# JAVA

# OBJECT ORIENTED PROGRAMMING

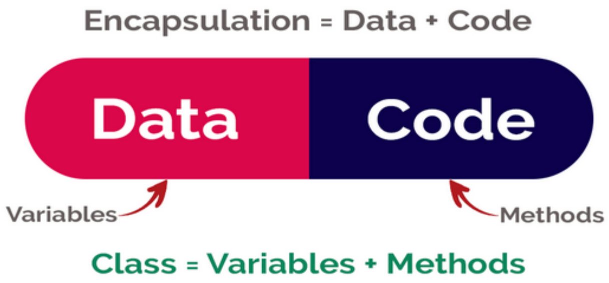
**Basic principles of Object Oriented Programming**

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

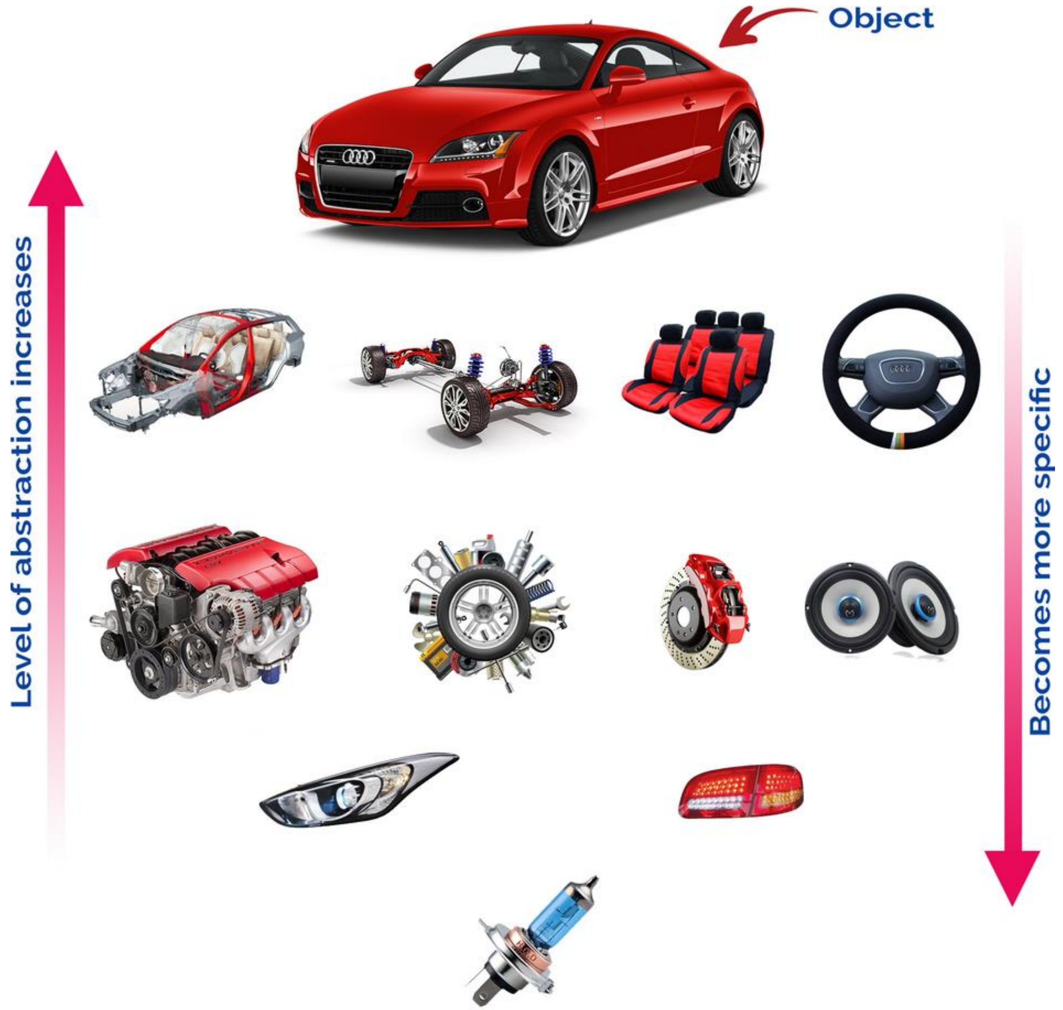
**Object Oriented approach offers advantages**

- Data Values are secured
- Mishandling of Data is protected
- Error Detection and Correction easier
- Easier in coding complex program

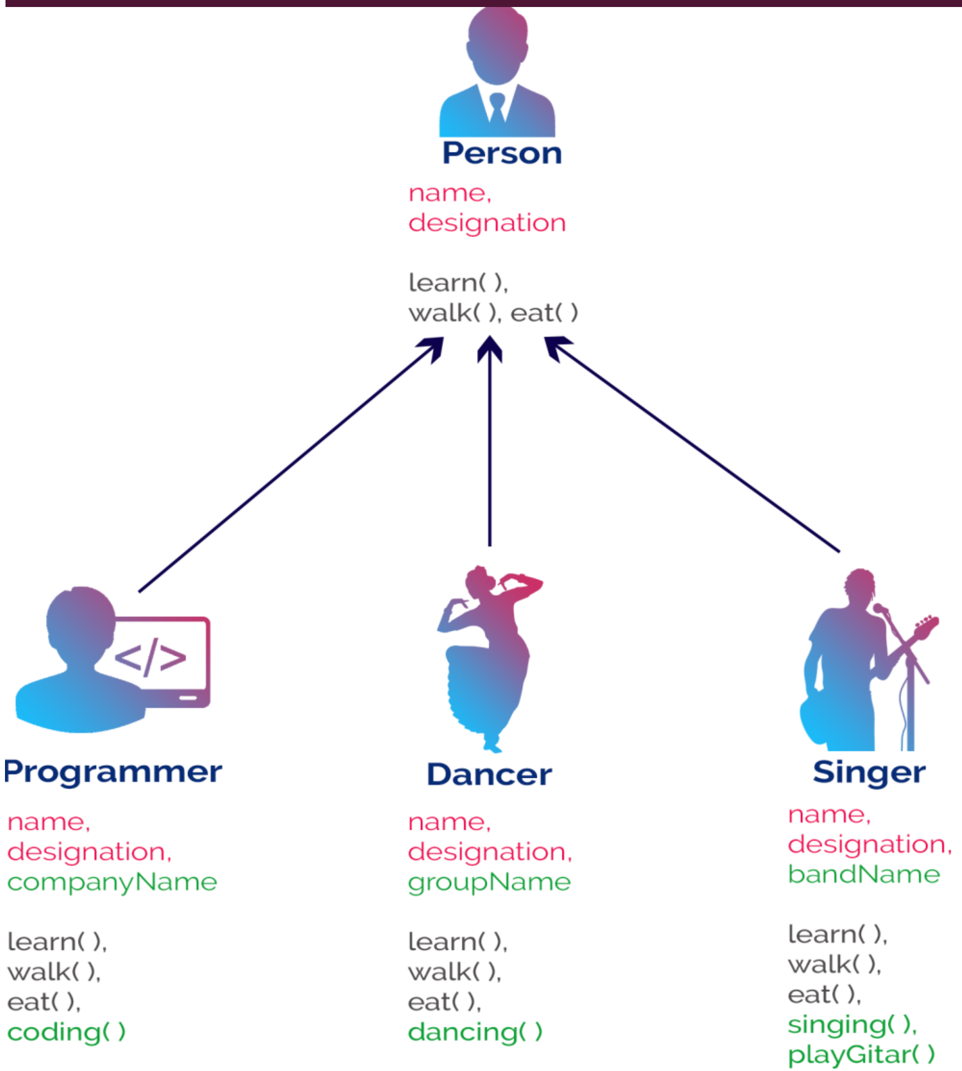
# ENCAPSULATION



# ABSTRACTION



# INHERITANCE



**Person**

name,
designation

learn( ),
walk( ), eat( )

**Programmer**

name,
designation,
companyName

learn( ),
walk( ),
eat( ),
coding( )

**Dancer**

name,
designation,
groupName

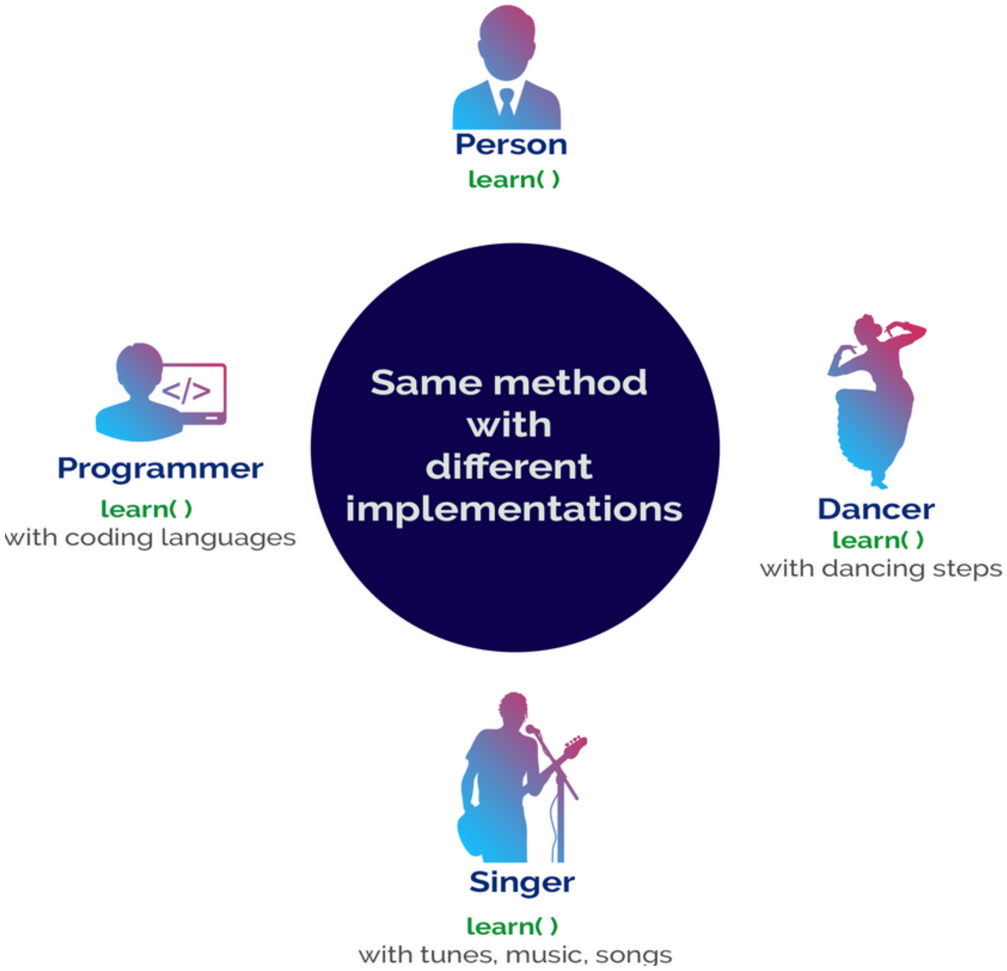
learn( ),
walk( ),
eat( ),
dancing( )

**Singer**

name,
designation,
bandName

learn( ),
walk( ),
eat( ),
singing( ),
playGitar( )

# POLYMORPHISM



Person
learn( )

Same method with different implementations

Programmer
learn( )
with coding languages

Dancer
learn( )
with dancing steps

Singer
learn( )
with tunes, music, songs

# CLASS, FUNCTIONS, VARIABLES

- Class
- Instance of Class (Object)
- Default Constructor
- Constructor with Arguments
- Class Variables and its Scope
- Local Variables and its Scope
- this operator
- Return type of functions
- Accessing member functions in static main method.

# DATA TYPES

## Primitive Data Types

| Type | Size (bits) | Minimum | Maximum | Example |
|---|---|---|---|---|
| *byte* | 8 | $-2^{7}$ | $2^{7}-1$ | *byte b = 100;* |
| *short* | 16 | $-2^{15}$ | $2^{15}-1$ | *short s = 30_000;* |
| *int* | 32 | $-2^{31}$ | $2^{31}-1$ | *int i = 100_000_000;* |
| *long* | 64 | $-2^{63}$ | $2^{63}-1$ | *long l = 100_000_000_000_000;* |
| *float* | 32 | $-2^{-149}$ | $(2-2^{-23}) \cdot 2^{127}$ | *float f = 1.456f;* |
| *double* | 64 | $-2^{-1074}$ | $(2-2^{-52}) \cdot 2^{1023}$ | *double f = 1.456789012345678;* |
| *char* | 16 | 0 | $2^{16}-1$ | *char c = ‘c';* |
| *boolean* | 1 | – | – | *boolean b = true;* |

## Non-Primitive Data Types

- String
- Array
- User Defined Classes

## Wrapper Classes

## (Sub Classes of Number)

- Integer
- Long
- Byte
- Double
- Float
- Short

# OPERATORS

**Unary Operator**

- Postfix (a++, a--)
- Prefix (++a, --a)

**Arithmetic Operator**

- +, - , * , % , /

**Relational Operator**

- <, >,  <=, >=, instanceOf
- ==, !=

**Logical Operator**

- && , ||

**Ternary Operator**

- ?, :

**Assignment Operator**

- =, +=, -=, *=, /=, %=

# CONDITIONAL STATEMENTS

- Normal Flow of Control
  - Regular flow without any condition or branching (top-down approach).
- Bi-directional flow of Control
  - If statement
  - If and only if statements (use of multiple if)
  - If – else statement
  - If – else If statement
  - Nested if statement with else
  - System.exit(0) to break and come out.

- Multiple branching of Control
  - Switch block
  - Break Statement
  - Fall Through
  - Default Case

# SYNTAX FOR CONDITIONAL STATEMENTS

```
if(condition1) {
// block of code to be executed if
condition is true
}
if(condition2) {
// block of code to be executed if
condition is true
}
if(condition3 || (condition3 &&
condition4)) {
// block of code to be executed if condition is true
}
else if (condition5){
// block of code to be executed, else if condition is true
}
else{
// reaches only if all the conditions are false
}
```

```
switch (number or char) {
    case 1:
      // block of code
      break;
    case 2:
      // block of code
      break;
    case n:
      // block of code
      break;
    default:
      // block of code
      break;
}
```

# BASIC CALCULATIONS

- Find greatest among 3 Numbers
- **Divisible Function**
  - Divisible by 3 ,
  - divisible by 5
  - divisible by both 3 & 5
- **Year**
  - Find Leap year,
  - Century year,
  - Century Year and leap Year ,
  - Century Year not Leap year

- **Square**
  - Check perfect square Number
  - Find Second Smallest Number
- **Greatest or Smallest**
  - Find Smallest number
  - Find 2nd greater number
- **Taxi Charges**
  - Till 5km – Rs.100
  - Next 10km – Rs.10/km
  - Next 10km – Rs.8/km
  - > 25 km – Rs.5/km

- **Percentage Calculation**
  - Till 2000 – 5 %
  - 2000 to 5000 – 10%
  - 5000 to 10000 – 15%
  - Above 10000 – 20%
- **Marks & Percentage**
- **Celsius | Fahrenheit**
  - c = (5/9)*(f-32)
  - f = 1.8*(c+32)
- **Feet – Meters**
  - 1 m = 3.28084 foot
  - 1 foot = 12 inch

# BASIC CALCULATIONS

- Math Formulae
  - **Area & Perimeter**
    - Square
    - Rectangle
    - Triangle
    - Circle
  - **Volume**
    - Cylinder
    - Cone
    - Sphere
    - cuboid

- **Print Message**
  - Message for Input
  - Message for Output
- **Interest**
  - Simple Interest
  - Compound Interest
- **Arithmetic operations**
  - Addition
  - Subtraction
  - Multiplication
  - Division (quotient, reminder)
- **String Concatenation**

- **Java Math**
  - Min()
  - Max()
  - Avg()
  - Round()
  - Ceil()
  - Floor()
  - Random()
  - Sqrt()
  - Cbrt()
- **Unary**
  - Increment
  - Decrement

# PROJECT 01 – EB BILL GENERATION

| | Option | Per unit(₹) |
|---|---|---|
| **Metro Lights** | **M** | |
| Per unit | | 6.35 |
| **Light Commercial** | **C** | |
| Per unit <100 | | 5 |
| >100 | | 8.05 |
| **Public Workshop** | **P** | |
| upto 120 | | 2.85 |
| >120 | | 5.75 |
| **Cottage & Tiny Industries** | **I** | |
| less 500 | | 4 |
| >500 | | 4.6 |
| **Power Looms** | **L** | |
| upto 750 | | |
| 751-1000 units | | 2.3 |
| 1001-1500 units | | 3.45 |
| >1500 | | 4.6 |
| **Temporary Supply** | **T** | |
| All units | | 12 |
| **Public light Town** | **W** | |
| All | | 6.35 |
| **Govt Schools** | **G** | |
| All | | 5.75 |
| **Private Hostpital Institution** | **H** | |
| All | | 7.5 |

**Input:**

- Enter your Name : ______________________

M: Metro lights

C: Light Commercial

P: Public Workshop

C: Cottage & Tiny Industries

L: Power Looms

T: Temporary Supply

W: Public light Town

G: Govt Schools

H: Private Hostpital Institution

- Enter the EB Connection Type: _____
- Total Consumed Units : __________

**Output:**

```
-------------------------------------------------------------------

EB Bill Generation

-------------------------------------------------------------------

Bill Generated for Mr. / Ms., XXXX XXX

Your Connection Type is <see the types left image>

You have consumed 00.0 Units

Your Bill Amount is Rs. 00.00 only.

-------------------------------------------------------------------
```

# PROJECT 02 – DOMESTIC EB BILL CALCULATION

| Domestic | Unit | Per unit(₹) | Fixed Charges Per Month |
|---|---|---|---|
| 0 to 100 | 0-100 | | |
| 0 to 200 | 0-100 | | 20 |
| | 101-200 | 1.5 | |
| 0 to 500 | 0-100 | | 30 |
| | 101-200 | 2 | |
| | 201-500 | 3 | |
| > 500 | 0-100 | | 50 |
| | 101-200 | 3.5 | |
| | 201-500 | 4.6 | |
| | >500 | 6.6 | |

**Input:**

- Enter your Name : ______________________
- Total Consumed Units : __________________

**Output:**

```
-----------------------------------------------------------------------
EB Bill Generation for Domestic Type
-----------------------------------------------------------------------
Bill Generated for Mr. / Ms., XXXX XXX
You have consumed 00.0 Units
Charges for consumed units : Rs. 00.00
Fixed Charges per Month: Rs. 00.00
----------------------------------------------------------------------
Total Bill Amount is Rs. 00.00 only.
-----------------------------------------------------------------------
```

# ITERATIVE CONSTRUCTS

- Entry Controlled Loop
  - For loop (Fixed Iteration)
    - ❑ Use of Break & Continue
    - ❑ Finite Loop
      - ❑ Continuous Loop
      - ❑ Step Loop
    - ❑ Infinite Loop
      - ❑ Eg.,
      - `for(m=1; m<=10) { … }`
      - `for(;;) { … }`
      - `for(m=1; m<=10; m++) {`

```
 //do nothing
} [called delay loop or null loop]
```

      - `for(m=1; m<=10; m++); [called delay loop or null loop]`

  - While loop (Unfixed Iteration)
    - ❑ Use of Break in for loop
    - ❑ Use of Continue in for loop
- Exit Controlled Loop
  - Do – while loop
    - ❑ Use of Break in for loop
    - ❑ Use of Continue in for loop
- Inter Conversion of Loops

# SYNTAX : FOR .. WHILE .. DO WHILE

```
for(statement1; statement2; statement3) {
        // List of statements.
}
```

Statement1 – Initialize the variable

Statement2 – set the upper / lower limit for loop

Statement3 – increment / decrement to reach limit set

```
While(condition){
        Statements
          …
          …
          …
}
```

```
Do {
        Statements
        …
        …
        …
} While(condition)
```

# BASIC CALCULATION

- Sequence Numbers:
  - 1, 4, 9, 16,
  - 1, 2, 4, 7, 11,
  - 3, 6, 9, 12,
  - 4, 8, 16, 32
  - 1.5, 3.0, 4.5, 6.0,
  - 0, 7, 26
  - 1, 9, 25, 49,
  - 4, 16, 36, 64,
  - 0, 3, 8, 15,
  - 24, 99, 224, 399,
  - 2, 5, 10, 17,
  - 1, 11, 111, 1111, 11111
  - Find Odd / Even Numbers

- Sum of Sequence
  - 1 + 4 + 9 + ...... + 400
  - 1 + (1/2) + (1/3) + ...... + (1/20)
  - 1 + (1/3) + (1/5) + ...... + (1/19)
  - (1/2) + (2/3) + (3/4) + ...... + (19/20)
  - 2 - 4 + 6 - 8 + ...... - 20
  - (1*2) + (2*3) + ...... + (19*20)
  - S = a^2 + a^2 / 2 + a^2 / 3 + ...... + a^2 / 10
  - S = a + a^2 / 2 + a^3 / 3 + ...... + a^10 / 10
  - S = (a*2) + (a*3) + ...... + (a*20)
  - S = a + a^2 + a^3 + ...... + a^n

- Sum of Sequence
  - S = 1 + 2^2 / a + 3^3 / a^2 + ...... to n terms
  - S = 1^2/a + 3^2 / a^2 + 5^2 / a^3 + ...... to n terms
  - S = 1/a + 1/a^2 + 1/a^3 + ...... + 1/a^n
  - S = x/2 + x/5 + x/8 + x/11 + ...... + x/20
- Find the Perfect Square Number
- Buzz Numbers (ends with 7 or divisble by 7)
- Count of Digits in a Number
- Reverse of a Number
- GCD (Greatest Common Divisor)

# BASIC CALCULATION

- Addition without + operator
- Product without * operator
- quotient and remainder without / and % operator
- Palindrome number
- Perfect number
- Prime Number
- Automorphic Number
- Fibonacci Series
- Factors
- Niven
- Spy Number

# NESTED LOOP

# LIBRARY CLASSES

- Important packages
- Frequently used in java programs

| Java Packages | Purpose |
|---|---|
| Java.lang | String Manipulations |
| Java.io | Input / output operations |
| Java.awt | Graphical User Interfaces |
| Java.util | Implement data structures |
| Java.applet | Implementing applets |
| Java.net | Supporting network operations |
| Java.math | Mathematical operations |

# WRAPPER CLASS

- Conversion from String to Primitive types
  - String to Integer
  - String to Long
  - String to Float
  - String to Double
- Character
  - Input a character using
    - Scanner
    - InputStreamReader & BufferReader

- Conversion from Primitive type data to String
  - Integer to String
  - Long to String
  - Float to String
  - Double to String
- Conversion from Characters to ASCII code
  - char x = ‘A’
  - int n = (int) x; // returns 65
  - int i = 98;
  - char c = (char) i; //returns ‘b’

| Characters | ASCII Codes |
|---|---|
| 0 – 9 | 48 – 57 |
| A – Z | 65 – 90 |
| a – z | 97 - 122 |

# CHARACTER ORIENTED FUNCTIONS

- Return type as Boolean
  - isLetter()
  - isDigit()
  - isLetterOrDigit()
  - isWhitespace()
  - isUpperCase()
  - isLowerCase()

- Return type as Same type
  - toUpperCase()
  - toLowerCase()

# CONVERSION OF DATA TYPES

- Autoboxing
  - Primitive datatype => wrapper class
- Unboxing
  - Wrapper class => primitive data type

# ARRAYS

Single Dimensional

Syntax: <data_type> <variable_name>[] =new <data_type>[actual Size of elements];

Multi Dimensional

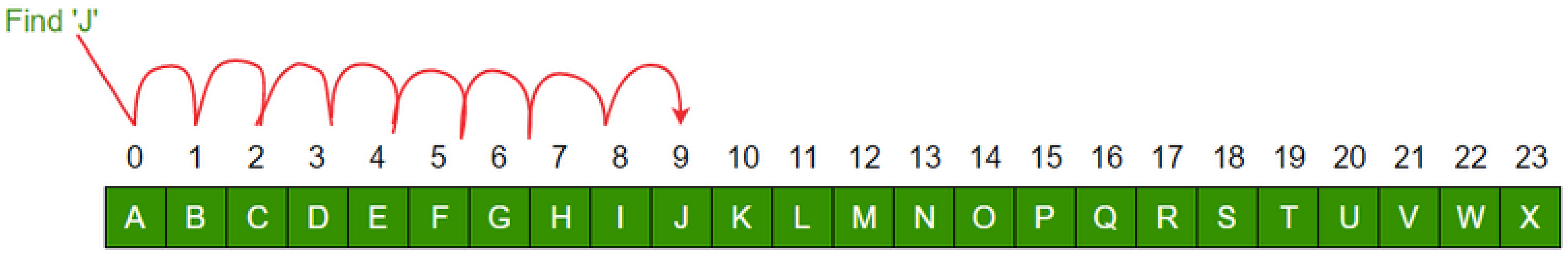
- Syntax: <data_type> <variable_name>[][] =new <data_type> [actual Size of elements] [actual Size of elements];

# BASIC OPERATIONS ON ARRAYS

- Searching
  - Linear Search
  - Binary Search
- Sorting
  - Selection Sort
  - Bubble Sort
- Inserting
- Deleting
- Merging

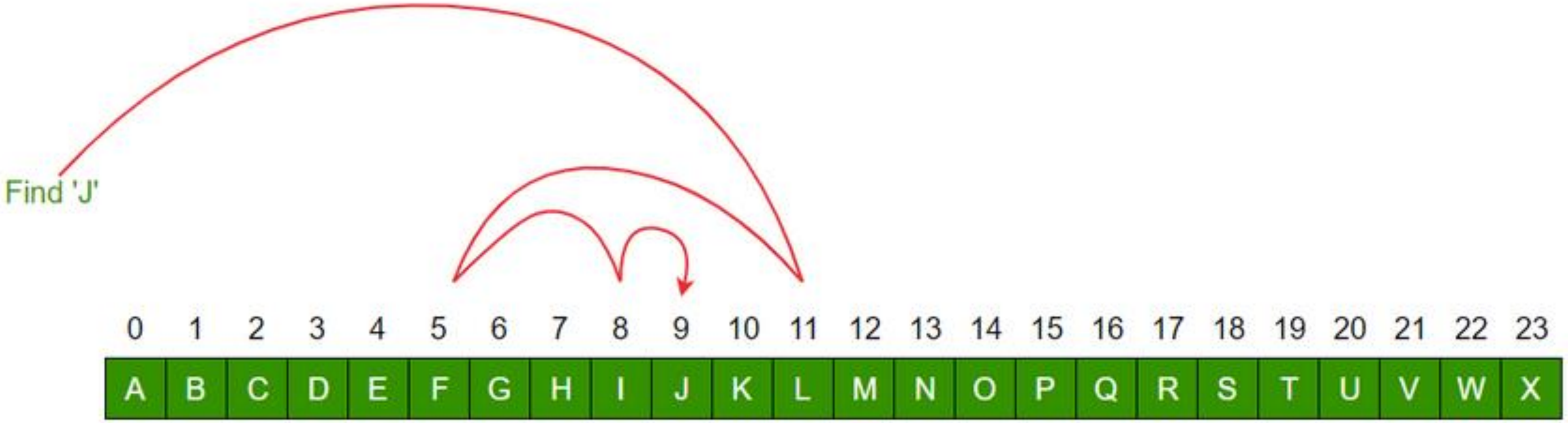
# SEARCHING

- Linear Search



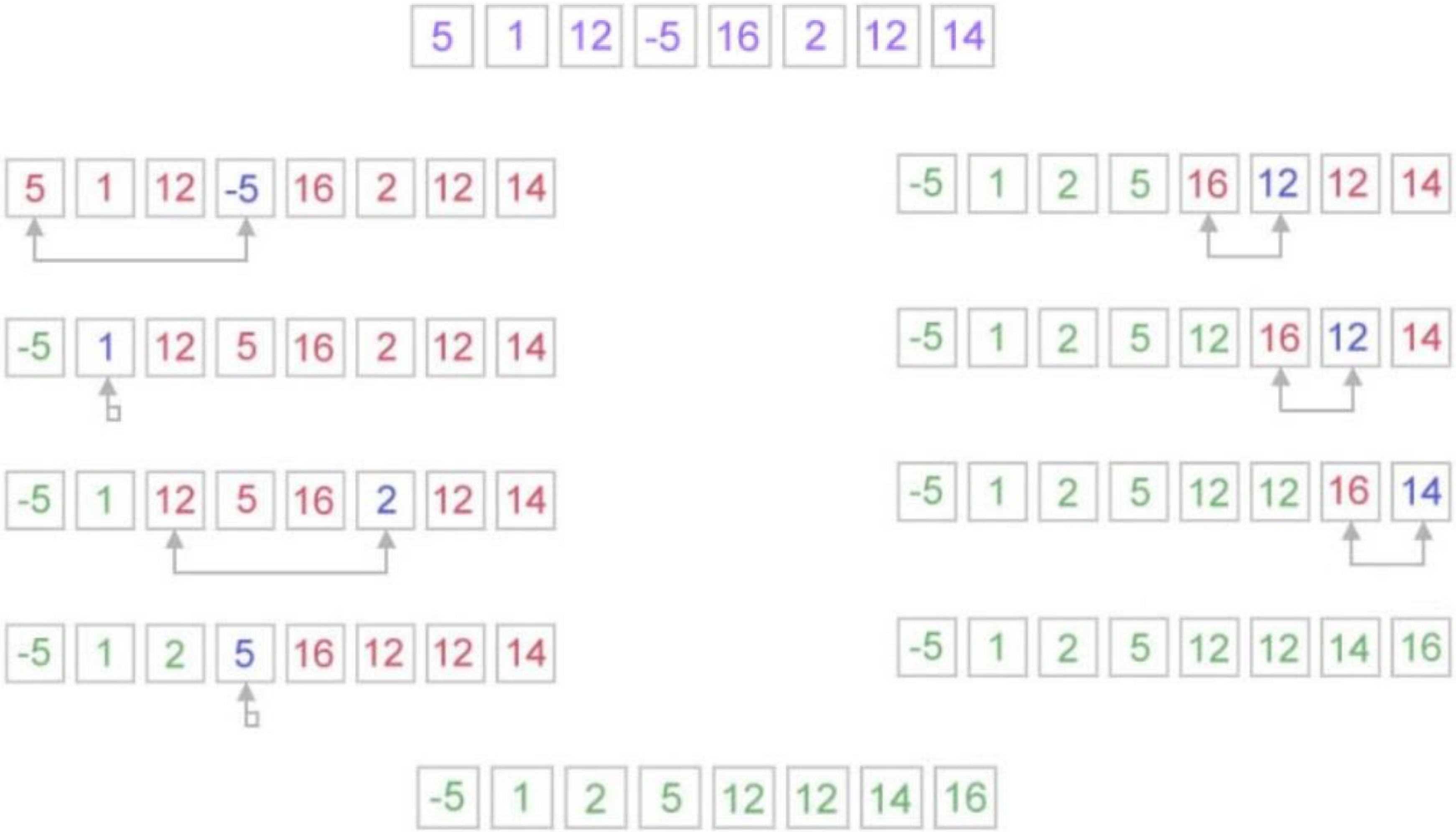
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X |

- Binary Search



| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X |

# SELECTION SORTING



5 1 12 -5 16 2 12 14

5 1 12 -5 16 2 12 14

-5 1 12 5 16 2 12 14

-5 1 12 5 16 2 12 14

-5 1 2 5 16 12 12 14

-5 1 2 5 16 12 12 14

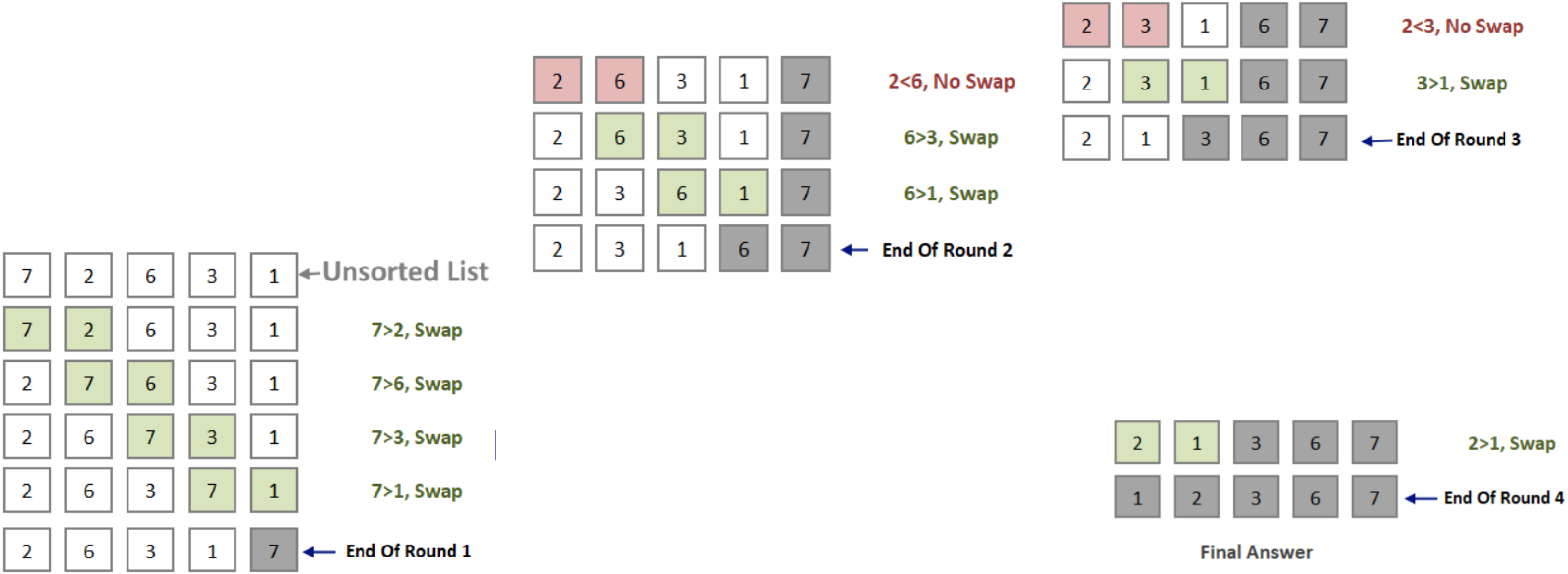
-5 1 2 5 12 16 12 14

-5 1 2 5 12 12 16 14

-5 1 2 5 12 12 14 16

-5 1 2 5 12 12 14 16

# BUBBLE SORTING



|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 7 | 2 | 6 | 3 | 1 | ← Unsorted List |
| 7 | 2 | 6 | 3 | 1 | 7>2, Swap |
| 2 | 7 | 6 | 3 | 1 | 7>6, Swap |
| 2 | 6 | 7 | 3 | 1 | 7>3, Swap |
| 2 | 6 | 3 | 7 | 1 | 7>1, Swap |
| 2 | 6 | 3 | 1 | 7 | ← End Of Round 1 |

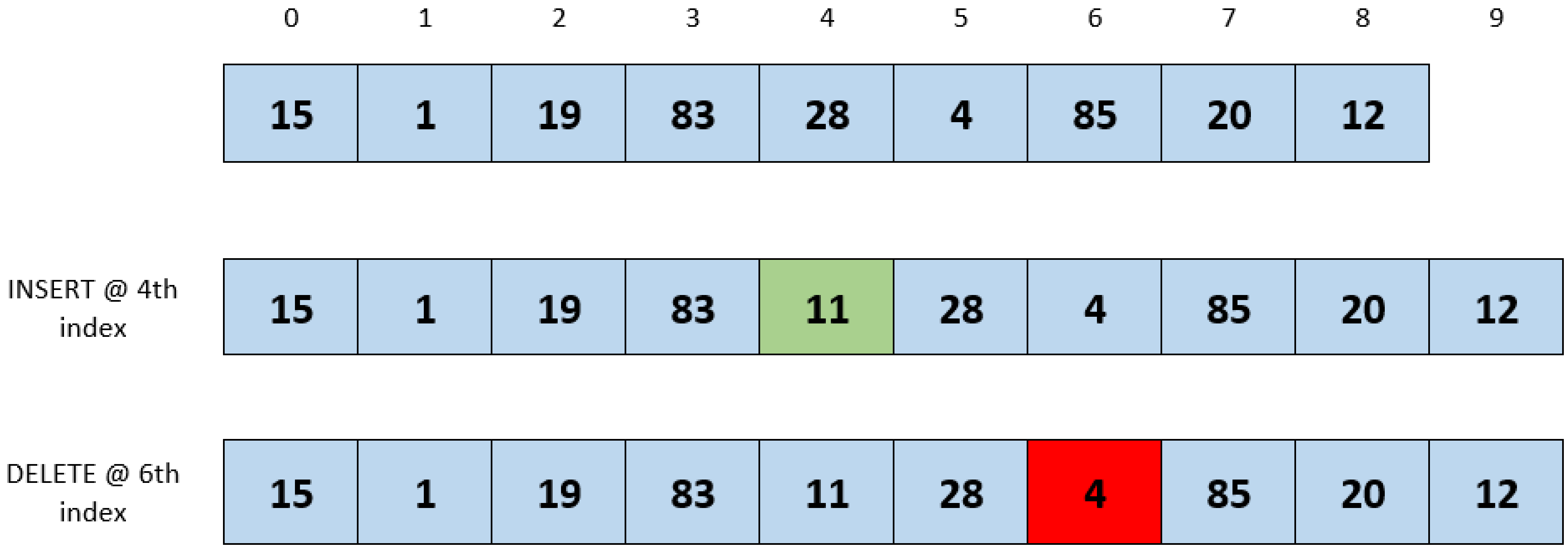
|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 2 | 6 | 3 | 1 | 7 | 2<6, No Swap |
| 2 | 6 | 3 | 1 | 7 | 6>3, Swap |
| 2 | 3 | 6 | 1 | 7 | 6>1, Swap |
| 2 | 3 | 1 | 6 | 7 | ← End Of Round 2 |

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 2 | 3 | 1 | 6 | 7 | 2<3, No Swap |
| 2 | 3 | 1 | 6 | 7 | 3>1, Swap |
| 2 | 1 | 3 | 6 | 7 | ← End Of Round 3 |

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 2 | 1 | 3 | 6 | 7 | 2>1, Swap |
| 1 | 2 | 3 | 6 | 7 | ← End Of Round 4 |

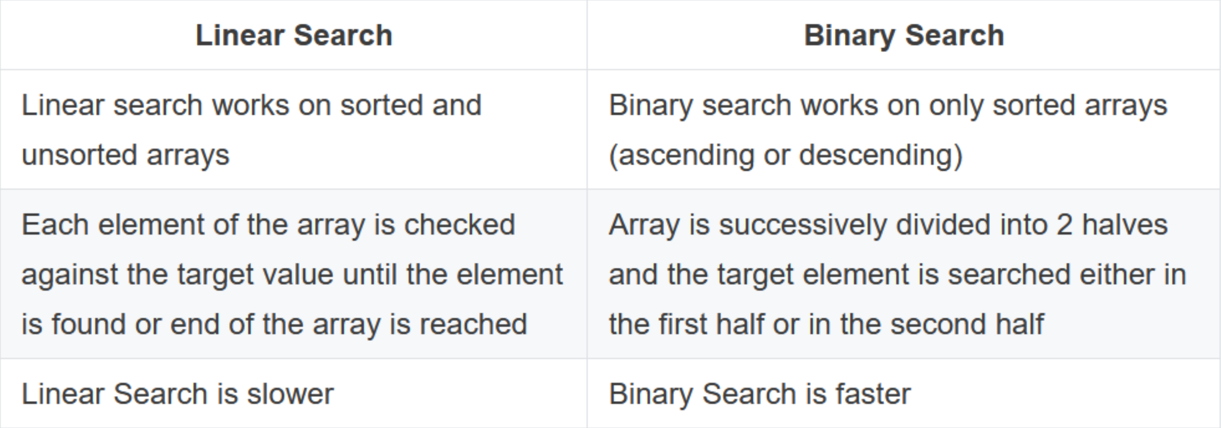
Final Answer

# INSERT AND DELETE ELEMENTS FROM ARRAY

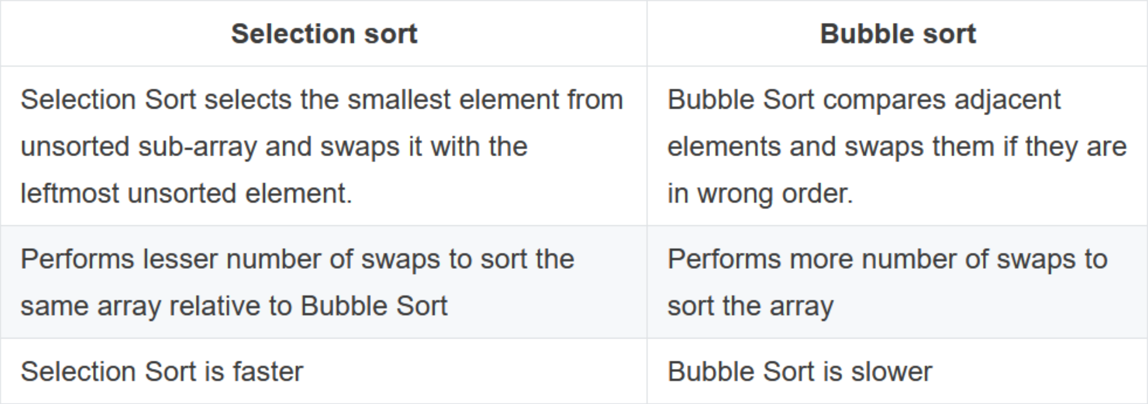


| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| | 15 | 1 | 19 | 83 | 28 | 4 | 85 | 20 | 12 | |
| INSERT @ 4th index | 15 | 1 | 19 | 83 | 11 | 28 | 4 | 85 | 20 | 12 |
| DELETE @ 6th index | 15 | 1 | 19 | 83 | 11 | 28 | 4 | 85 | 20 | 12 |

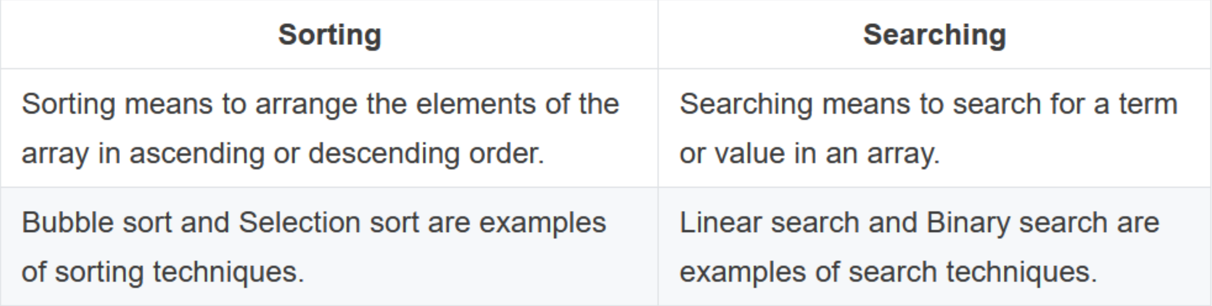
# DIFFERENCES



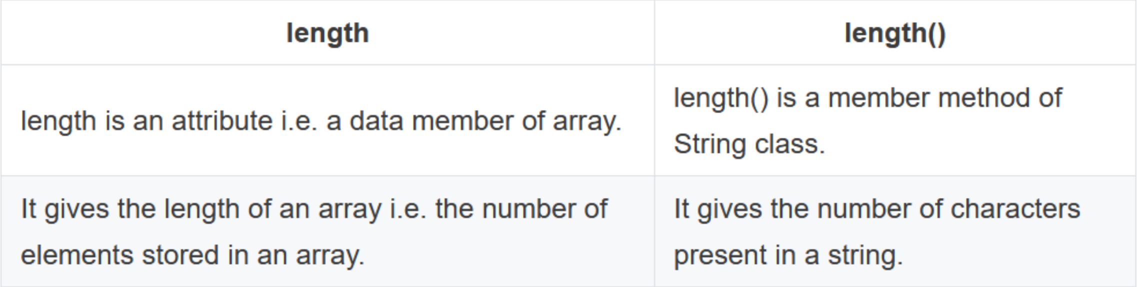
| Linear Search | Binary Search |
|---|---|
| Linear search works on sorted and unsorted arrays | Binary search works on only sorted arrays (ascending or descending) |
| Each element of the array is checked against the target value until the element is found or end of the array is reached | Array is successively divided into 2 halves and the target element is searched either in the first half or in the second half |
| Linear Search is slower | Binary Search is faster |



| Selection sort | Bubble sort |
|---|---|
| Selection Sort selects the smallest element from unsorted sub-array and swaps it with the leftmost unsorted element. | Bubble Sort compares adjacent elements and swaps them if they are in wrong order. |
| Performs lesser number of swaps to sort the same array relative to Bubble Sort | Performs more number of swaps to sort the array |
| Selection Sort is faster | Bubble Sort is slower |



| Sorting | Searching |
|---|---|
| Sorting means to arrange the elements of the array in ascending or descending order. | Searching means to search for a term or value in an array. |
| Bubble sort and Selection sort are examples of sorting techniques. | Linear search and Binary search are examples of search techniques. |



| length | length() |
|---|---|
| length is an attribute i.e. a data member of array. | length() is a member method of String class. |
| It gives the length of an array i.e. the number of elements stored in an array. | It gives the number of characters present in a string. |