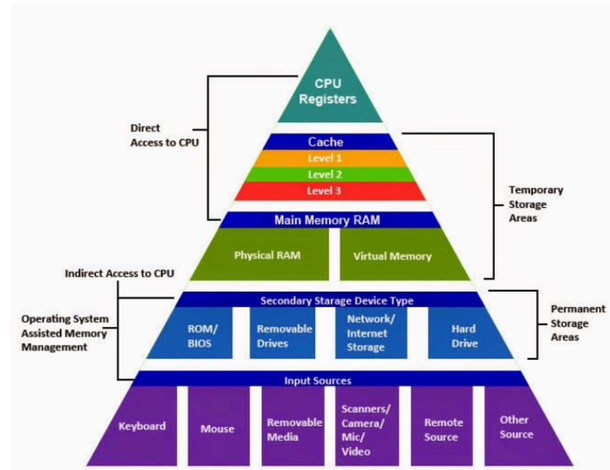
Lecture 20:
Heaps & Heapsort

CS 62
Fall 2017
Kim Bruce & Alexandra Papoutsaki

Memory Hierarchies



Access Time

Registers: Typical access time: One clock cycle.

Cache: Tens to hundreds of clock cycles.

Main Memory: Hundreds of clock cycles.

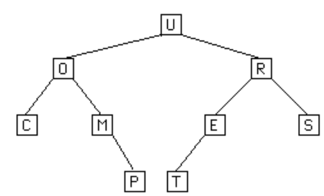
Secondary Memory: Millions of clock cycles.

Removable memory: Tens of millions of clock cycles

3 Ghz processor performs 3 billion clock cycles per second

Array Representation of Trees

- data[0..n-1] can hold values in trees
  - left subtree of node i in 2*i+1, right in 2*i+2,
  - parent in (i-1)/2



Indices:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
data[]:	U	O	R	C	M	E	S	–	–	–	P	T	–	–	–

Min-Heap

- Min-Heap H is complete binary tree s.t.
  - H is empty, or
  - Both of the following hold:
    - The value in root position is smallest value in H
    - The left and right subtrees of H are also heaps.
      Equivalent to saying parent ≤ both left and right children
- Excellent implementation for priority queue
  - Dequeue elements w/lowest priority values before higher

PriorityQueue

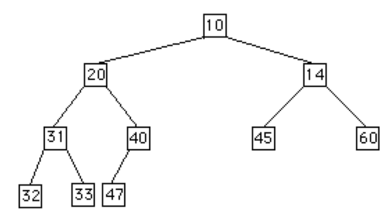
```
public interface PriorityQueue<E extends Comparable<E>>
{

    /**
     * @pre !isEmpty()
     * @return The minimum value in the queue.
     */
    public E remove();
    public E getFirst();
    public void add(E value);
    public boolean isEmpty();
    public int size();
    public void clear();
}
```

Implementations

- As regular queue (array or linked) where either keep in order or search for lowest to remove:
  - One of add or remove will be O(n)
- Heap representation (in arraylist) is more efficient: O(log n) for both add and remove.
  - Insert into heap:
    - Place in next free position,
    - "Percolate" it up.
  - Delete:
    - remove root,
    - move last element in array up to root. "Push" it down.

Insert 15:



IndexRange:	0	1	2	3	4	5	6	7	8	9	10
data:	10	20	14	31	40	45	60	32	33	47	-

IndexRange:	0	1	2	3	4	5	6	7	8	9	10
data:	10	20	14	31	*40*	45	60	32	33	47	15

IndexRange:	0	1	2	3	4	5	6	7	8	9	10
data:	10	*20*	14	31	15	45	60	32	33	47	40

IndexRange:	0	1	2	3	4	5	6	7	8	9	10
data:	10	15	14	31	20	45	60	32	33	47	40

Deleting from Heap

- Trickier!
- Remove top (smallest element)
- Move last element in array to top
  - *This is a large element!!*
- "Push" it down while larger than either child
  - *Swap with smallest child if larger than it.*
- What is worst case complexity?

See VectorHeap code

Called PriorityQueue class in standard Java

Sorting with Trees

Tree Sort

- Build Binary search tree (later)
- Do Inorder traversal, adding elts to array
  - Inorder traversal: O(n)
  - Building tree:
    - log 1 + log 2 + ... + log n = O(n log n) in best (& average) case
    - $O(n^2)$ in worst case
- O(n log n) in best & average case
- $O(n^2)$ in worst case :-(*What is worst case?*
- Heapsort is always better!