```
@articles = ( 1..10, 150..2000, 2017 );
undef $read;
for (@articles) { vec($read,$_,1) = 1 }
```

Now check whether `vec($read,$n,1)` is true for some `$n`.

Please do not use

```
($is_there) = grep $_ eq $whatever, @array;
```

or worse yet

```
($is_there) = grep /$whatever/, @array;
```

These are slow (checks every element even if the first matches), inefficient (same reason), and potentially buggy (what if there are regex characters in $whatever?). If you're only testing once, then use:

```
$is_there = 0;
foreach $elt (@array) {
    if ($elt eq $elt_to_find) {
        $is_there = 1;
        last;
    }
}
if ($is_there) { ... }
```

20.5.5 How do I compute the difference of two arrays? How do I compute the intersection of two arrays?

Use a hash. Here's code to do both and more. It assumes that each element is unique in a given array:

```
@union = @intersection = @difference = ();
%count = ();
foreach $element (@array1, @array2) { $count{$element}++ }
foreach $element (keys %count) {
    push @union, $element;
    push @{ $count{$element} > 1 ? \@intersection : \@difference }, $element;
}
```

Note that this is the *symmetric difference*, that is, all elements in either A or in B but not in both. Think of it as an xor operation.

20.5.6 How do I test whether two arrays or hashes are equal?

The following code works for single-level arrays. It uses a stringwise comparison, and does not distinguish defined versus undefined empty strings. Modify if you have other needs.

```
$are_equal = compare_arrays(\@frogs, \@toads);

sub compare_arrays {
    my ($first, $second) = @_;
    no warnings;  # silence spurious -w undef complaints
    return 0 unless @$first == @$second;
    for (my $i = 0; $i < @$first; $i++) {
        return 0 if $first->[$i] ne $second->[$i];
    }
    return 1;
}
```