Lecture 14.5

Topics:

1. Array Based Sorting – quickSort

---

1. Array Based Sorting – `quickSort`

Example 1 – quickSort

The general algorithm can be summarized as follows,

```
if ( array size is greater than 1 )
{
     1. Partition the array into L and R arrays.
     2. quickSort L.
     3. quickSort R.
     4. Combine the sorted L and R arrays.
}
```

In the **`quickSort`** algorithm, the array is partitioned in such a way that combining the sorted **L** and **R** arrays is very easy done (or trivial).

To partition the list into two sublists, an element is chosen and stored in `pivotValue`. The elements in the array are arranged so that the elements in **L** are smaller than the `pivotValue`, and the elements in **R** are greater than `pivotValue`.

There are different ways to select the `pivotValue`. The given example will take the middle element as `pivotValue`. Thus, if **L** and **R** were sorted then reassemble **L**, `pivotValue` and **R** should be trivial.

The detailed discussion of the partition procedure is described as follows,

1. Determine the `pivotValue` and swap the `pivotValue` with the first element of the array.
2. The index `pivotIndex` is initialized to the first element of the array.
3. For the remaining elements in the array (starting at the second element),

   If the current element is smaller than the `pivotValue` then

   a. Increment `pivotIndex`.

   b. Swap the current element with the array element pointed to by `pivotIndex`.
4. Swap the first element, that is `pivotValue`, with the array element pointed to by `pivotIndex`.

Let's apply the previous partition procedure to the array given below.

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
array	40	81	27	93	55	62	74	36	91

Step 1

```
pivotValue = 55;
swap(array[0], array[4]);
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
55	81	27	93	40	62	74	36	91

Step 2

```
pivotIndex = 0;
```

Step 3 (Setting up a for-loop with loop control variable **`scan`**)

`scan = 1;` (Starting loop #1)

55	0	1
pivotValue	pivotIndex	scan

Because `array[scan] > pivotValue`, the **if** condition fails. Thus, next iteration will start.

`scan = 2;` (Starting loop #2)

55	0	2
pivotValue	pivotIndex	scan

55	81	27	93	40	62	74	36	91

Before second iteration

Because `array[scan] = 25 < pivotValue = 55`, the **if** condition is true. Increment `pivotIndex` so that `pivotIndex = 1`. Next `swap(array[pivotIndex], array[scan]);` that means `swap(array[1], array[2]).`

55	1	2
pivotValue	pivotIndex	scan

55	27	81	93	40	62	74	36	91

After second iteration

`scan = 3;` (Starting loop #3)

55	1	3
pivotValue	pivotIndex	scan

55	27	81	93	40	62	74	36	91

Before third iteration

Because `array[scan] = 93 > pivotValue = 55`, the **if** condition fails. The next iteration starts.

`scan = 4;` (Starting loop #4)

55	1	4
pivotValue	pivotIndex	scan

55	27	81	93	40	62	74	36	91

Before fourth iteration

Because `array[scan] = 40 < pivotValue = 55`, the **`if`** condition is true. Increment `pivotIndex` so that `pivotIndex = 2`. Next, `swap(array[pivotIndex], array[scan]),` that means `swap(array[2], array[4]).`

55	2	4
pivotValue	pivotIndex	scan

55	27	40	93	81	62	74	36	91

After fourth iteration

`scan = 5;` (Starting loop #5)

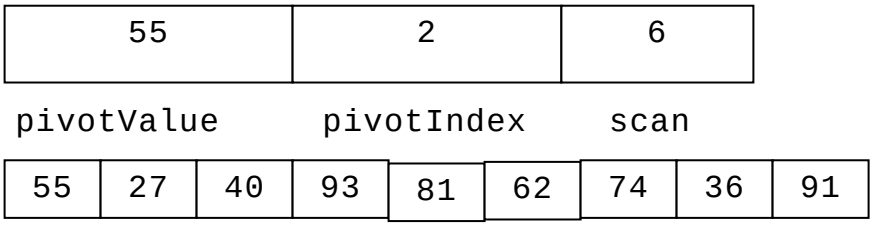
55	2	5
pivotValue	pivotIndex	scan

55	27	40	93	81	62	74	36	91

Before fifth iteration

Because `array[scan] = 62 > pivotValue = 55`, the **if** condition fails. The next iteration starts.

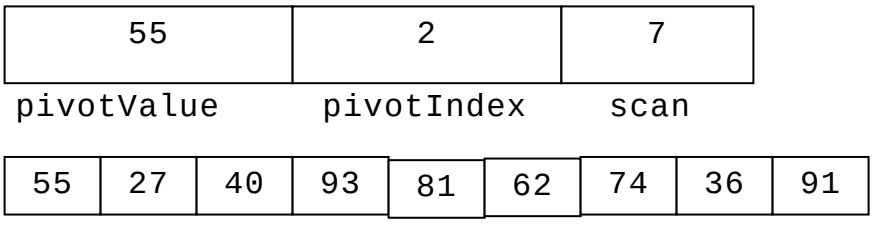
`scan = 6`; (Starting loop #6)



Before sixth iteration

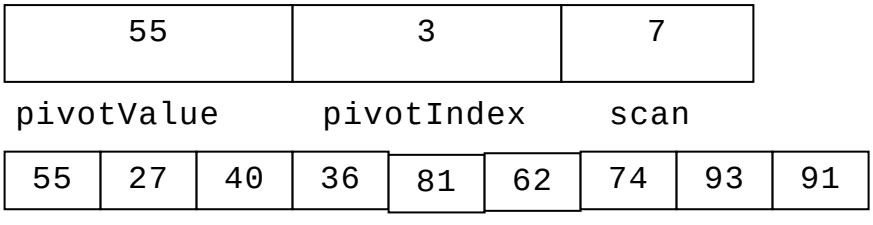
Because `array[ scan ] = 74 > pivotValue = 55`, the **if** condition fails. The next iteration starts.

`scan = 7`; (Starting loop #7)



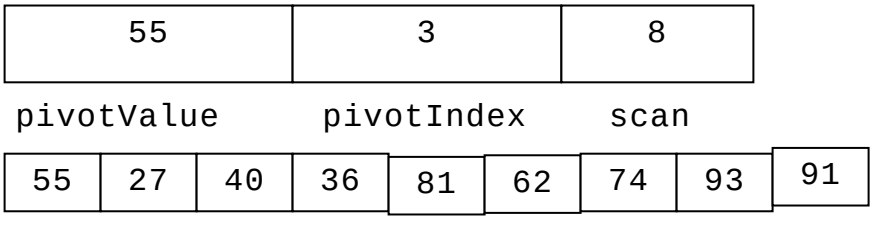
Before seventh iteration

Because `array[ scan ] = 36 < pivotValue = 55`, the **if** condition is true. Increment `pivotIndex` so that `pivotIndex = 3`. Next, `swap( array[ pivotIndex ], array[ scan ] )`, that means `swap( array[ 3 ], array[ 7 ] ).`



After seventh iteration

`scan = 8;` (Starting loop #8)



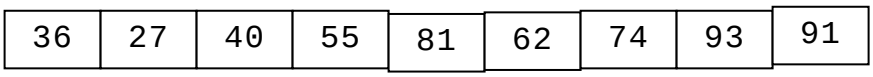
Before eighth iteration

Because `array[scan] = 91 > pivotValue = 55`, the **if** condition fails. The next iteration starts.

The eighth iteration is the last one. The loop will stop.

Step 4

```
swap(array[pivotIndex], array[0]);
```



After completing one partition

Clearly, the **L**, `pivotValue`, and **R** should already have been arranged to form the final sorted array after all recursion calls.

Code example of `Quicksort`

```
/**
 *Program:    cis27L1451_c.c
 *Discussion: Recursive Sorting Algo -- quickSort
 */
#include <stdio.h>

void quickSort(int data[], int size);
void qSort(int data[], int start, int end);
int partition(int data[], int start, int end);
void displayArray(int data[], int size);

void quickSort(int data[], int size) {
  qSort(data, 0, size - 1);

  return;
}

void qSort(int data[], int start, int end) {
  int pivotPoint;

  if (start < end) {
    pivotPoint = partition(data, start, end);

    qSort(data, start, pivotPoint - 1);

    qSort(data, pivotPoint + 1, end);
  }

  return;
}

int partition(int data[], int start, int end) {
  int pivotIndex;
  int pivotValue;
  int midIndex;

  int temp;
  int scan;

  midIndex = (start + end) / 2;

  temp = data[start];
  data[start] = data[ midIndex];
```

```
  data[midIndex] = temp;

  pivotIndex = start;
  pivotValue = data[start];

  for (scan = start + 1; scan <= end; scan++) {
    if (data[scan ] < pivotValue) {
      pivotIndex++;

      temp = data[pivotIndex];
      data[pivotIndex] = data[scan];
      data[scan] = temp;

    }
  }

  temp = data[start];
  data[start] = data[pivotIndex];
  data[pivotIndex] = temp;

  return pivotIndex;
}

void displayArray(int data[], int size) {
  int index;

  for (index = 0; index < size; index++) {

    printf("Value of array element %d : %d\n",
            index, data[ index ]);
  }

  return;
}

int main() {
  int ch;
  int data[] = {26, 13, 19, 4, 40, 30, 20, 3};

  printf("\nBefore sorting:\n\n");

  displayArray(data, 8);

  quickSort(data, 8);

  printf("\nAfter sorting:\n\n");
  displayArray(data, 8);

  printf("\nEnter a key to quit ... ");
  scanf( "%d", &ch);

  return 0;
}
```

OUTPUT

```
Before sorting:

Value of array element 0 : 26
```

```
Value of array element 1 : 13
Value of array element 2 : 19
Value of array element 3 : 4
Value of array element 4 : 40
Value of array element 5 : 30
Value of array element 6 : 20
Value of array element 7 : 3

After sorting:

Value of array element 0 : 3
Value of array element 1 : 4
Value of array element 2 : 13
Value of array element 3 : 19
Value of array element 4 : 20
Value of array element 5 : 26
Value of array element 6 : 30
Value of array element 7 : 40

Enter a key to quit ... q
```