# **Set ADT**

**Set**: A collection of unique values (no duplicates allowed) that can perform the following operations efficiently

1. add
2. remove
3. contains

The client doesn't think of a set as having indexes like a list.

We just add things to the set in general and don't worry about order.

A picture containing text, device

Description automatically generated

**Implementation Strategies**

|  |  |  |  |
| --- | --- | --- | --- |
| Data structure | add | remove | contains |
| Unsorted collection | O(1) | O(n) | O(n) |
| Sorted Collection | O(n) | O(n) | O(logn) |
| Balanced BST | O(logn) | O(logn) | O(logn) |

**Unsorted Collection Set**

* It doesn't really matter what order the elements appear in a set, so long as they can be added, removed, and searched quickly.
* If we add new elements to the end of the collection (either add a node or add to the next available index) like we do in a list, then
  + **add** takes O(1) because we simply append to the end of the list
  + **remove** takes O(N) because we must shift elements down
  + **contains** takes O(N) because we must search the whole array

Table

Description automatically generated

**Sorted Collection Set**

* Suppose we ordered the elements in the array in sorted order instead of insertion order.
* If we add new elements to the end of the list (or the next available index), as we do in a list, then
  + **add** takes O(N) because we must shift elements up to make room
  + **remove** takes O(N) because we must shift elements down to make room
  + **contains** takes O(logN) because we can perform binary search on a sorted array to find a value

Table

Description automatically generated

Is there a way we can efficiently get these operations down to O(1) time?

**A New Idea**

Now suppose when client adds value i, store it at index i in the array.

Text

Description automatically generated with medium confidence

Table

Description automatically generated with medium confidence

**Hashing**

* **Hash**: To map a large domain of values to a smaller fixed domain.
  + Typically, mapping a set of elements to integer indexes in an array.
* **Idea**: Store any given element value in a particular predictable index.
  + That way, adding / removing / looking for it are constant-time (O(1)).
* **Hash Table**: An array that stores elements via hashing.
* **Hash Function**: An algorithm that maps values to indexes.
  + **Hash Code**: The output of a hash function for a given value.
* In our previous ‘A New Idea’ example, our "hash function" was: **hash(i) → i**
  + Potentially requires a large array (a.length > i).
  + Doesn't work for negative numbers.
  + Array could be very sparse, mostly empty (memory waste).