This is nice in that it doesn't use much extra memory, simulating uniq(1)'s behavior of removing only adjacent duplicates. The ", 1" guarantees that the expression is true (so that grep picks it up) even if the $_ is 0, "", or undef.

b)

If you don't know whether @in is sorted:

```
undef %saw;
@out = grep(!$saw{$_}++, @in);
```

c)

Like (b), but @in contains only small integers:

```
@out = grep(!$saw[$_]++, @in);
```

d)

A way to do (b) without any loops or greps:

```
undef %saw;
@saw{@in} = ();
@out = sort keys %saw;  # remove sort if undesired
```

e)

Like (d), but @in contains only small positive integers:

```
undef @ary;
@ary[@in] = @in;
@out = grep {defined} @ary;
```

But perhaps you should have been using a hash all along, eh?

20.5.4 How can I tell whether a certain element is contained in a list or array?

Hearing the word "in" is an *in*dication that you probably should have used a hash, not a list or array, to store your data. Hashes are designed to answer this question quickly and efficiently. Arrays aren't.

That being said, there are several ways to approach this. If you are going to make this query many times over arbitrary string values, the fastest way is probably to invert the original array and maintain a hash whose keys are the first array's values.

```
@blues = qw/azure cerulean teal turquoise lapis-lazuli/;
%is_blue = ();
for (@blues) { $is_blue{$_} = 1 }
```

Now you can check whether $is_blue{$some_color}. It might have been a good idea to keep the blues all in a hash in the first place.

If the values are all small integers, you could use a simple indexed array. This kind of an array will take up less space:

```
@primes = (2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31);
@is_tiny_prime = ();
for (@primes) { $is_tiny_prime[$_] = 1 }
# or simply  @istiny_prime[@primes] = (1) x @primes;
```

Now you check whether $is_tiny_prime[$some_number].

If the values in question are integers instead of strings, you can save quite a lot of space by using bit strings instead: