Garbage Collection

A **garbage collector** automatically frees up memory that a program isn't using anymore.

For example, say we did this in C#:

C# (beta)

```
public int GetMin(int[] nums)
{
    // NOTE: this is NOT the fastest way to get the min!
    int[] numsSorted = nums;
    Array.Sort(numsSorted);
    return numsSorted[0];
}

int[] myNums = { 5, 3, 1, 4, 6 };
Console.WriteLine(GetMin(myNums));
```

Look at `numsSorted` in `getMin()`. We allocate that whole array inside our function, and once the function returns we don't need the array anymore. In fact, once the function returns we *don't have any references to it anymore*!

What happens to that array in memory? The C# garbage collector will notice we don't need it anymore and free up that space.

> How does the garbage collector know when something can be freed?
>
> One option is to start by figuring out what we *can't* free. For example, we definitely can't free local variables that we're going to need later on. And, if we have an array, then we also shouldn't free any of the array's elements.
>
> This is the main intuition behind one garbage collector strategy:
>
> 1. Carefully figure out what things in memory we might still be using or need later on.

> 2. Free everything else.
>
> This strategy is often called **Tracing Garbage Collection**, since we usually implement the first step by tracing references from one object (say, the array) to the next (an element within the array).
>
> A different option is to have each object keep track of the number of things that reference it—like a variable holding the location of an array or multiple edges pointing to the same node in a graph. We'll call this number an object's **reference count**.
>
> In this case, a garbage collector can free anything with a reference count of zero.
>
> This strategy is called **Reference Counting**, since we are *counting* the number of times each object is *referenced*.
>
> In languages with reference counting, make sure you know who is responsible for maintaining the reference counts for your objects. Some languages, like C++, take care of incrementing and decrementing the count for you automatically, but others, like Objective-C, force you to manually update it.

Some languages, like C, don't have a garbage collector. So we need to manually free up any memory we're not using anymore:

```c
// make a string that can hold 15 characters
// including the terminating null byte ('\0')
str = malloc(15);

// ... do some stuff with it ...

// we're done. free that memory!
free(str);
```

We sometimes call this **manual memory management**.

Some languages have both manual and automatic memory management. In C#, most resource management is done automatically by the garbage collector, but we can also implement the `IDisposable` interface to clean up resources that the garbage collector might hold on to longer than we need.

What's next?

If you're ready to start applying these concepts to some problems, check out our mock coding interview questions (/next).

They mimic a real interview by offering hints when you're stuck or you're missing an optimization.

Try some questions now ➜
