



Quickheaps

Navarro,  
Paredes

Introduction

Related work  
Incremental sorting

Quickheaps

Main idea  
Quickheap  
operations  
External Quickheaps  
Experimental results

Conclusions

# Quickheaps

Gonzalo Navarro   Rodrigo Paredes

Department of Computer Science, University of Chile  
{raparede, gnavarro}@dcc.uchile.cl

2nd Workshop on Compression, Text and Algorithms  
2007



# Outline

## Quickheaps

Navarro,  
Paredes

### Introduction

Related work

Incremental sorting

### Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

Experimental results

### Conclusions

- 1 Introduction
  - Related work
  - Incremental sorting
- 2 Quickheaps
  - Main idea
  - Quickheap operations
  - External Quickheaps
  - Experimental results
- 3 Conclusions



# Outline

Quickheaps

Navarro,  
Paredes

Introduction

Related work

Incremental sorting

Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

Experimental results

Conclusions

- 1 Introduction
  - Related work
  - Incremental sorting
- 2 Quickheaps
  - Main idea
  - Quickheap operations
  - External Quickheaps
  - Experimental results
- 3 Conclusions



# Outline

## Quickheaps

Navarro,  
Paredes

### Introduction

Related work

Incremental sorting

### Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

Experimental results

### Conclusions

- 1 Introduction
  - Related work
  - Incremental sorting
- 2 Quickheaps
  - Main idea
  - Quickheap operations
  - External Quickheaps
  - Experimental results
- 3 Conclusions



# Introduction

## Quickheaps

Navarro,  
Paredes

## Introduction

Related work

Incremental sorting

## Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

Experimental results

## Conclusions

- A **priority queue** is a data structure which allows:
- **insert**(*key*, *item*), which inserts element (*key*, *item*);
- **findMin**(), which returns an element of the queue with lowest *key* value;
- **extractMin**(), which in addition removes that lowest *key* value element.
- Other operations are **heapify**(*A*), **increaseKey**(*key*,  $\delta$ ), **decreaseKey**(*key*,  $\delta$ ), **find**(*key*), **delete**(*key*), and so on.



# Introduction

## Quickheaps

Navarro,  
Paredes

## Introduction

Related work

Incremental sorting

## Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

Experimental results

## Conclusions

- A **priority queue** is a data structure which allows:
- **insert**(*key*, *item*), which inserts element (*key*, *item*);
- **findMin**(), which returns an element of the queue with lowest *key* value;
- **extractMin**(), which in addition removes that lowest *key* value element.
- Other operations are **heapify**(*A*), **increaseKey**(*key*,  $\delta$ ), **decreaseKey**(*key*,  $\delta$ ), **find**(*key*), **delete**(*key*), and so on.



# Quickheaps

## Quickheaps

Navarro,  
Paredes

## Introduction

Related work  
Incremental sorting

## Quickheaps

Main idea  
Quickheap  
operations  
External Quickheaps  
Experimental results

## Conclusions

- Based on the INCREMENTALQUICKSORT algorithm (IQS) [ALENEX 2006], we develop yet another priority queue, the **Quickheaps**.
- **Quickheaps** enable efficient element insertion, minimum extraction, deletion of arbitrary elements and modification of the priority of elements within the heap.
- **Quickheaps** require  $O(\log m)$  extra integers.
- **Quickheaps** exhibit a local access pattern  $\Rightarrow$  they are alternatives for a secondary memory implementation.



# Outline

Quickheaps

Navarro,  
Paredes

Introduction

Related work

Incremental sorting

Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

Experimental results

Conclusions

- 1 Introduction
  - **Related work**
  - Incremental sorting
- 2 Quickheaps
  - Main idea
  - Quickheap operations
  - External Quickheaps
  - Experimental results
- 3 Conclusions



# Related work

## Internal memory priority queues

### Quickheaps

Navarro,  
Paredes

### Introduction

Related work

Incremental sorting

### Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

Experimental results

### Conclusions

heap	insert	findMin	extractMin	merge	decreaseKey	increaseKey
binary	$O(\log m)$	$O(1)$	$O(\log m)$	$O(m)$	$O(\log m)$	$O(\log m)$
binomial	$O(\log m)$	$O(\log m)$	$O(\log m)$	$O(\log m)$	$O(\log m)$	$O(\log m)$
Fibonacci	$O(1)$	$O(1)$	$O(\log m)$	$O(1)$	$O(1)$	
leftist	$O(\log m)$	$O(1)$	$O(\log m)$	$O(\log m)$	$O(\log m)$	$O(\log m)$
min-max	$O(\log m)$	$O(1)$	$O(\log m)$	$O(m)$	$O(\log m)$	$O(\log m)$
pairing	$O(1)$	$O(1)$	$O(\log m)$	$O(1)$	$O(1)$	
skew	$O(\log m)$	$O(1)$	$O(\log m)$	$O(\log m)$	$O(\log m)$	$O(\log m)$
vEB	$O(\log \log C)$	$O(\log \log C)$	$O(\log \log C)$		$O(\log \log C)$	$O(\log \log C)$

Operation costs of some min-order priority queues.

Amortized time for pairing and skew heaps, yet for pairing heaps it's a conjecture.

The van Emde Boas heaps are subject to the restriction that the universe of keys is the set  $\{1, \dots, C\}$ .



# Related work

## External memory priority queues

### Quickheaps

Navarro,  
Paredes

### Introduction

#### Related work

#### Incremental sorting

### Quickheaps

#### Main idea

#### Quickheap operations

#### External Quickheaps

#### Experimental results

### Conclusions

- Only implements **insert**, **findMin** and **extractMin**.
- **buffer trees**.
- **$M/B$ -ary heaps**.
- **Array Heaps**.
- They achieve  $O((1/B) \log_{M/B}(m/B))$  amortized I/Os per operation.
- Yet they are impractical: complex to implement and use too much extra space and time.
- Some practical examples: **R-Heaps**, a simplification of **Array-Heaps**, and **Sequence heaps**.



# Related work

## External memory priority queues

### Quickheaps

Navarro,  
Paredes

### Introduction

#### Related work

#### Incremental sorting

### Quickheaps

#### Main idea

#### Quickheap operations

#### External Quickheaps

#### Experimental results

### Conclusions

- Only implements **insert**, **findMin** and **extractMin**.
- **buffer trees**.
- **$M/B$ -ary heaps**.
- **Array Heaps**.
- They achieve  $O((1/B) \log_{M/B}(m/B))$  amortized I/Os per operation.
- Yet they are impractical: complex to implement and use too much extra space and time.
- Some practical examples: **R-Heaps**, a simplification of **Array-Heaps**, and **Sequence heaps**.



# Outline

Quickheaps

Navarro,  
Paredes

Introduction

Related work

Incremental sorting

Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

Experimental results

Conclusions

- 1 Introduction
  - Related work
  - Incremental sorting
- 2 Quickheaps
  - Main idea
  - Quickheap operations
  - External Quickheaps
  - Experimental results
- 3 Conclusions



# Incremental sorting

## Problem formulation

### Quickheaps

Navarro,  
Paredes

### Introduction

Related work

Incremental sorting

### Quickheaps

Main idea

Quickheap  
operations

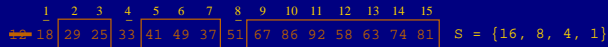
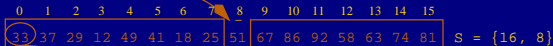
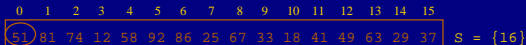
External Quickheaps

Experimental results

### Conclusions

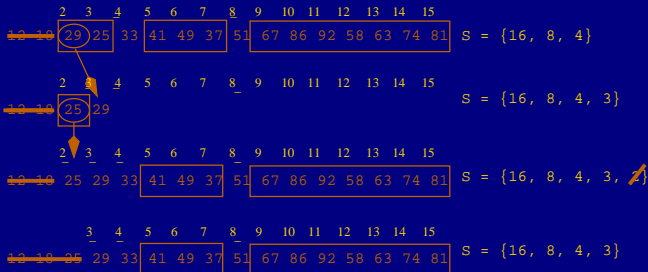
*Given set  $A$  of  $m$  numbers, output the elements of  $A$  from smallest to largest, so that the process can be stopped after  $k$  elements have been output, for any  $k \leq m$  that is unknown to the algorithm.*

Let's look for the minimum element.



Now, the second minimum is the index on top of S.

The third element is within the first chunk.



The next minima are either pivots or within the first chunk, and so on...



# Incremental sorting

## Quickheaps

Navarro,  
Paredes

## Introduction

Related work

Incremental sorting

## Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

Experimental results

## Conclusions

## IQS (Set $A$ , Index $idx$ , Stack $S$ )

1. **If**  $idx = S.top()$  **Then**  $S.pop()$ , **Return**  $A[idx]$
2.  $pidx \leftarrow \text{random}[idx, S.top() - 1]$
3.  $pidx' \leftarrow \text{partition}(A, A[pidx], idx, S.top() - 1)$
4.  $S.push(pidx')$
5. **Return**  $IQS(A, idx, S)$

- Stack  $S$  is initialized as  $S \leftarrow \{|A|\}$ .
- Note that the search is limited to  $A[idx, S.top() - 1]$ .



# Outline

## Quickheaps

Navarro,  
Paredes

### Introduction

Related work

Incremental sorting

### Quickheaps

**Main idea**

Quickheap  
operations

External Quickheaps

Experimental results

### Conclusions

## 1 Introduction

- Related work
- Incremental sorting

## 2 Quickheaps

- **Main idea**
- Quickheap operations
- External Quickheaps
- Experimental results

## 3 Conclusions



# Quickheaps

## Core idea

### Quickheaps

Navarro,  
Paredes

### Introduction

Related work  
Incremental sorting

### Quickheaps

Main idea  
Quickheap  
operations  
External Quickheaps  
Experimental results

### Conclusions



$$S = \{16, 8, 4, 1\}$$

- From right to left, there is a pivot and a chunk, a pivot and a chunk .... The first chunk can be empty.
- This resembles us a heap :-).
- This time **IQS** doesn't pop  $S$ .

### Quickheaps

Navarro,  
Paredes

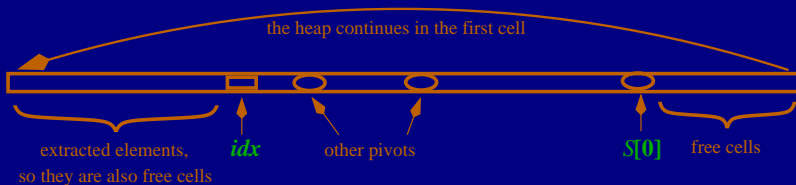
### Introduction

Related work  
Incremental sorting

### Quickheaps

**Main idea**  
Quickheap  
operations  
External Quickheaps  
Experimental results

### Conclusions



- An array *heap*.
- The stack *S*. The last pivot is on the bottom of *S*.
- An integer *idx* indicating the first cell of the quickheap.
- An integer *capacity* indicating the size of the quickheap.



# Outline

## Quickheaps

Navarro,  
Paredes

### Introduction

Related work

Incremental sorting

### Quickheaps

Main idea

**Quickheap  
operations**

External Quickheaps

Experimental results

### Conclusions

- 1 Introduction
  - Related work
  - Incremental sorting
- 2 **Quickheaps**
  - Main idea
  - **Quickheap operations**
  - External Quickheaps
  - Experimental results
- 3 Conclusions



# Quickheaps

## Basic operations

### Quickheaps

Navarro,  
Paredes

### Introduction

Related work

Incremental sorting

### Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

Experimental results

### Conclusions

**Quickheap**(Integer  $N$ ) // empty quickheap

1.  $capacity \leftarrow N + 1$ ,  $heap \leftarrow \text{new Array}[capacity]$ ,  $S \leftarrow \{0\}$
2.  $idx \leftarrow 0$

**Quickheap**(Array  $A$ , Integer  $N$ ) // heapifying  $A$

1.  $capacity \leftarrow \max\{N, |A|\} + 1$ ,  $heap \leftarrow \text{new Array}[capacity]$
2.  $S \leftarrow \{|A|\}$ ,  $idx \leftarrow 0$ ,  $heap.\text{copy}(A)$

**findMin**()

1.  $\text{IQS}(heap, idx, S)$
2. **Return**  $heap[idx \bmod capacity]$

**extractMin**()

1.  $\text{IQS}(heap, idx, S)$ ,  $idx \leftarrow idx + 1$ ,  $S.\text{pop}()$
2. **Return**  $heap[(idx - 1) \bmod capacity]$



# Quickheaps

Operation **insert** 1

## Quickheaps

Navarro,  
Paredes

### Introduction

Related work  
Incremental sorting

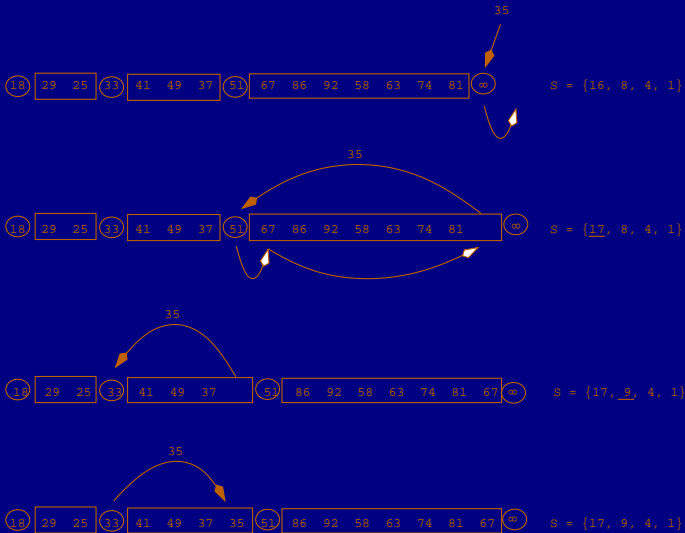
### Quickheaps

Main idea

Quickheap  
operations

External Quickheaps  
Experimental results

### Conclusions





# Quickheaps

Operation **insert** 2

Quickheaps

Navarro,  
Paredes

Introduction

Related work

Incremental sorting

Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

Experimental results

Conclusions

**insert**(Elem  $x$ )

1.  $pidx \leftarrow 0$
2. **While** TRUE **Do** // moving pivots, starting from pivot  $S[pidx]$
3.      $heap[(S[pidx] + 1) \bmod capacity] \leftarrow heap[S[pidx] \bmod capacity]$
4.      $S[pidx] \leftarrow S[pidx] + 1$
5.     **If**  $(|S| = pidx + 1)$  OR // we are in the first chunk  
        $(heap[S[pidx + 1] \bmod capacity] \leq x)$  **Then** // we found the chunk  
           $heap[(S[pidx] - 1) \bmod capacity] \leftarrow x$ , **Return**
6.     **Else**
7.          $heap[(S[pidx] - 1) \bmod capacity] \leftarrow$   
           $heap[(S[pidx + 1] + 1) \bmod capacity]$
8.          $pidx \leftarrow pidx + 1$  // go to next chunk
- 9.



# Quickheaps

## Other operations

### Quickheaps

Navarro,  
Paredes

### Introduction

Related work  
Incremental sorting

### Quickheaps

Main idea  
Quickheap  
operations  
External Quickheaps  
Experimental results

### Conclusions

- Operation **delete**( $x$ ) is the dual of **insert** (we need to know the position of  $x$ ).
- To implement operation **decreaseKey**( $x, \delta$ ), we need the position of  $x$ ). Next, we decrease *value* and move  $x$  towards *idx*.
- To implement operation **increaseKey**( $x, \delta$ ), we need the position of  $x$ ). Next, we increase *value* and move  $x$  towards  $S[0]$ .



# The Quickheap's Self-Similarity Property

## Quickheaps

Navarro,  
Paredes

## Introduction

Related work

Incremental sorting

## Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

Experimental results

## Conclusions

- Key observation: quickheaps follow a *self-similar structure*:  $p\mathit{c}p\mathit{c}c\mathit{p}cccc\mathit{p}cccccccc\mathit{p} \dots p$ .
- The distribution of elements within a quickheap seen from the last chunk is the same as seen from the second last chunk, and so on.
- Formally:

Theorem (quickheap's self-similarity property)

*Given a segment heap[idx .. S[pidx] - 1], the probability of that its pivot is large is smaller than or equal to  $\frac{1}{2}$ , that is,*

$$\mathbb{P}(\text{pivot is large}) \leq \frac{1}{2}.$$



# Outline

Quickheaps

Navarro,  
Paredes

Introduction

Related work

Incremental sorting

Quickheaps

Main idea

Quickheap  
operations

**External Quickheaps**

Experimental results

Conclusions

- 1 Introduction
  - Related work
  - Incremental sorting
- 2 Quickheaps
  - Main idea
  - Quickheap operations
  - **External Quickheaps**
  - Experimental results
- 3 Conclusions

## Quickheaps

Navarro,  
Paredes

### Introduction

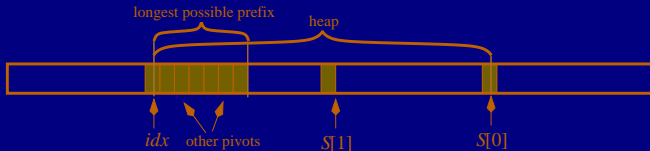
Related work  
Incremental sorting

### Quickheaps

Main idea  
Quickheap  
operations  
External Quickheaps  
Experimental results

### Conclusions

- Only **findMin**, **extractMin** and **insert**.
- We keep in main memory:
  - $S$ ,  $idx$  and  $capacity$ .
  - One disk block per pivot.
  - The longest possible prefix of *heap*.





# Outline

## Quickheaps

Navarro,  
Paredes

### Introduction

Related work

Incremental sorting

### Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

**Experimental results**

### Conclusions

- 1 Introduction
  - Related work
  - Incremental sorting
- 2 Quickheaps
  - Main idea
  - Quickheap operations
  - External Quickheaps
  - **Experimental results**
- 3 Conclusions



# Isolated quickheap operations

## Quickheaps

Navarro,  
Paredes

## Introduction

Related work

Incremental sorting

## Quickheaps

Main idea

Quickheap  
operations

External Quickheaps

Experimental results

## Conclusions

	Quickheaps	Binary heaps	Pairing heaps
<b>insert</b>	41	95	23
<b>extractMin</b>	$35 \log_2 m$	$53 \log_2 m$	$201 \log_2 m$
<b>increaseKey</b>	$18 \log_2 m$	$18 \log_2 m$	—
<b>decreaseKey</b>	$18 \log_2 m$	$20 \log_2 m$	$16 \log_2 m$



# Conclusions

## Quickheaps

Navarro,  
Paredes

## Introduction

Related work  
Incremental sorting

## Quickheaps

Main idea  
Quickheap  
operations  
External Quickheaps  
Experimental results

## Conclusions

- We have presented another priority queue, the **Quickheaps**.
- We experimentally compare **Quickheaps** state-of-the-art alternatives.
- We prove that the average amortized complexity of the basic operations is  $O(\log m)$  CPU time and  $O((1/B) \log(m/M))$  I/O time.



Quickheaps

Navarro,  
Paredes

Introduction

Related work  
Incremental sorting

Quickheaps

Main idea  
Quickheap  
operations  
External Quickheaps  
Experimental results

Conclusions

# Quickheaps

Gonzalo Navarro   Rodrigo Paredes

Department of Computer Science, University of Chile  
{raparede, gnavarro}@dcc.uchile.cl

2nd Workshop on Compression, Text and Algorithms  
2007