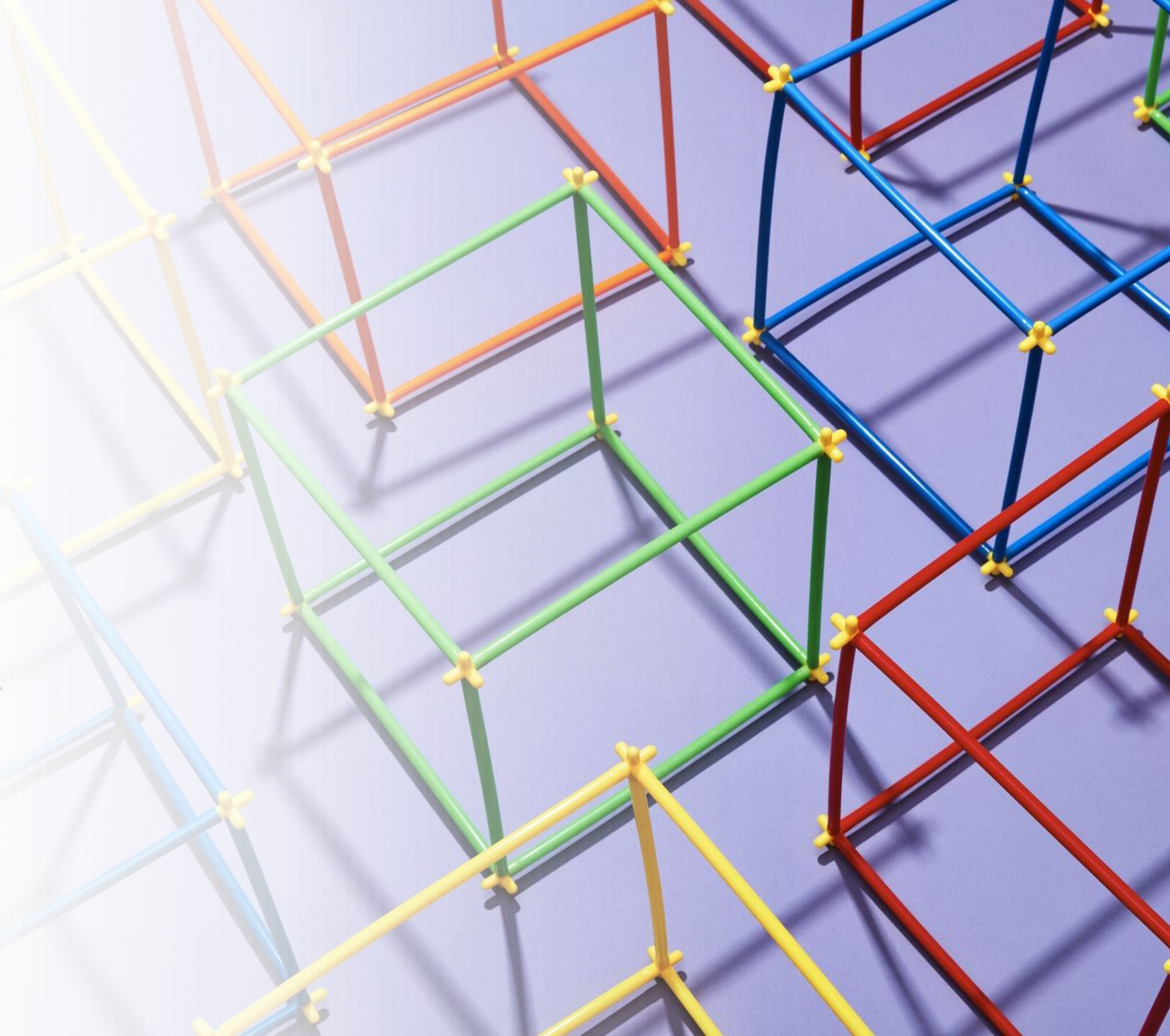
Data Structure Lecture 2: Algorithms and Complexity

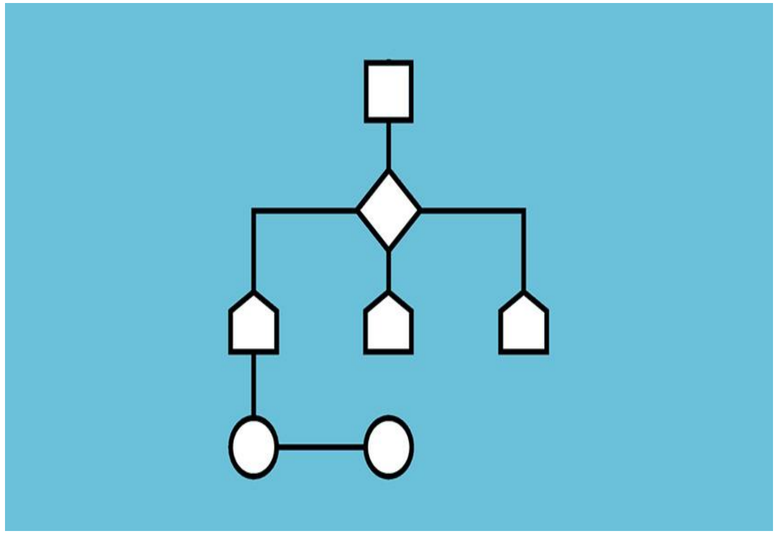


Prepared by

Dr. Mohammed Salah Al-Obiadi

What is an Algorithm?

- Steps of preparing a frying egg.
  1. Get the frying pan.
  2. Get the oil.
     a. Do we have oil?
        i. If yes, put it in the pan.
        ii. If no, do we want to buy oil?
           a. If yes, then go out and buy.
           b. If no, no egg today.
  3. Turn on the stove, etc...



An algorithm is the step-by-step clear instructions to solve a given problem.

Criteria for judging Algorithms

- There are two main criteria for judging Algorithms:
    1. Correctness: does the algorithm give solution to the problem in a finite number of steps?
    2. Efficiency: how much resources (in terms of memory and time) does it take to execute the program.

Complexity of an Algorithm

1. **Space Complexity of a program** is the amount of memory it needs to run to completion.
2. **Time complexity of a program** is the amount of computer time it needs to run to completion. The time complexity is of two types such as
   a) Compilation time
   b) Runtime

Asymptotic Notations: Big-O Notation

- **Big-O Notation [Upper Bounding Function]:** The O(g(n)) represents the upper bound computation a program can cause to the computer. f(n) = O(g(n)) (read as f of n is big oh of g of n)
- **Example-1** Find upper bound for:
  - $f(n) = 3n + 8$
    - solution $f(n) = O(n)$
  - $f(n) = n^2 + 1$
    - Solution $f(n) = O(n^2)$
  - $f(n) = 16n^3 + 45n^2 + 12n$
    - Solution $f(n) = O(max(n^3, n^2, n)) = O(n^3)$
  - $f(n) = n^4 + 100n^2 + 50$
    - Solution $f(n) = O(max(n^4, n^2)) = O(n^4)$
  - $f(n) = 410$
    - *Solution* $f(n) = O(1)$

Example a program of O(1):

- Problem: To find out the greater between two numbers

```
bool max_value (int a, int b) // function that accept two numbers
{
   if (a> b) // Compare the two numbers
         return true; // if first is greater return true
   else
         return false. // otherwise return false
}
```

- This function does not have any loop and will not cost the computer a lot of computations, so it's *f(n)=O(1)* means a constant computations.

Example a program of O(n):

- Problem: Program to search a number from a list of numbers

```
bool search (int arr [], int number, int n)
{
bool found=false;
for (int i=0; i<n; i++)
{
      if (arr[i] ==number)
         {
         found=true;
         break;
         } // end of if
} // end of for
return found;
} // end of function
```

- This function has a for loop that require *n* time implementations from the computer, so it's $f(n)=O(n)$.

Example a program of O(n^2):

Problem: Write a program to sort the series of numbers using Bubble sort

```
void array (int arr [], int n)
{
int i, j;
for (i=0; i<n; i++) // start of outer loop
{
    for (j=1; j<n-i; j++) // inner loop
    {
        if (arr [j+1] > arr[j]) // comparing the elements
            { // swapping if the adjacent is larger
            temp=arr [j+1];
            arr [j+1] =arr[j];
            arr[j] =temp;
            } // end of if
    } //end of inner for loop
} // end of outer for loop
```

- This function has two *for loops* that require *n* x *n* time implementations from the computer, so it's $f(n)=O(n \times n)=O(n^2)$.

Time Compexity Chart