Lesson 3

Data Structures and Algorithms
DSA

In this lesson we will talk about: data structures, analysing and designing algorithms, algorithm efficiency, searching and sorting algorithms

Data Structures

What is a data structure?

In computer science, a **data structure** is a data organization and storage format that is usually chosen for efficient access to data.[1][2][3] More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data,[4] i.e., it is an algebraic structure about data.

Data Structures

- How to **organize** data
- For **efficient** access

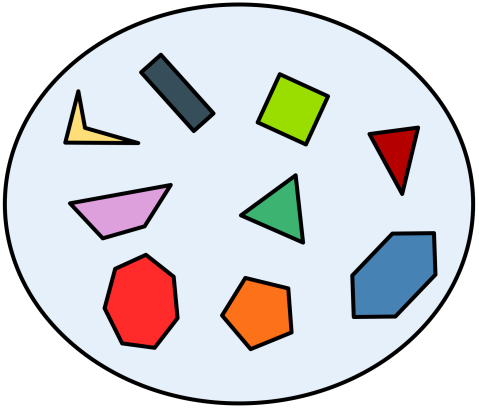
Its a collection of data values, and the relationships among these values

Sets

The basic, fundamental data structure: $\{1,2,4,51,9\}$

- mathematical set
- unchanging, unique elements, no duplicates
- contains a fixed number of elements or infinite

A set of polygons



Sets

A set is fixed, not changing. Sets which are manipulated by algorithms are dynamic. These sets can change in size, grow or shrink, basically change over the time.

- static set - $\{1,2,4,51,9\}$
- dynamic set - can add or remove elements

But what is a set?

Sets

A set is a **mathematical model** for a collection of different things, a set contains elements or members, which can be mathematical objects of any kind numbers, symbols, points in space, lines, other geometrical shapes, variables, or even other sets.

Sets

- $\{\text{white, blue, red, yellow}\}$
- The empty set $\{\}$
- Natural numbers: $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$
- Natural numbers except 0: $\mathbb{N}^* = \{1, 2, 3, \ldots\}$
- Integers: $\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$
- Positive integers: $\mathbb{Z}_+ = \{0, 1, 2, 3, \ldots\}$

$$\{\mathbf{1,2,3,4}\}$$

Roster or the enumeration notation defines a set by listing its elements between curly brackets, separated by commas.

Basic Operations on Sets

- Insert
- Delete
- Test - if element X belongs to a set or not

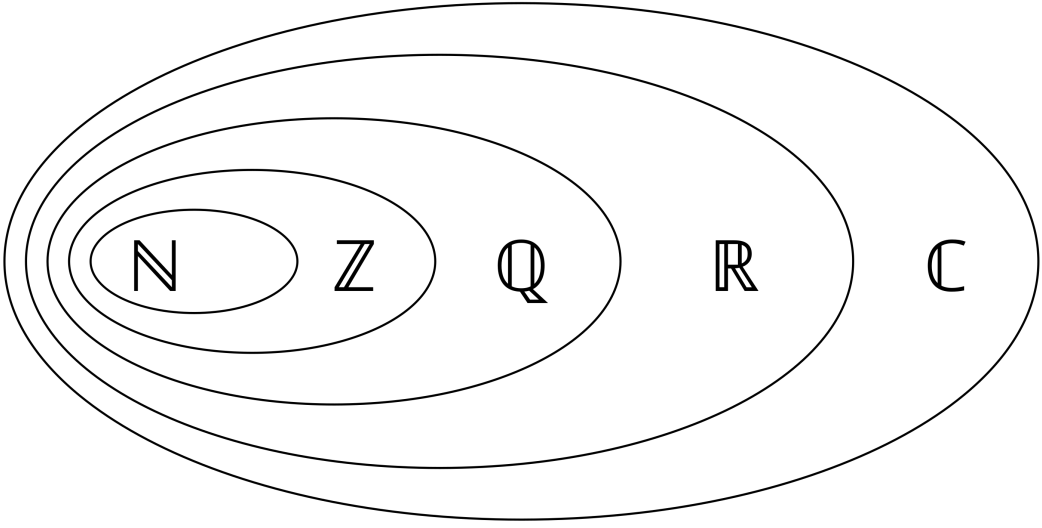
A dynamic set which supports all these basic operations: **a dictionary**

Data Structures

The main advantage of using a set data structure is that it allows you to perform operations on a collection of elements in an efficient and organized way.

Lesson 3

Natural Numbers (Counting Numbers) ($\mathbb{N}$)	Numbers you use for counting: 1, 2, 3 ...	It's "natural" to count on your fingers: 1, 2, 3, ...
Whole Numbers	The natural numbers, plus 0: 0, 1, 2, 3 ...	The word "whole" has an "o" in it, so include 0.
Integers ($\mathbb{Z}$)	Whole numbers, their opposites (negatives), plus 0: ... −2, −1, 0, 1, 2 ...	Integers can be separated into negative, 0, and positive numbers.
Rationals ($\mathbb{Q}$)	Integers and all fractions, positive and negative, formed from integers. These include repeating fractions, such as $\frac{1}{3}$, or .33333.. or $\overline{3}$.	The word "rational" is a derivation of "ratio", and rational numbers are numbers that can be written as a ratio of two integers. "Q" stands for quotient.
Irrationals	Numbers that cannot be expressed as a fraction, such as π, $\sqrt{2}$, e. (We'll learn about these later).	If something is "irrational", it's not easy to explain or understand.
Real Numbers ($\mathbb{R}$)	Rational numbers and Irrational Numbers. The real number system can be represented on a number line: 	If a number exists on a number line that you can see, it must be "real". Note that the "smallest" real number is negative (–) infinity ($-\infty$), and the largest real number is infinity (∞). We can never really get to these "numbers" ($-\infty$ and ∞), but we can indicate them as the "end" of the real numbers.
Complex Numbers ($\mathbb{C}$)	Real numbers, plus imaginary numbers (concept only, such as $\sqrt{-2}$).	"Imaginary" numbers are difficult to imagine, since they are so "complex".


ℕ
ℤ
ℚ
ℝ
ℂ

Data structures

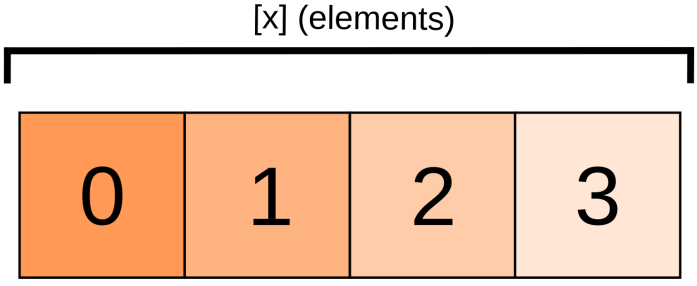
- Arrays
- Matrices
- Stacks
- Queues
- Linked lists
- Trees

Arrays

Arrays

In computer science, an **array** is a data structure consisting of a collection of *elements* (values or variables), of same memory size, each identified by at least one *array index* or *key*. An array is stored such that the position of each element can be computed from its index tuple by a mathematical formula.[1][2][3] The simplest type of data structure is a linear array, also called a one-dimensional array.

Typical "1 Dimensional" array



Element indexes are typically defined in the format *[x]*
[x] being the number of elements
For example: this array could be defined as *array[4]*

Example 1: Traversing the array A

```
1: procedure GETARRAY(A)                ▷ Returns the max value in A
2:     L ← length(A)
3:     for i=0 to L-1 do
4:         print A[i]
5:     end for
6: end procedure
```

Example 2: Find the max value in the array A

```
1:  procedure MaxArray(A)                    ▷ Returns the max value in A
2:      N ← length(A)
3:      MAX ← A[0]
4:      for from i=1 to N-1 do
5:          if A[i] > MAX then
6:              MAX = A[i]                   ▷ The MAX is A[i]
7:          end if
8:      end for
9:      return MAX
10: end procedure
```

Example 3: Search element X in array A

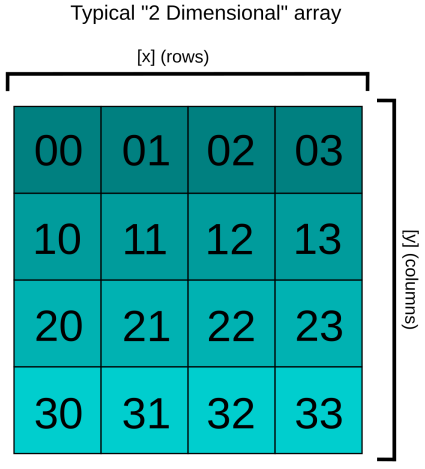
```
1:  procedure SEARCHARRAY(A)   ▷ Returns the max value in A
2:      X ← MyElement
3:      N ← length(A)
4:      for from i=0 to N-1 do
5:          if X = A[i] then
6:              return i                    ▷ The index for my match
7:          end if
8:      end for
9:      return -1                           ▷ otherwise return -1
10: end procedure
```

Matrices

Matrices

Because the mathematical concept of a matrix can be represented as a two-dimensional grid, two-dimensional arrays are also sometimes called "matrices". In some cases the term "vector" is used in computing to refer to an array, although tuples rather than vectors are the more mathematically correct equivalent. Tables are often implemented in the form of arrays, especially lookup tables; the word "table" is sometimes used as a synonym of array.

Matrices



Element indexes are typically defined in the format *[x][y]*
[x] being the number of rows
[y] being the number of columns
For example: this array could be defined as *array[4][4]*