Runway reservation system
- Definition
- How to solve with lists

Binary Search Trees
- Operations

---

Runway Reservation System:

- Airport with single (very busy) runway (Boston 6→1)
- "Reservations" for future landings
- When plane lands, it is removed from set of pending events

Reserv req. specify "requested landing time" $t$
Add $t$ to the set if no other landings are scheduled within $< 3$ minutes either way.
- else error, don't schedule
Example

\[ \begin{array}{cccccc}
37 & 41 & 46 & 49 & 56 & \text{time (mins)} \\
\downarrow & & \downarrow & \downarrow & \downarrow & \text{now} \\
\times & & \times & \times & \times & \text{reserved landing times} \\
\end{array} \]

\[ R = \{ 41, 46, 49, 56 \} \]

Request for time: \( 44 \) not allowed \((46 \in R)\)

\( 53 \) OK

\( 20 \) not allowed \((\text{already past})\)

\(|R| = n \) goal: run this system efficiently \(O(\log n)\) time

---

Keep \( R \) as sorted list.

Init: \( R = \emptyset \)

req(t): if \( t < \text{now} \): return "error"

\[ \Theta(n) \]

for \( i \) in range (\( \text{len}(R) \)):

\[ \text{if abs}(t-R[i]) < 3: \text{return error} \]

\( R \). append(t)

\( R = \text{sorted}(R) \)

land: \( t = R[0] \)

if \( t \neq \text{now} \): return error

\( R = R[1:] \) \((\text{drop } R[0] \text{ from } R)\)
Can we do better?

Sorted list: Can insert new time/plan pair in $O(n)$ time, but insertion takes $O(n)$ time.

Sorted array? Can do binary search to find place to insert in $O(\log n)$ time.

Unsorted list/array: Search takes $O(n)$ time.

Dictionary or Python Set: Insertion is $O(1)$ time.

3-min check takes $\Omega(n)$ time.

What if times are in whole minutes?

Large array indexed by time: does the trick.

Doesn't work for arbitrary precision time or

Need fast insertion into Sorted list!

New requirement: rank($t$): how many

planes scheduled to land at times $\leq t$?

design amendment?
BST = nil

| Insert (49) | BST → 49 |
| "79" | 49 → 79 |
| 41, 64, |

Find the min element in a BST.

Delete - min:
find min & eliminate

Just go left!
(till you can't anymore)

All are O(h) where h is height of BST

next-larger(x) if right child ≠ NIL, return minimum(right)
else y = parent(x)
while y ≠ NIL and x = right(y)
x = y
y = parent(y)
return y
What about rank(k)?
Can't solve it efficiently with what we have, but can augment the BST structure.

What happens before 79?
Keep track of size of subtrees, during insert & delete.

Walk down tree to find desired time
Add in nodes that are smaller, and add in subtree sizes to the left
O(h) time

(Have we accomplished anything?)
Height h of the tree should be O(\log n).

Insert into BST in sorted order.

Looks like a linked list.
O(n) not O(\log n)

Balanced BSTs to the rescue!