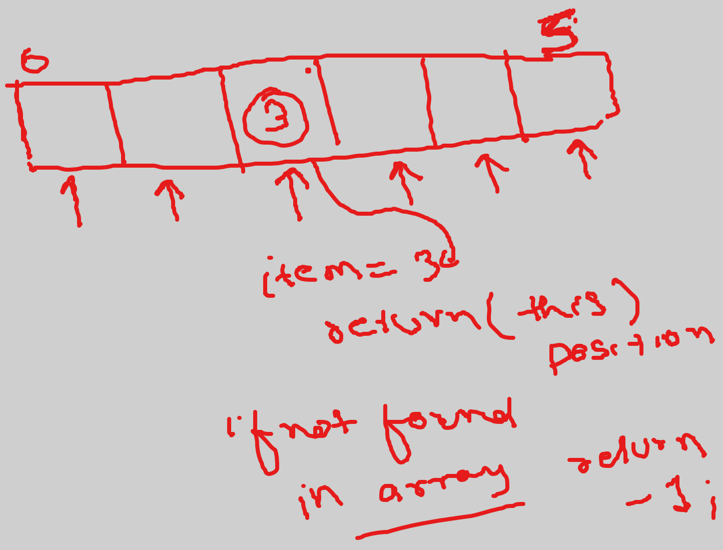
SEARCHING AND SORTING IN C PROGRAMMING

Searching and sorting through arrays is one of the most labor-intensive tasks. There are two different approaches to searching through arrays: linear or sequential search, and binary search.

In a linear search, each element of the array is checked until a match is found. Below is an example of a function that searches an array for a specific item, and returns its location if the item is found or returns -1 if it was not found.

```
int search (int aray[], int size, int item)
{
  int cnt = 0;
    while (cnt < size)
    {
    if (item ==  array[cnt])
      return (cnt);
    cnt++;
    }
    return (-1);
}
```



In a binary search, the data in the array is ordered and then the middle of the contracted array is tested until the required match is found. Next is an example of a binary search:

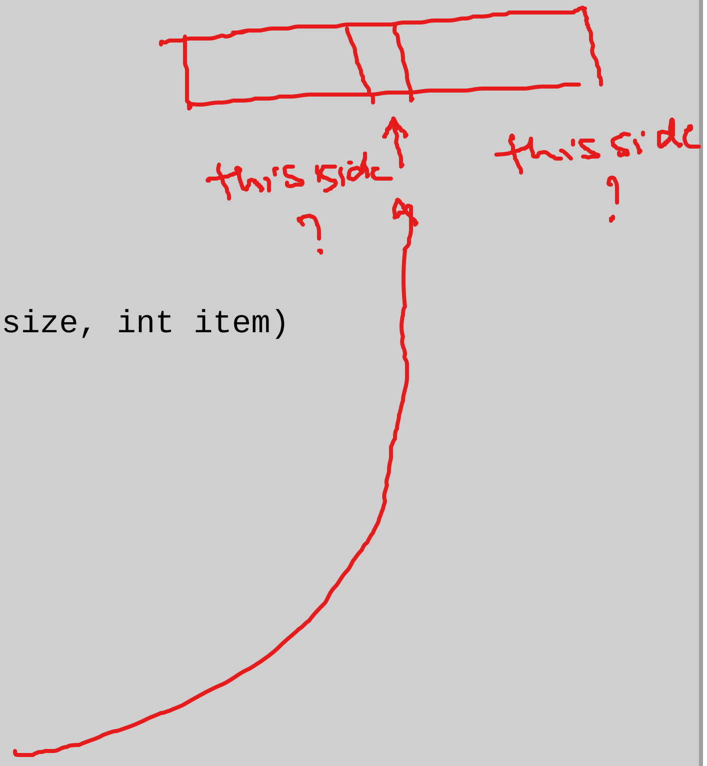
```
Assume that the value passed to this function
```

```
are:
    int array = [50,8,20,3,40];
    int item = 40;
    int size = 5;


int BinSearch (int array[], int size, int item)
{
  int low = 0, count = 0;
  int high = size - 1;
  int middle;
    while (low <= high)
    {
      count++;
      middle = (high + low) / 2;
      if (item == array[middle])
        return middle;
      else if (item < array[middle])
        high = middle - 1;
      else
        low = middle + 1;
    }
  return -1;
}



What is the value of count when searching for:
a)    50           R: 3
b)    8            R: 2
c)    20           R: 1
```



```
d)    3           R: 2
e)    40          R: 3
```

Sorting data

There are three approaches to sorting arrays: selection sort, insertion sort, and bubble sort. As you will notice, whereas searching involves a single *for* loop and visiting each array location, sorting involves nested *for* loops, and n-1 passes through the array.

In a **selection sort**, we start with the first position in the array, find the smallest value in a first pass through the array, and swap the value in the first position in the array with the value in the position of the newly found smallest value; then we take each subsequent position in the array, find the smallest value in the remaining array, and swap position. Here is an example of a selection sort:

```
void selectionSort (int a[], int size)
{
  int temp, pass, k, min, minIndex;
    for (pass = 0; pass < size - 1; pass++)
    {
      min = a[pass];
      minIndex = pass;
      for (k = pass + 1; k < size; k++)
      {
        if (a[k] < min)
        {
          min = a[k];
          minIndex = k;
        }
      }
      temp = a[pass];
      a[pass] = a[minIndex];
      a[minIndex] = temp;
    }
    return;
}
```

In an **insertion sort**, you start with one item, take a new item and sort the two items relative to each other, then take a new item and sort the three items relative to each other (swapping the new item with consecutive values until it is no longer lower, and thus inserting it in that position), and so on. It is like sorting a deck of cards with your hands.Below is an example:

```
void insertionSort (int a[], int size)
{
  int temp, pass, k;
    for (pass = 1; pass <= size - 1; pass++)
    {
      for ( k = pass; k >= 1; k--)
      {
        if (a[k] < a[k-1])
        {
          temp = a[k];
          a[k] = a[k-1];
          a[k-1] = temp;
        }
      }
    }
    return;
}
```

In a **bubble sort**, you swap neighbors; the larger items drop down while the smaller ones bubble up, in n-1 passes through the array. Here's an example:

```
void bubbleSort (int a[], int size)
{
int temp, k, pass;
  for (pass = 1; pass <= size - 1; pass++)
  {
    for (k = 0; k < size - pass ; k++)
    {
      if (a[k] > a[k+1])
      {
        temp = a[k];
        a[k] = a[k+1];
        a[k+1] = temp;
      }
    }
  }
  return;
}
```