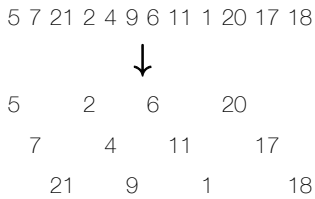
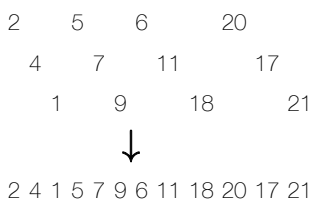
Shell Sort, Reading

In the early 1970's, an American computer scientist named Donald Shell thought about what we just saw in the previous reading. He thought about the big oh effect of starting with a nearly-sorted array when applying the insertion sort algorithm. Of course, in general there's no way we could ever count on such a fortunate circumstance. But what if there was a way to rearrange the array so that it was *kind of* in order -- not perfect and not costing so much (in runtime) that it defeats the purpose? Then the final step would be to run a standard insertion sort, at about O(n). How much worse than O(n) would the preparation step make this?

Shell's idea went something like this: given an array of size 12 to sort, break them into 3 separate arrays like this:



Then sort the three, and collapse them back to the full array:



It's not perfect, but it's pretty close! Let's see what it "cost":

Each short array's sorting takes $^1/_9$ the runtime of the whole array, because the size is $^1/_3$ and it's n^2. There are 3 of then to do, so that's $^3/_9$, or $^1/_3$ of the runtime to get it to *kind of* in order vs the full runtime to get it fully in order. If the extra step of one last insertion sort on the whole array is as efficient are we hope, then this has promise.

The actual solution has more than just these two steps -- it has several different first steps with fewer and longer separate arrays, going through a series of separating and collapsing. The separating is really not hard -- all the "separated" arrays are in the same actual array -- they just start at different indexes and traverse in jumps that are greater than one.

```
// delta is the separation between elements of the separated arrays
for (int delta = n / 3;; delta = delta / 3 + 1) // the last cycle is a pure insertion sort
{
  for (int offset = 0; offset < delta; offset++) // for each of the separated arrays
  {
    // standard insertion sort with "delta" distance between compared values' indexes
    for (int i = offset + delta; i < n; i += delta) // bring in each new value one-at-a-time
    {
      for (int j = i - delta; j >= 0; j -= delta) // traverse towards the front
      {
        if (a[j] < a[j + delta]) break;
        swap(a[j], a[j + delta]);
      }
    }
  }
  if (delta == 1) break;
}
```

There are different "flavors" of the Shell sort, depending on how many separation-collapse cycles there are. In the above, for n=100, we start with 33

There are different "flavors" of the Shell sort, depending on how many separation-collapse cycles there are. In the above, for n=100, we start with 33 separate arrays of size 3 (or 4) and sort each. In the next cycle, we have 9 separate arrays of size 11 (or 12). Then 4 arrays of 25, then 2 arrays of 50, until finally just one array with all 100. By then the values should be very close to being sorted.

At some point from the first cycle to the last we transition from $O(n^2)$ to O(n). It's difficult to calculate analytically, but empirical studies show that the big oh for Shell is about:

$$O(n^{1.25})$$