# Worst Case Efficient Data Structures

**Gerth Stølting Brodal**

**BRICS**
**Department of Computer Science**
**University of Aarhus**
and
**Max-Planck-Institut für Informatik**
**Saarbrücken**

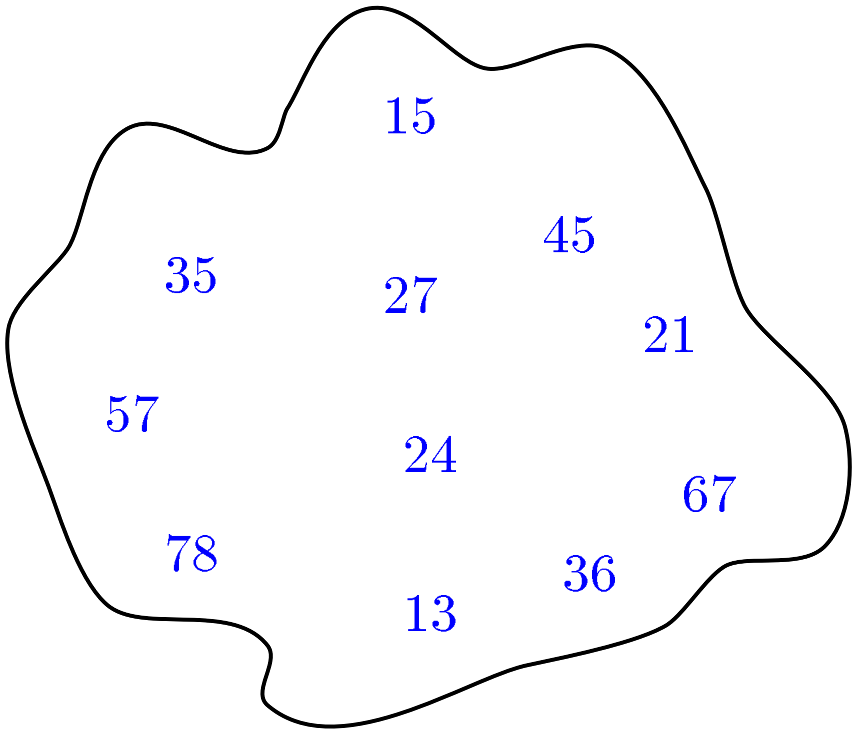
## Overview

- Priority queues
  - Comparison based data structures
  - Lower bounds for comparison based data structures
  - RAM data structures
  - Parallel data structures
- Partial persistent data structures

# Priority Queues

Maintain a set of $n$ elements from a totally ordered universe (say integers) under the operations:

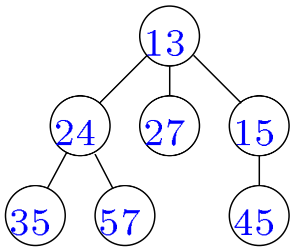
- FindMin($Q$)
- Insert($Q, e$)
- DeleteMin($Q$)
- Delete($Q, e$)*
- Meld($Q_1, Q_2$)
- DecreaseKey($Q, e, e'$)*

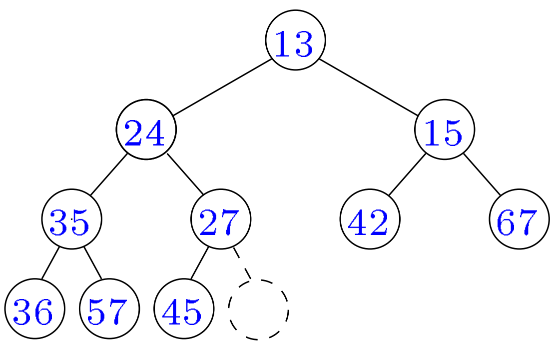



*Assume the location of $e$ is known.

# Priority Queues

Many priority queues are based on heap ordered trees, *i.e.*, each node stores an element and the element stored at a node is $\geq$ the element stored at the node's parrent.



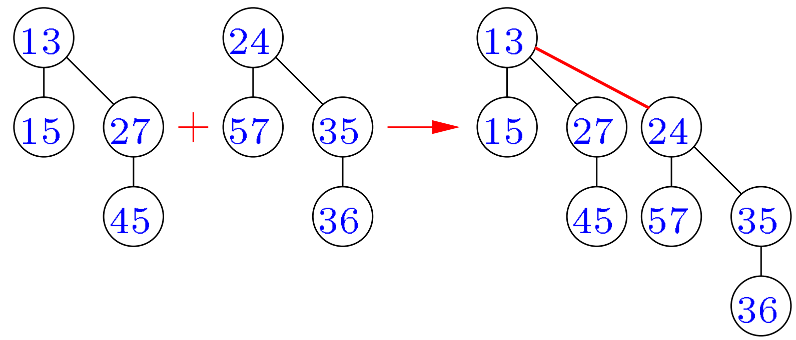




The priority queues (heaps) of Williams are based on one heap ordered binary tree.
$\Rightarrow$ INSERT and DELETEMIN can be performed in $O(\log n)$ time.

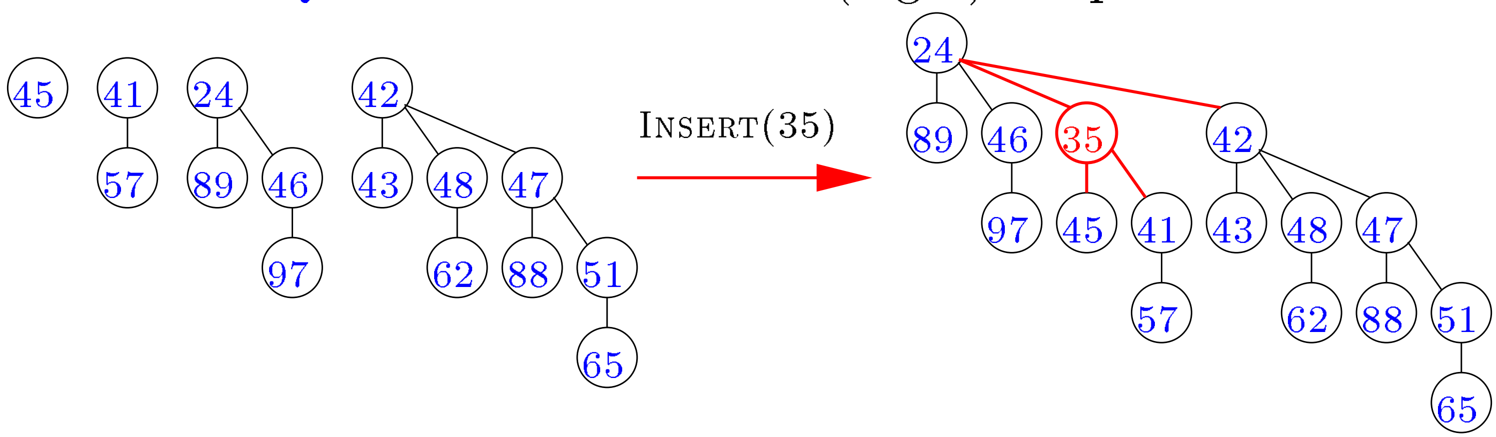
Williams 1964

# Linking Heap Ordered Trees

More recent data structures are based on linking heap ordered trees of the same size.




Ex.: Binomial Queues are based on $O(\log n)$ heap ordered trees.




**Theorem** Binomial queues support INSERT and MELD in amortized constant time and DELETEMIN in amortized $O(\log n)$ time.

Vuillemin 1978

# Constant Time Meldable Priority Queues

**Theorem** INSERT and MELD can be supported in worst case constant time and DELETEMIN in worst case $O(\log n)$ time.

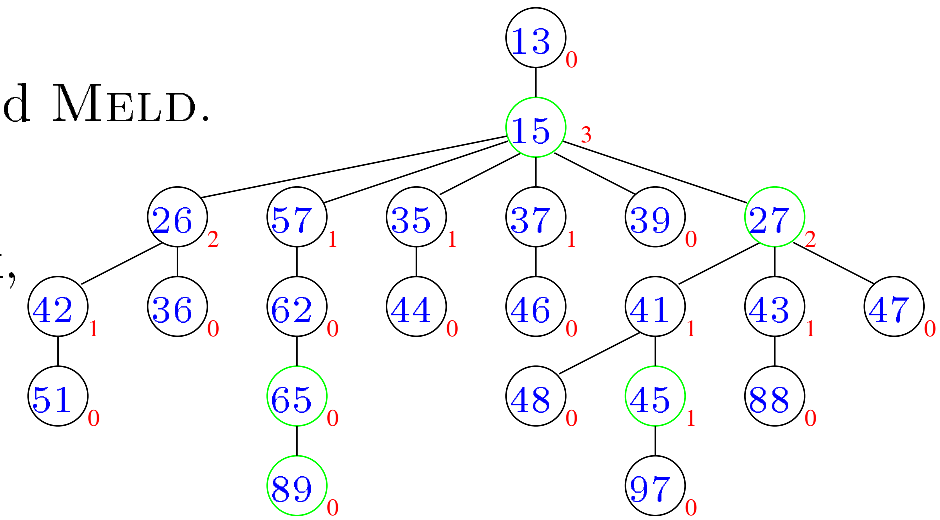
$\Leftarrow$ One heap ordered tree.

$\Leftarrow$ One linking per INSERT and MELD.

$\Leftarrow$ 1,2 or 3 sons of each rank less than the parent's rank,

$+$ one arbitrary ranked son.

$\Leftarrow$ The root has rank 0.




Brodal 1995

## Constant Time DecreaseKey

**Theorem** Fibonacci heaps support DecreaseKey in amortized constant time and DeleteMin in amortized $O(\log n)$ time.

Fredman, Tarjan 1984

**Theorem** Relaxed heaps support DecreaseKey in worst case constant time and DeleteMin in worst case $O(\log n)$ time. Meld requires $\Theta(\log n)$ time.

Driscoll, Gabow, Shrairman, Tarjan 1988

**Theorem** DecreaseKey and Meld can be supported in worst case constant time and DeleteMin in worst case $O(\log n)$ time.

Brodal 1996

$\Leftarrow$ $O(1)$ heap ordered trees.

$\Leftarrow$ Relaxed heaps.

$\Leftarrow$ A number of invariants to solve the technical dependencies **!**

# Comparison Based Priority Queues

| | Williams 1964 Heaps | Vuillemin 1978 Binomial Queues* | Brodal 1995 | Fredman Tarjan 1984 Fibonacci Heaps* | Driscoll Gabow Shrairman Tarjan 1988 Relaxed Heaps | Brodal 1996 |
|---|---|---|---|---|---|---|
| FindMin | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| Insert | $O(\log n)$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| Meld | $O(n)$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(\log n)$ | $O(1)$ |
| Delete(Min) | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ |
| DecreaseKey | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ | $O(1)$ | $O(1)$ | $O(1)$ |

*Amortized bounds

# Comparison Based Priority Queues
# Lower Bounds

**Theorem** Comparison based sorting requires $\Omega(n \log n)$ comparisons.

$\Rightarrow$ Insert or DeleteMin require $\Omega(\log n)$ comparisons.

**Theorem** If Insert and Delete make $O(t)$ comparisons, then FindMin requires $\dfrac{n}{2^{O(t)}}$ comparisons.

$\Rightarrow$ a doubly linked list is an optimal priority queue implementation!

**Theorem** If Meld makes $o(n)$ comparisons, and FindMin $O(n^\epsilon)$ comparisons, $\epsilon < 1$, then Delete and DeleteMin require $\Omega(\log n)$ comparisons.

Brodal, Chaudhuri, Radhakrishnan 1996

# RAM Priority Queues

**Model:** A unit cost Random Access Machine with word size $w$.

**Word operations:** $+$, shifting, bit-wise boolean operations.

**Elements:** Integers in the range $0..2^w - 1$.

**Operations:**

- $\textsc{FindMin}(Q)$
- $\textsc{Insert}(Q, e)$
- $\textsc{Delete}(Q, e)$
- $\textsc{Pred}(Q, e)$, $\qquad \textsc{Pred}(\ \{13, 15, 24, 36, 45, 67\}\ ,\ 31\ ) = 24.$

**Theorem** The above operations can be performed in $O(\log w)$ time.

van Emde Boas 1977

$\Rightarrow$ an $O(\log\log n)$ priority queue for $w = \log^{O(1)} n$.

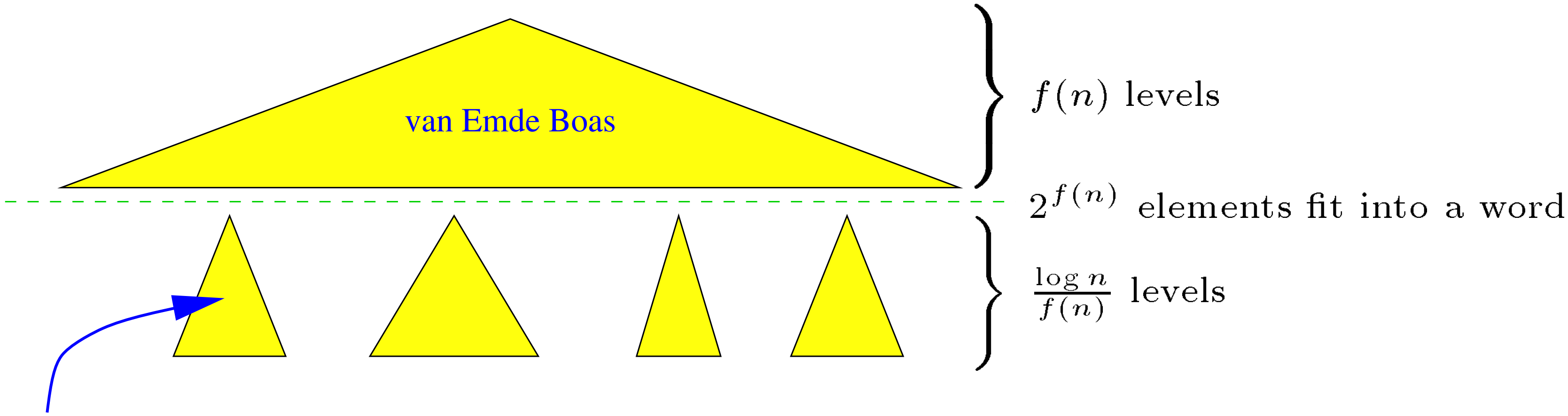
# RAM Priority Queues

| | van Emde Boas 1977 | Thorup* 1996 | Andersson 1995 | Brodal 1997 | |
|---|---|---|---|---|---|
| FindMin | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ | $O(1)$ |
| Insert | $O(\log w)$ | $O(\log\log n)$ | $O(\sqrt{\log n})$ | $O(f(n))$ | $O(\log\log n)$ |
| Delete | $O(\log w)$ | $O(\log\log n)$ | $O(\sqrt{\log n})$ | $O(f(n))$ | $O(\log\log n)$ |
| Pred | $O(\log w)$ | | $O(\sqrt{\log n})$ | $O(\frac{\log n}{f(n)})$ | $O(\frac{\log n}{\log\log n})$ |

$$\log\log n \leq f(n) \leq \sqrt{\log n}$$

*Amortized bounds

# Outline of RAM Priority Queue




Packed search trees of degree $2^{\Theta(f(n))}$ with buffers of delayed INSERT and DELETE operations, supporting INSERT and DELETE in worst case $O(f(n))$ time.

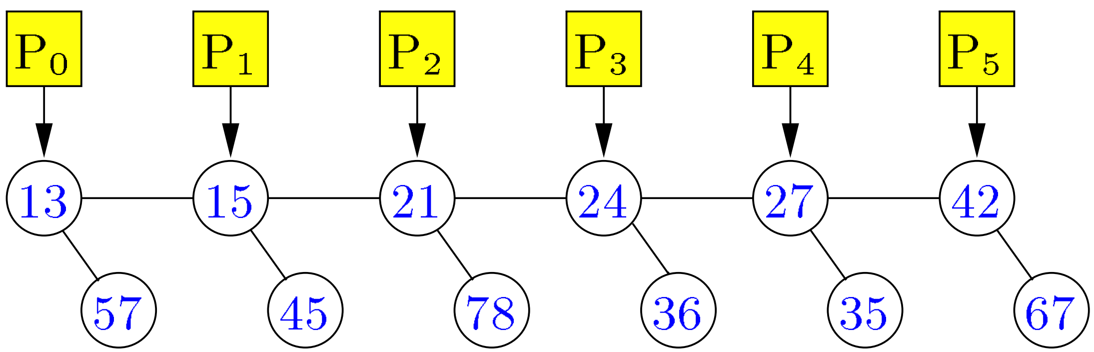
$\Leftarrow$ Two level data structure (van Emde Boas and packed).
$\Leftarrow$ Packed search trees, Andersson 1995.
$\Leftarrow$ Buffer trees for external memory, Arge 1995.
$\Leftarrow$ List merging in $O(1)$ words, Albers, Hagerup 1992.
$\Leftarrow$ Standard deamortization techniques.

Brodal 1997

# Adopting Parallelism to Priority Queues

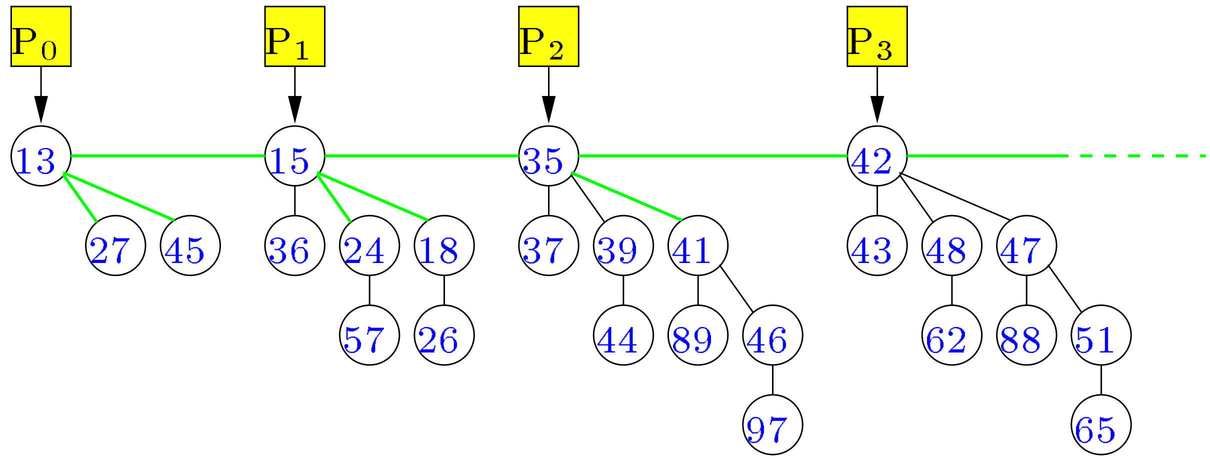
Question: Is it possible to obtain comparison based priority queues supporting operations in $o(\log n)$ time by using a non-constant number of processors ?

Answer: Yes, $O(n)$ processors can support INSERT and DELETEMIN in constant time.




Folklore

# Outline of Parallel Priority Queue




⇐ Processor $P_i$ maintains 1, 2 or 3 trees of size $2^i$.

⇐ Parallel linking and unlinking of trees.

**Theorem** $O(\log n)$ processors can support INSERT, MELD and DELETEMIN in constant time. An extension of the data structure supports DELETE and DECREASEKEY in constant time too.

Brodal 1996

# Parallel Priority Queues

| | Folklore | Pinotti Pucci 1992 | Pinotti Das, Crupi 1996 | Ranade *et al.* 1994 | Brodal 1996 |
|---|---|---|---|---|---|
| FindMin | $O(1)$ | $O(1)$ | $O(\log\log n)$ | $O(1)$ | $O(1)$ |
| Insert | $O(1)$ | $O(\log\log n)$ | $O(\log\log n)$ | $O(1)$ | $O(1)$ |
| DeleteMin | $O(1)$ | $O(\log\log n)$ | $O(\log\log n)$ | $O(1)$ | $O(1)$ |
| Meld | | | $O(\log\log n)$ | | $O(1)$ |
| Delete | | | $O(\log\log n)$ | | $O(1)$ |
| DecreaseKey | | | $O(\log\log n)$ | | $O(1)$ |
| #processors | $n$ | $\frac{\log n}{\log\log n}$ | $\frac{\log n}{\log\log n}$ | $\log n$ | $\log n$ |
| Model | Array | EREW PRAM | CREW PRAM | Array | CREW PRAM or Array |

## Parallel DecreaseKey operations

Essential to algorithms like Dijkstra's algorithm for the single-source shortest path problem are the operations

- DeleteMin($Q$)
- DecreaseKey($Q, (e_1, e'_1), \ldots, (e_k, e'_k)$)

**Theorem** There exists an EREW PRAM data structure supporting the above operations in constant time, provided $e'_1 \leq e'_2 \leq \cdots \leq e'_k$.

Brodal, Träff, Zaroliagis 1997

Previous parallel data structures only supported parallel Insert and DeleteMin operations.

# The Single-Source Shortest Path Problem

**Theorem** The single-source shortest path problem can be solved by Dijkstra's algorithm in sequential $O(n \log n + m)$ time by using Fibonacci heaps. Fredman, Tarjan 1987

| | Han Pan Reif 1992 | Driscoll Gabow Shrairman Tarjan 1988 | Paige Kruskal 1985 | | Brodal Träff Zaroliagis 1997 |
|---|---|---|---|---|---|
| Time | $O(\log^2 n)$ | $O(n \log n)$ | $O(n \log \log n)$ | $O(n)$ | $O(n)$ |
| Work | $O(n^3 \log n)$ | $O(n \log n + m)$ | $O(n^2)$ | $O(n^{2+\epsilon})$ | $O(m \log n)$ |
| Model | EREW | EREW | CRCW | CRCW | CREW |

# Partial Persistence

A data structure is said to be partial persistent if

- old versions are remembered and can be accessed,
- only the latest version can be modified.

Some naive approaches:

- Store a copy of each version of the data structure
  $\Rightarrow$ space and time overhead per update operation is $O(n)$.

- Only store the sequence of changes done to the data structure
  $\Rightarrow$ update steps in constant time and space, but accesses require $O(n)$ time.

# Partial Persitence Techniques

| | Driscoll Sarnak Sleator Tarjan 1989 | Brodal 1996 | Dietz Raman° 1991 | Dietz$^{\triangle}$ 1989 |
|---|---|---|---|---|
| Data structures | pointer based | pointer based | pointer based | arrays |
| Indegree | $O(1)$ | $O(1)$ | $\log^{O(1)} n$ | $\infty$ |
| Access steps | $O(1)$ | $O(1)$ | $O(1)$ | $O(\log\log n)$ |
| Update steps | $O(1)^*$ | $O(1)$ | $O(1)$ | $O(\log\log n)$ |

$^*$amortized, °technique requires the RAM, $^{\triangle}$expected amortized.

## Summary

### Comparison based priority queues [WADS'95, SODA'96, NJC'96]

- MELD can be supported in worst case constant time.
- MELD and DECREASEKEY can be supported in worst case constant time.
- A lower bound tradeoff between updating a priority queue and the query time.

### RAM priority queues [STACS'97]

- DELETE can be supported in worst case $O(\log \log n)$ time.
- PRED can be supported in $o(\log n)$ time while having $O(\log \log n)$ update time.

### Parallel priority queues [SWAT'96, IPPS'97]

- $O(\log n)$ processors can support all operations in constant time.
- Parallel DECREASEKEY in constant time.

### Partial persistent data structures [NJC'96]

- Bounded indegree data structures can be made partially persistent in worst case constant time.

# Approximate Dictionary Queries

$\text{Dist}_H(u, v)$ is the Hamming distance between two binary strings $u$ and $v$ of equal length.

$$\begin{array}{l} u : 1\,0\,0\,1\,1\,0\,0\,0\,1\,1\,1 \\ v : 1\,0\,0\,0\,1\,0\,1\,1\,1\,1\,0 \\ \quad\;\;\;\;\;\;\;\;\uparrow\;\;\;\;\;\;\uparrow\,\uparrow\;\;\;\;\;\;\uparrow \end{array}$$

$$\text{Dist}_H(u, v) = 4$$

- Dictionary $W = \{w_1, w_2, \ldots, w_n\}$, $w_i \in \{0, 1\}^m$ for $i = 1, \ldots, n$.
- Query string $\alpha \in \{0, 1\}^m$.
- Question Is there any $w_i \in W$ such that $\text{Dist}_H(\alpha, w_i) \leq d$ ?

| $d = 1$ | Yao, Yao 1995 | Brodal, Gąsieniec 1996 |
|---|---|---|
| Query time | $O(m \log \log n)$ | $O(m)$ |
| Space | $O(nm \log m)$ | $O(nm)$ |
| Preprocessing time | $O(nm \log m)$ | $O(nm)$ |