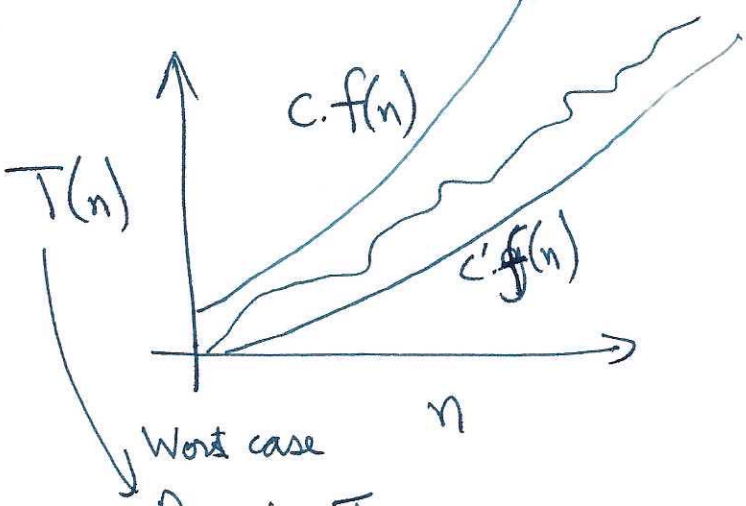



Worst case Running Time of Insertion Sort

To express $T(n)$ as an approximation of a "nice fn".

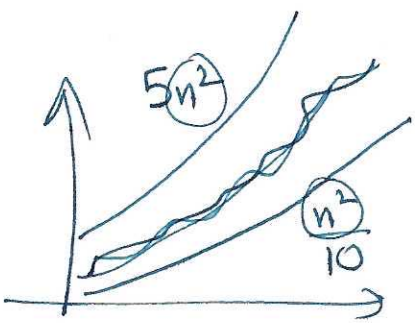
→ polynomial

$T(n)$ is approximately a "nice fn"

"$T(n) \leq f(n)$"

$T(n) = O(f(n))$

$T(n) = \Omega(g(n))$ → hide constant factors




$T(n) = \Theta(n^2)$

$T(n) = \frac{n}{100}$

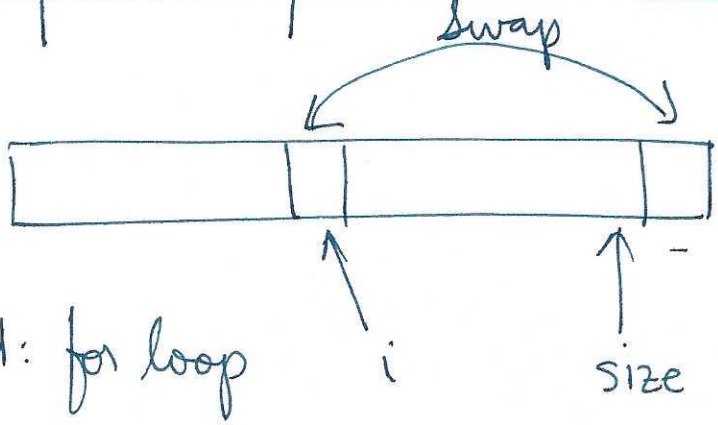
$T(n) = \Omega(n)$

$T(n) \geq c' n$ → for some constant

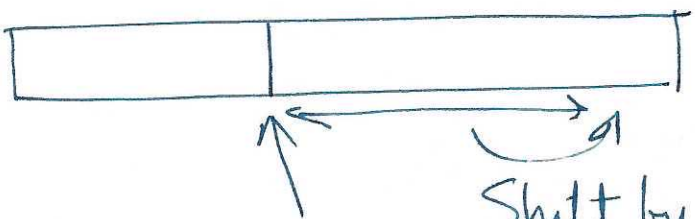
$T(n) = \Theta(n)$

$T(n) = \Theta(1)$

| Data Structure | Find | Insert | Delete (with appropriate ptr) | |
|---|---|---|---|---|
| Linked List | $\Theta(n)$ | $\Theta(1)$ | $\Theta(1)$ | |
| Unsorted Array | $\Theta(n)$ | $\Theta(1)$ | $\Theta(1)$ | (Swap with end) |
| Sorted Array | $\Theta(\lg n)$ | $\Theta(n)$ | $\Theta(n)$ | |




Find: for loop

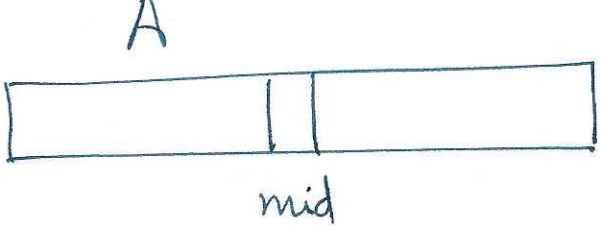



In worst case, insert the smallest element, WHOLE array shifts

## Binary Search

Faster searching in sorted arrays

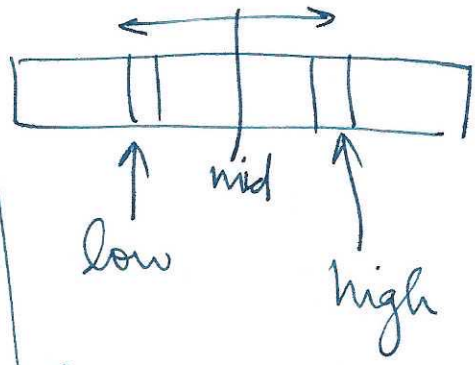
Search for val




Compare val with A[mid]

```
boolean BinSearch(A, low, high, val)
{
    if (low == high)
    {
        if (A[low] == val) return True;
        else return False;
    }
    if (low > high) return False;
```

O(1)




Base Case

O(1) mid = $\left\lfloor \frac{low + high}{2} \right\rfloor$

O(1) if (A[mid] == val) return True

if (A[mid] < val)

return BinSearch(A, mid+1, high, val)

if (A[mid] > val)

return BinSearch(A, low, mid-1, val)

---

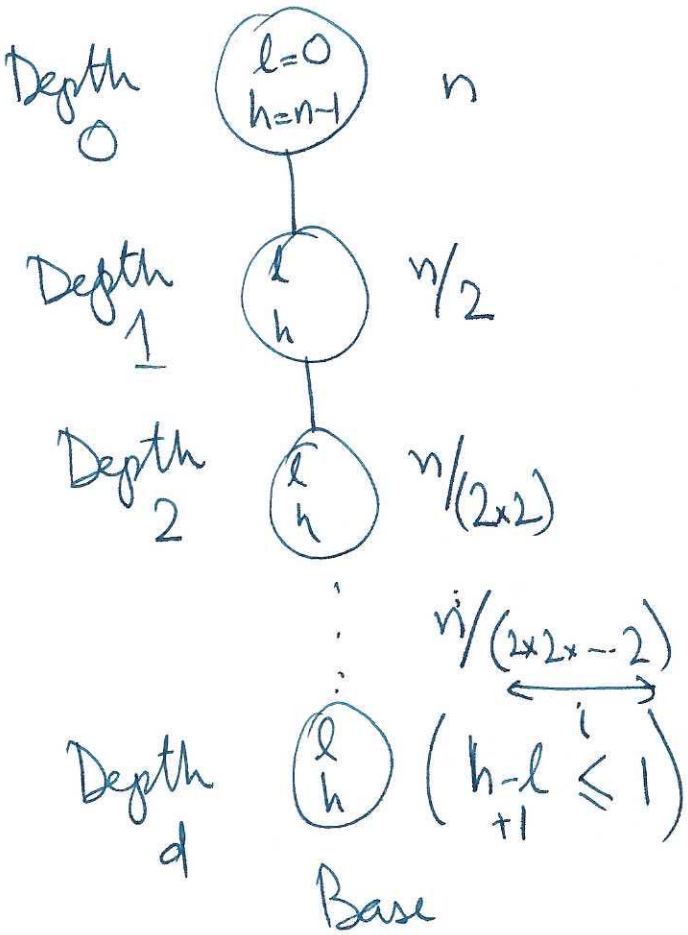
What is the running time of binary search? → T(n)

$T(n) = O($ worst/largest #recursive calls made on an array of size $n)$

At the recursion progresses, keep track of the size of the array (high - low + 1) in which search is done.

BinSearch( A, 0, n-1, val)

l h → length of array




Q. At depth i, what is (h-l+1)? Size of the array being considered?

(R) $n/(i+1)$

~~(G)~~

$\boxed{n/2^i}$

(B) Something else

At what depth is the base case?

At what value of i is $(h-l+1) <= 1$?

$$\frac{n}{2^i} = 1 \qquad 2^i = n$$

$$\boxed{i = \log_2 n}$$

There are at most $\lceil \log_2 n \rceil$ recursive calls.

So each line of code runs at most $\lceil \log_2 n \rceil$ times.

The worst-case run time is $O(\log_2 n)$

$O(\lg n)$.

(By careful analysis, one can argue that if val is NOT in $A_{mn}$, then at least $\log_2 n$ recursive calls are made.) $\rightarrow T(n) = \Omega(\lg n)$

So, $\boxed{T(n) = \Theta(\lg n)}$ Logarithmic Time

---

## Binary Heap

Priority Queue : Objects with a "priority" key

Operations (1) Insert

(2) Find Max (find highest priority object)

(3) Extract Max (delete max)

(4) Increase key (with ptr to object)

| | Insert | FindMax | Extract Max with ptr | Increase key with ptr |
|---|---|---|---|---|
| Linked List | $\Theta(1)$ | $\Theta(n)$ | $\Theta(1)$ | $\Theta(1)$ |
| Sorted Array | $\Theta(n)$ | $\Theta(1)$ | $\Theta(1)$ | $\Theta(n)$ |
| Binary Heap | $\Theta(\lg n)$ | $\Theta(1)$ | $\Theta(\lg n)$ | $\Theta(\lg n)$ |

Why is logarithmic time great?

$n = 10^6$ = million

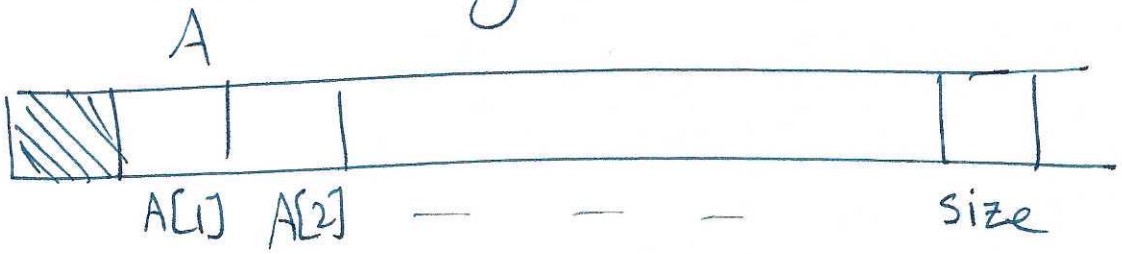
$\log_2 10^6 = 6 \log_2 10$

$10^3 \approx 2^{10}$

$\leq \log_2 (2^{10})^2 = \log_2 2^{20} = 20$

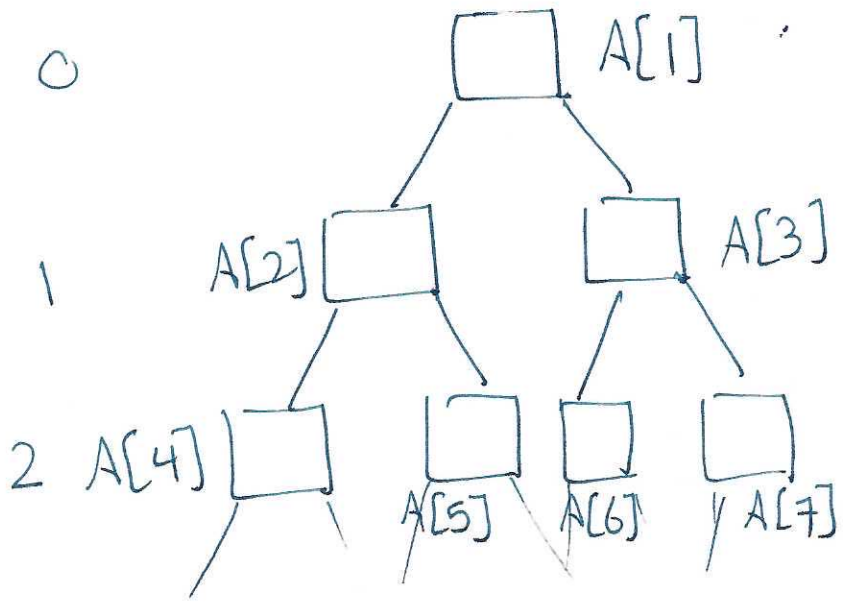
---

Binary Heap is just an array of object, but we INTERPRET array as a tree.




Conventionally A[0] is empty.

Size is the largest index storing something in Bin Heap.




Q. What is the parent of $A[i]$? size

~~(R)~~ $A[\lfloor i/2 \rfloor]$ (G) $A[(i-1)/2]$ (B) $A[i - n/2]$

Children of $A[i]$ are $A[2i]$ and $A[2i+1]$

Q. Let root be at level 0. How many nodes at level $l$? (assuming filled)

(R) $2l$ (G) $2+l$ ~~(B)~~ $2^l$

Q. If there are $n$ nodes/objects in heap, what is the maximum level/depth?

(R) $\Theta(n)$ ~~(G)~~ $\Theta(\lg n)$ (B) $\Theta(\sqrt{n})$

Technically, ~~if the heap is "filled out"~~

the max. level is between $\lg n$ and $\lg n + 1$

$\lg n$ might not be integer

$-1$

$\Theta(\lg n)$ $\lg n + \Theta(1)$