**Space Complexity**

**Space Complexity**

* The ***Space Complexity***of an algorithm is a measure of the amount of storage an algorithm needs with respect to the input size.
* That means how much memory, in the worst case, is needed at any point in the algorithm.
* Space complexity is a parallel concept to time complexity.
* We're mostly concerned with how the space needs grow, in big-Oh terms, as the size N of the input problem grows.
* If we need to create an array of size n, this will require O(n) space.
* If we create a two-dimensional array of size n\*n, this will require O(n2) space.

**Example: O(1) Space**

**int sum(int x, int y, int z) {**

**int r = x + y + z;**

**return r;**

**}**

* This algorithm requires 3 units of space for the parameters and 1 for the local variable, and this never changes, so this is O(1).

**Example: O(N) Space**

**public int[] copy(int c1[]) {**

**int[] c2 = new int[c1.length];**

**for (int i = 0; i < n; ++i) {**

**c2[i] = c1[i];**

**}**

**return c2;**

**}**

* This algorithm requires N extra units of space for the new array c2.

**Recursion and Space**

* Stack space in recursive calls also account for space memory.
* Consider the following recursive function:

int sum(int n)

{

if (n <= 0) {

return 0;

}

return n + sum(n-1);

}

* Each call adds a level to the stack.

sum(4)

🡪 sum(3)

🡪 sum(2)

🡪 sum(1)

🡪 sum(0)

* Each of these calls is added to the call stack and takes up actual memory.
* However, if you have a non-recursive function, then making O(n) calls to a function does not take O(n) space.
* This is because with recursion, there can be up to n frames existing on the call stack simultaneously.
* What happens if a recursive function calls itself recursively N times?
* Then its space can't be reused because every call is still in progress, so it needs O(N2) units of space.
* But be careful here. If things are passed by pointer or reference, then space is shared.