While folks are joining

Get you laptops ready and login to www.crio.do.
We will be coding away in the session!

DSA-1

Session 2

What’s for this session?

- Introduction to Time and Space Complexity
    - What is Time Complexity?
    - What is Space Complexity?
    - Tradeoff between the two
    - Activities
- Matrix
    - What and Why?
    - Indices
    - Traversals
    - Solve Problems

Time Complexity Examples

1. You have ‘n’ books and you go through each of them to find the one you’re looking for
   - O(n)
   - Does this definition change if `n` changes?

2. ```
   int Sum(int a, int b) {
       return a+b;
   }
   ```
   - 2 units of time(constant). One for the arithmetic operation and one for the return - O(1)

3. ```
   void main() {
       int n = 100;
       for (int i = 1; i <= n; i++) {
           printf("Hello World!\n");
       }
   }
   ```
   - Loop executes n number of times, thus O(n)

What is Time Complexity?

- Compare Algorithms/DSA Solutions. Which algorithm is faster?
- How to measure its speed?
  - Time taken in seconds?
    - But that would change depending on the input data set size. Also on the machine on which its being run.
  - Number of statements executed?
    - This would depend on the language and style of the code.
  - How to make it independent of these?
- **Time Complexity** of an algorithm is the time taken for it to complete its operation as a function of its data input size, n.
  - Standards - Big O notation (others - Big Theta, Big Omega)
  - Examples - **O(1), O(n), O(n^2), O(n logn)**

Big O Notation

Way to measure how well our algorithm scales as the amount of data increases

- Example: Input set of 10 elements compared to input set of million elements
- Example: What has the biggest effect on the answer in this equation? → **$2*n^3 + 5*n^2 + 19$**
    - When n = 2, the answer is 2*8 + 5*4 + 19 = 55
    - When n = 3, the answer is 2*27 + 5*9 + 19 = 118
    - When n = 10, the answer is 2*1000 + 5*100 + 19 = 2519
    - 19 is insignificant as n increases
    - In fact, n^2 is also dwarfed by n^3 as n increases
    - **n^3** is the main contributor (even the constant, 2, doesn't contribute much)
    - **Hence, the complexity of this equation is O(n^3)**

What is Space Complexity?

- Algorithm also uses memory to store data for its operations
- **Which algorithm takes less space** (desired)?
- How would you measure their memory usage?
  - Compare memory used by different algorithms, for same input?
    - Not possible to test for all inputs. Also, this depends on compiler, language, machine etc.
  - How to make it generic across all inputs?
- Space Complexity of an algorithm is the amount of memory needed for its operation as a function of its data input size, n.
  - Standards - Big O notation
  - This space includes the inputs as well as additional space used by variables and DS

Space Complexity Examples

1.
```
int Sum(int a, int b) {
    return a+b;
}
```

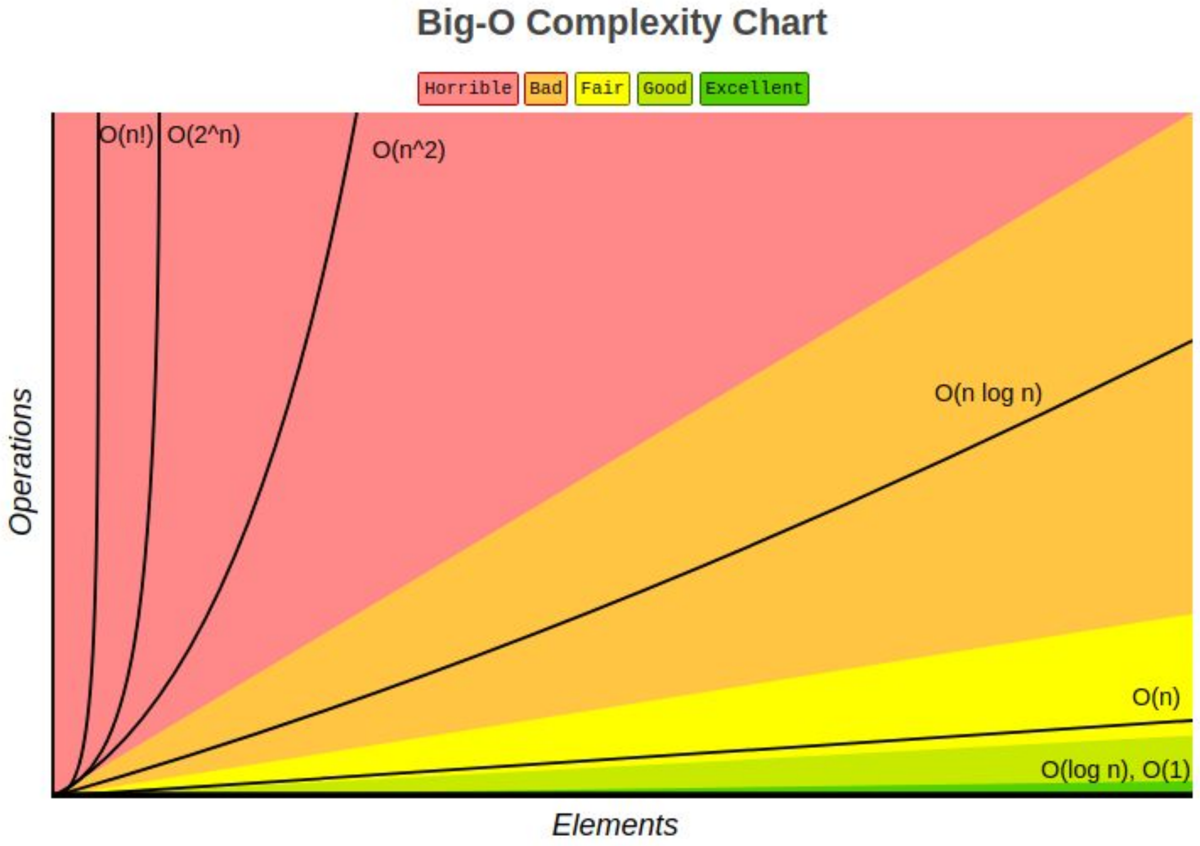
- Three integers used here - a, b and the result. But this is constant - O(1).

2.
```
public int sumArray(int[] array) {
    int size = size of input array;
    int sum = 0;
    for (int iterator = 0; iterator < size; iterator++) {
        sum += array[iterator];
    }
    return sum;
}
```

- *array* – the argument – space taken is equal to 4*n* bytes, *n* is length of array
- *size* – a 4-byte integer, *sum* – a 4-byte integer, *iterator* – a 4-byte integer
- Total space needed is 4*n* + 4 + 4 + 4 (bytes). The highest order in this equation is *n*.
- Thus, space complexity is *O(n)*.

Visualisation of Common Big O Complexities

- Applies to both Space and Time Complexities
- Think very large n, to realize why O(1) is so much better than O(n) or why O(n) is so much better than O(n^2), etc.
- Can you arrange these in increasing order?



Trade off between Time and Space

- Algorithms use **Memory** and need **Time** to complete
- In most cases, we can have trade off between Space and Time i.e. we can solve a problem
    - Either in less time by using more memory
    - Or using less memory but spending more time

The choice depends on the constraints of a problems.

Example 1 - Merge Sort algorithm is exceedingly fast but requires a lot of space to do the operations. At the other end, Bubble Sort is exceedingly slow but requires the least space.

Example 2 - We can store already calculated results in some recursion problems, instead of calculating them multiple times. (Example: Fibonacci problem)

Activity #1 What's the Time/Space complexity here?

```
public void searchForValue(int valueToFind, int[] arr, int sizeOfArray) {
        for (int it = 0; it < sizeOfArray ; it++) {
            if (arr[it] == valueToFind) {
                System.out.println("Success");
                break;
            }
        }
        if (it == sizeOfArray)
            System.out.println("Failure");
        return;
}
```

Try it out - https://onlinegdb.com/HyKRBpQdd

Time Complexity - O(n) - Linear Search, grows directly in proportion to the input data size

Space Complexity - O(1) - No size specific data structure used

Other examples of O(n)

- Get the max/min value in an array.
- Find a given element in a collection. (What happens if this is sorted?)
- Print all the values in a list.
- Every time a list or array gets iterated over, it is most likely in O(n) time.

Activity #2 What's the Time/Space complexity here?

```
for(int i = 0; i < n; i++){
    for(int j = 0; j < m; j++){
       Matrix[i][j] = i+j;
  }
}
```

Time Complexity - O(m*n) - Running 2 loops but each one iterates to a different length

Space Complexity - O(1) - constant number of variables

Activity #3 What's the Time/Space complexity here?

```
for (int i = 1; i < n; i = i * 2){
    System.out.println("I just processed: " + i);
}

function indexOf(array, element, offset = 0) {
 // split array in half
 const half = parseInt(array.length / 2);
 const current = array[half];
 if(current == element) {
   return offset + half;
 } else if(element > current) {
   const right = array.slice(half);
   return indexOf(right, element, offset + half);
 } else {
   const left = array.slice(0, half)
   return indexOf(left, element, offset);
 }
}
```

Time Complexity - O(log n) - Binary Search. As n increases, increase in (log n) is quite slow. Base is 2 here.

Space Complexity - O(log n) - Function call stack due to recursion

Other examples

- Traversing Balanced Binary Search Tree
- Problems where the number of elements in the problem space gets halved each time, it will most probably be in O(logn) runtime.

What about this?

```
for(int i = 1; i <= n; i++){
        i = i * k;
}
```

- O(log n (base k))

Matrix - Why do we need to know this?

- Matrix related problems common in DSA
- Some examples
    - You are standing on a square matrix, there are some blockages in some of the cells, you cannot enter them. Is there a way to reach the end of the matrix starting from the first cell?

What is a Matrix?

- What is a matrix?
- Different dimensions of matrix?
- How is Matrix represented?
  - 2D Array

[5]
[6]
[7]
[8] -> 4 rows, 1 column

[5 6 7 8] -> 1 row, 4 columns

[4] -> 1 row, 1 column

[5 6 7]
[1 3 4]
[5 8 1] -> 3 rows, 3 columns, Square matrix

[5 6]
[1 3]
[5 8] -> rows != columns, Rectangular matrix

Declaration and Indexing for a matrix

Example:

arr[3][2] = [5 6]
[1 3]
[5 8]

- To access a particular value in the matrix, we use arr[i][j], where ‘i’ is row index and ‘j’ is column index

- What is the value for arr[0][0]?
- What is the value for arr[3][2]?
- What is the value for arr[2][3]?

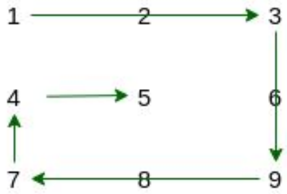
Traversals - order of visiting all cells

- Problem: Find a given element in the matrix
  - Solution: Traverse each row, one at a time to find the element
  - How to increment indices, do we need loops, how many?
  - How to keep the index from going out of bounds - beyond the matrix boundary?

- Traverse each column, one at a time

- Diagonal traversal
  - What is a diagonal? How many diagonals does a matrix have? What's a forward/backward diagonal?
  - Pseudocode for Forward Diagonal Traversal

- Spiral Traversal



Activity 1 - Check if matrix is a magic square

Questions?

Traversal pseudocode

Take home exercises

- Diagonal sum in a matrix
- Addition of two matrices

To be solved before the next session on Saturday, 11:00 AM

Feedback

Thank you for joining in today.

We’d love to hear your thoughts and feedback - https://bit.ly/dsa-nps

Thank you