MINOS
LABS

Data Structures

MINOS
LABS

Complexities

Intro

- Data structures are what you would expect them to be: different ways of organizing and storing data
- These structures are extremely important in Computer Science, both the conceptual aspect and its implementation
- We use Data structures to manipulate data, and the structures used are tailored to the situation

Complexity

- Complexity is a measure of the efficiency of an algorithm, whether it be time efficiency or space efficiency
- Expressed in **Big O** notation
- The Big O of an algorithm represents the limiting case or worst case scenario of an executed function

Time Complexity

- A measure of the efficiency of an algorithm
- Generally represents the amount of time or number of basic operations it takes to execute an algorithm as a function of the size of the input, n
- This is usually expressed in Big O notation, since Big O represents the worst case scenario i.e. “limiting behavior”

MINOS LABS

Time Complexity Example

- Suppose an algorithm adds 10 to the first element of any array. This would be O(1) since it doesn't depend on the size of the array
- On the contrary if I add 10 to every element this would be O(n) as it would depend on the array size

MINOS LABS

Time Complexity Example

```java
public static void main(String[] args){
    int[] arr = new int[100];

    //Single for loop -> O(n)
    for(int i=0;i<arr.length;i++){
        arr[i] = i; //O(1)
    }

    //Nested for loop -> O(n)*O(n) = O(n^2)
    for(int i=0;i<arr.length;i++){
        for(int j=0;j<arr.length;j++){
            arr[i]++; //O(1)
        }
    }

    //Traverse to print -> O(n)
    for(int i=0;i<arr.length;i++){
        System.out.println("arr["+i+"] = "+ arr[i]);
    }
}
```

Space Complexity

- Another important term, space complexity, is used to refer to the amount of space in memory required to execute a particular program.
- This requirement depends on the size of the data structure.
- Algorithms often trade time for space or vice versa depending on the situation.

Space Complexity Example

- Suppose we want to find the total product of all elements of an array of size n, containing integers
- Since integers have 4 bytes each, we would need 4*n bytes of space to store the array and some extra bytes for extra variables
- Space complexity would therefore be O(n), i.e. space required increases linearly with the array size n

Space Complexity Example

```java
public class spacecomp {
    Run | Debug
    public static void main(String[] args){
        int[] arr = {1,3,4,5,8};
        int num = 1;

        //4*n bytes to store array, 8 bytes to store num and
        //i variables so 4n+8 = O(n)
        for(int i=0;i<arr.length;i++){
            num*=arr[i];
        }

        System.out.println(num);


    }
}
```

Different time complexities

- **Constant, O(1):** Algorithm doesn’t depend on the size of the input
  - Ex: accessing any single element in an array b/c only one operation has to be performed to locate it
- **Linear, O(n):** algorithm completion time increases linearly with input size
  - Common complexity for algorithms with single loop
  - Examples: Find a given element in an array or Print all values in a list

MINOS LABS

Different Time Complexities (cont.)

- **Quadratic O(n^2):** has a growth rate of n^2. If the input size is 4, it will do c*16 operations where c is a constant
  - Common complexity for an algorithm with a loop within a loop (nested loop), ex. for loop within another for loop
  - Examples:
    - Check if a collection has duplicated values
    - Sorting items (bubble sort, insertion sort, or selection sort)
    - Find all ordered pairs in an array

MINOS LABS

Different Time Complexities (cont.)

- **Polynomial, O(n^c):** when c > 1, it is considered polynomial and we want to stay away from polynomial running times
  - If there are three nested for loops, the runtime will be O of n cubed O(n^3)
- **Logarithmic, O(Log n):** Logarithmic time complexities usually apply to algorithms that divide in half every time
  - For example, in a ***binary search***, we want to find the index of an element in an ascending sorted array, we continuously divide our pool of data in half until we find our target.

Different Time Complexities (cont.)

- **Linearithmic, O(n log n):** This is slightly slower than a linear O(n) algorithm but faster than a quadratic O(n^2) algorithm
  - Examples: efficient sorting algorithms like merge sort, quicksort and others

Order of complexities (best to worst):

$O(\log n) < O(n) < O(n \log n) < O(n^c) < O(c^n) < O(n!)$

Additional Resources

- Download Java
  - https://adoptopenjdk.net/
- Big O notation and cheat sheet
  - http://web.mit.edu/16.070/www/lecture/big_o.pdf
  - https://www.bigocheatsheet.com/
- Overall Java help
  - https://www.codecademy.com/learn/learn-java