Priority Queues + Heaps I

Naive Solution:
- Store items in linear list
- Order?
  - Insert order - fast insert / slow extract
  - Priority order - fast extract / slow insert

Priority Queue:
- Stores key-value pairs
- Key ≡ priority
- Ops: insert(x,v) - insert value v with key x
  - extract-min - remove/return pair with min key value

```
void insert(Key x)        (ignore value)
    n++; i ← sift-up(n,x)
    A[i] ← x

int sift-up(int i, Key x)
    while (i>1 && x<A[par(i)])
        A[i] ← A[par(i)]
        i ← par(i)
    return i
```

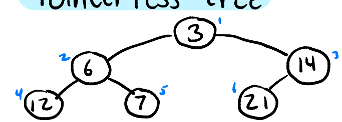
Heap: Tree-based structure

(min) heap order: for all nodes, parent's key ≤ node's key

[Reverse: max-heap order]

Many variants: Binary, leftist, binomial, Fibonacci, pairing, quake, skew ... heaps

"Pointerless" tree

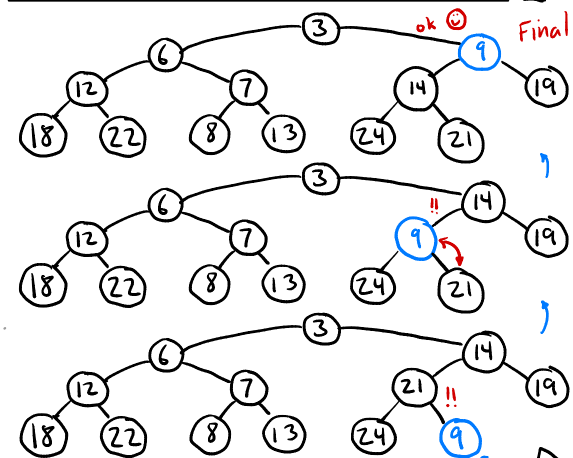


A:

0	1	2	3	4	5	6	...
X	3	6	14	12	7	21	

- left(i): if(2·i ≤ n) 2·i else null
- right(i): if(2·i+1 ≤ n) 2·i+1 else null
- par(i): if (i ≥ 2) ⌊i/2⌋ else null

Binary Heap:
- simple, elegant, efficient
- old (1964)
- basic: insert/extract - $O(\log n)$
  build - $O(n)$



Insert(x):

insert(9)

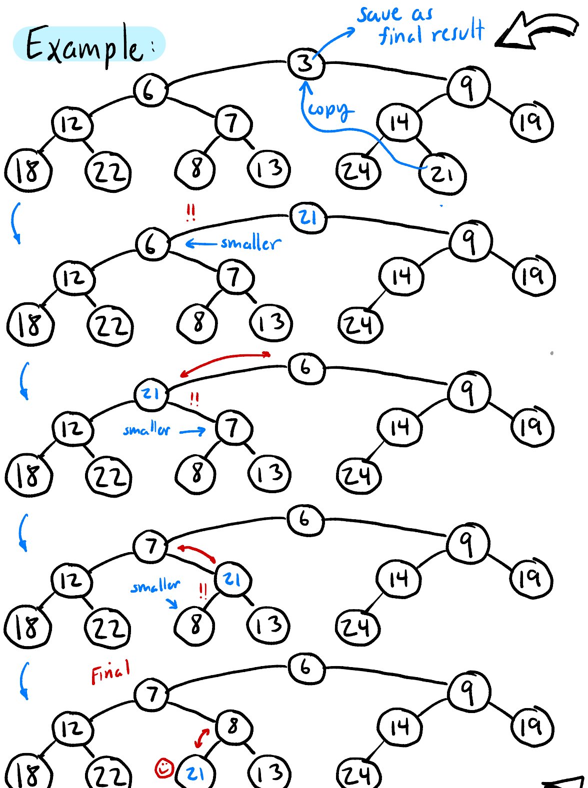
- Append x to end of array
- Sift x up until its parent's key is smaller (or reaching root)

Priority Queues & Heaps II

Binary Heap - Extract Min

- Min key at root → save it
- Copy A[n] to root (A[1]) & decrement n
- Sift the root key down
  - find smaller of two children
  - if larger, swap with this child
- Return saved root key

Example:




```
Key extract-min()
    if (n == 0) Error - Empty heap
    result ← A[1]
    z ← A[n--]        // get replacement
    i ← sift-down(i, z)
    A[i] ← z
    return result
```

```
int sift-down(int i, Key z)
    while (left(i) ≠ null)
        u ← left(i); v ← right(i)
        if (v ≤ n && A[v] < A[u])
            u ← v       // A[u] is smaller child
        if (A[u] < z)
            A[i] ← A[u]; i ← u
        else break
    return i
```

Analysis: Both insert & extract-min take time proportional to tree height. Tree is complete ⇒ $O(\log n)$ time

Leftist Heaps: Meldable heaps

- Can merge two heaps into single heap
- E.g. One processor breaks. Awaiting jobs must be merged with another processor.

Leftist Property: Null path length

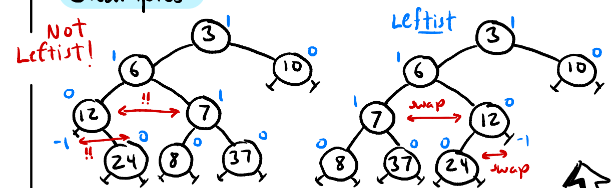
npl(v) = length of shortest path to null

$$\mathrm{npl}(v) = \begin{cases} -1 & \text{if } v = \text{null} \\ 1 + \min\begin{pmatrix}\mathrm{npl}(v.\mathrm{left}) \\ \mathrm{npl}(v.\mathrm{right})\end{pmatrix} & \text{o.w.} \end{cases}$$

Def: Leftist Heap is binary tree where:

- Keys are heap ordered
- ∀ nodes v, npl(v.left) ≥ npl(v.right)

Examples:



Priority Queues + Heaps III

Class structure:

```
LeftistHeap<Key>{
    private class LHNode{
        Key x
        LHNode left, right
        int npl
    }
    private LHNode root
    public LeftistHeap() {root ← null}
    "   void insert(Key x)
    "   Key extractMin()
    "   void mergeWith(LeftistHeap H2)
    ... (other private/protected utilities)
```

inner class - used only by LeftistHeap

references root node

constructor

public functions

```
public mergeWith(LeftistHeap H2){
    root ← merge(this.root, H2.root)
    H2.root ← null
}
```

helper function

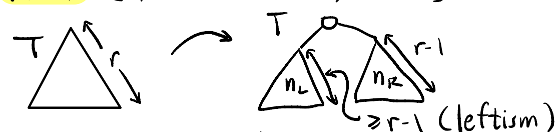
merger destroys H2

Lemma:

A leftist tree with $r \geq 1$ nodes along its rightmost path has $n \geq 2^r - 1$ nodes

Proof:

(Sketch - see latex notes)



By induction: $n_L \geq 2^{r-1} - 1$, $n_R \geq 2^{r-1} - 1$

$n = 1 + n_L + n_R \geq (2 \cdot 2^{r-1} - 1) + 1 = 2^r - 1$ $\square$

Analysis:

Time ~ Rightmost path $= \mathcal{O}(\log n)$

Insert + Extract-min? Exercises

```
LHNode merge(LHNode u, LHNode v){
    if (u == null) return v
    if (v == null) return u
    if (u.key > v.key)          // swap so u is smaller
        swap u ↔ v
    if (u.left == null) u.left ← v
    else
        u.right ← merge(u.right, v)
        if (u.left.npl < u.right.npl)
            swap u.left ↔ u.right
        u.npl ← u.right.npl + 1
    return u
```

Merge helper:

2 phases

1. Merge right paths by order of keys + update npl's
2. Check leftist property + swap