Data Structures

Week 6 – The set data structure: representations and operations

Dr. Ferenc Koczka · Eszterházy Károly University · 2026

Review

- **Array:** direct access O(1), insertion/deletion O(n)
- **Binary search** in a sorted array: O(log n)
- **Sorting:** bubble, selection, insertion – $O(n^2)$
- **Merging** (week 3, exercise 5): two sorted arrays into one sorted array, O(n + m)

Questions for today

- How can a mathematical set be represented in a computer?
- How fast are the set operations in each representation?

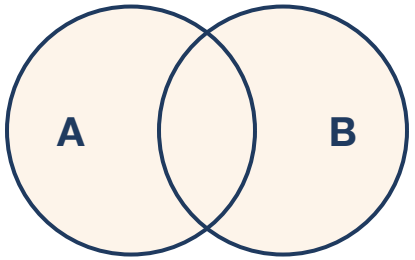
Mathematical background

Set: an **unordered** collection of distinct elements.

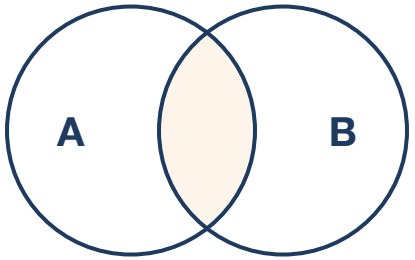
- Every element occurs **at most once**: {1, 2, 2} = {1, 2}
- The order **does not matter**: {1, 2} = {2, 1}
- Notation: $x \in A$ (element of) · $|A|$ (number of elements) · $\varnothing$ (empty set) · **U** (universe)

Operation	Notation	Meaning
Membership	$x \in A$	is x in A
Subset	$A \subseteq B$	every element of A is also an element of B
Union	$A \cup B$	elements of A **or** B
Intersection	$A \cap B$	common elements of A **and** B
Difference	$A \setminus B$	elements of A that are **not** in B

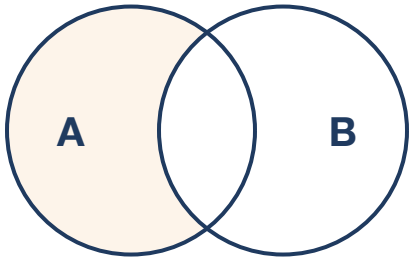
Set operations illustrated



A ∪ B (union)



A ∩ B (intersection)



A \ B (difference)

Example – U = the days of the week:

A = {Mon, Wed, Thu, Sat} (training days) · B = {Tue, Wed, Thu} (class days)

A ∪ B = {Mon, Tue, Wed, Thu, Sat} · A ∩ B = {Wed, Thu} · A \ B = {Mon, Sat}

The set as a data structure

Property	Set
Elements	homogeneous
Relationship between elements	**none** (no order)
Size	dynamic (can grow and shrink)
Duplicates	none

Possible representations (all based on arrays):

1. **Boolean vector** (characteristic vector, bit vector)
2. **Unsorted** array (sequence)
3. **Sorted** array (sequence)
4. Hash table (week 11), search tree (week 15)

Representation by a boolean vector

Boolean vector

If the **universe is small and known** ($U = \{u_1, \ldots, u_n\}$), every set is a boolean vector of length **n**: **A[i] = true ⇔ $u_i \in A$**

	Mon	Tue	Wed	Thu	Fri	Sat	Sun	
U	1	1	1	1	1	1	1	
A	1	0	1	1	0	1	0	A = {Mon, Wed, Thu, Sat}
B	0	1	1	1	0	0	0	B = {Tue, Wed, Thu}
A ∪ B	1	1	1	1	0	1	0	element-wise: or
A ∩ B	0	0	1	1	0	0	0	element-wise: and
A \ B	1	0	0	0	0	1	0	element-wise: and not

Boolean vector – operations

Pseudocode

```
Function Member(A, i)          // O(1)
  Return A[i]
End function

Procedure Union(A, B, C, n)     // O(n)
  For i ← 1 to n
    C[i] ← A[i] or B[i]
  End for
End procedure

// Intersection: C[i] ← A[i] and B[i]
// Difference:   C[i] ← A[i] and not B[i]

Function Subset(A, B, n)        // A ⊆ B ?
  i ← 1
  While i ≤ n and (not A[i] or B[i])
    i ← i + 1
  End while
  Return i > n
End function
```

Java

```java
boolean[] in = new boolean[7];

BitVectorSet union(BitVectorSet o) {
    BitVectorSet r = new BitVectorSet();
    for (int i = 0; i < in.length; i++)
        r.in[i] = in[i] || o.in[i];
    return r;
}
```

Operation	Steps
membership, insertion, deletion	**O(1)**
union, intersection, difference	O(|U|)
subset	O(|U|)
number of elements	O(|U|)

Bit mask: one bit = one element

If $|U| \le 64$, the whole set fits into **a single integer**: the **i**-th bit shows whether $u_i \in A$.

Operation	Java	Go	Example (A = 0101101, B = 0001110)
union	`a | b`	`a | b`	0101111
intersection	`a & b`	`a & b`	0001100
difference	`a & ~b`	`a &^ b`	0100001
membership (i)	`(a >> i & 1) == 1`	`a>>i&1 == 1`	
subset	`(a & ~b) == 0`	`a&^b == 0`	
number of elements	`Integer.bitCount(a)`	`bits.OnesCount(a)`	4

One set operation = **a single processor instruction**. Applications: permissions (read/write/execute), days, settings.

Boolean vector – advantages and limits

Advantages:

- The fastest: membership **O(1)**, set operations in one step with bit masks
- Simple to implement

Limits:

- The universe must be **small and known in advance**
- The space needed is proportional to |U|, not to the number of elements
  e.g. a set of integers between 0 … 10^9: 10^9 bits ≈ **125 MB –** even if it contains only 3 elements!
- Elements must be identified by a **sequence number** (Mon → 0, Tue → 1, …)

Representation by an unsorted array

Unsorted array

The elements are stored in an array, **in any order**, without duplicates.

A = `[8, 2, 11, 5]` · B = `[9, 5, 14, 3, 11]`

- **Membership:** linear search – O(n)
- **Insertion:** first check whether it is already present (O(n)), then append at the end – O(1)
- **Deletion:** find it (O(n)) and **move the last element into its place** – no shifting needed, since the order does not matter!

> The space needed is proportional to the number of elements – no universe is needed, any element can be stored.

Unsorted array – operations

Subset: A ⊆ B

```
Function Subset(A, n, B, m)
  i ← 1
  While i ≤ n and Member(B, m, A[i])
    i ← i + 1
  End while
  Return i > n
End function
```

Intersection: A ∩ B

```
Procedure Intersection(A, n, B, m, C, k)
  k ← 0
  For i ← 1 to n
    If Member(B, m, A[i]) then
      k ← k + 1 ; C[k] ← A[i]
    End if
  End for
End procedure
```

Union: every element of A + the elements of B that are not in A

Difference: the elements of A that are not in B

Operation	Steps
membership	O(n)
insertion	O(n)
deletion	O(n)
subset	**O(n · m)**
union, intersection, difference	**O(n · m)**

A linear search in the other set for every element → quadratic!

Representation by a sorted array

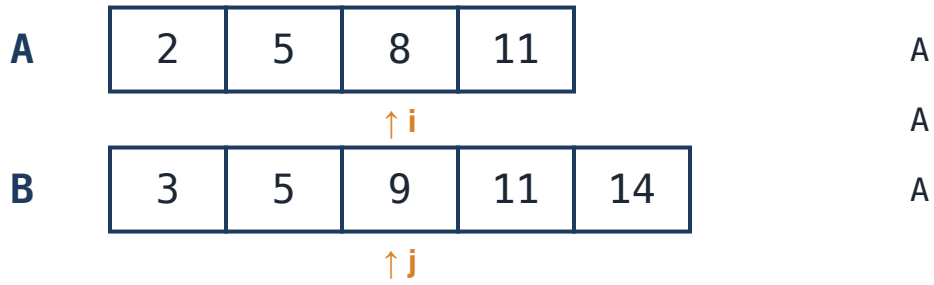
Sorted array

The elements are stored in **increasing order**, without duplicates.

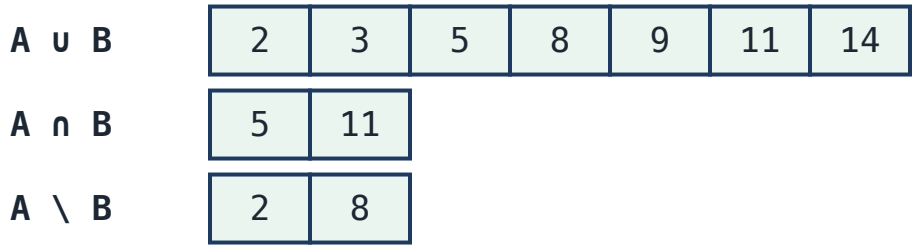
A = `[2, 5, 8, 11]` · B = `[3, 5, 9, 11, 14]`

- **Membership: binary search** – O(log n)
- **Insertion / deletion:** the position is found by binary search, but elements must be **shifted** – O(n)
- **Set operations: merging** the two sorted arrays – a single joint pass

Merging: the basis of the set operations




```
A[i] < B[j] → take A[i], i ← i + 1
A[i] = B[j] → take it once, both advance
A[i] > B[j] → take B[j], j ← j + 1
```



all three: a single joint pass

$O(n + m)$

Union by merging

Pseudocode

```
Procedure Union(A, n, B, m, C, k)
  i ← 1 ; j ← 1 ; k ← 0
  While i ≤ n and j ≤ m
    k ← k + 1
    If A[i] < B[j] then
      C[k] ← A[i] ; i ← i + 1
    else if A[i] > B[j] then
      C[k] ← B[j] ; j ← j + 1
    else                          // equal
      C[k] ← A[i] ; i ← i + 1 ; j ← j + 1
    End if
  End while
  While i ≤ n                     // rest of A
    k ← k + 1 ; C[k] ← A[i] ; i ← i + 1
  End while
  While j ≤ m                     // rest of B
    k ← k + 1 ; C[k] ← B[j] ; j ← j + 1
  End while
End procedure
```

Java

```java
int i = 0, j = 0, k = 0;
while (i < a.length && j < b.length) {
    if (a[i] < b[j])      c[k++] = a[i++];
    else if (a[i] > b[j]) c[k++] = b[j++];
    else { c[k++] = a[i++]; j++; }
}
while (i < a.length) c[k++] = a[i++];
while (j < b.length) c[k++] = b[j++];
```

Intersection: write into C only in the equal branch, do not copy the rest.

Difference: write only in the `A[i] < B[j]` branch, and copy the rest of A.

Subset by merging

```
Function Subset(A, n, B, m)              // A ⊆ B ?
  i ← 1 ; j ← 1
  While i ≤ n and j ≤ m and A[i] ≥ B[j]
    If A[i] = B[j] then i ← i + 1
    j ← j + 1
  End while
  Return i > n
End function
```

- If `A[i] < B[j]`, then A[i] is **certainly not** in B (the further elements of B are even larger) → not a subset
- Steps: **O(n + m)**

Comparing the representations

Operation	Boolean vector	Unsorted array	Sorted array
membership	**O(1)**	O(n)	O(log n)
insertion, deletion	**O(1)**	O(n)	O(n)
subset	O(|U|)	O(n · m)	**O(n + m)**
union, intersection, difference	O(|U|)	O(n · m)	**O(n + m)**
space	|U| bits	n elements	n elements
precondition	small, known U	–	sortable elements

There is no “best” representation: the choice depends on the **size of the universe** and on the **most frequent operations**.

Built-in sets

Java

```java
Set<String> a = new HashSet<>();  // O(1) avg.
a.add("Wuhan");
a.contains("Wuhan");    // true
a.remove("Wuhan");

Set<String> u = new HashSet<>(a);
u.addAll(b);            // union
u.retainAll(b);         // intersection
u.removeAll(b);         // difference

TreeSet<Integer> t = new TreeSet<>(); // O(log n)
t.first(); t.last();    // sorted!
```

Go – no set type, use a map

```go
type Set map[string]struct{}

s := Set{}
s["Wuhan"] = struct{}{}      // insertion
_, ok := s["Wuhan"]          // membership
delete(s, "Wuhan")           // deletion

// common shortcut:
seen := map[int]bool{}
if !seen[x] {
    seen[x] = true
}
```

`HashSet` and Go `map`: **hash table** (week 11) · `TreeSet`: **balanced search tree** (week 15)

Summary

- **Set:** an unordered collection of distinct elements; operations: $\in$, $\subseteq$, $\cup$, $\cap$, $\setminus$
- **Boolean vector / bit mask:** the fastest for a small, known universe – O(1), O(|U|)
- **Unsorted array:** simple, but the set operations are O(n · m)
- **Sorted array:** binary search O(log n), set operations by **merging** in O(n + m)
- Built-in: Java `HashSet` / `TreeSet`, Go `map[T]struct{}`

Next week: the stack – the LIFO principle, array implementation, evaluating postfix expressions.

Review questions

1. The universe is U = {1, 2, …, 8}. Write as boolean vectors: A = {2, 3, 5, 7}, B = {1, 2, 3, 4}! Compute A ∪ B, A ∩ B, A \ B!
2. True or false: with the unsorted array representation, intersection takes O(n + m) steps.
3. Perform the merging on paper: A = [1, 4, 6, 9], B = [2, 4, 9, 10]! Give A ∪ B, A ∩ B and A \ B!
4. Why is it not practical to represent the set of student IDs (8-digit numbers) by a boolean vector?
5. With bit masks: `a = 0b1010`, `b = 0b0110`. What are `a & b`, `a | b`, `a & ~b`?

Practice

✎ Demo programs

Program	Java	Go	What does it show?
Boolean vector	`BitVectorSet.java`	`w06/bitvectorset`	a set of days, bit mask operations
Unsorted	`UnorderedSet.java`	`w06/unorderedset`	O(n · m) operations, deletion with the last element
Sorted	`SortedSet.java`	`w06/sortedset`	binary search, merging
Built-in	`BuiltinSets.java`	`w06/builtin`	`HashSet`, `TreeSet`, Go `map`

✎ Exercises

1. **Removing duplicates:** make a set from an array by sorting! (`[4,1,4,2,1] → [1,2,4]`)
2. **Symmetric difference:** A △ B = (A \ B) ∪ (B \ A) – on sorted arrays, with a single merge!
3. **Disjoint?** Decide in O(n + m) steps whether two sorted sets have no common element!
4. **Power set:** print every subset of an **n**-element set using bit masks! How many are there?
5. **Distinct letters:** how many different English letters occur in a text? (boolean vector)
6. **Pangram:** does a sentence contain every letter of the English alphabet?
7. **Common friends:** the intersection of two friend lists using a built-in set (map).

Solutions: `Exercises06.java`, `w06/exercises`

✎ Homework

1. Implement a `SortedSet` class with `add(x)` and `remove(x)` operations that always stays sorted! (Hint: find the position by binary search.)
2. **Multiset** (bag): an element may occur several times. How would you represent it instead of a boolean vector? Implement union and intersection (maximum and minimum of the counts)!
3. Compute the intersection of three sorted sets with a **single** joint pass!
4. In an array of 1 000 000 numbers between 1 and 100 000, how many **distinct** numbers are there? Solve it with a boolean vector and compare it with the sorting-based solution!