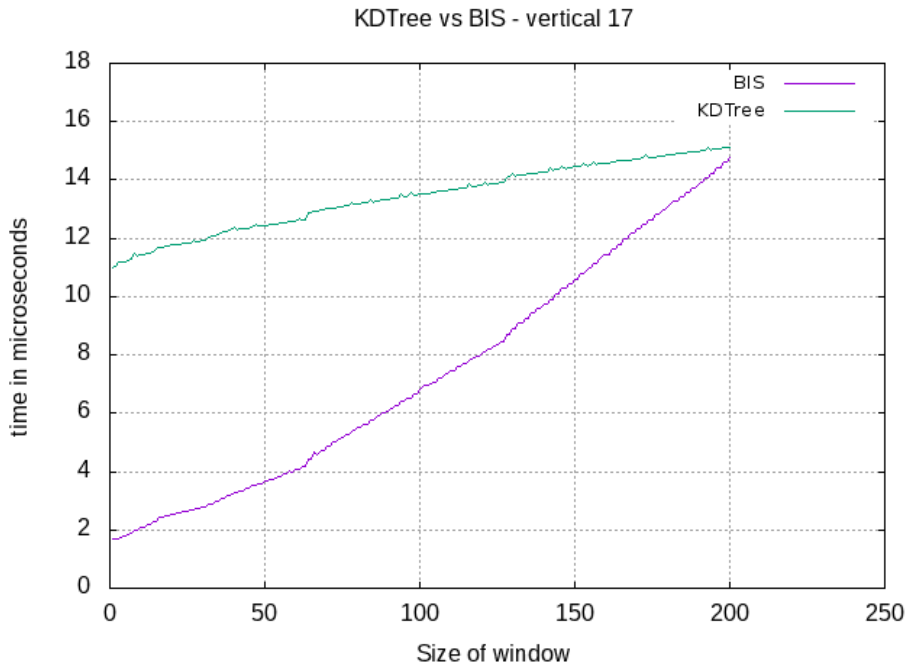
horizontal

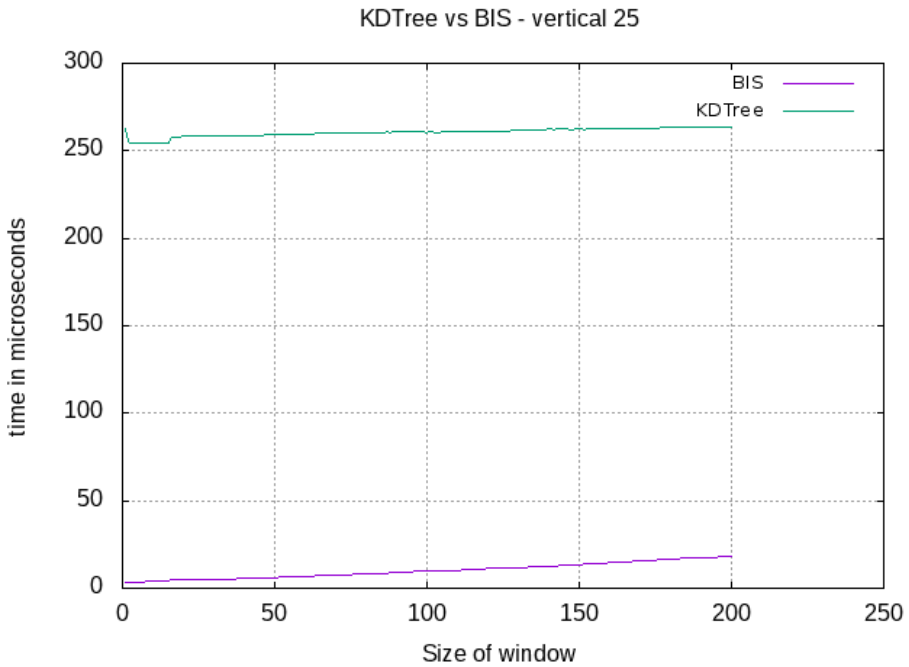




AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

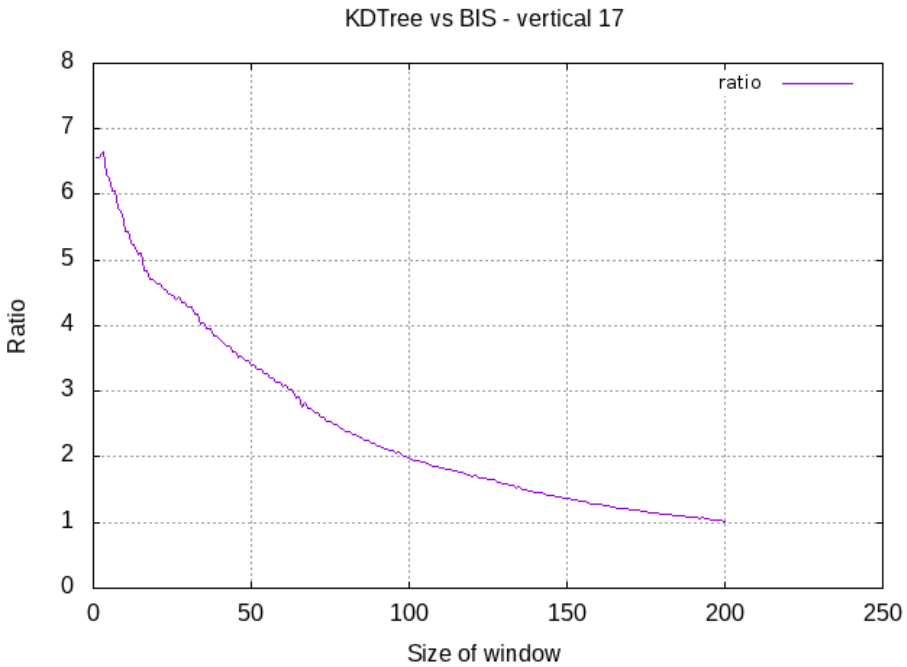
small k

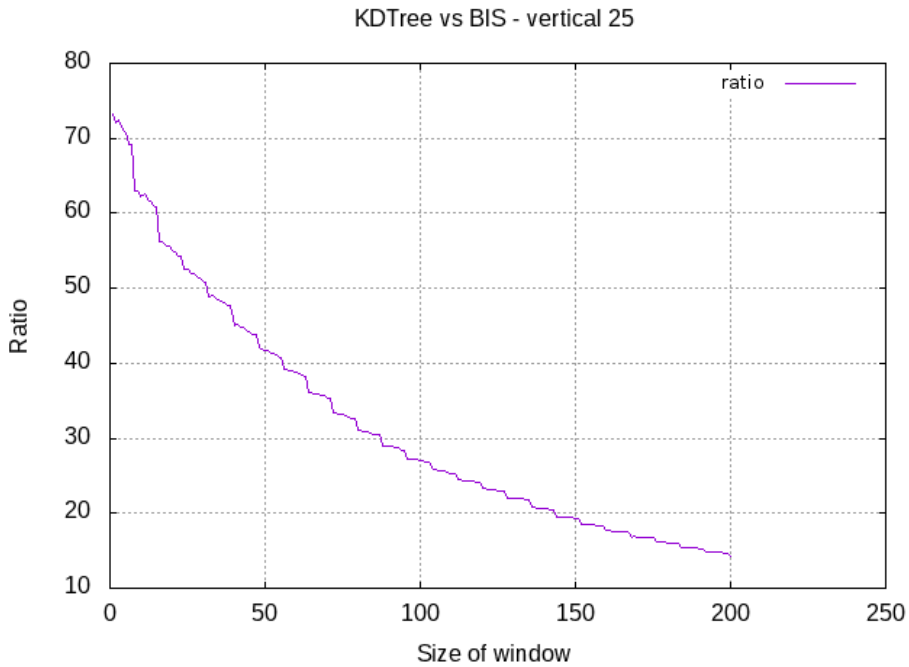




AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

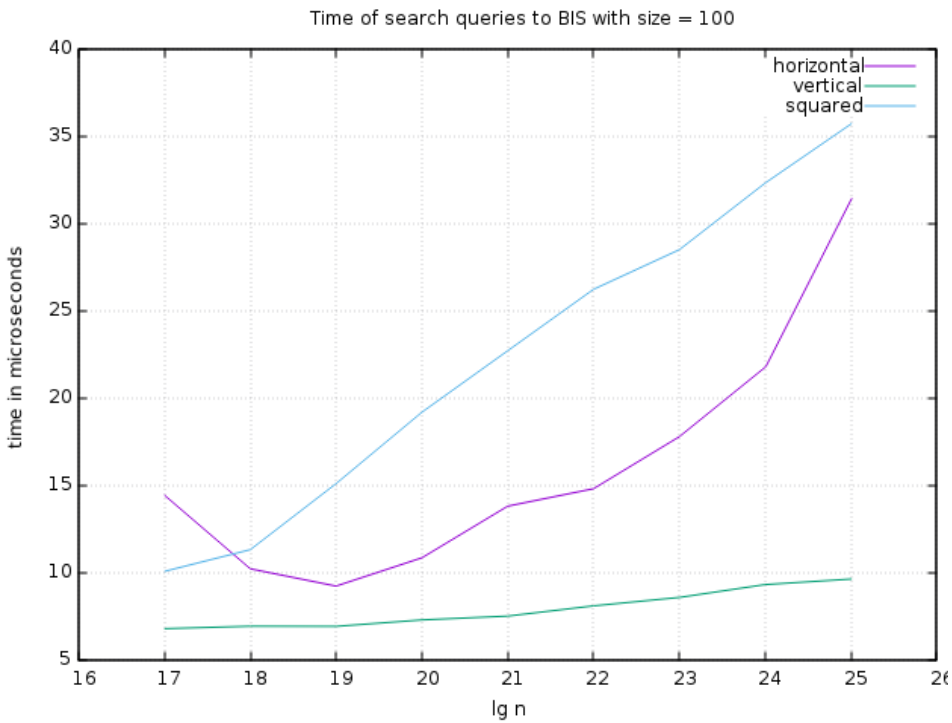
small k

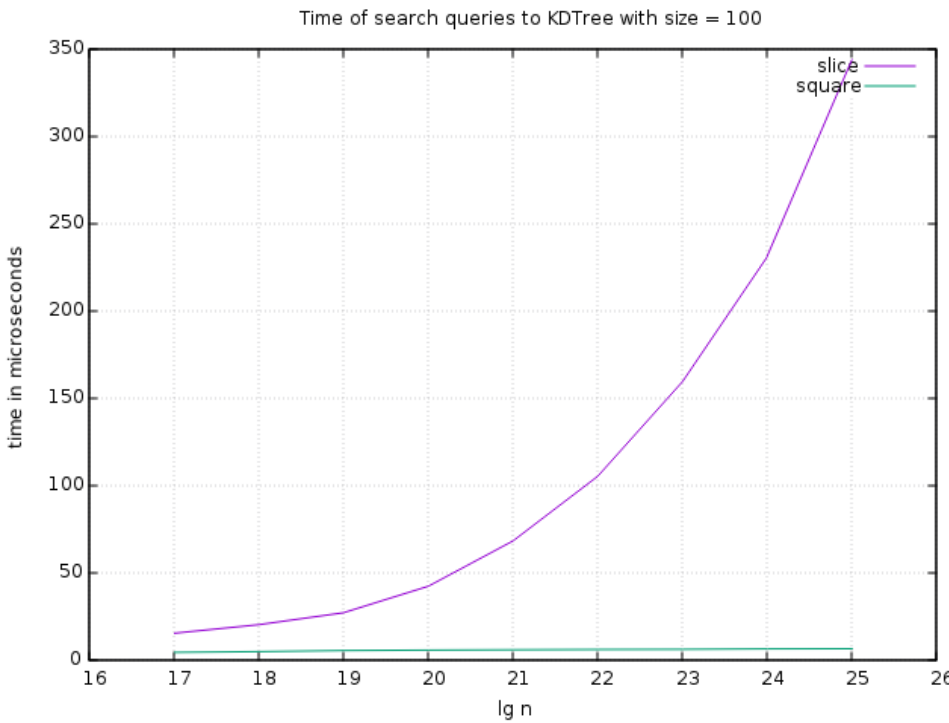




AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

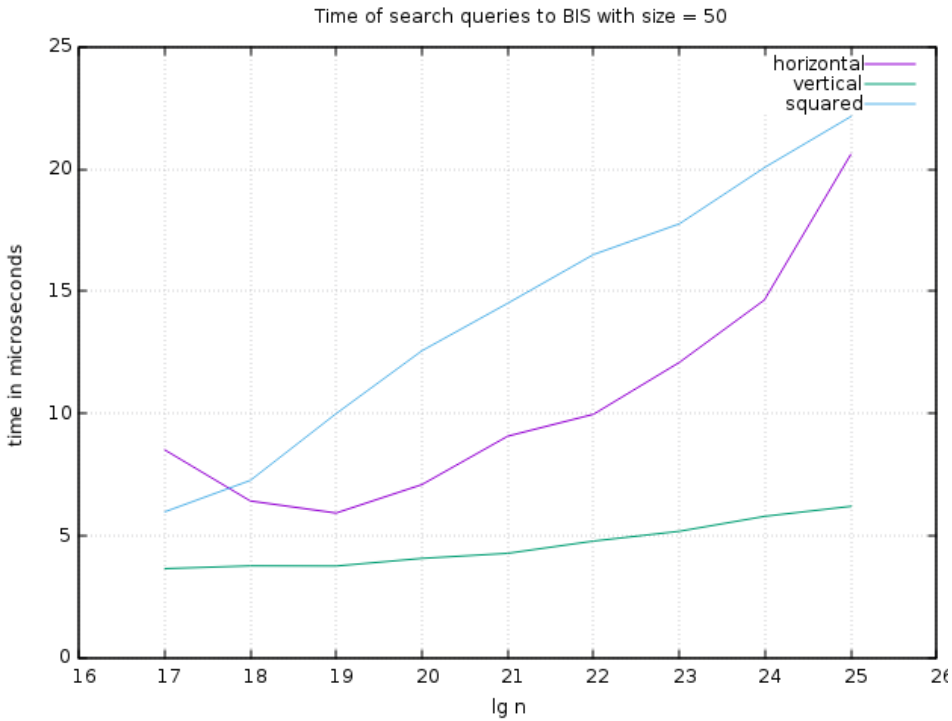
small k

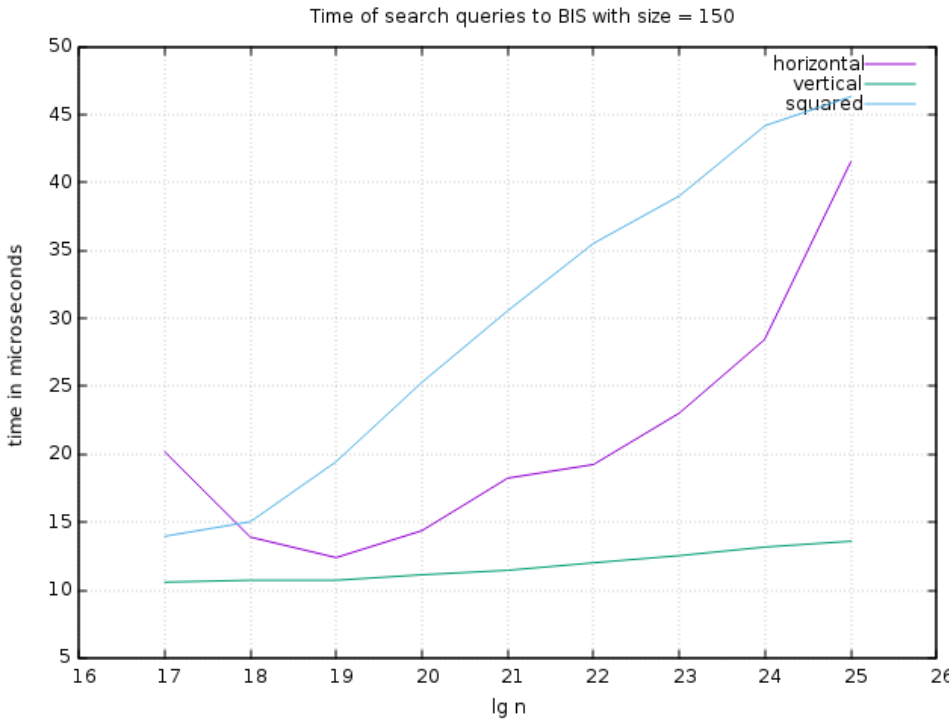




AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

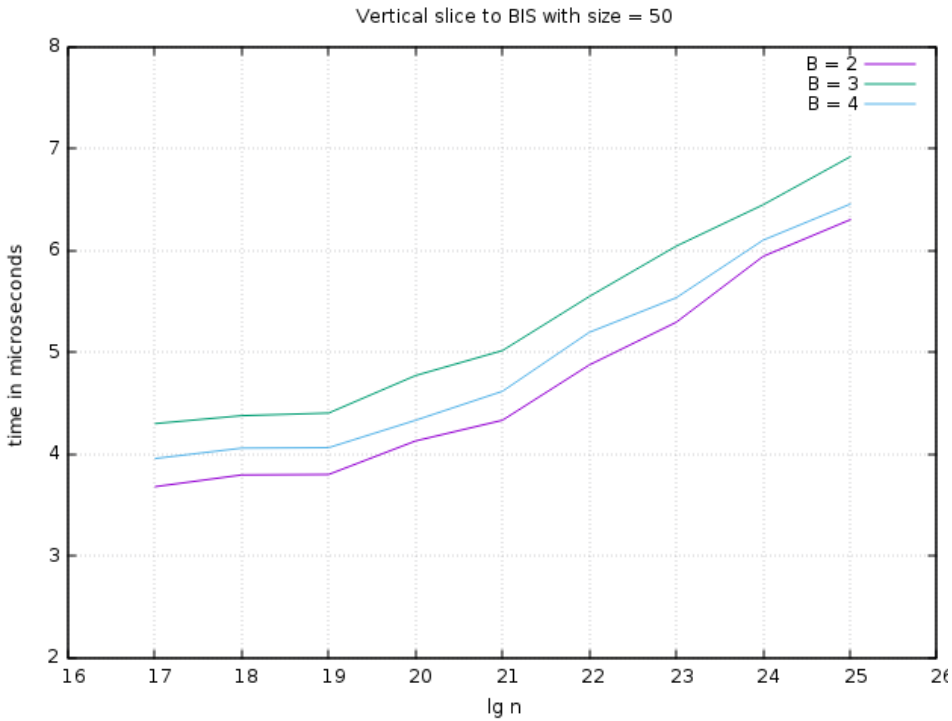
small k

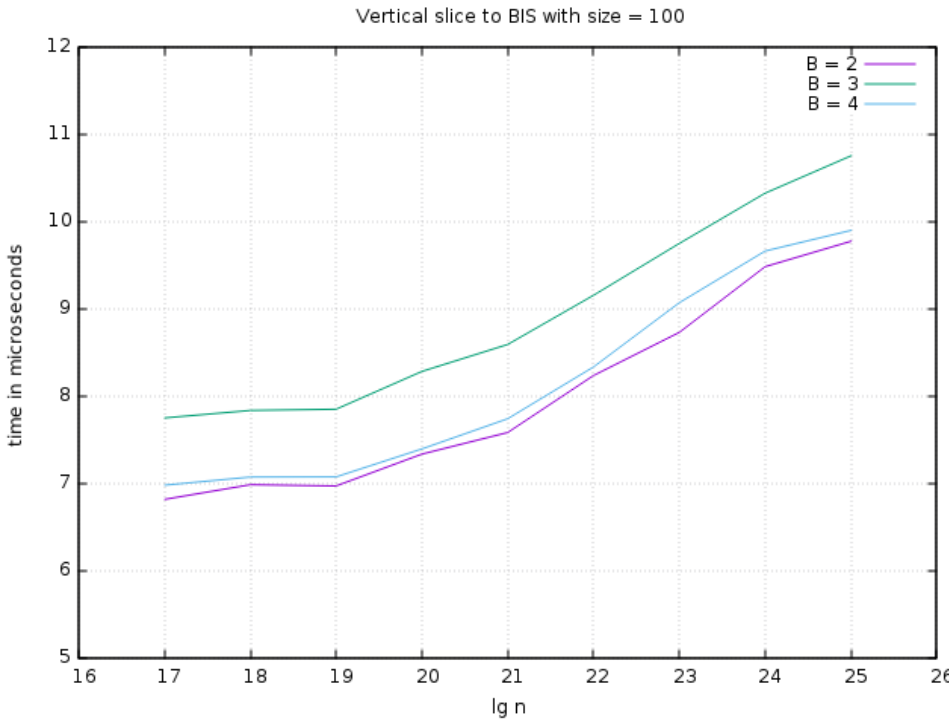




AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

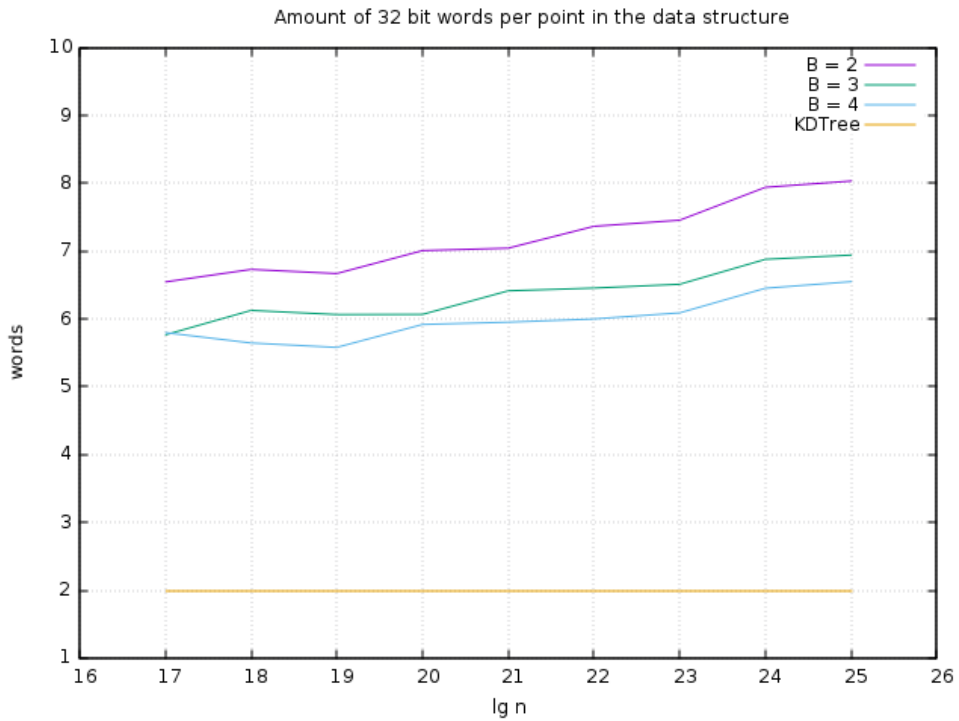
small k





AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

sizes



AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Future work

- Bedre udpakning
- Cache
- Concurrency
- Bedre kd-træ

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

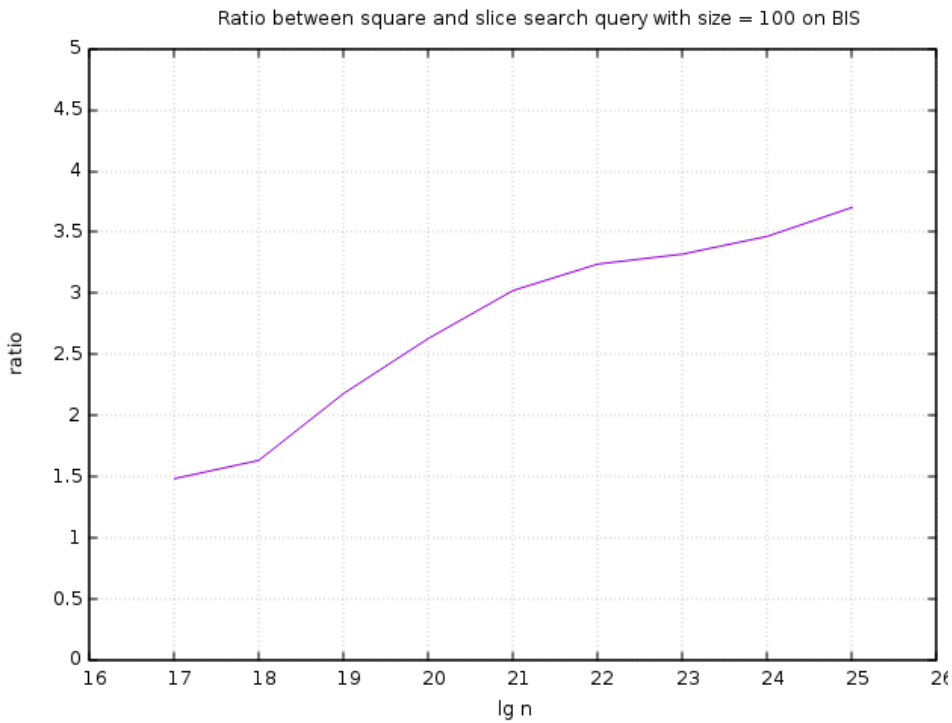
$$\sum_{i=1}^{\lg_B \lg n} \frac{\lg n}{B^i} \cdot \mathcal{O}(B^i) = \mathcal{O}(\lg n \cdot \lg_B \lg n)$$

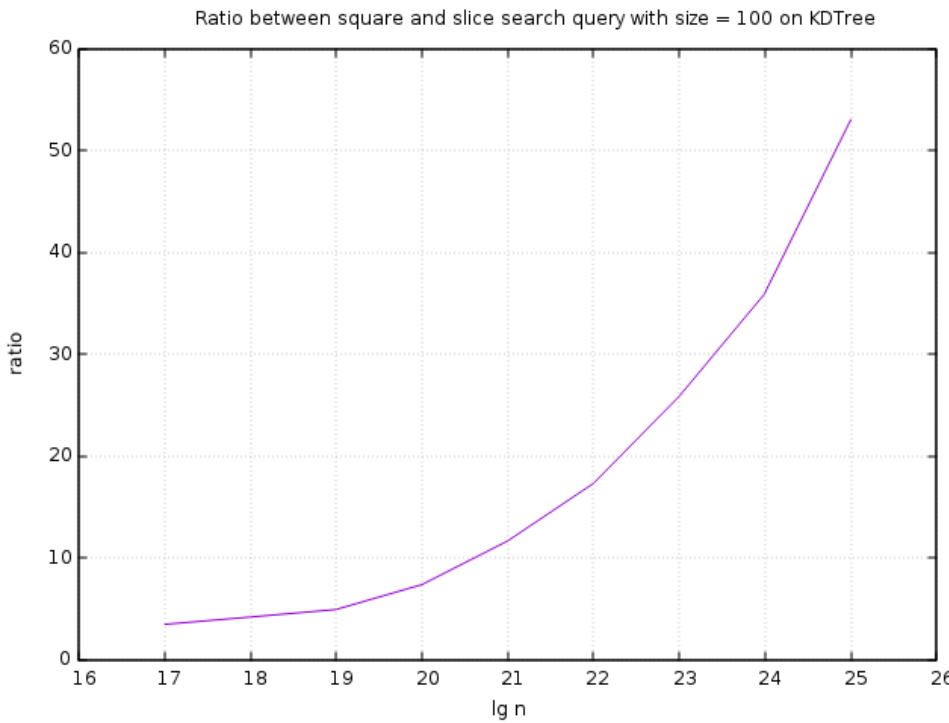
for hele kæden. Vælg nu $B = \Omega(\lg^\epsilon n)$.

Store hop

Tiden for de store hop er højst $\mathcal{O}(B \lg_B \lg n)$. Vælg $B = \lg^{\epsilon/2} n = \Omega(\lg \lg n)$.

small k





AARHUS UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

References I

 Timothy M. Chan, Kasper Green Larsen, Mihai Patrascu.
Orthogonal Range Searching on the RAM, Revisited.