Copenhagen, 26 May 2000

Methodological Issues in Algorithm Experimentation: A Case Study of Heaps

Jesper Bojesen

UNI•C, Danish Computing Centre for
Research and Education
Technical University of Denmark

Jyrki Katajainen

Department of Computing
University of Copenhagen

Operations Are Free; Data Motion Isn't

Internal data structures:
Designed for a random access machine (RAM) with flat memory

External data structures:
Designed for a RAM with two-level memory

Reality:
A RAM with hierarchical memory

	Access time	**Volume**
Registers	$\geq 3/$ cycle	$\approx 2^{10}$ bytes
L1 cache	1–2 cycles	$\approx 2^{16}$ bytes
L2 cache	3–10 cycles	$\approx 2^{20}$ bytes
Internal memory	50–100 cycles	$\approx 2^{26}$ bytes
External memory	10^6–10^7 cycles	$\approx 2^{32}$ bytes

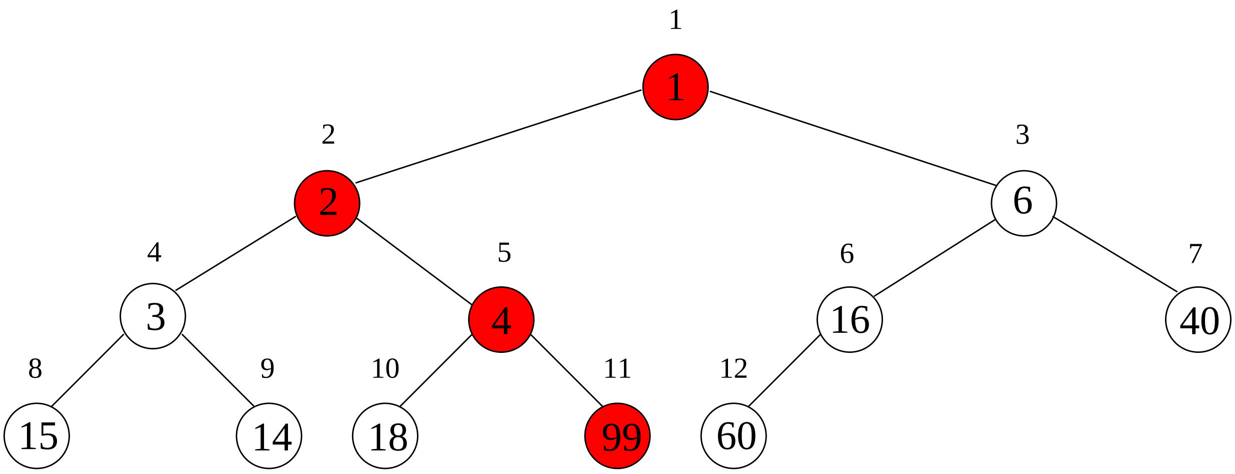
What Have We Done?

We have tried / will try

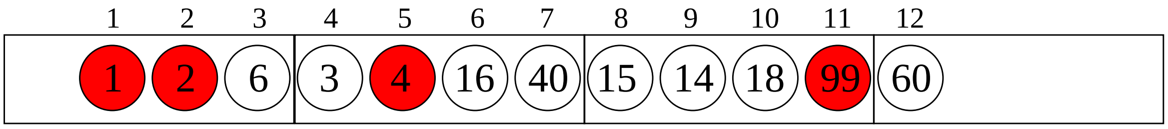
- to reproduce the results of some earlier experiments on the performance of various heap structures,
- to point out the anomalies observed, and
- to come to a conclusion what is the fastest heap structure.

Binary Heaps

A binary heap of 12 integers



and its implicit representation in an array



Internal Competitors

- AVL-trees
- leftist trees
- splay trees
- all other search trees
- binomial heaps
- skew heaps
- Fibonacci heaps
- pairing heaps
- etc.

But the implementation of all these structures requires **pointers**, and pointers are enemies of modern computers.

External Alternatives

- B-trees
- buffer trees
- multi-way heaps
- generalized binary heaps where each node stores the same number of elements in sorted order
- generalized multi-way heaps where nodes are at least half-full as in B-trees
- sequence heaps
- etc.

Typical Benchmarks

Heapsort:

$\texttt{insert}^N \ \texttt{delete\_min}^N$

Hold model:

$\texttt{insert}^N \ (\texttt{delete\_min insert})^T$

$e \leftarrow H.\texttt{delete\_min}()$
increase the priority of e by $-\texttt{ln}(\texttt{drand}())$
$H.\texttt{insert}(e)$

Minimum spanning trees:

$\texttt{insert}^N \ (\texttt{delete\_min decrease\_key}^{M_i})^N$,

$$\sum_{i=1}^{N} M_i \leq 2M$$

Shortest paths:

$\texttt{insert}^N \ (\texttt{delete\_min decrease\_key}^{M_i})^N$,

$$\sum_{i=1}^{N} M_i \leq 2M$$

Jones 1986

Figure 12 from his paper.

Operation sequence:
Hold model

Input data:
element size: 4 B; # elements: 1–$2^{13.5}$

Environment:
computer: VAX 11/780 running UNIX (BSD 4.2); cache: 8 kB; TLB: 64 entries; compiler: Berkeley Pascal with optimization enabled

LaMarca & Ladner 1996

Figure 17 from their paper.

Operation sequence:

Hold model?

`#define NOTSORANDNUM(x) (x + RANDNUM())`

Input data:

element size: 8 B; # elements: 2^{10}–2^{23}

Environment:

computer: DEC Alphastation 250; processor: Alpha 21064A 266 MHz; L1 cache: 8 kB; L2 cache: direct-mapped, 2 MB, 32 B per line; compiler?: `cc`

Sanders 1999

Figure 6 from his article.

Operation sequence:
$(\texttt{insert}\ (\texttt{delete\_min}\ \texttt{insert})^s)^N\ (\texttt{delete\_min}\ (\texttt{insert}\ \texttt{delete\_min})^s)^N$, $s \in \{0, 1, 4, 16\}$

Input data:
element size: 4 B, drawn randomly; satellite data: 4 B; # elements: 2^8–2^{23}

Environment:
computer: Sun Ultra-10 workstation; processor: Ultra-SparcIIi 300 MHz; L1 cache: 16 kB, 16 B per line; L2 cache: 512 kB, 32 B per line; compiler: Sun Workshop C++ 4.2 with options `-fast -O4`

Brengel et al. 1999

Table 1 from their paper.

Operation sequence:
`insert`^{N} / `delete_min`^{N}

Input data:
element size: 4 B, drawn randomly from $[0..10^7]$;
\# elements: $1 \cdot 10^6$–$200 \cdot 10^6$

Environment:
computer: Sparc Ultra 1/143; main memory: 256 MB, 8 kB per page; local disk: 9 GB fast-wide SCSI; logical block size: 64 kB; buffer size: 16 MB

Varieties in Computers

superscalar processors
speculative execution
branch prediction
instruction scheduling
pipelining

Be aware that there exists computers with

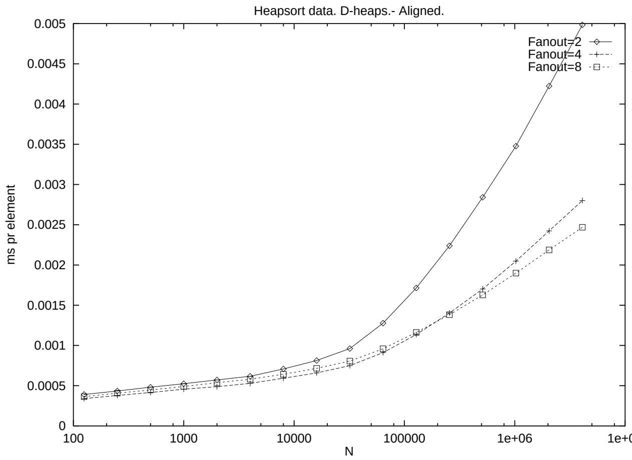
- a slow CPU and a fast memory and
- a fast CPU and a slow memory,

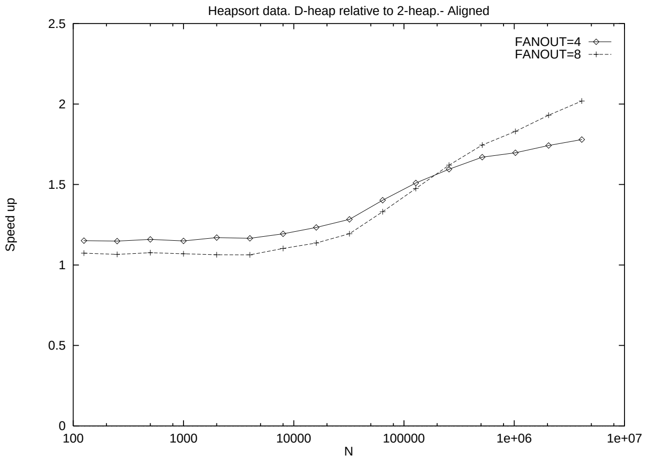
and anything there between; and modern computers seem to belong to the latter category.

Our fast-CPU slow-memory environment:
computer: IBM F50; processor: 332 MHz PowerPC; L1 cache: 4-way set associative, 32 kB, 32 B per line; L2 cache: 8-way set associative, 256 kB, 64 B per line

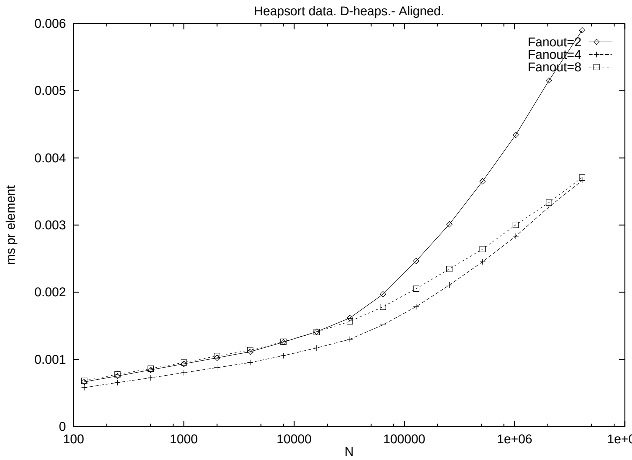
Our slow-CPU fast-memory environment:
computer: IBM RS/6000 397; processor: 160 Mhz P2SC; L1 cache: 4-way set associative, 128 kB, 128 B per line

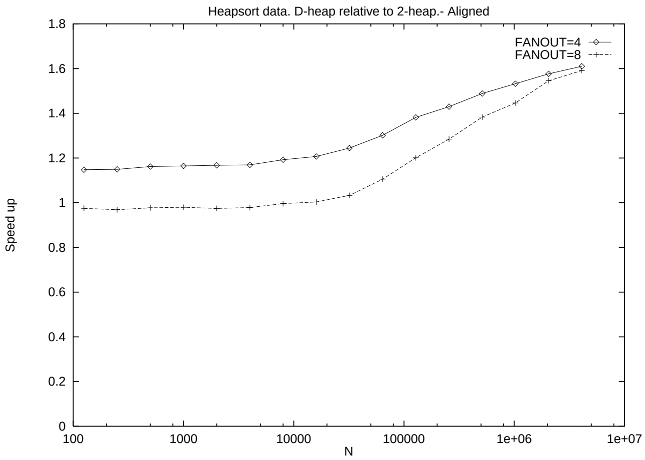
Quality of Compilers





Our fast-CPU slow-memory environment:
compiler: `KCC` from Kuch and Associates





Our fast-CPU slow-memory environment:
compiler: GNU **`g++`** version 2.8.1

Which Interface to Use?

LaMarca & Ladner (specialized, static):

```
typedef struct {
   long key; // long is 64 bits
} element;
void insert(element*, int, element);
```

Sanders (generic, static):

```
template <class key, class value, int capacity>
inline void heap<key, value>::insert(key, value);
```

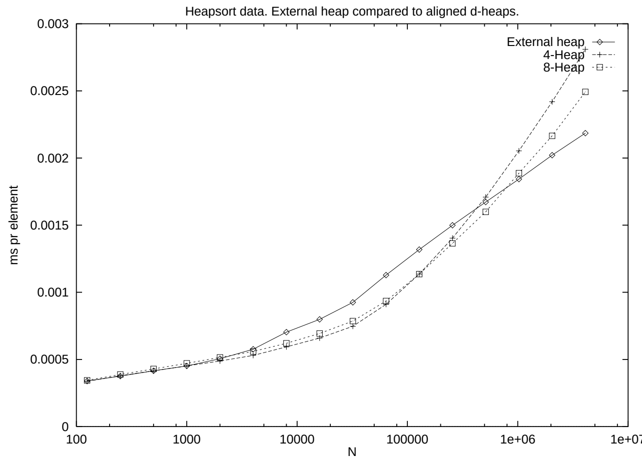
LEDA (generic, semi-dynamic):

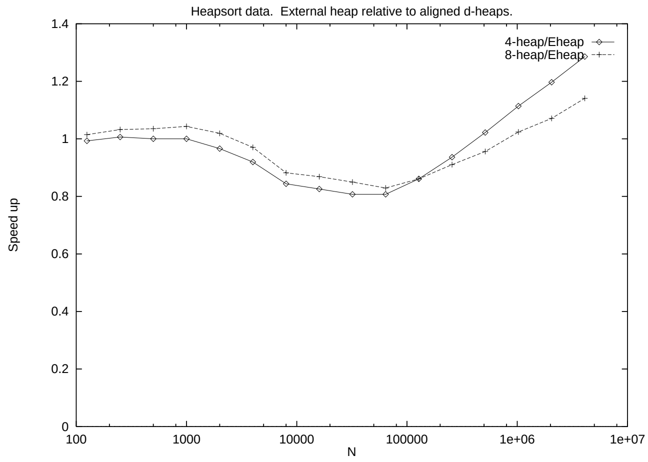
```
heap_item heap::insert(GenPtr k, GenPtr i);
```

STL (generic, semi-dynamic):

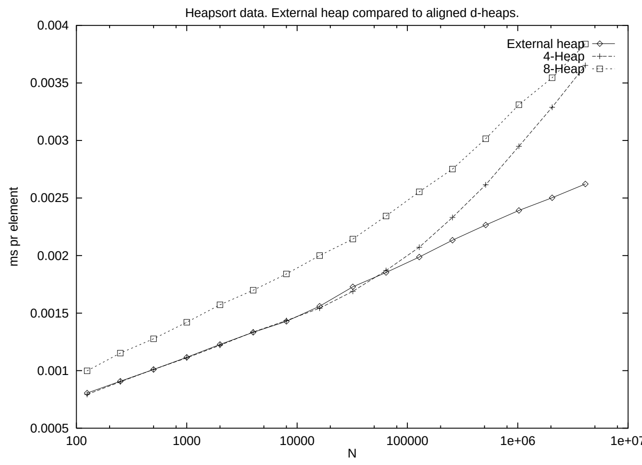
```
templete <class T, class C = vector<T>,
  class Cmp = less<typename C::value_type> >
class std::priority_queue<T, C, Cmp>::
push(const C::value_type&);
```

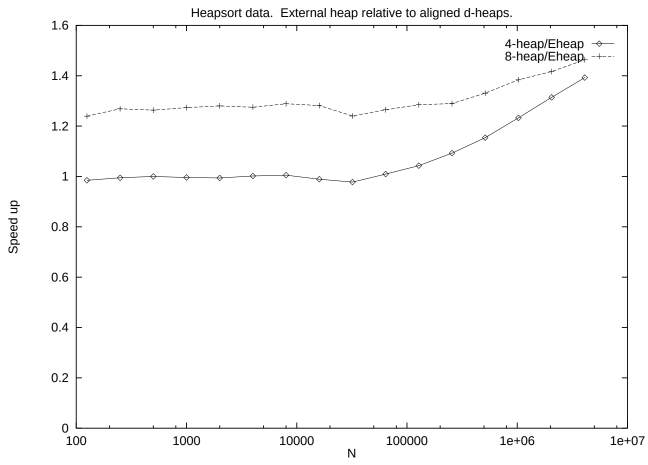
Generalized Heaps vs. d-Heaps





Our slow-CPU fast-memory environment:





Our fast-CPU slow-memory environment:

What Has to Be Done?

- Some more programs have to be written.
- Some more experiments have to be done.
- External-memory data structures require still a careful analysis.
- The programs should be stylized to be suitable for public scrutinies.

Conclusion

Instead of saying

> My program is two to three times faster than his.

I should say

> My program is two to three times faster than his,
>
> **if** I run the programs on a computer with a fast CPU and a slow memory,
>
> **if** I use the best available C++ compiler,
>
> **if** the size of the problem exceeds the capacity of the second-level cache, and
>
> **if** I do such and such tunings,
>
> **but not if** . . .