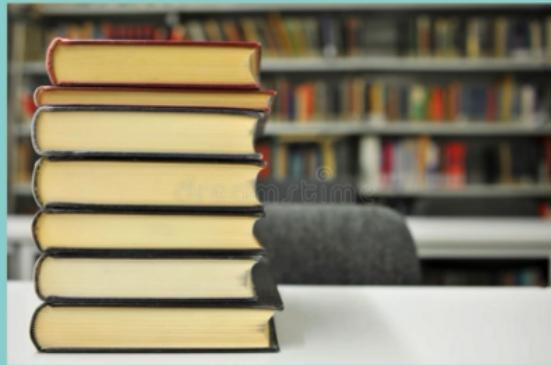


# What are Data Structures?

1

- **Data Structures** are different ways of organizing data on your computer, that can be used effectively.

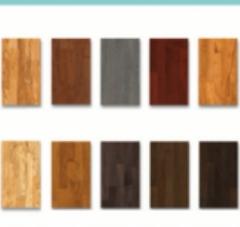


Importance of data structure and algorithm???

# What is an Algorithm?

2

- Set of steps to accomplish a task



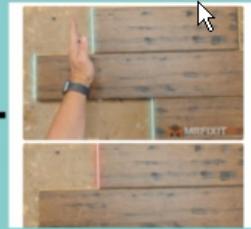
Step 1: Choosing flooring



Step 2: Purchase and bring



Step 3: Prepare sub flooring



Step 4: Determine the layout



Step 5: Trim door casing



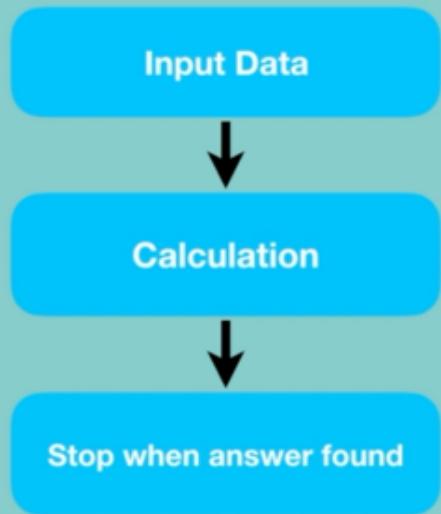
AppMillers  
[www.appmillers.com](http://www.appmillers.com)



# Algorithms in Computer Science

3

- Set of rules for a computer program to accomplish a task

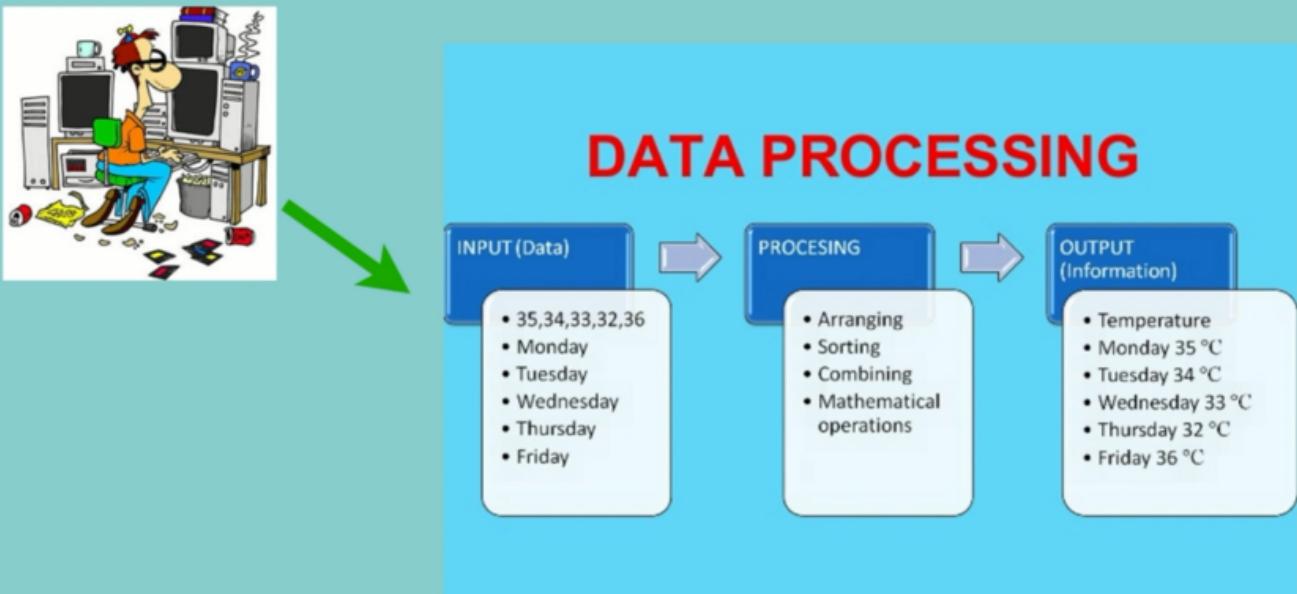


## What makes a good algorithm?

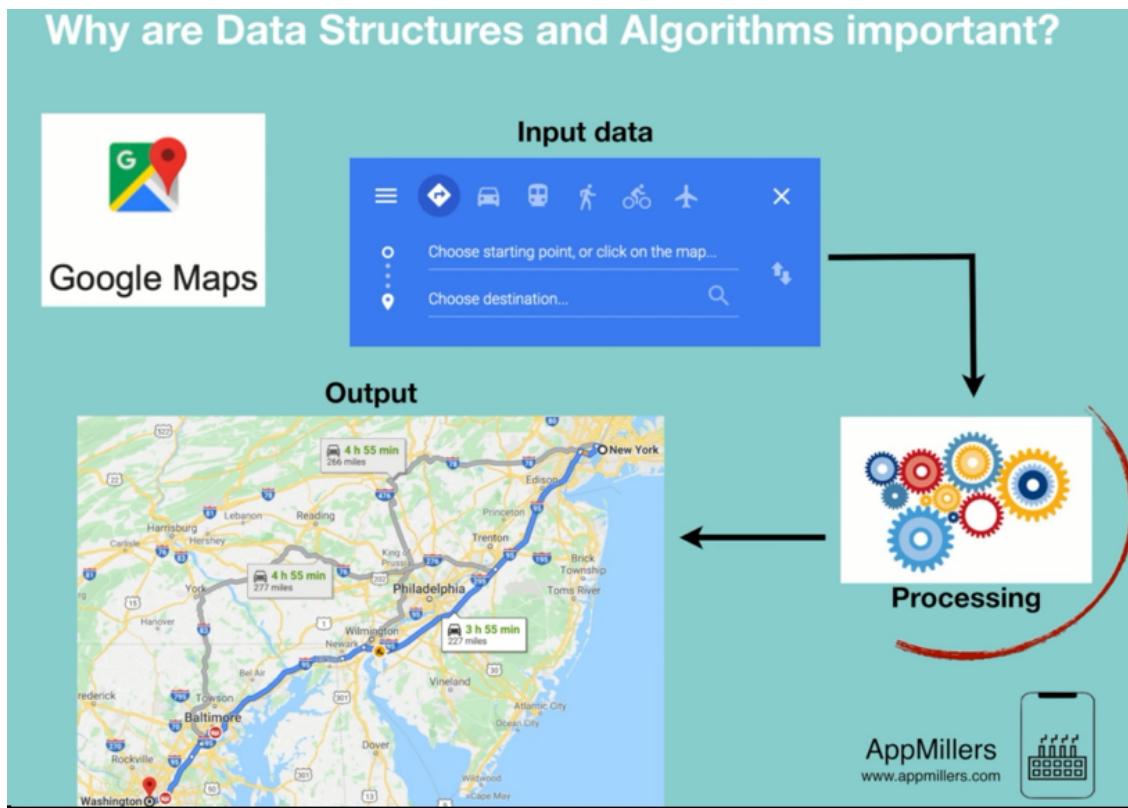
1. Correctness
2. Efficiency

# Why are Data Structures and Algorithms important?

4



## Why are Data Structures and Algorithms important?



# Why are Data Structures and Algorithms important?

5



books = data

arranging book = data structure

finding book= algorithm



# Why are Data Structures and Algorithms in INTERVIEWS?

6



- Problem solving skills
- Fundamental concepts of programming in limited time

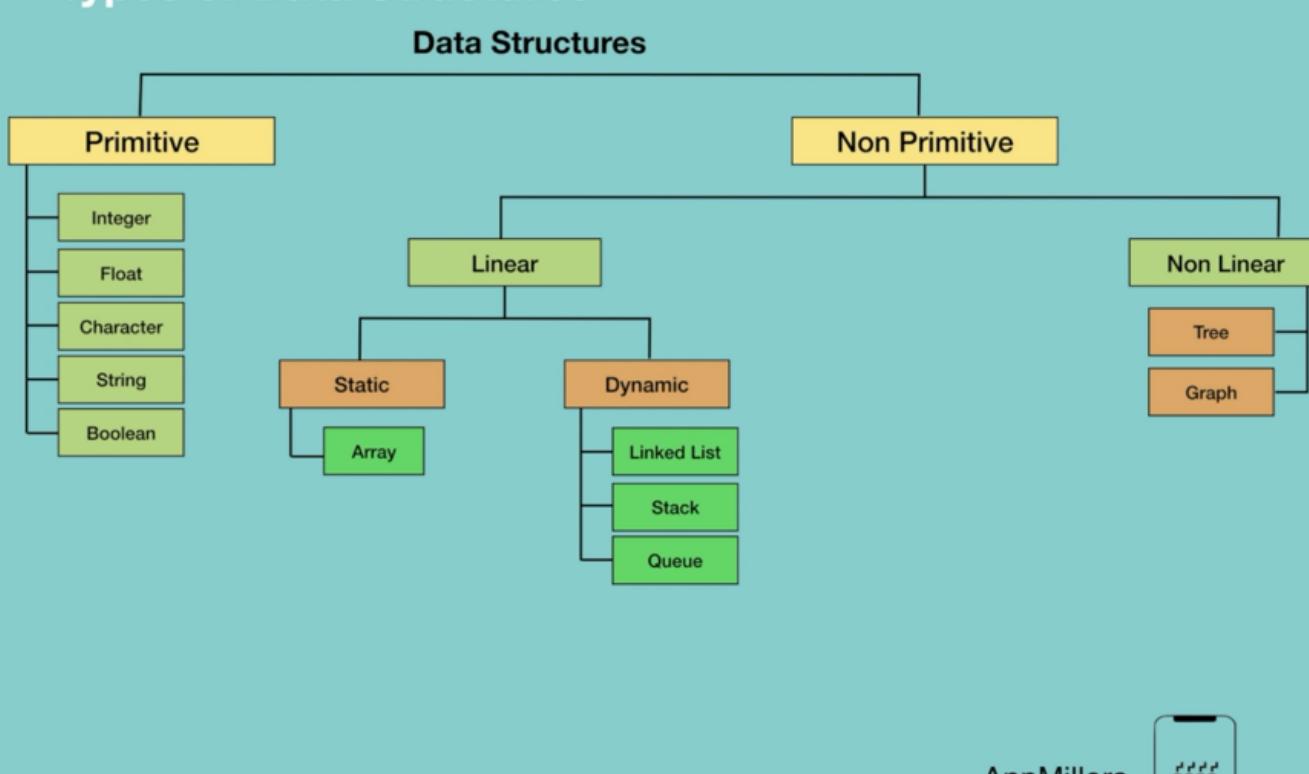


AppMillers  
[www.appmillers.com](http://www.appmillers.com)



# Types of Data Structures

7



AppMillers



# Types of Algorithms

- Simple recursive algorithms
- Divide and conquer algorithms
- Dynamic programming algorithms
- Greedy algorithms
- Brute force algorithms
- Randomized algorithms

## Divide and conquer algorithms

- Divide the problem into smaller subproblems of the same type, and solve these subproblems recursively
- Combine the solutions to the subproblems into a solution to the original problem

Examples: Quick sort and merge sort

## Dynamic programming algorithms

- They work based on memoization
- To find the best solution

## Greedy algorithms

- We take the best we can without worrying about future consequences.
- We hope that by choosing a local optimum solution at each step, we will end up at a global optimum solution

## Primitive Data Structures

DATA STRUCTURE	Description	Example
INTEGER	Numbers with our decimal point	1, 2, 3, 4, 5, 1000
FLOAT	Numbers with decimal point	3.5, 6.7, 6.987, 20.2
CHARACTER	Single Character	A, B, C, F
STRING	Text	Hello, Data Structure
BOOLEAN	Logical values true or false	TRUE, FALSE

## Definition and Usage

The **assert** keyword lets you test if a condition in your code returns True, if not, program will raise an `AssertionError` than can be written by ourselves.

## Syntax

---

```
assert {condition}, {message}
```

---

## Examples

### Example 1

Create a sequence of numbers from 0 to 6, and print each item in the sequence:

```
x = 2
assert x < 1, 'x is not less than 1'
```

The output is:

`AssertionError: x is not less than 1`

Because the condition here (`x<1`) is not met, so it raises error.

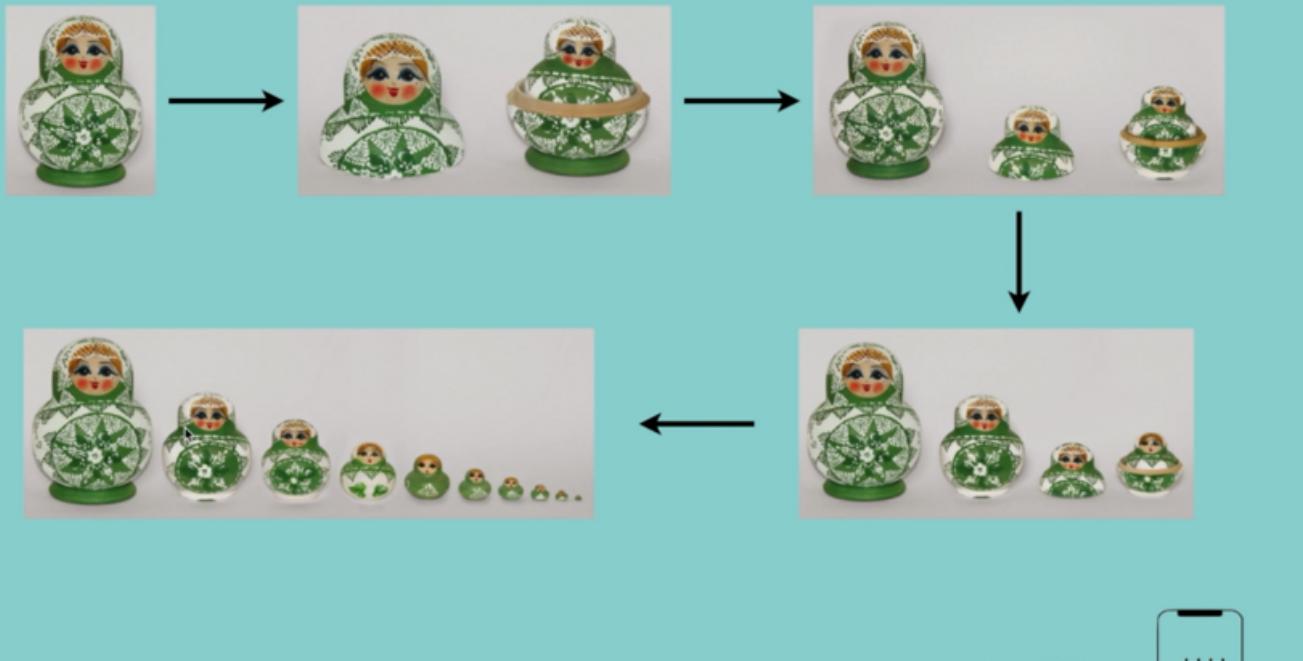
## What is Recursion?

11

Recursion = a way of solving a problem by having a function calling itself

## What is Recursion?

Recursion = a way of solving a problem by having a function calling itself



# What is Recursion?

12

Recursion = a way of solving a problem by having a function calling itself



```
def openRussianDoll(doll):  
    if doll == 1:  
        print("All dolls are opened")  
    else:  
        openRussianDoll(doll-1)
```

## Why Recursion?

13

1. Recursive thinking is really important in programming and it helps you break down big problems into smaller ones and easier to use
  - when to choose recursion?
    - ▶ If you can divide the problem into similar sub problems
    - ▶ Design an algorithm to compute nth...
    - ▶ Write code to list the n...
    - ▶ Implement a method to compute all.
    - ▶ Practice
2. The prominent usage of recursion in data structures like trees and graphs.
3. Interviews
4. It is used in many algorithms (divide and conquer, greedy and dynamic programming)

## How Recursion works?

1. A method calls it self
2. Exit from infinite loop

```
def recursionMethod(parameters):  
    if exit from condition satisfied:  
        return some value  
    else:  
        recursionMethod(modified parameters)
```

## How Recursion works?

14

```
def recursiveMethod(n):
    if n<1:
        print("n is less than 1")
    else:
        recursiveMethod(n-1)
        print(n)
```

recursiveMethod(4)  
└ recursiveMethod(3)  
  └ recursiveMethod(2)  
    └ recursiveMethod(1)  
      └ recursiveMethod(0)      n is less than 1

recursiveMethod(1)  
└ recursiveMethod(2)  
└ recursiveMethod(3)  
└ recursiveMethod(4)

STACK Memory

## How Recursion works?

```
def recursiveMethod(n):
    if n<1:
        print("n is less than 1")
    else:
        recursiveMethod(n-1)
        print(n)
```

recursiveMethod(4)  
└ recursiveMethod(3)  
  └ recursiveMethod(2)  
    └ recursiveMethod(1)  
      └ recursiveMethod(0)      n is less than 1

  4 ←  
  3 ←  
  2 ←  
  1 ←  
soura

STACK Memory

## Recursive vs Iterative Solutions

15

```
def powerOfTwo(n):
    if n == 0:
        return 1
    else:
        power = powerOfTwo(n-1) ↴
        return power * 2
```

```
def powerOfTwoIt(n):
    i = 0
    power = 1
    while i < n:
        power = power * 2
        i = i + 1
    return power
```

## Recursive vs Iterative Solutions

```
def powerOfTwo(n):
    if n == 0:
        return 1
    else:
        power = powerOfTwo(n-1)
        return power * 2
```

```
def powerOfTwoIt(n):
    i = 0
    power = 1
    while i < n:
        power = power * 2
        i = i + 1
    return power
```

Points	Recursion	Iteration	
Space efficient?	No	Yes	No stack memory require in case of iteration
Time efficient?	No	Yes	In case of recursion system needs more time for pop and push elements to stack memory which makes recursion less time efficient
Easy to code?	Yes ↴	No	We use recursion especially in the cases we know that a problem can be divided into similar sub problems.









































































































































































































































































































































































































































































































































































































































































































































































































































































































































































































