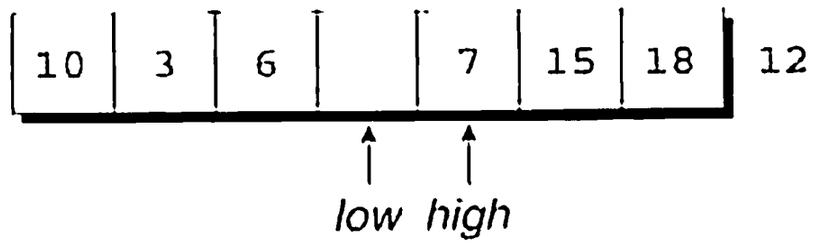
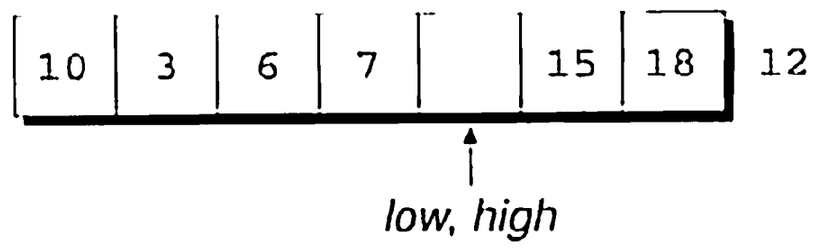
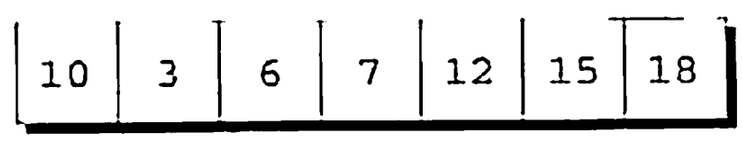
high points to 15, which is greater than 12 and thus doesn't need to be moved. We shift *high* to the left and continue.



high points to 7, which is out of position. After moving 7 to the hole, we shift *low* to the right.



low and *high* are now equal, so we move the partitioning element to the hole.



At this point, we've accomplished our objective: all elements to the left of the partitioning element are less than or equal to 12, and all elements to the right are greater than or equal to 12. Now that the array has been partitioned, we can use Quicksort recursively to sort the first four elements of the array (10, 3, 6, and 7) and the last two (15 and 18).

PROGRAM Quicksort

Let's develop a recursive function named `quicksort` that uses the Quicksort algorithm to sort an array of integers. To test the function, we'll have `main` read 10 numbers into an array, call `quicksort` to sort the array, then print the elements in the array:

```
Enter 10 numbers to be sorted: 9 16 47 82 4 66 12 3 25 51
In sorted order: 3 4 9 12 16 25 47 51 66 82
```

Since the code for partitioning the array is a bit lengthy, I'll put it in a separate function named `split`.

qsort.c

```
/* Sorts an array of integers using Quicksort algorithm */

#include <stdio.h>

#define N 10

void quicksort(int a[], int low, int high);
int split(int a[], int low, int high);

int main(void)
{
  int a[N], i;

  printf("Enter %d numbers to be sorted: ", N);
  for (i = 0; i < N; i++)
    scanf("%d", &a[i]);
```