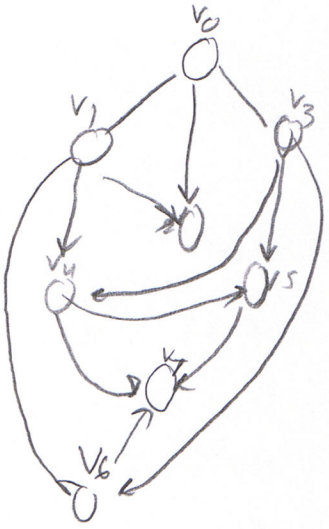
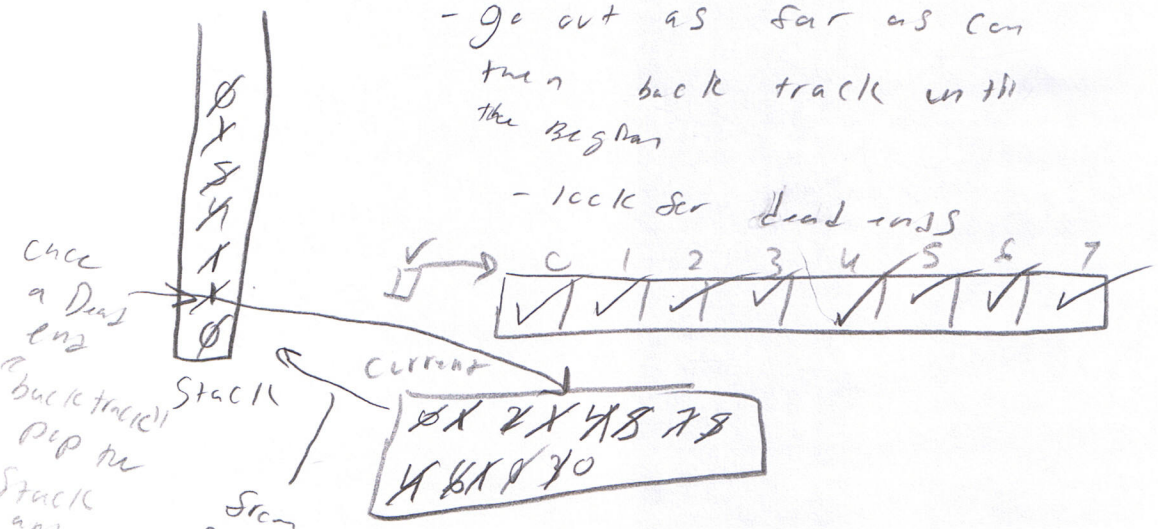
Depth first



- Starts with
  stack Empty
  Current = 0
- zero visited

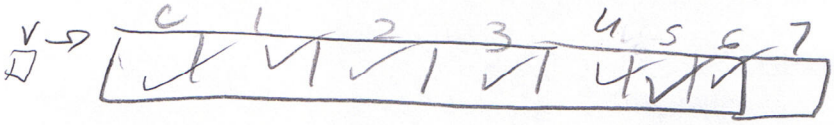
Depth first Traversal

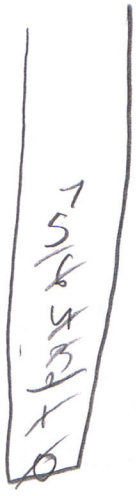
- go out as far as can then back track until the beginning
- look for dead ends



- who can you go to from this vertex
  (pick one and take that root)
  (pick the lowest vertex order) ← although it doesn't matter
- once there's no where to go backtrack

Breath First Traversal

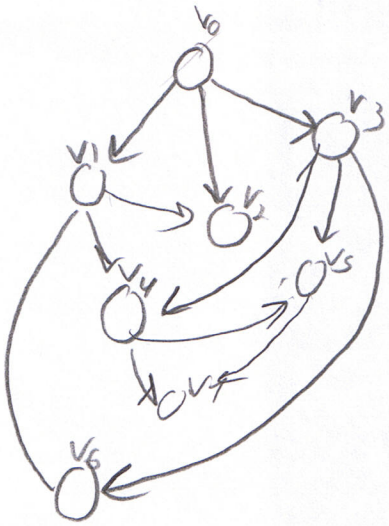




current

0 1 2 3 4 5 6 7

- Starts with zero on the queue current empty
- Zero visited



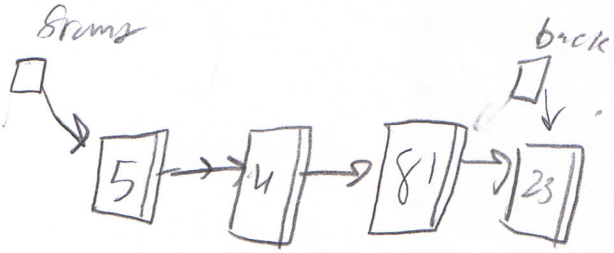
Queue
Start with zero then dequeue and whatever is dequeued becomes current Then every vertex it touches is put on the queue and marked visited

Cmp 338

12.15.09

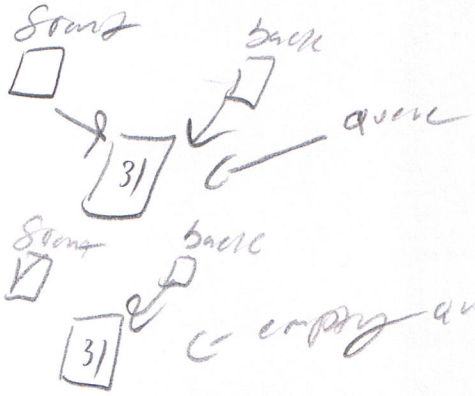
Test B

2) e

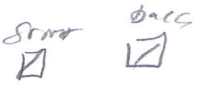


If all of the nodes are dequeued front becomes null

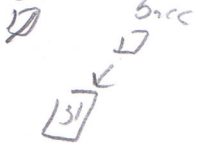
If back points to sentinel and front is null it's an empty queue



- If there both front back null the Queue is empty



- If the front is null and the back points to a node



CMP 336

12·15·01

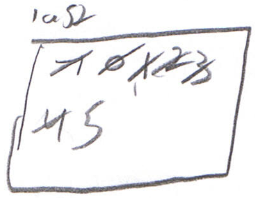
(4)

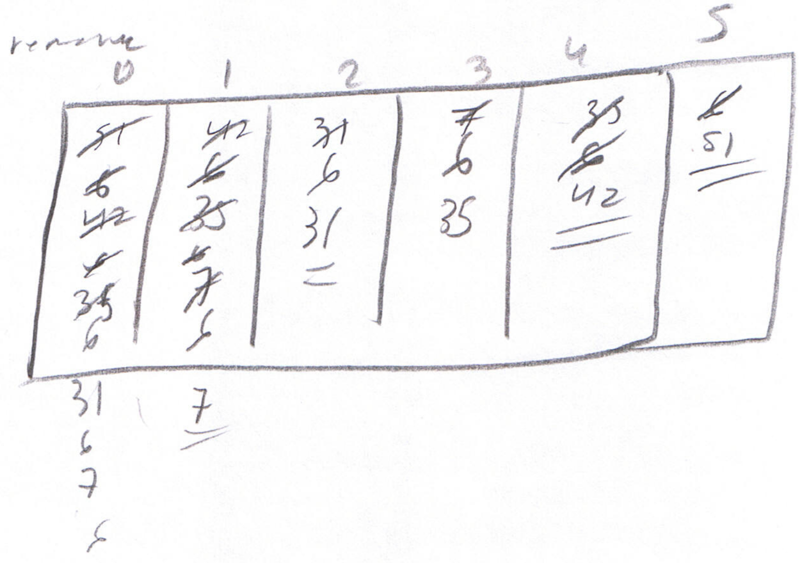
Heap

31, 7, 42, 35, 51, 6

Parent of

$\frac{n-1}{2}$

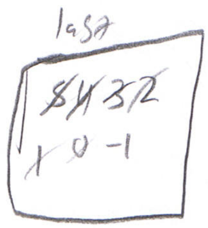




Child of

(2n)+1

(2n)+2



Code

Insert

last ++

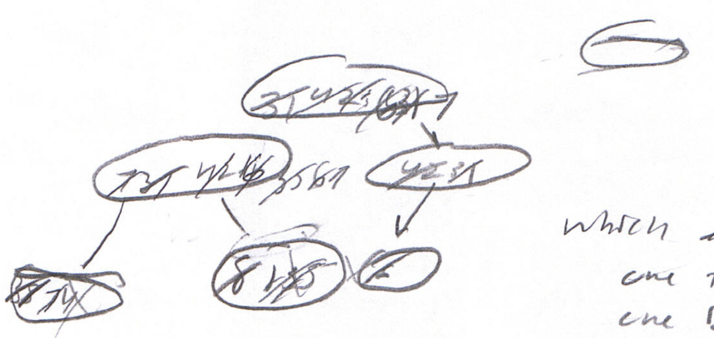
heapsify upwards

Remove

Swap(A, Root, last)

last --;

heapsify Downwards



Which ever one the last one is

Comp 338

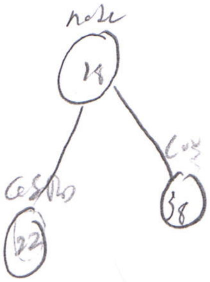
12.15.01

(5)

heapifydown ward

there has to be a right node first

public void heapifyDownward (int node){



if (right(node <= last) && (value [right(node)] > value of left node) && value [right(node)] > value[node]);

← If there is a right node & its bigger than the left node and its bigger than its parent

swap(value right(node)

If theres then swap(value, right(node), node);

else

if ((left(Node) <= last) && (value[left node])] > value[node]);

// If there is left node

}

public int left (int node){

return (2 node)+1

}

If thers a right node theres got to be a left node because its complete

public int right (int node){

return (2 node)+2;

}

Comp 328

12.15.05


```
if ((right(node) <= last) && (value[right(node)] > value[last(node)])
                          && (value[right(node)] > value[node])){
    swap(value, right(node), node);
    heapifyDownward(right(node));
}
else
if ((left(node) <= last) && (value(node) > value[node])){
    swap(value, left(node), node);
    HeapifyDownward(left(node), node);
```

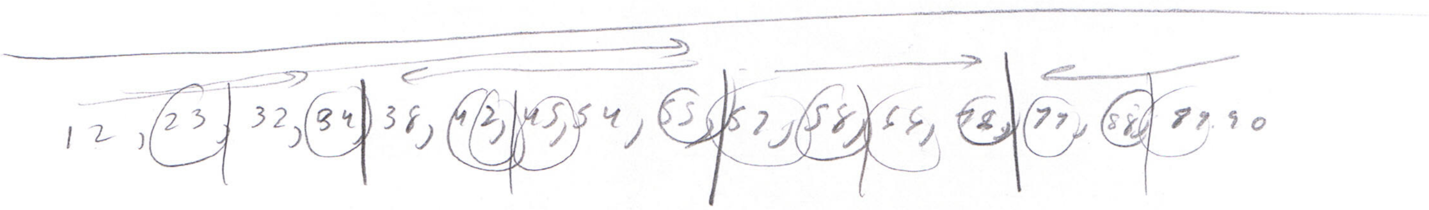
Heap Sort complexity

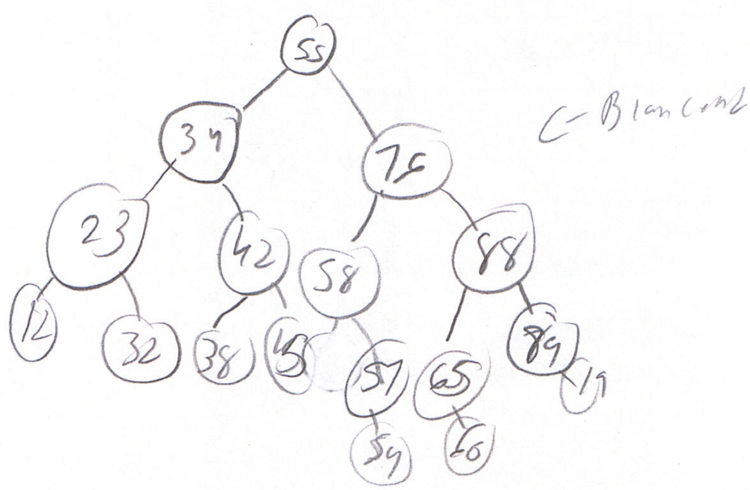
Insert Θn
remove Θn

- the hight of a complete Binary tree is Θ log n
- always complete

Insert & removing
→ n Log n

n → Insert → moving throw the heap (log n) = n log n





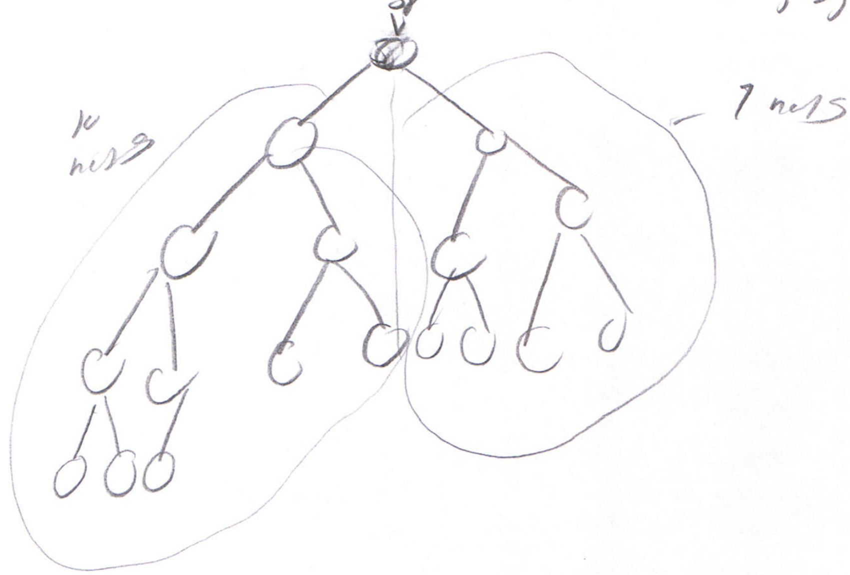
Cmp 338

12-15-09

← 10 nodes

12, 23, 32, 34, 38, 42, 45, 54, 55, 57, 58, 65, 66, 76, 77, 88, 89, 90

7 nodes →



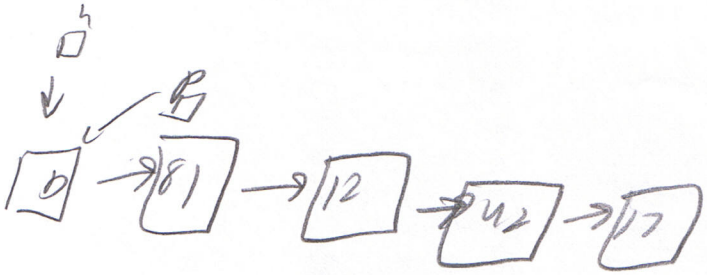
$Left(n) = 2n+1$

Better to store heap in an array than a linked structure

Random Access

Less memory

Just stores entries




```
public void remove(int x)
prev=head
while prev.next.x != x
    prev=prev.next;
prev.next=prev.next.next
}
```