Descent into Cache-Oblivion

Benjamin Sach

March 13, 2008

University of BRISTOL

Outline

What is an External Memory Algorithm?

- The External Memory Model
- Two (simple) examples

Searching Cache-Obliviously

- Why aren't classic Binary Search Trees good enough?
- Static Cache-Oblivious Search Trees
- Making things Dynamic - allowing insertions and deletions

Sorting Cache-Obliviously

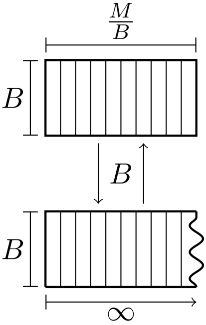
- Why isn't (binary) MergeSort good enough?
- FunnelSort - a Cache-Oblivious MergeSort

Summary and some Empirical Results

External Memory (EM) algorithms

The Problem

- Analysis in the RAM model relies on constant time memory access.



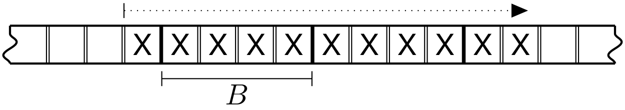
The Solution: The External Memory Model

- Two levels of memory, internal (of size M) and external (unbounded).
- Data transfer occurs in blocks of size B
- We analyse asymptotic I/O complexity.

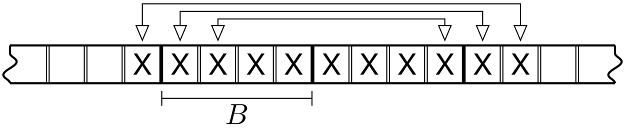
- A *Cache-Aware* algorithm knows M and B
- A *Cache-Oblivious* algorithm doesn't

Two (simple) examples of Cache-Oblvious algorithms

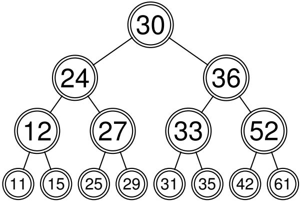
- Example 1: *Scanning* requires $\lceil \frac{N}{B} \rceil + 1 \in \Theta(\frac{N}{B})$ I/Os



- Example 2: *Array Reversal* requires $\lceil \frac{N}{B} \rceil + 1 \in \Theta(\frac{N}{B})$ I/Os

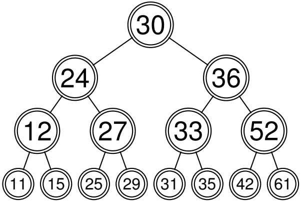


Why aren't classic Binary Search Trees good enough?



Data Values

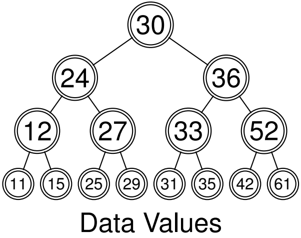
Why aren't classic Binary Search Trees good enough?



Data Values

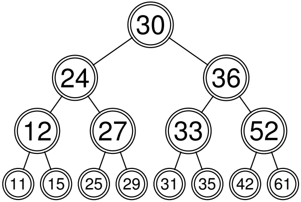
- a left child has a smaller or equal value

Why aren't classic Binary Search Trees good enough?



- a left child has a smaller or equal value
- a right child has a greater value

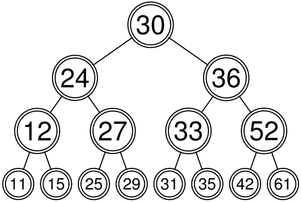
Why aren't classic Binary Search Trees good enough?



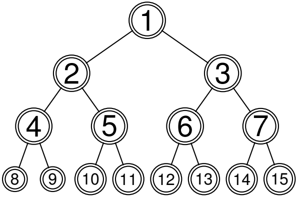
Data Values

- a left child has a smaller or equal value
- a right child has a greater value
- We can find an element in $O(\log N)$ time

Why aren't classic Binary Search Trees good enough?



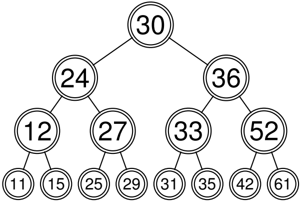
Data Values



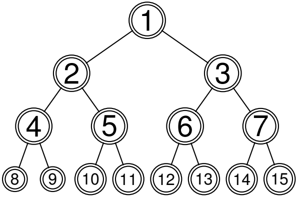
Breadth First Search Layout

- a left child has a smaller or equal value
- a right child has a greater value
- We can find an element in $O(\log N)$ time

Why aren't classic Binary Search Trees good enough?



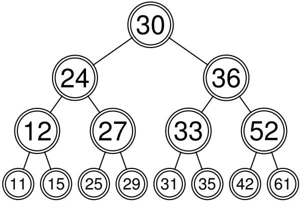
Data Values



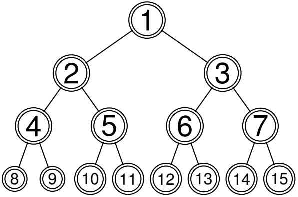
Breadth First Search Layout

- a left child has a smaller or equal value
- a right child has a greater value
- We can find an element in $O(\log N)$ time
- The two children of node with index i are at positions $2i$ and $2i+1$

Why aren't classic Binary Search Trees good enough?



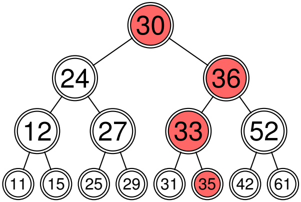
Data Values



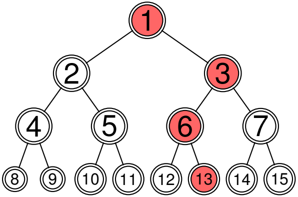
Breadth First Search Layout

- a left child has a smaller or equal value
- a right child has a greater value
- We can find an element in $O(\log N)$ time
- The two children of node with index i are at positions $2i$ and $2i+1$
- How many I/Os does this require?

Why aren't classic Binary Search Trees good enough?



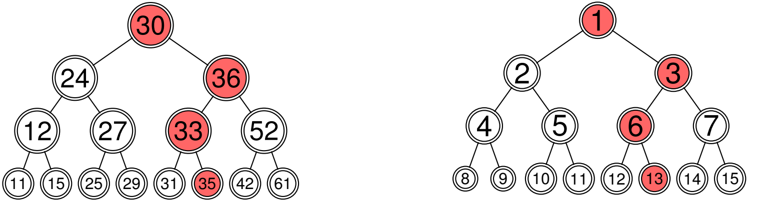
Data Values



Breadth First Search Layout

- a left child has a smaller or equal value
- a right child has a greater value
- We can find an element in $O(\log N)$ time
- The two children of node with index i are at positions $2i$ and $2i + 1$
- How many I/Os does this require?

Why aren't classic Binary Search Trees good enough?



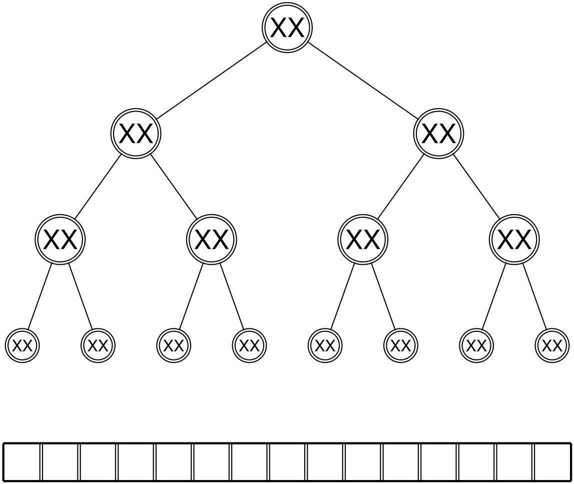
Data Values

Breadth First Search Layout

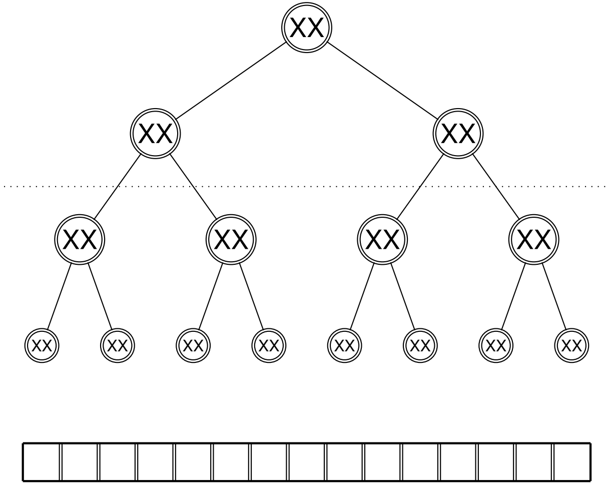
- a left child has a smaller or equal value
- a right child has a greater value
- We can find an element in $O(\log N)$ time
- The two children of node with index i are at positions $2i$ and $2i + 1$
- How many I/Os does this require?

$$O(\log N - \log B) = O(\log(N/B)) \text{ I/Os}$$

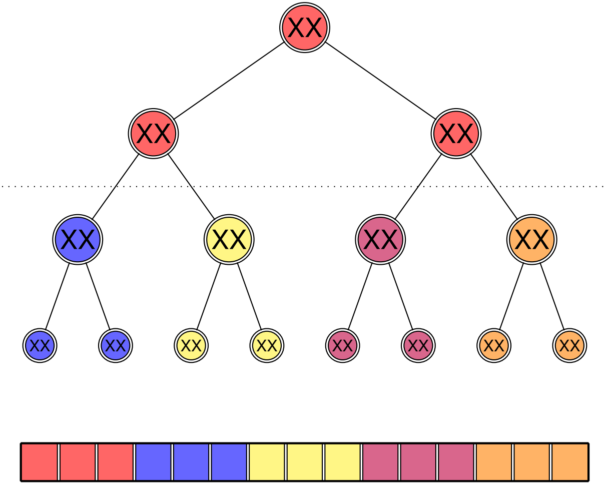
An example of the Van Emde Boas layout



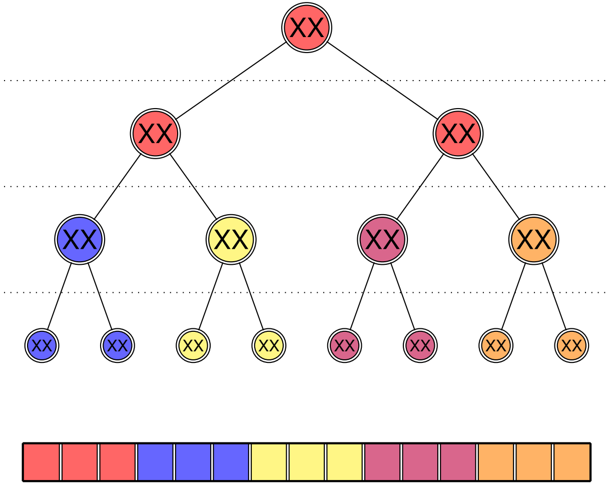
An example of the Van Emde Boas layout



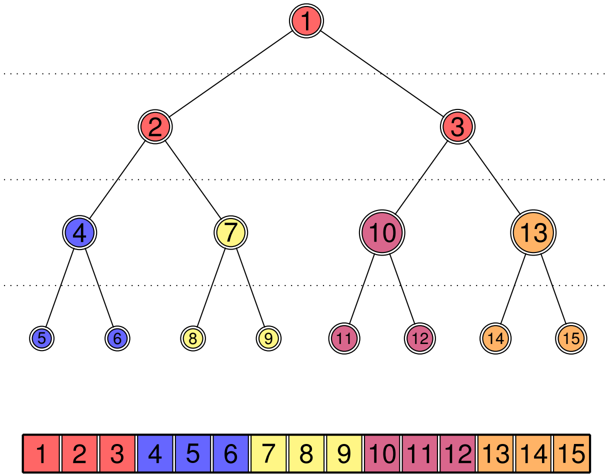
An example of the Van Emde Boas layout



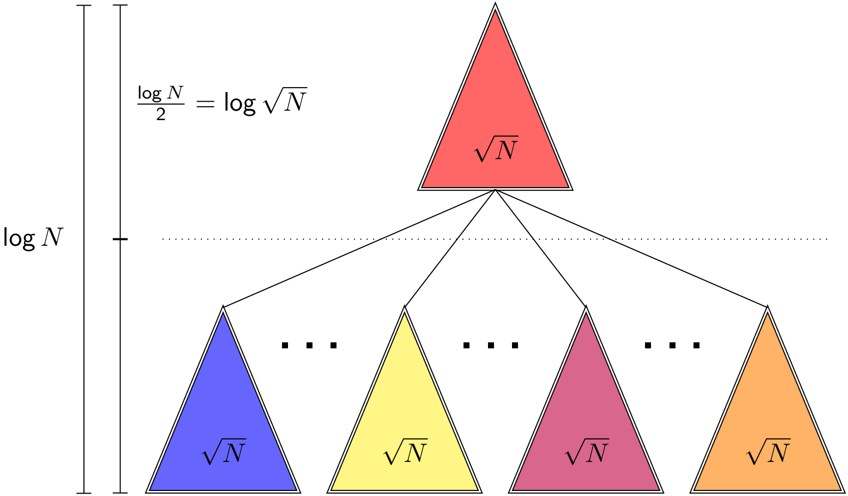
An example of the Van Emde Boas layout



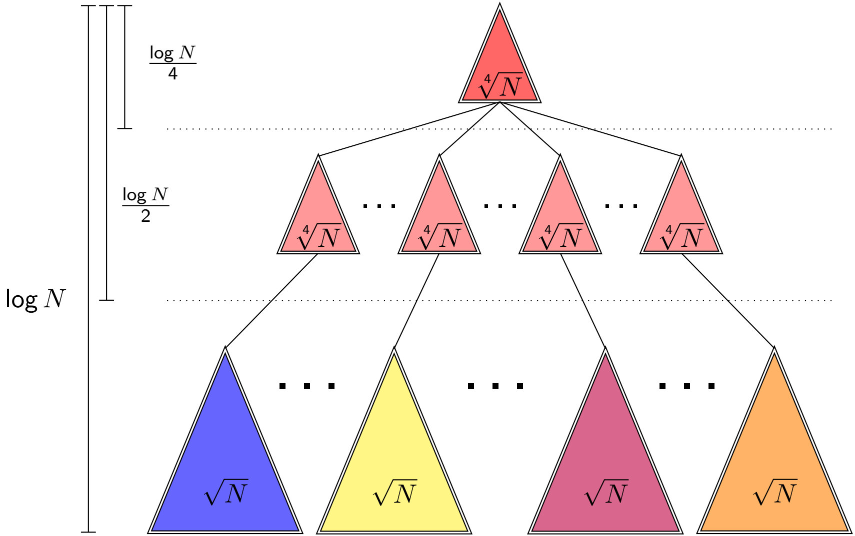
An example of the Van Emde Boas layout



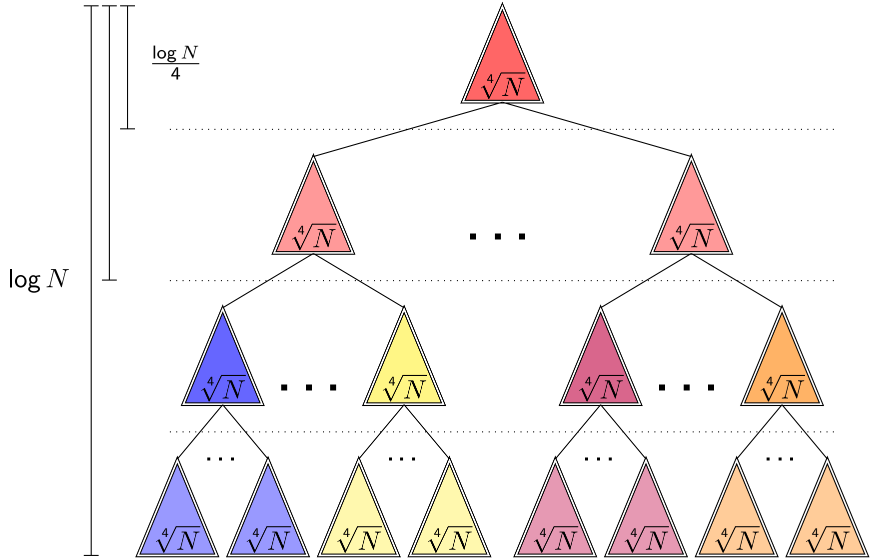
The Van Emde Boas layout



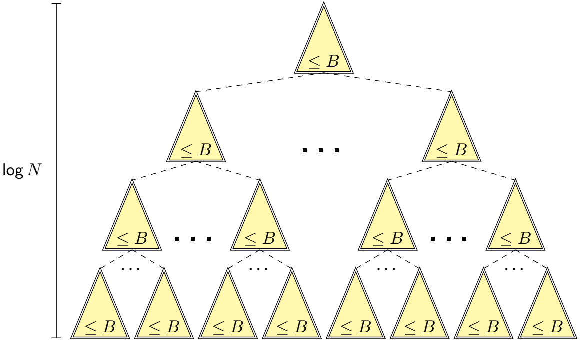
The Van Emde Boas layout -top expanded



The Van Emde Boas layout - level of detail $\sqrt[4]{N}$

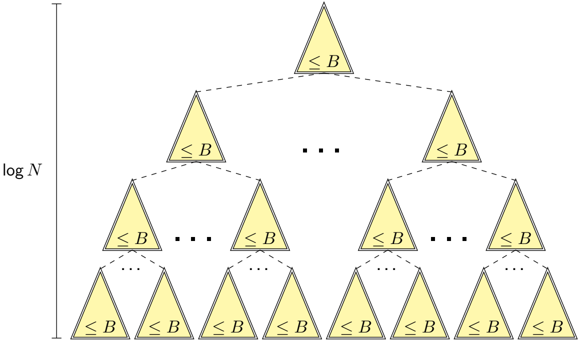


The Van Emde Boas layout - level of detail B



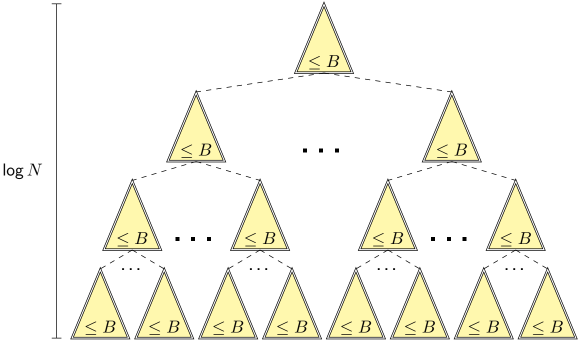
The Van Emde Boas layout - level of detail B

- conceptually recurse until all subtrees are smaller than B



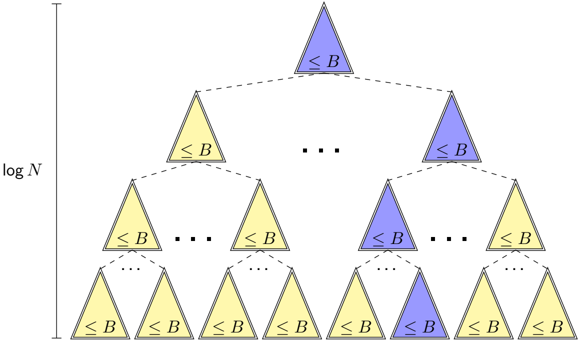
The Van Emde Boas layout - level of detail B

- conceptually recurse until all subtrees are smaller than B
- the subtrees are at least as large as $\sqrt{B}$



The Van Emde Boas layout - level of detail B

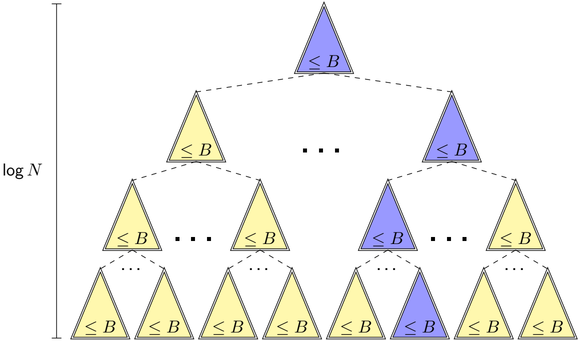
- conceptually recurse until all subtrees are smaller than B
- the subtrees are at least as large as $\sqrt{B}$
- how many such subtrees are on a path from root to leaf?



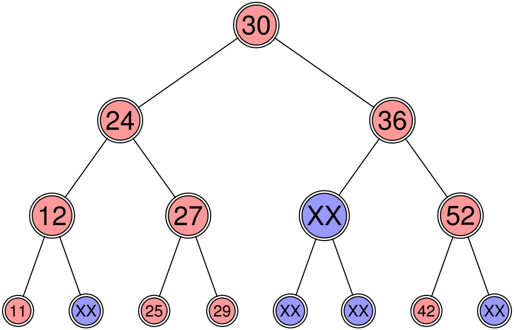
The Van Emde Boas layout - level of detail B

- conceptually recurse until all subtrees are smaller than B
- the subtrees are at least as large as $\sqrt{B}$
- how many such subtrees are on a path from root to leaf?

$$\sim \log N / \log \sqrt{B} \in O(\log N / \log B) = O(\log_B N)$$

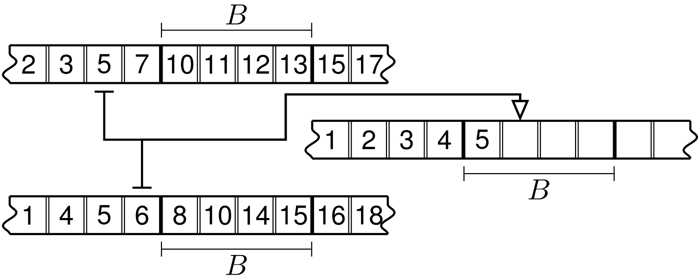


Making things Dynamic - allowing insertions and deletions

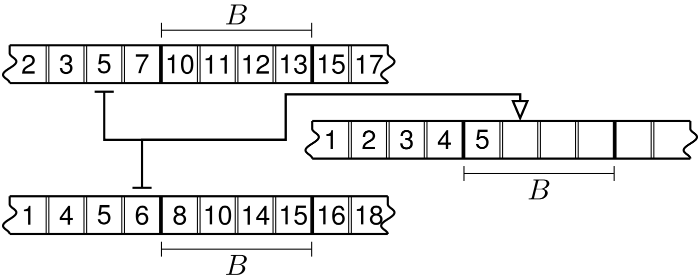


- Idea : embed the dynamic tree in a larger static tree
- If the dynamic tree becomes too unbalanced,re-distribute nodes
- If the dynamic tree becomes too large, re-construct
- Insertions and Deletions can be performed in $O(\log^2 N/B)$ I/Os

Why isn't (binary) MergeSort good enough?

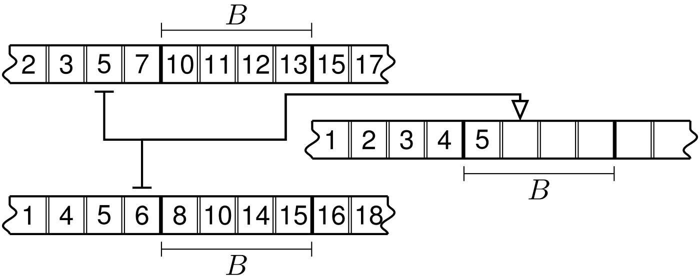


Why isn't (binary) MergeSort good enough?



$$T(N) = 2T(N/2) + O(N/B)$$

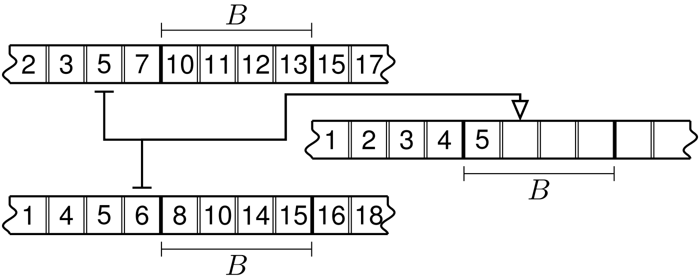
Why isn't (binary) MergeSort good enough?



$$T(N) = 2T(N/2) + O(N/B)$$

- base case : $T(O(1)) = O(1) \implies T(N) \in O(\frac{N}{B} \log N)$ I/Os

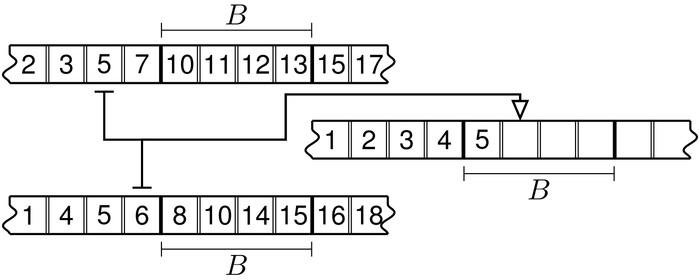
Why isn't (binary) MergeSort good enough?



$$T(N) = 2T(N/2) + O(N/B)$$

- base case : $T(O(1)) = O(1) \implies T(N) \in O(\frac{N}{B} \log N)$ I/Os
- base case : $T(O(B)) = O(1) \implies T(N) \in O(\frac{N}{B} \log \frac{N}{B})$ I/Os

Why isn't (binary) MergeSort good enough?



$$T(N) = 2T(N/2) + O(N/B)$$

- base case : $T(O(1)) = O(1) \implies T(N) \in O(\frac{N}{B} \log N)$ I/Os
- base case : $T(O(B)) = O(1) \implies T(N) \in O(\frac{N}{B} \log \frac{N}{B})$ I/Os

(M/B)-way MergeSort gives $\Theta(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$ I/Os (Cache-Aware)

A Cache-Oblivious MergeSort?

What if we had a black box which merged K sorted lists of total size K^3 in $O(\frac{K^3}{B} log_{\frac{M}{b}} \frac{K^3}{B} + K)$ I/Os?

A Cache-Oblivious MergeSort?

What if we had a black box which merged K sorted lists of total size K^3 in $O(\frac{K^3}{B} log_{\frac{M}{b}} \frac{K^3}{B} + K)$ I/Os?

1. Split the array into $K = N^{1/3}$ segments of length $N/K = N^{2/3}$
2. Recursively sort each segment
3. Merge the sorted segments in $O(\frac{N}{B} log_{\frac{M}{b}} \frac{N}{B} + N^{1/3})$ I/Os

A Cache-Oblivious MergeSort?

What if we had a black box which merged K sorted lists of total size K^3 in $O(\frac{K^3}{B} log_{\frac{M}{b}} \frac{K^3}{B} + K)$ I/Os?

1. Split the array into $K = N^{1/3}$ segments of length $N/K = N^{2/3}$
2. Recursively sort each segment
3. Merge the sorted segments in $O(\frac{N}{B} log_{\frac{M}{b}} \frac{N}{B} + N^{1/3})$ I/Os

$$T(N) = N^{1/3} T(N^{2/3}) + O(\frac{N}{B} log_{\frac{M}{b}} \frac{N}{B} + N^{1/3}) \in (\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$$

A Cache-Oblivious MergeSort?

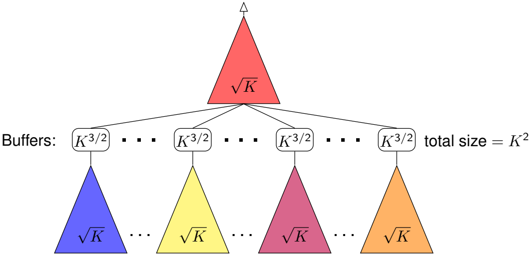
What if we had a black box which merged K sorted lists of total size K^3 in $O(\frac{K^3}{B} log_{\frac{M}{b}} \frac{K^3}{B} + K)$ I/Os?

1. Split the array into $K = N^{1/3}$ segments of length $N/K = N^{2/3}$
2. Recursively sort each segment
3. Merge the sorted segments in $O(\frac{N}{B} log_{\frac{M}{b}} \frac{N}{B} + N^{1/3})$ I/Os

$$T(N) = N^{1/3}T(N^{2/3}) + O(\frac{N}{B} log_{\frac{M}{b}} \frac{N}{B} + N^{1/3}) \in (\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$$

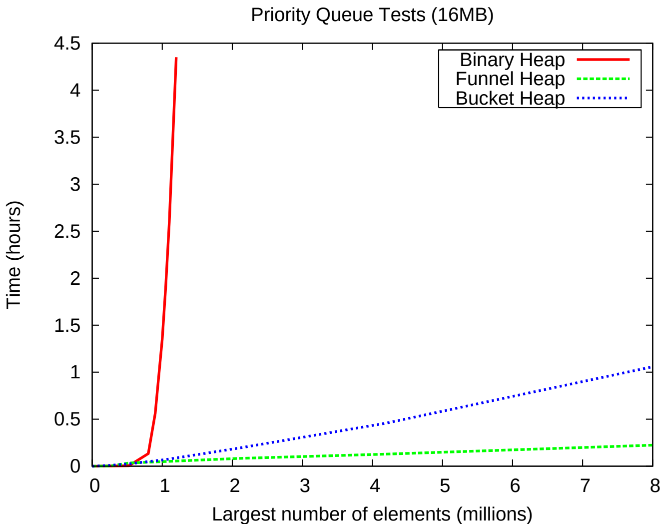
(assuming the tall cache assumption that $M > B^2$)

A K-funnel - the idea

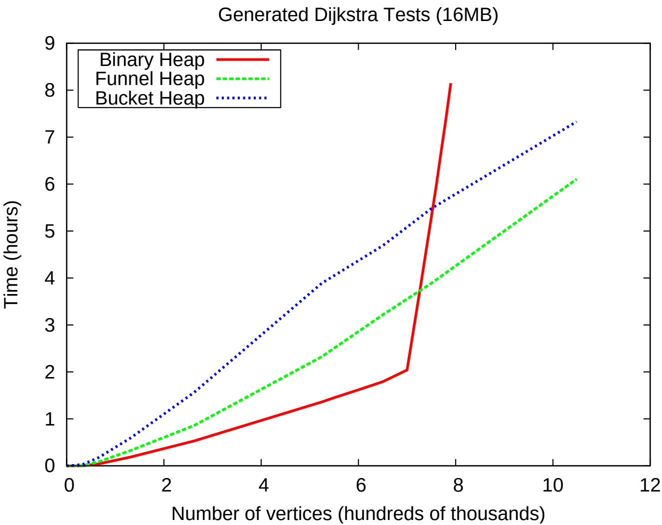


- Look at the largest level of detail J such that a J-funnel occupies less than $M/4$ space
- We can hold a J-funnel and one block of each of its input buffers in memory.
- Empty input buffers are completely refilled. This may push the original Funnel out of memory, which the 'new' J^3 elements pay for.

Some empirical results (1)



Some empirical results (2)



Summary and Questions

- Scanning/Array Reversal: $\Theta(N/B)$ I/Os
- Searching: $\Theta(\log_B N)$ I/Os
- Sorting: $\Theta(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$ I/Os

Questions?