Some Algorithms, running times & "data structures" (ways to store & access data).

Arrays

- sequence of locations, one value stored at each location.
- $a[1], a[2], \ldots, a[n]$.
- For any i, $1 \leq i \leq n$, $a[i]$ can be accessed in one step.
- Read, write.

Problem: Given n numbers stored in an array, find their maximum.

Algo:

```
Let $M = a[1]$.                          "pseudo-code"
for $i = 2, 3, \ldots, n$ {
    if $a[i] > M$ then $M = a[i]$.
}
Output $M$.                       I prefer text descriptior
```

Running time Clearly $O(n)$.

English/text description

We'll scan the array $a[1], .., a[n]$, one by one and keep updating the "current maximum" M. Initialize $M = a[1]$, and scan $a[2], .., a[i], .., a[n]$. At i^{th} step, if the number currently scanned, $a[i]$ is larger than M, then update $M \leftarrow a[i]$. After all array entries are scanned, clearly M equals the maximum over all array entries.

Array Search

Problem Given array of size n and number b, does b appear in the array?

Algo. Scan array left to right, one entry at a time, and compare each entry to b.

Running time $O(n)$.

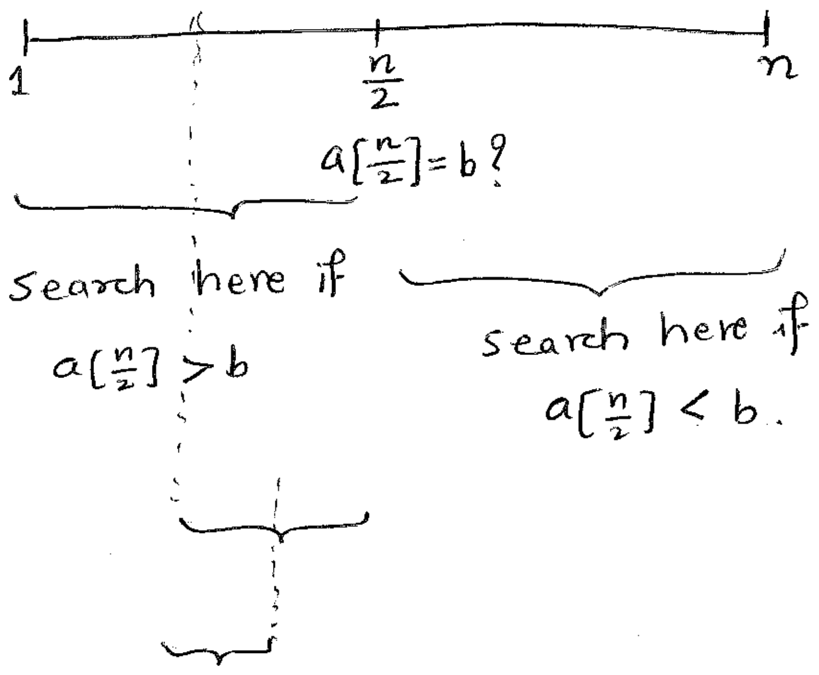
Binary Search

Problem Given a sorted array i.e.

$a[1] < a[2] < \cdots < a[n]$,

and number b, does b appear in array?

Algo.



1
$\frac{n}{2}$
n
$a[\frac{n}{2}] = b$?
search here if $a[\frac{n}{2}] > b$
search here if $a[\frac{n}{2}] < b$.

Algo + Main Idea We compare b with $a[\frac{n}{2}]$. (assume n even).

- if $b = a[\frac{n}{2}]$, we found b in the array. done.
- if $b < a[\frac{n}{2}]$, then if b appears in the array at all, it must be among

  $a[1], \ldots, a[\frac{n}{2}]$ (remaing "right half" entries are larger than b)

- if $b > a[\frac{n}{2}]$, then b can only be among

  $a[\frac{n}{2}], \ldots, a[n]$ similarly,

In either case, the "length of interval or search space" is reduced by half. We search now in this reduced "search interval."

The process can be continued, reducing the "search interval" by half at each step.

In $O(\log n)$ steps, the process terminates.

$\therefore$ Running time $= O(\log n)$.

Recurrence relation

$T(n)$ = Number of searches (running time) by the algo.

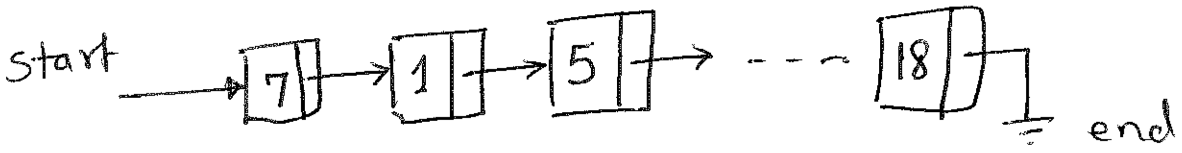
$$T(n) \leq T\left(\frac{n}{2}\right) + C$$

solution

$T(n) \leq c \cdot \log n$.

proof
- unrolling recursion
- guess & verify,

Linked List (Array: inserting item in the middle takes $O(n)$ time)



- sequence of nodes
- given: pointer to first node

- i^{th} node contains pointer to $(i+1)^{th}$ node
- each node contains value / data-item.

<u>Note</u> - Accessing i^{th} node takes i steps.

- Can create, add, delete nodes

<u>Problem</u> Given two linked lists L_1, L_2,

- each of size n,
- each a sorted list of integers

merge then into a single linked list of size $2n$ that is sorted.

L_1 : $\boxed{2} \to \boxed{3} \to \boxed{10} \to \boxed{22} \to \boxed{37}$

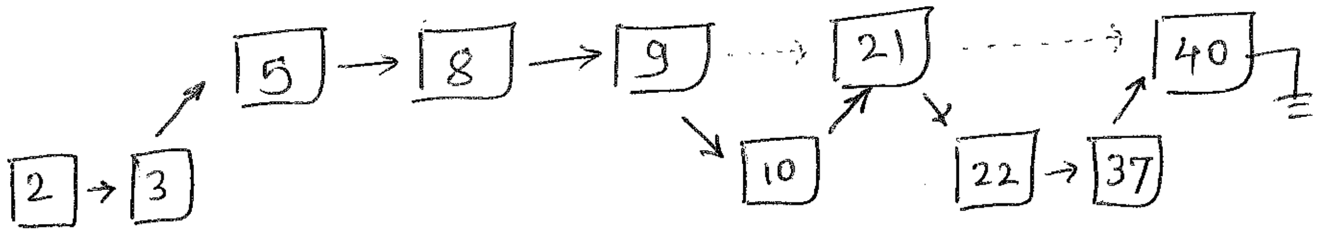
L_2 : $\boxed{5} \to \boxed{8} \to \boxed{9} \to \boxed{21} \to \boxed{40}$

<u>output</u>

$\boxed{2} \to \boxed{3} \to \boxed{5} \to \boxed{8} \to \boxed{9} \to \boxed{10} \to \boxed{21} \to \boxed{22} \to \boxed{37} \to \boxed{40}$

<u>Obvious algorithm</u>

- Remove one node from the front of L_2 in each step and insert into L_1 at the "correct place".



This could take

$$n + (n+1) + \cdots + 2n \approx \frac{3}{2} n^2 \text{ time.}$$

Clever Algorithm

- Let L be a new list, initially empty.
- maintain a pointer to the head of list L_1 as well as L_2.

  Call the values at the heads v_1 and v_2.
- compare v_1, v_2.
- If $v_1 < v_2$ then remove v_1 from L_1 and append it to the end of L.

  If $v_1 > v_2$ then remove v_2 from L_2 and append it to the end of L.

- Takes $O(n)$ time because each step takes $O(1)$ time and size $|L_1| + |L_2|$ is reduced by 1.
- Space can be reused if one wants.

Merge Sort $O(n \log n)$ time (later)

Problem Sort given list of n integers.

Algo
- Let L be given list.
- Divide L into two lists L_1 and L_2. $\}$ $O(n)$
- Sort (recursively) L_1. $\}$ $T(\frac{n}{2})$
- " L_2 $\}$ $T(\frac{n}{2})$
- Merge L_1, L_2 into single sorted list L. $\}$ $O(n)$

Recurrence relation $T(n) \leq 2T(\frac{n}{2}) + O(n)$.

Solution to RR $T(n) = O(n \log n)$.

- Does binary search on array be used to design $O(n \log n)$ - sorting algorithm?

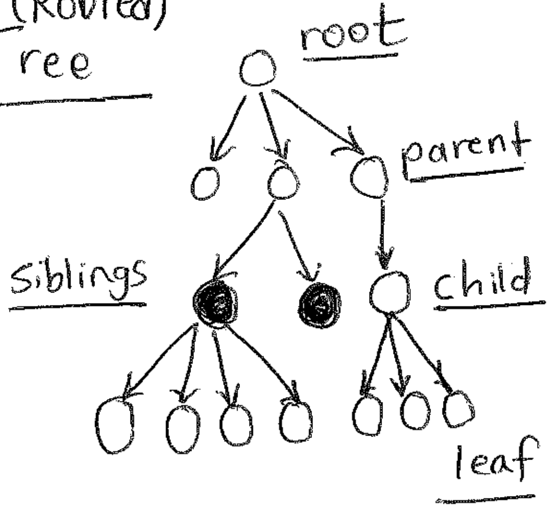
  NO: Cannot insert elements in the middle of an array.

- Linked list: can insert but no binary search.

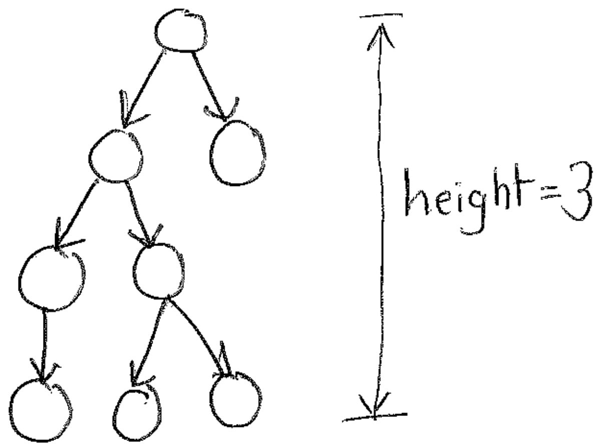
$O(n \log n)$ - Sorting using Heaps (also called Priority Queues)

- Data structure that stores n values.
- These operations can be done in $O(\log n)$ time.
  - Insert a value
  - Extract / delete the minimum value.
- This implies $O(n \log n)$ sorting algo. by n insertions and then n delete-min operations.

(Rooted)
Tree

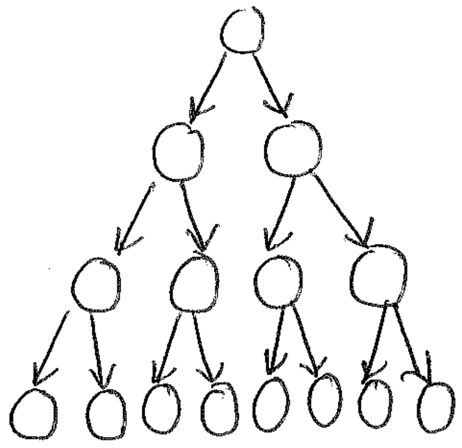


Binary Tree



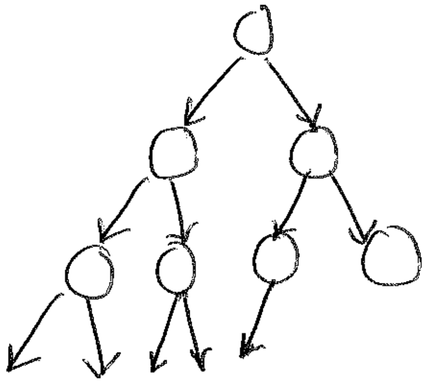
Every node has ≤ 2 children.

Complete binary tree



- height $= h$
- #leafs $= 2^h$
- #nodes $= 1 + 2 + 2^2 + \cdots + 2^h$
  $= 2^{h+1} - 1$.

Almost complete Binary Tree



- same as complete binary tree except at last level, some ℓ leftmost nodes are present.

Deaf A **Heap** is :

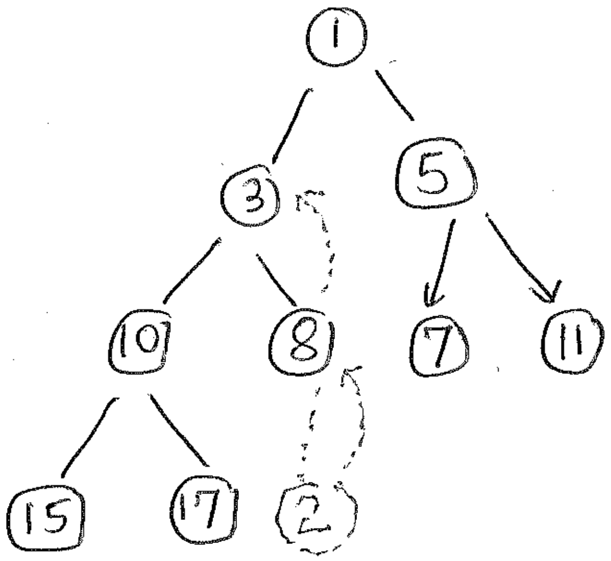
- Almost complete binary tree.
- A value at each node.
- (Heap order/property) :

  If u has a child v then

  $\text{value}(u) < \text{value}(v)$.

Fact Root has the minimum value among all nodes.

Example.



Extract (find) Min Return value at root.

Add a value

- Add it at next available node/location in A.C.B.T.
- Push the value up to preserve the heap order.

Delete-Min

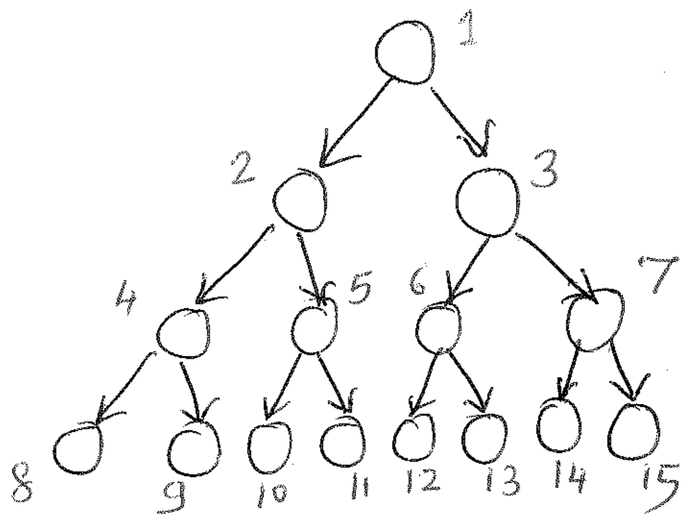
- Erase the value at the root.
- Delete the last node in A.C.B.T. and move its value to the root.
- Push down the $\overset{\text{(new)}}{\wedge}$ value at the root to preserve heap order.

Note: Any value/node in the heap can be deleted similarly, not just the root.

Running time Add, delete-min both take $O(h)$ time, h = height of ACBT.

$$n = \#\text{nodes} \geq (1+2+\cdots+2^{h-1})+1$$
$$= 2^h \quad \therefore \; h \leq \log n.$$

Array implementation of heap



- Number nodes left to right, top to bottom.
- For node i, its children are precisely $2i$ and $2i+1$. (Exercise: Prove!)

Thus all nodes can be maintained in an array and all operations can be carried out on the array! Very fast!