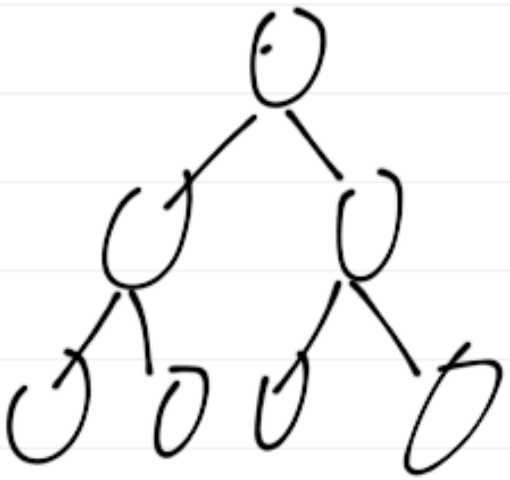
Theory

Practice

binary heap



Lab 3: quicksort

Lab 4: OMP

Lab 1: OMP

- first tree data structure
- but still uses array (same array as in lab 3)

$2i, 2i+1$

- Use array_t class similar to STL containers

STANDARD TEMPLATE LIB

---

e.g. insertion sort using STL Vector (like array_t)

```
template<typename T>
void insertion_sort(
        std::vector<T>& a)
```

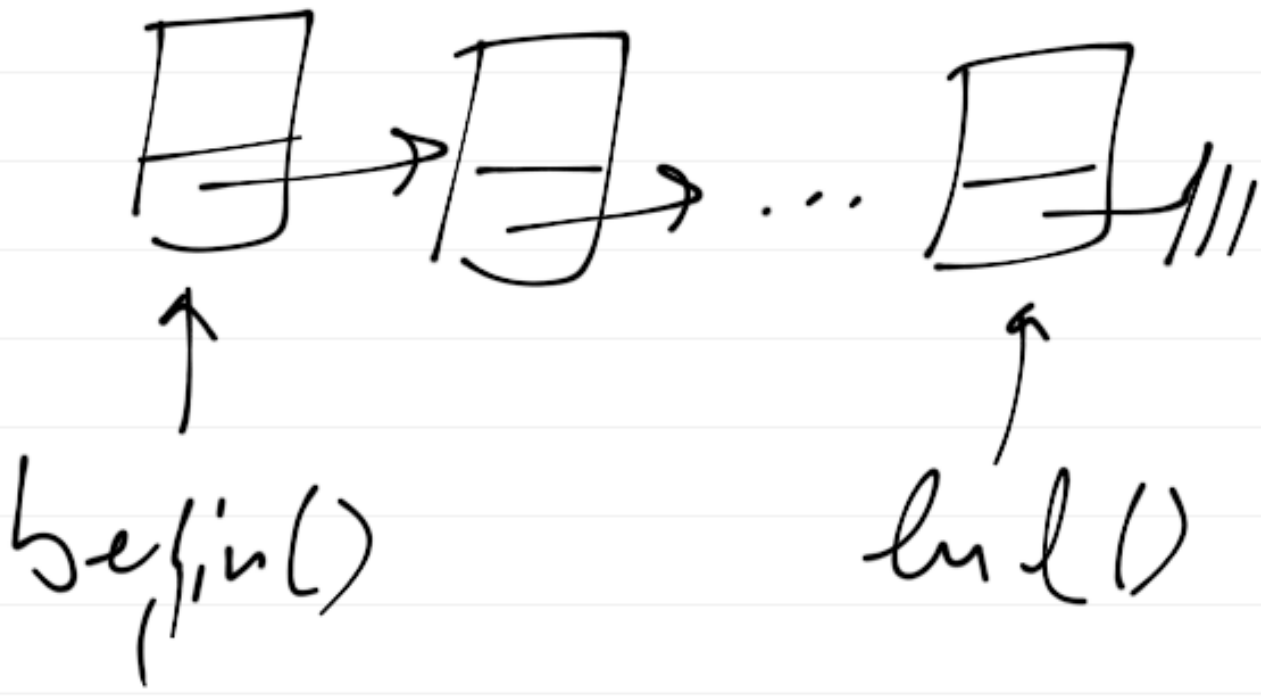
(#include <vector>)

```
for (int p=1; p<a.size(); p++){
    T tmp = a[p];
    for (int j = p; j>0 &&
            tmp < a[j-1];
            j--)
        a[j] = a[j-1];
    a[j] = tmp;
}
```

Vector member fth & operator[]

- What if operator[] isn't available or doesn't make sense (e.g. linked list)

- Use iterators (like pointers)



Insertion sort with iterators

```
template <typename iterator,
          typename T>
void insertion_sort(
    const iterator& begin,
    const iterator& end,
    const T& obj);
for(iterator p = begin+1;
    p != end; p++) iterator ops
  T tmp = *p;
```

```
for (iterator j = p;
     j != begin() &&
     tmp < *(j - 1);
     j++)
  *j = *(j - 1);
*j = tmp;
```

iterator operators

- insertion sort, quicksort, can degenerate to $O(n^2)$
- want $O(n \lg n)$ sort:

binary heap

(priority queue)

- binary tree, but implemented as array

- properties:

min heap:

smallest element always at root

→ you have to maintain this property when inserting & deleting

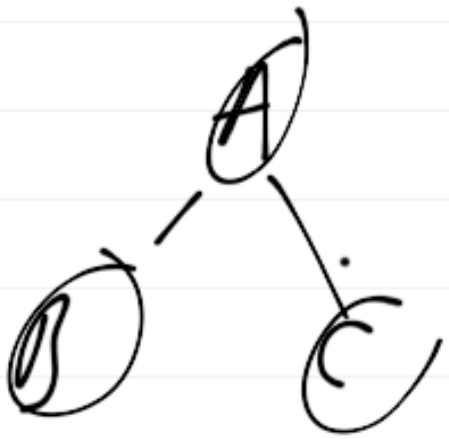
- height : h, $2^h \leq n \leq 2^{h+1} - 1$

(for n nodes, height of tree is $\lg n$)

(A) $n = 1$

$h = 0$, 2^h

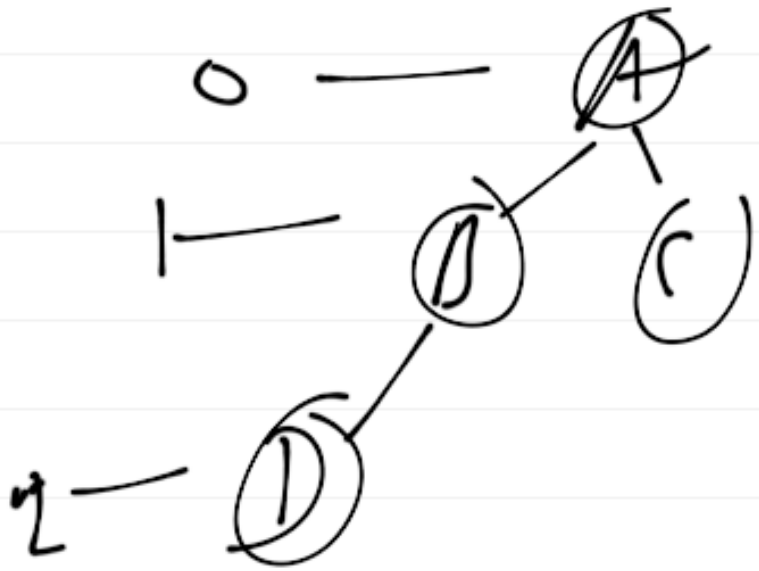
$2^0 = 1$ ✓



$n = 3$

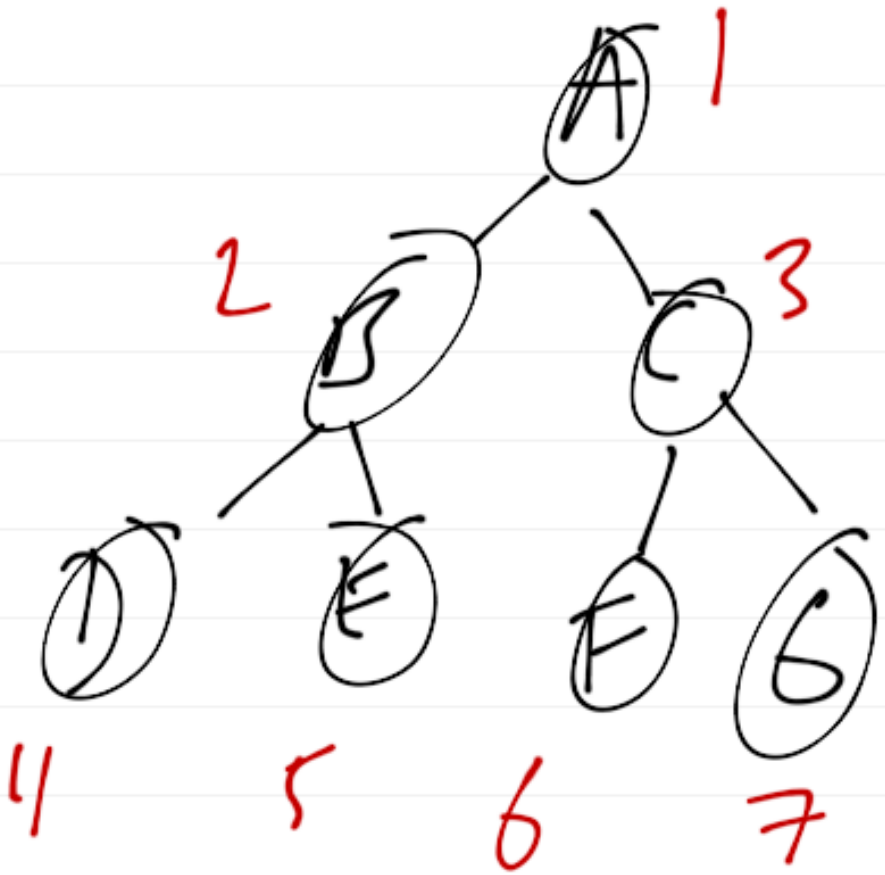
$h = 1$

$2^1 \leq n \leq 3^1$



$h = 2$

$n = 4$



for element at pos. i, parent @ $\lfloor i/2 \rfloor$

	A	B	C	D	E	F	G
0	1	2	3	4	5	6	7

Use 0th pos to store n

for each element at pos. i,

left child: $2i$ right child: $2i+1$

- heap order property:
  smallest el. at top (root)

→ findMin() ∈ O(1)
  (constant time op.)
  (STL top())
  return (arr[1])

- insert() &
  deleteMin() (pop)

insert(x) :

1. create 'hole' in the next available location

int hole = ++n;

↑ no. of elements stored in binheap

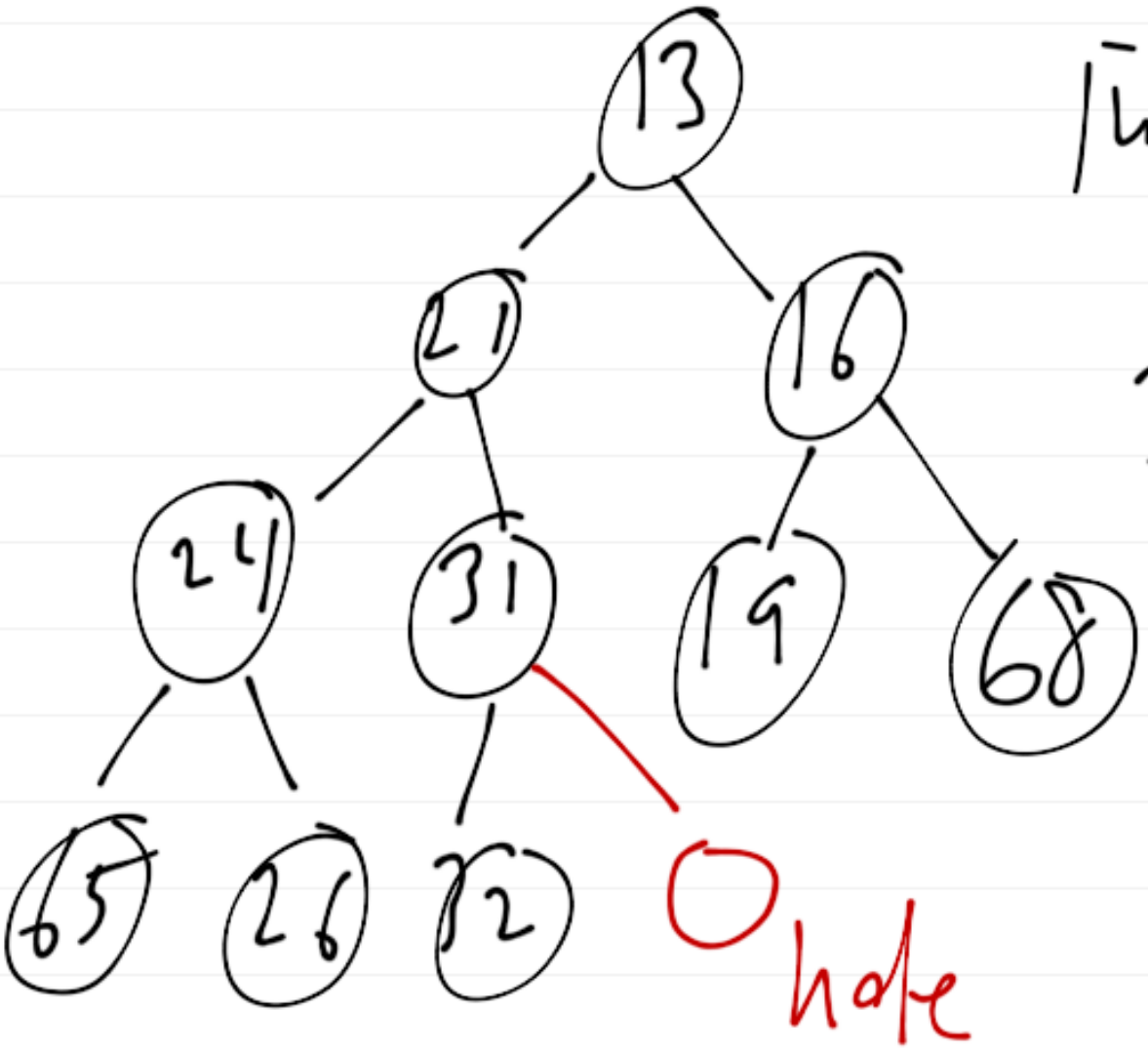
2. if x can be placed in hole w/out violating heap order, do so & end

else

percolate hole up the tree to hole's parent $\left(\left\lfloor \frac{hole}{2} \right\rfloor\right)$

arr[hole] = arr[hole/2];

Continue until x
can be inserted
(may be at root)



Insert 14

31 < 14?

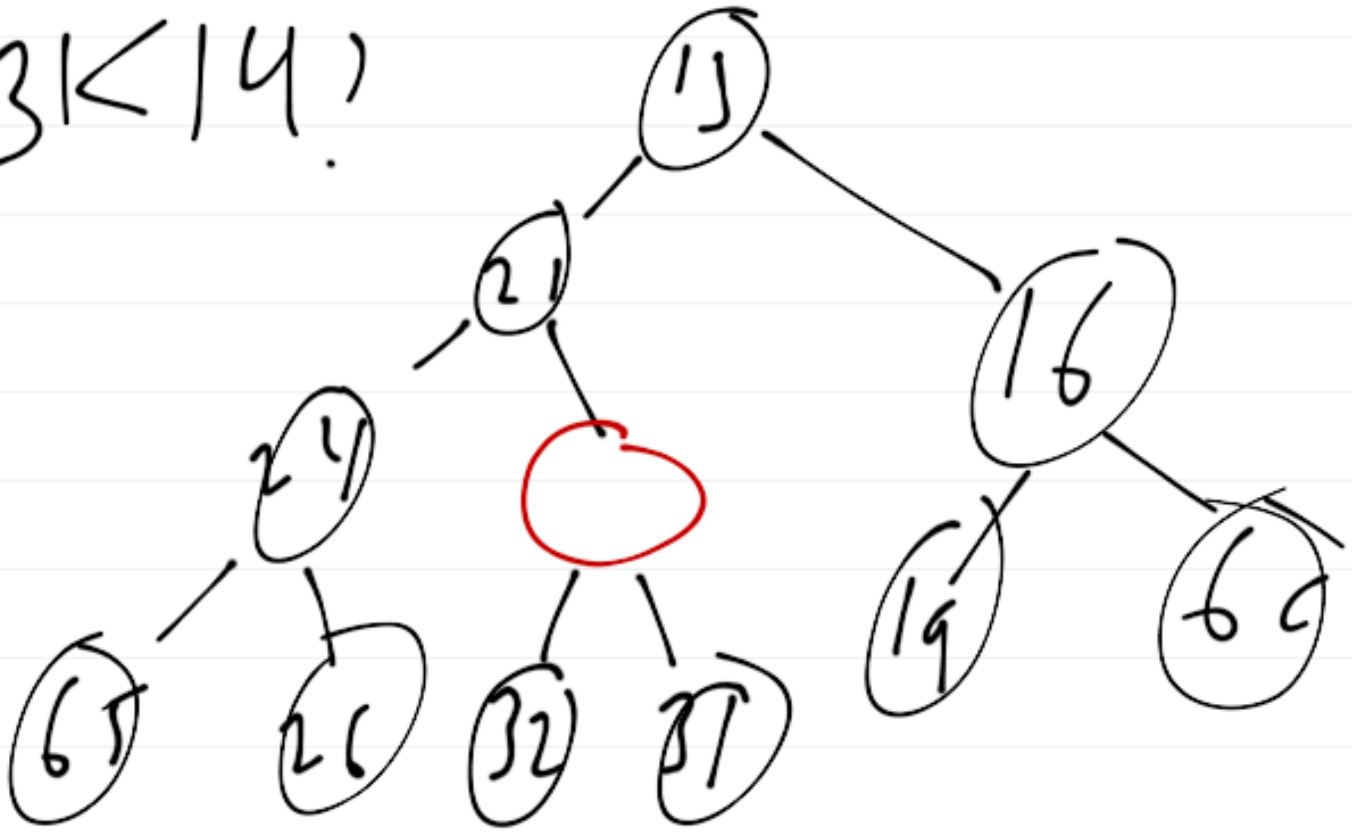
NO

→

tree always being filled in left-to-right

(complete binary tree)

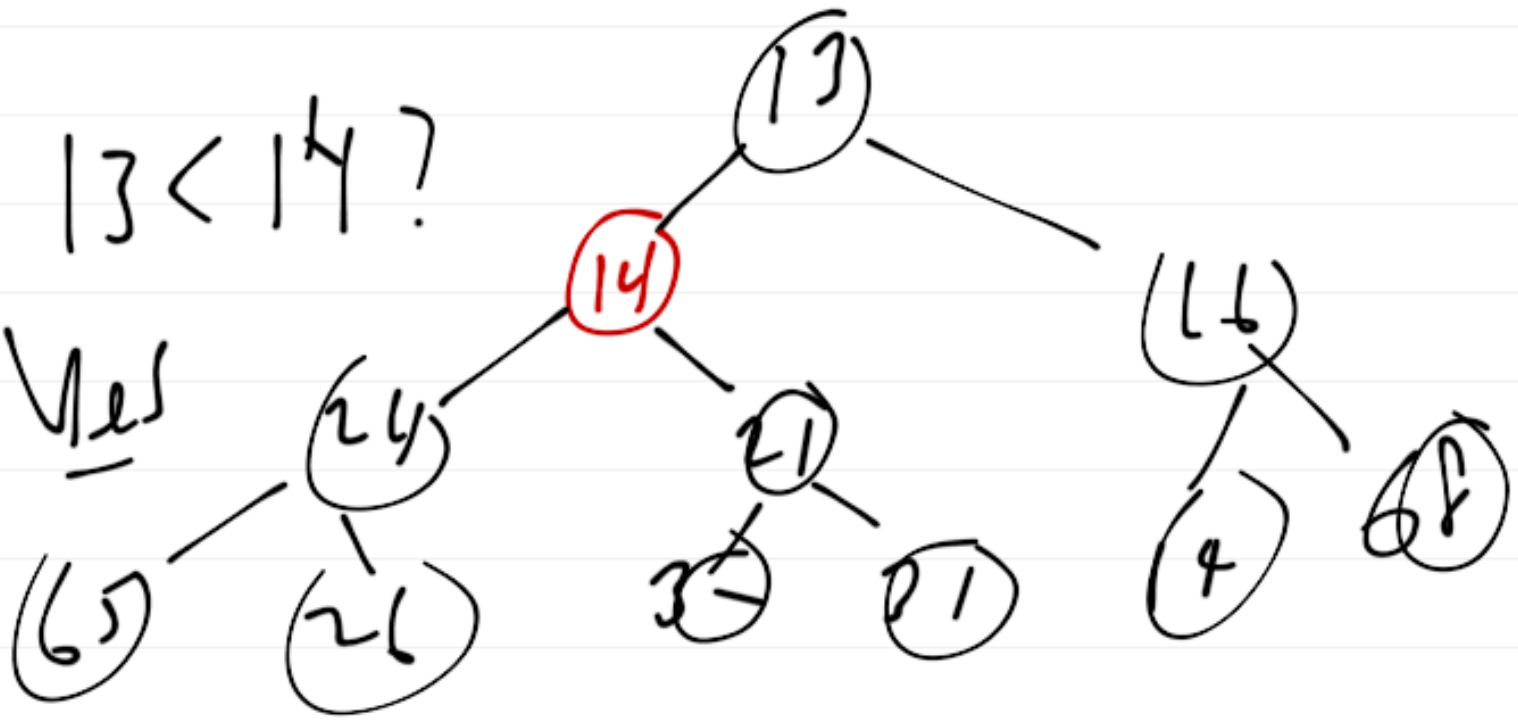
31<14?



21 < 14? no

13<14?

Yes



insert(const T& x)

{

if (n == arr.size() - 1)

capacity()?

arr.resize(size()*2)

=

See text

(doubling size of array

if need be)

```
// percolate up
int hole = ++size;

for ( ; hole > 1 &&
        x < arr[hole/2];
        hole /= 2 )
    arr[hole] = arr[hole/2];
arr[hole] = x;
} // See p. 219
```

- See text for deleteMin() & percolateDown

See text for incKey(P, Δ) & decKey(P, Δ) changing values of elements on heap (DOESN'T MAKE SENSE)

```
T deleteMin()
{
  if (isEmpty())
    return -1;
  T tmp = arr[1];
  arr[1] = arr[n--];
  percolateDown(1)
  return(tmp);
}
```

member function you write

decrement n ← move last element to top

percolateDown(int hole)

// see text:

watch out for testing

child = hole * 2 &

child + 1

forgetting to test right child is common mistake

see text for

incKey(p, Δ)

≜

decKey(p, Δ)

~~these really should not~~
be used:

STL priority_queue
does not provide them.