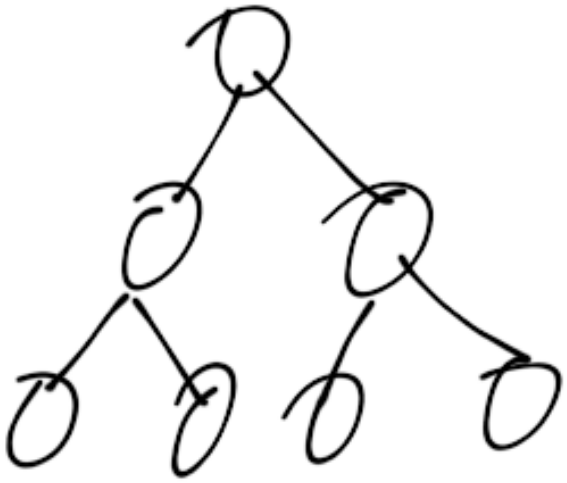
Theory

binary heap



Practice

Lab3 : quicksort

Lab4 : omp

Asgl : omp

- first tree data structure
- but still use array

$2i$, $2i+1$

insertion sort using STL

Standard Template Library

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

STL array

like your array_t

(#include <vector>)

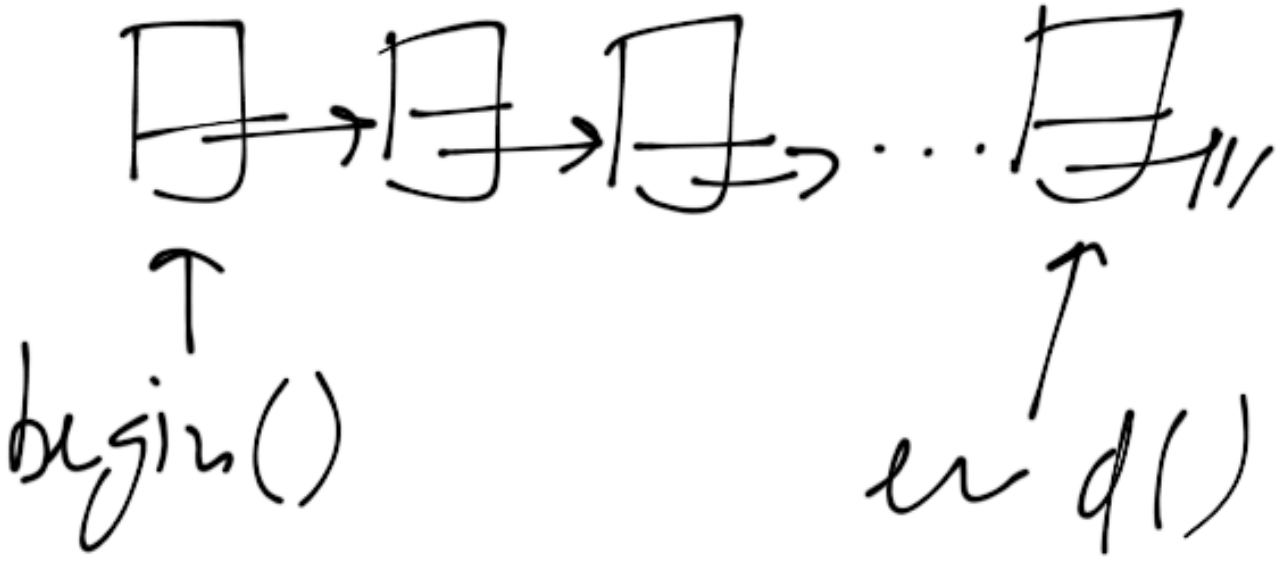
```
for(int p=1; p<a.size(); p++)
    T tmp = a[p];
```

vector member fn : operator[]

```
for(int j=p; j>0 &&
        tmp < a[j-1];
        j--)
    a[j] = a[j-1];
a[j]=tmp
```

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

- use iterators
(like pointers)



```
template <typename iterator,
          typename T>
void insertionSort(
    const iterator & begin,
    const iterator & end,
    const T & obj)

for(iterator p = begin+1;
    p != end; p++)
  T tmp = *p;
```

iterator ops.

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

iterator operators

- insertion sort, quicksort, etc. all 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:

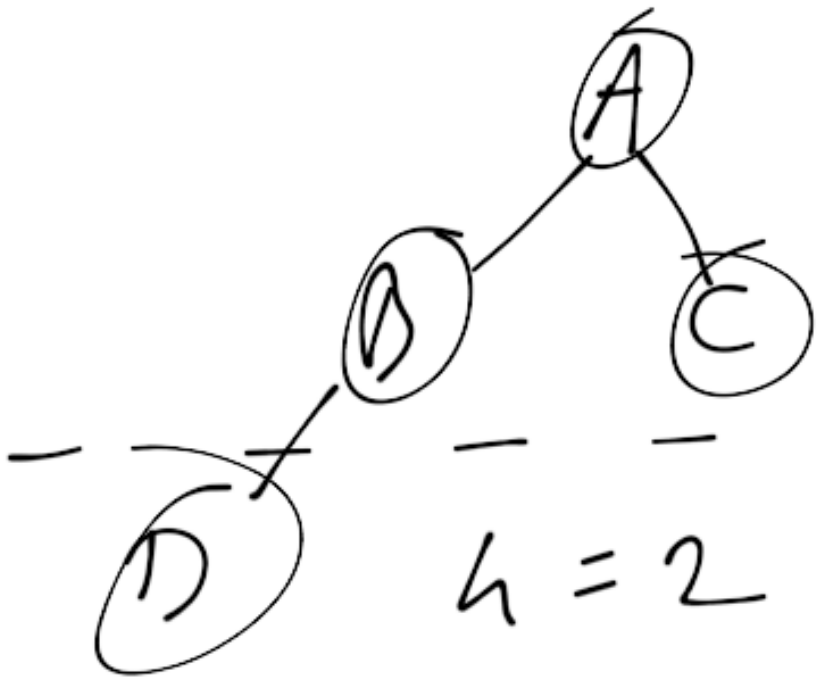
smallest element always at root

→ you have to maintain this property when inserting & deleting

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

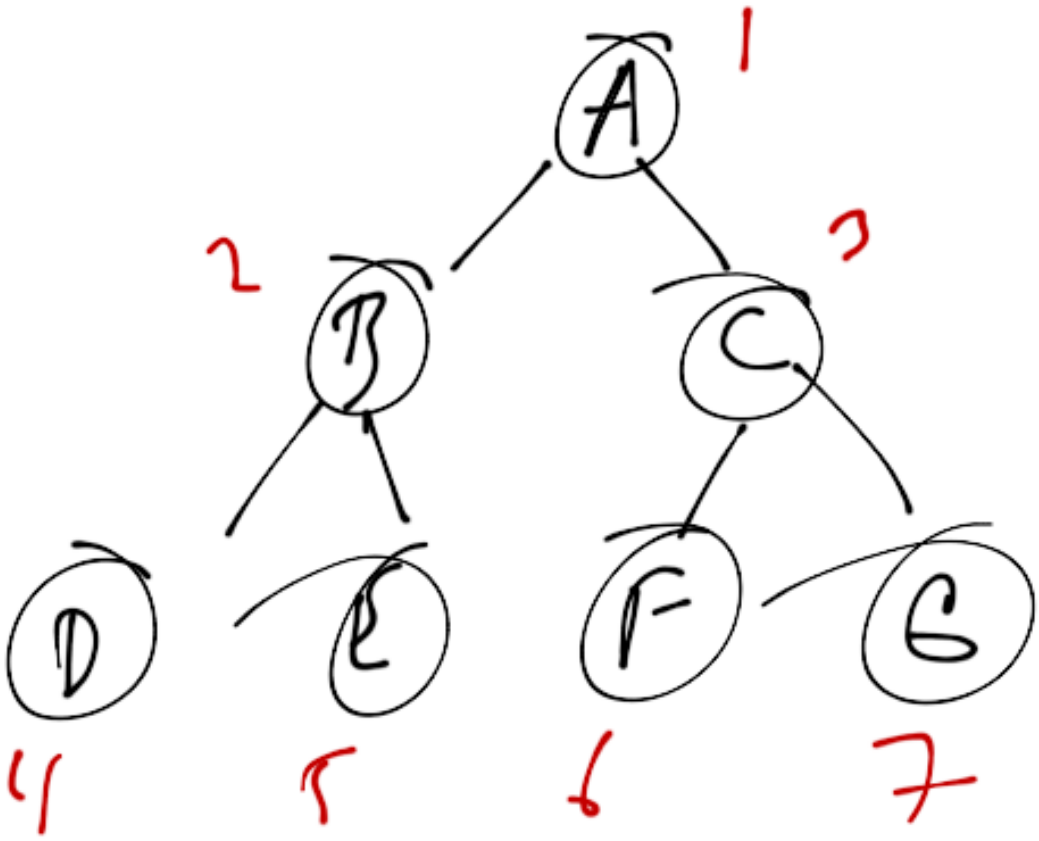
(for n nodes, $\lg n$ height)

$h = 0$, 2^0 node ✓



$h = 1$, $2^1 \leq n \leq 3$ ✓

$h = 2$



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

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

for any element at pos. i

left child @ $2i$

right " @ $2i+1$

- heap order property:

  smallest el. at top (root)

$\Rightarrow$ findMin() $\in O(1)$

((constant time op.)

(STL top())

return (arr[1])

- insert() & deleteMin()

  (pop())

insert (x) :

1. create 'hole' in the nearest available loc.

int hole = ++n;

↑

no. of elements stored in binheap data structure

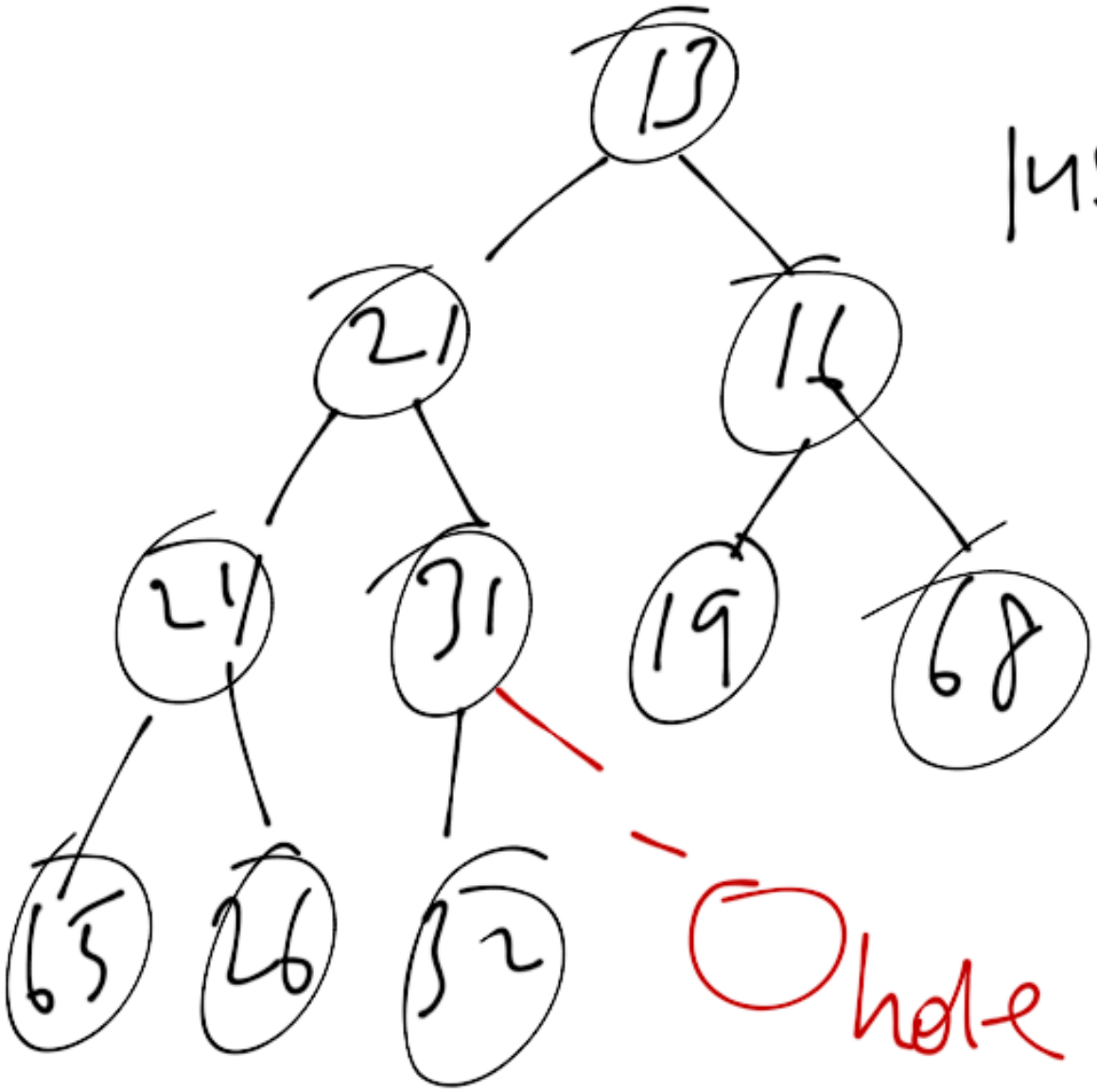
2. if x can be placed in hole w/out violating heap order, do so & end

else

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

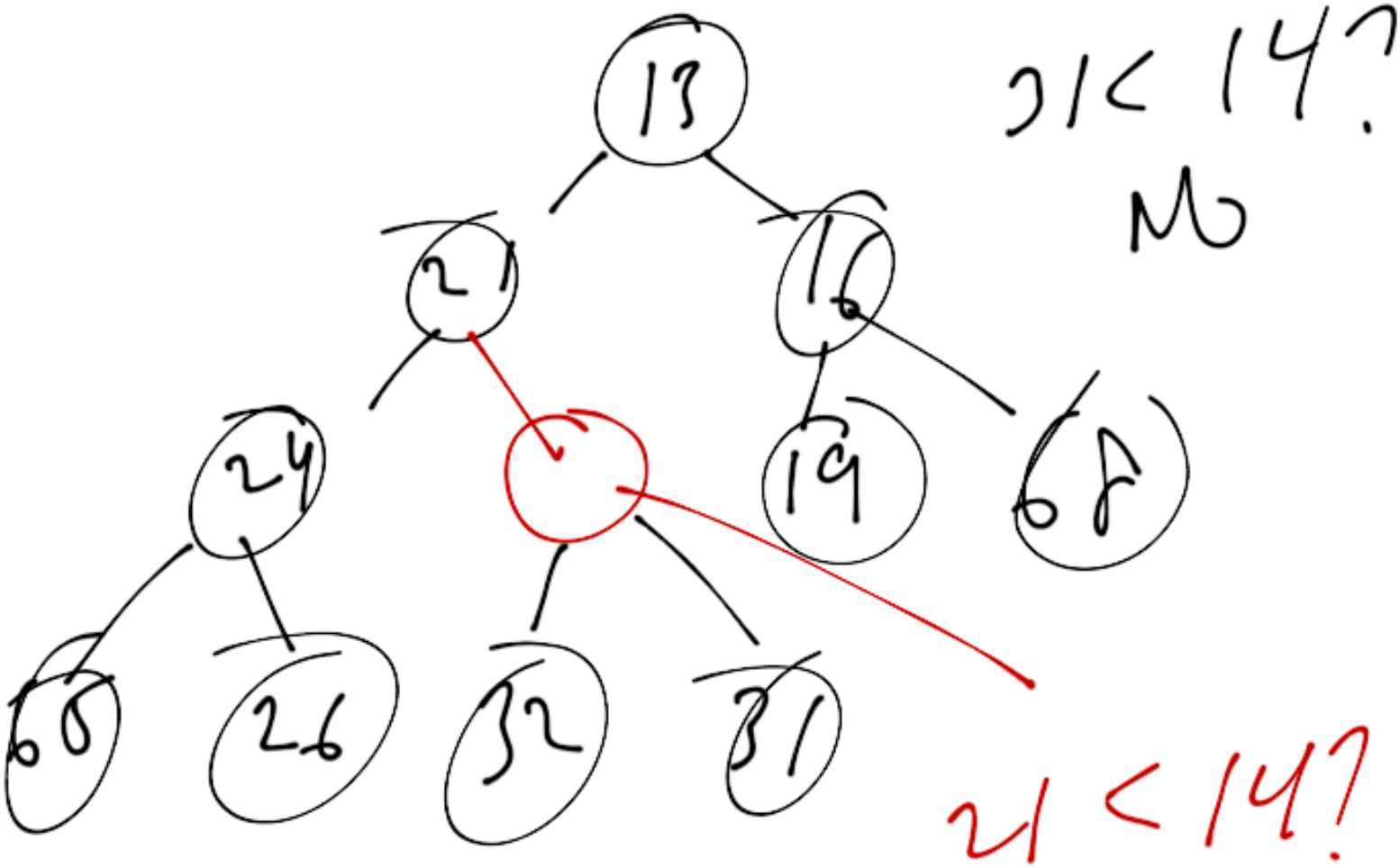
arr[hole] = arr[hole/2];

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



Insert 14

the always being filled in left-to-right (complete binary tree)



21 < 14?

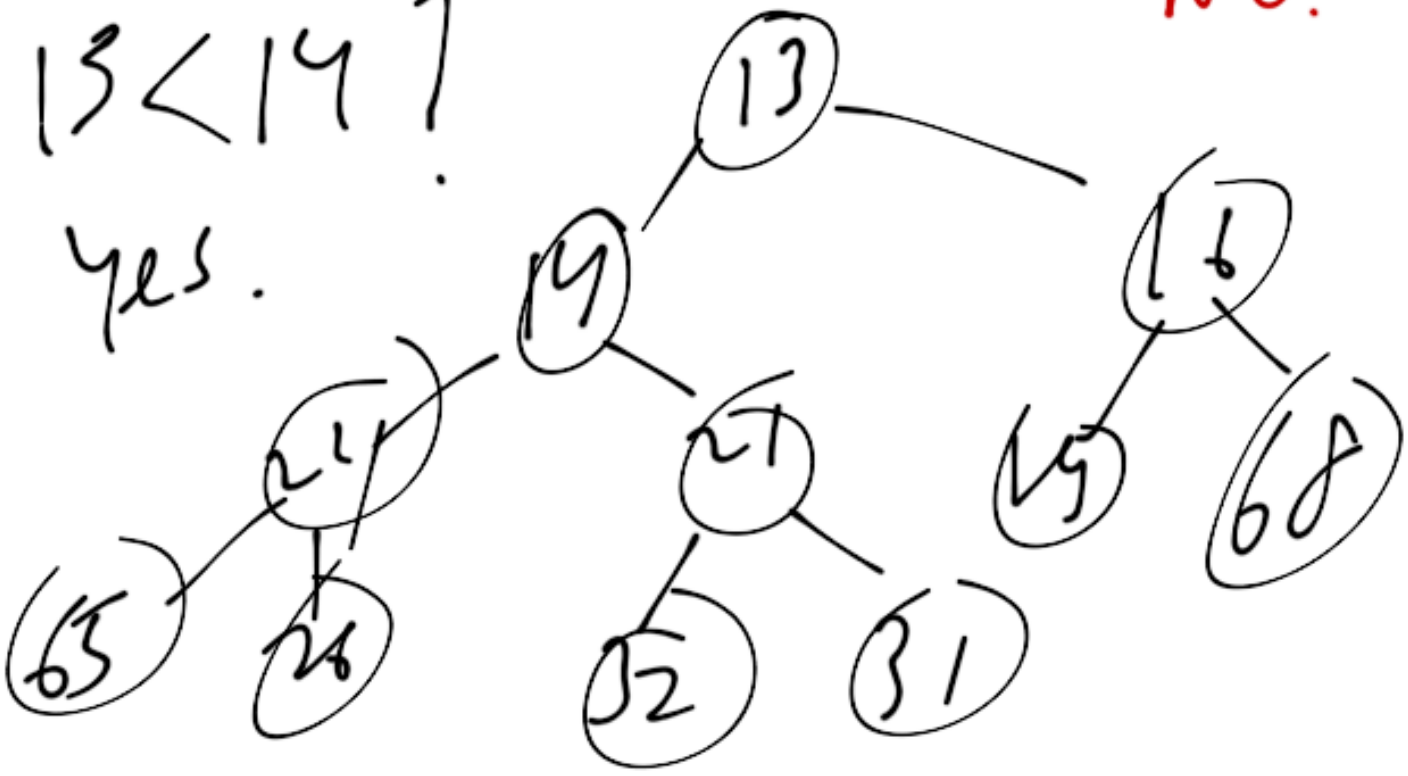
No

21 < 14?

No.

13 < 14?

Yes.



insert(const T& x)
{
  if (n == arr.size() - 1)
    arr.resize(size() * 2)
    capacity() ?
    //re. alloc space if
    not enough
    (for STV vector<T>)

reserve

```
// percolate up
int hole = ++n;
for ( ; hole > 1 &&
        x < arr[hole/2];
        hole /= 2)
    arr[hole] = arr[hole/2];
arr[hole] = x;
} // see p. 219
```

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

```
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

increase(p, Δ)

:

decrease(p, Δ)

→ then change values of elements on heap.

(Does it really make sense to do so!!)