# Space complexity

Space complexity is a measure of the amount of working storage an algorithm needs. That means how much memory, in the worst case, is needed at any point in the algorithm. As with 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**.

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

int r = x + y + z;

return r;

}

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

int sum(int a[], int n) {

int r = 0;

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

r += a[i];

}

return r;

}

requires **N units for a, plus space for n, r and i, so it's O(N).** What are the space complexities of these next two functions?

void matrixAdd(int a[], int b[], int c[], int n) {

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

c[i] = a[i] + b[j]

}

}

void matrixMultiply(int a[], int b[], int c[][], int n) { // not legal C++

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

for (int j = 0; j < n; ++j) {

c[i] = a[i] + b[j];

}

}

}

If a function A uses M units of its own space (local variables and parameters), and it calls a function B that needs N units of local space, then A overall needs M + N units of temporary workspace.

**What if A calls B 3 times**? When a function finishes, its space can be reused, so if A calls B 3 times, it still only needs M + N units of workspace.

What if A calls itself recursively N times? Then its space can't be reused because every call is still in progress, so it needs O(MN) units of workspace.

But be careful here. If things are passed by pointer or reference, then space is shared. If A passes a C-style array to B, there is no new space allocated.