ECE 220 Computer Systems & Programming

Lecture 11: Problem Solving with Pointers and Arrays

ECE ILLINOIS

ILLINOIS

Exercise: Exchange two rows in matrix

```
// Swap x-th row and y-th row
void matrix_change(int matrix[N][M], int x, int y)
{

}



// Transpose src matrix and store in des matrix
void matrix_tr(int des[M][N], int src[N][M])
{

}
```

Exercise: implement a function that exchanges two rows of an M x N matrix.

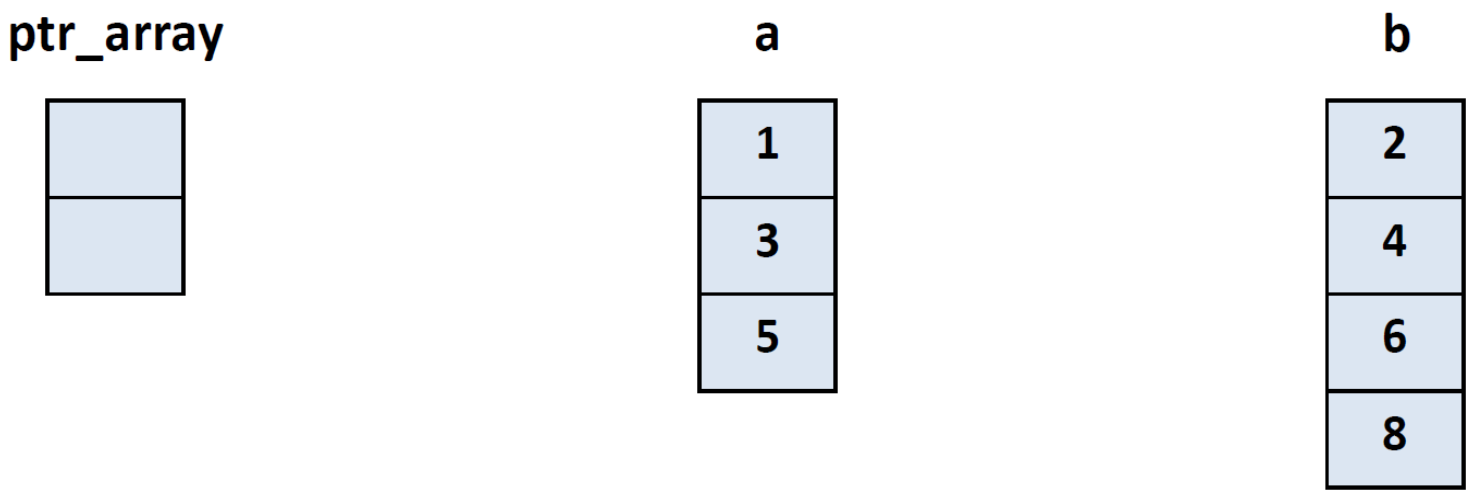
```
#define M 3 /* row size */
#define N 4 /* col size */
void row_exchange(int matrix[M][N], int row_x, int row_y){




}
```

Exercise: implement a function that transpose a 2-D matrix

```
#define ROW 2 /* row size */
#define COL 3 /* col size */
void transpose(int in_matrix[ROW][COL], int out_matrix[COL][ROW]){




}
```

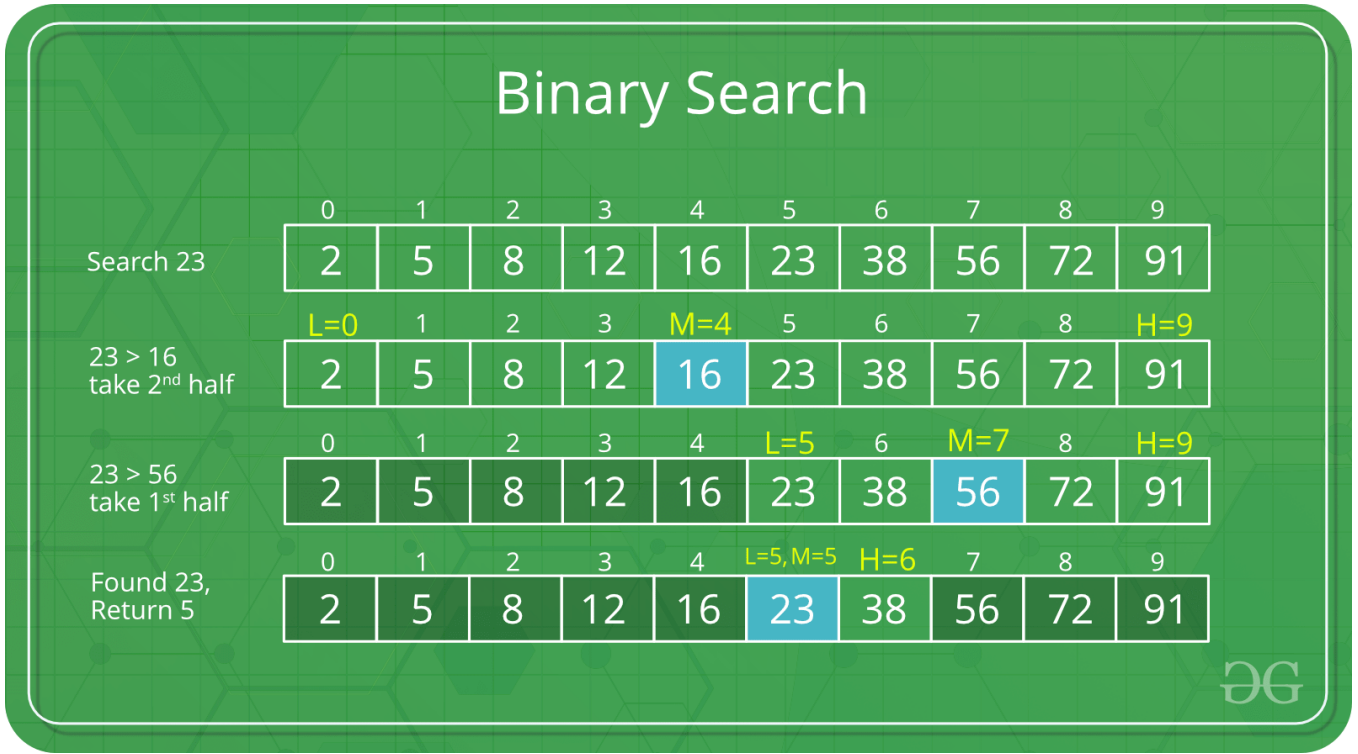
Pointer Array vs. Pointer to an Array

```
/* declare two integer arrays */
int a[3] = {1,3,5};
int b[4] = {2,4,6,8};

int *ptr_array[2];
ptr_array[0] = a;
ptr_array[1] = b;
```



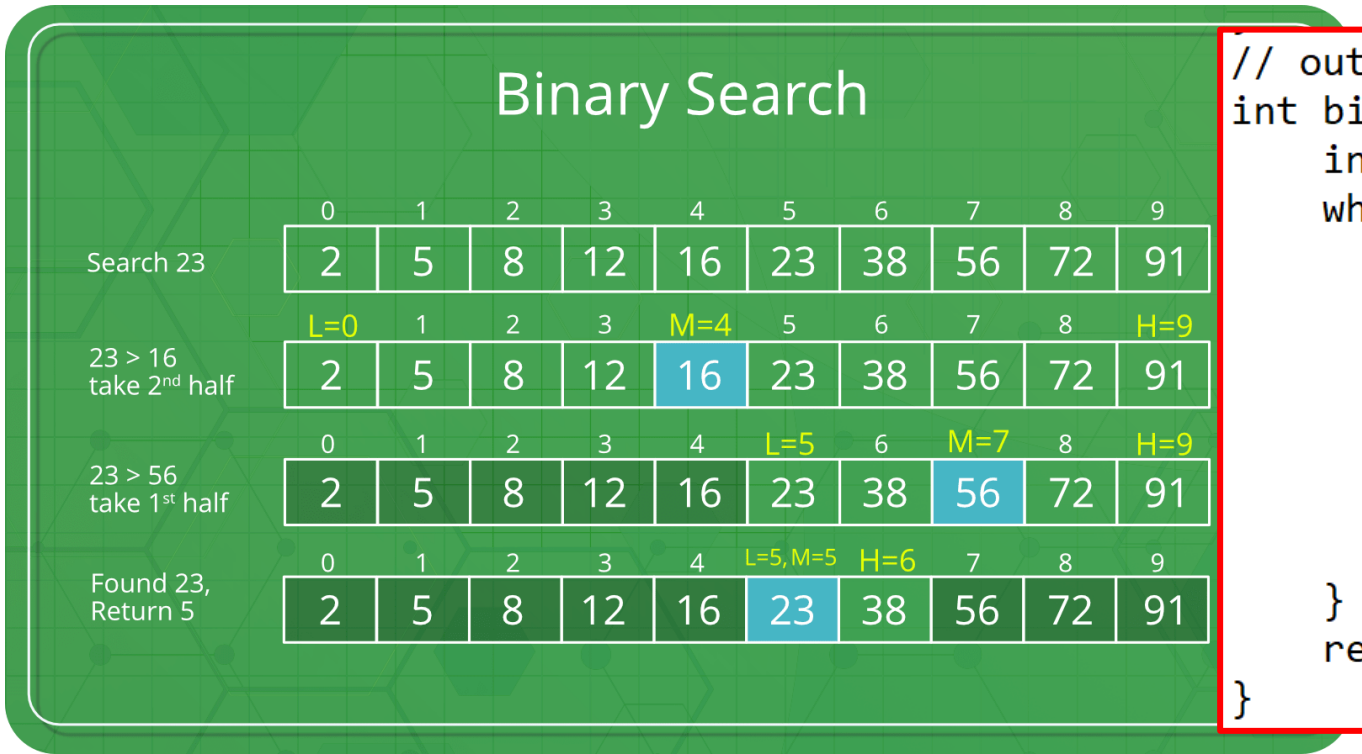
Search Algorithms

- Binary search (for sorted array)
  1. Find the middle of array and check if it's the search key
  2. If key < middle → search the first half
     If key > middle → search the second half
     If key == middle → return the key
  3. Repeat 1 and 2 until search space contains no items



1. Search Algorithms

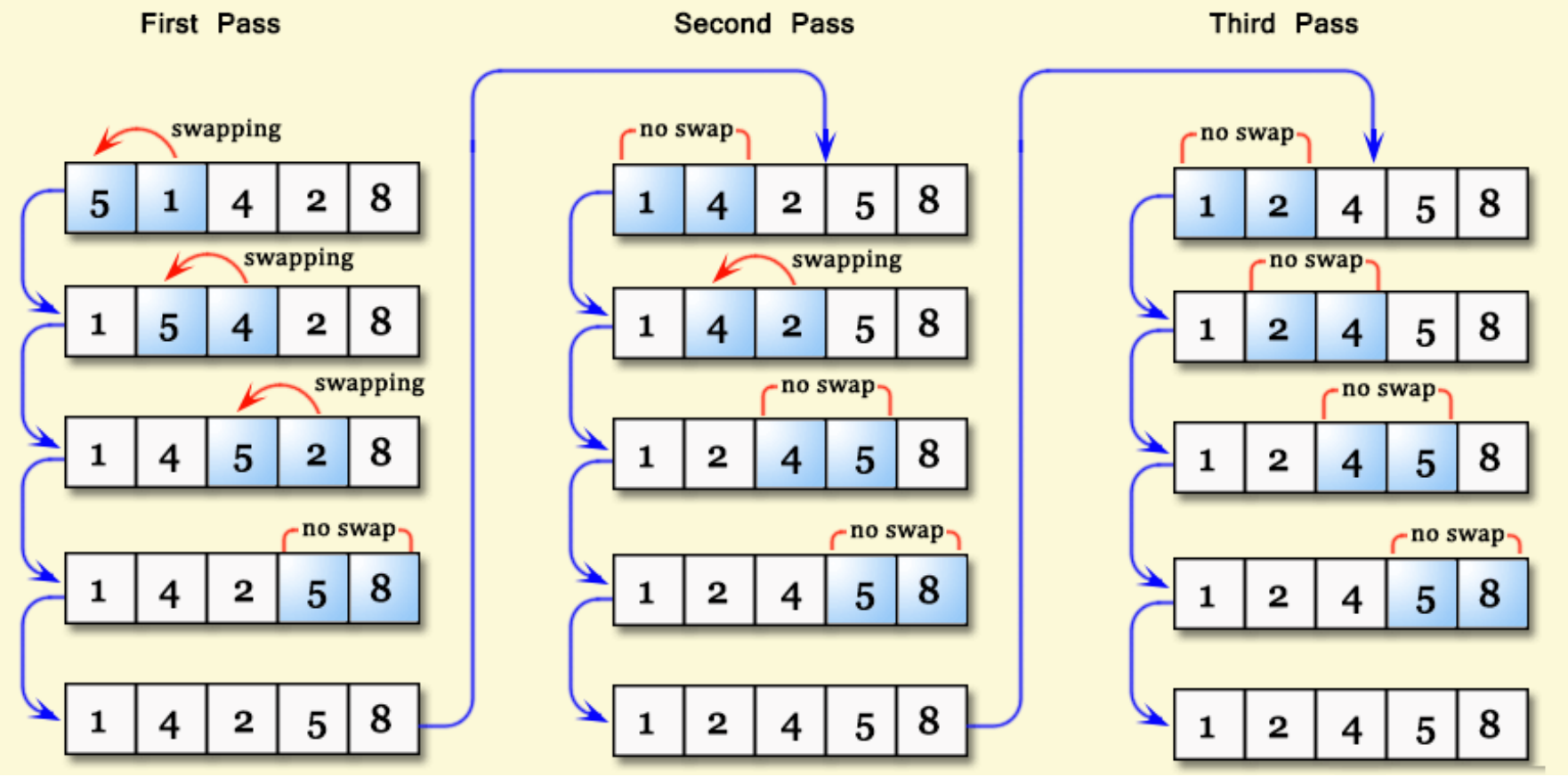
- Binary search (for sorted array)
  1. Find the middle of array and check if it's the search key
  2. If key < middle → search the first half
     If key > middle → search the second half
     If key == middle → return the key
  3. Repeat 1 and 2 until search space contains no items




```
// output index of the matched item
int binary_search(int array[], int L, int H, int key){
    int M;
    while( H >= L ){
        M = (L+H)/2;
        if( array[M] == key)
            return M;
        else if( array[M] < key)
            L = M + 1;
        else // array[M] > key
            H = M - 1;
    }
    return -1;
}
```

Bubble Sort

1. Compare items next to each other and swap them if needed.
2. Repeat this process until the entire array is sorted.



```
void swap_sort(                 )
{
     int temp;



}
```

```
#include "sort_header.h"
void bubble_sort(int *a, int size)
{

     int  i;


          for(i=0;i<size-1;i++)
          {

               {
                    swap_sort(&a[i],&a[i+1]);

               }
          }


}
```

```
void bubble_sort1(int *a, int size)
{

	int flag, i;
		flag=0;
		while(!flag)
		{
			flag=1;
		for(i=0;i<size-1;i++)
		{
			if(a[i]>a[i+1])
			{
				swap_sort(&a[i],&a[i+1]);
				flag=0;
			}
		}
		}
}
```

```c
#include "sort_header.h"
void bubble_sort(int *a, int size)
{

    int flag, i;
    do{
        flag=0;
        for(i=0;i<size-1;i++)
        {
            if(a[i]>a[i+1])
            {
                swap_sort(&a[i],&a[i+1]);
                flag=1;
            }
        }
    }while(flag);

}
```

```c
void swap_sort(int *a, int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
```

Insertion Sort

1. Remove item from array, insert it at the proper location in the sorted part by shifting other items.
2. Repeat this process until the end of array is reached.

6 5 3 1 8 7 2 4

```
void insert_sort(int *a, int size)
{

    int sorted_ind,unsortedItem,unsorted_ind;

    for(unsorted_ind=1;unsorted_ind<size;unsorted_ind++)
    {
        unsortedItem=;
        sorted_ind=;
        while(  )
        {
            // slide the current book to the right


            sorted_ind--;
        }

        //insert the unsortemItem to i


    }

}
```

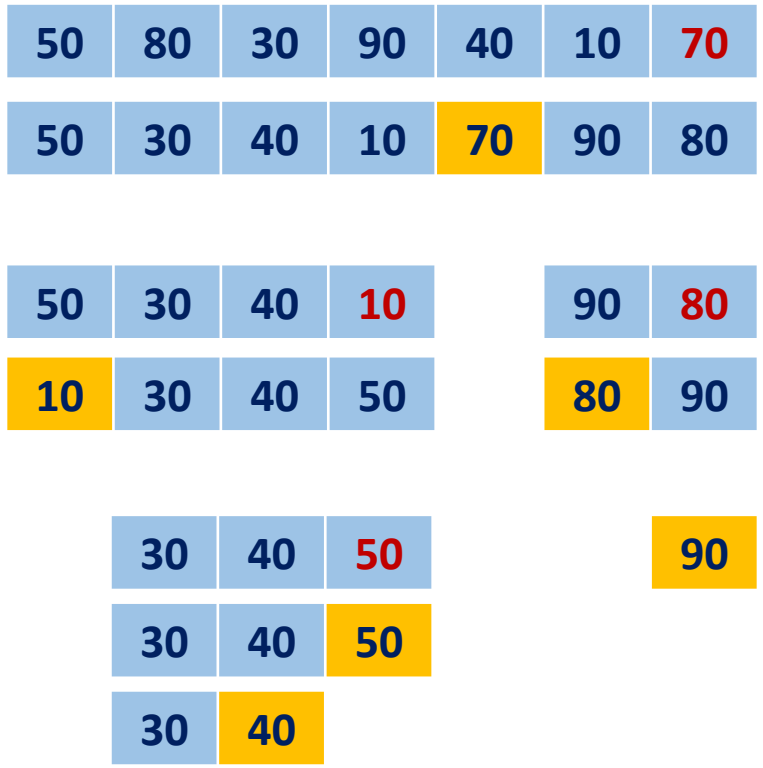
Insertion Sort

1. Remove item from array, insert it at the proper location in the sorted part by shifting other items.
2. Repeat this process until the end of array is reached.

```c
void insert_sort(int *a, int size)
{

    int sorted_ind,unsortedItem,unsorted_ind;

    for(unsorted_ind=1;unsorted_ind<size;unsorted_ind++)
    {
        unsortedItem=a[unsorted_ind];
        sorted_ind=unsorted_ind-1;
        while((sorted_ind>=0) && (unsortedItem<a[sorted_ind]))
        {
            a[sorted_ind+1]=a[sorted_ind];
            sorted_ind--;
        }

        a[sorted_ind+1]=unsortedItem;
    }

}
```

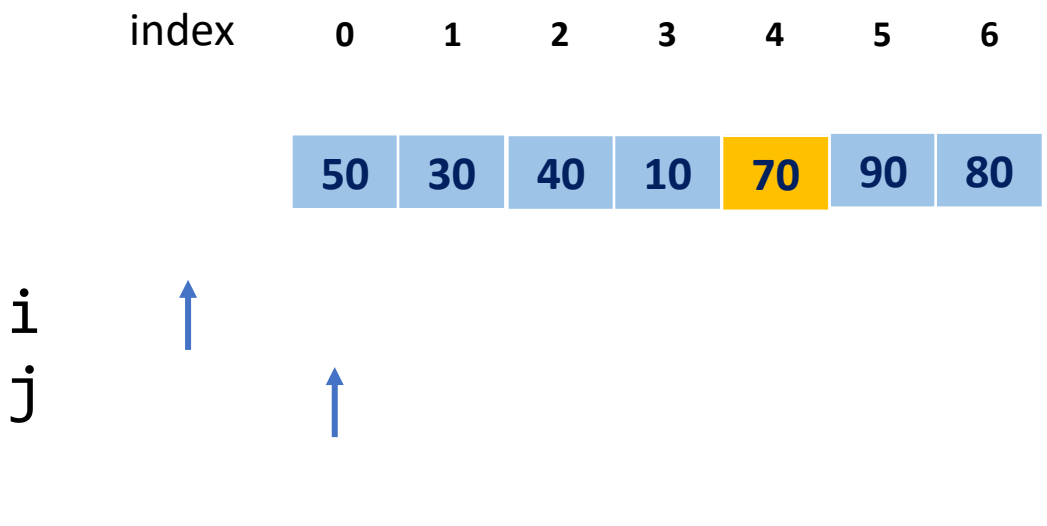
Quick Sort (divide-and-conquer)

1. Pick a pivot and partition array into 2 subarrays (smaller elements than the pivot on the left and greater elements on the right)
2. Sort the 2 subarrays using the same method



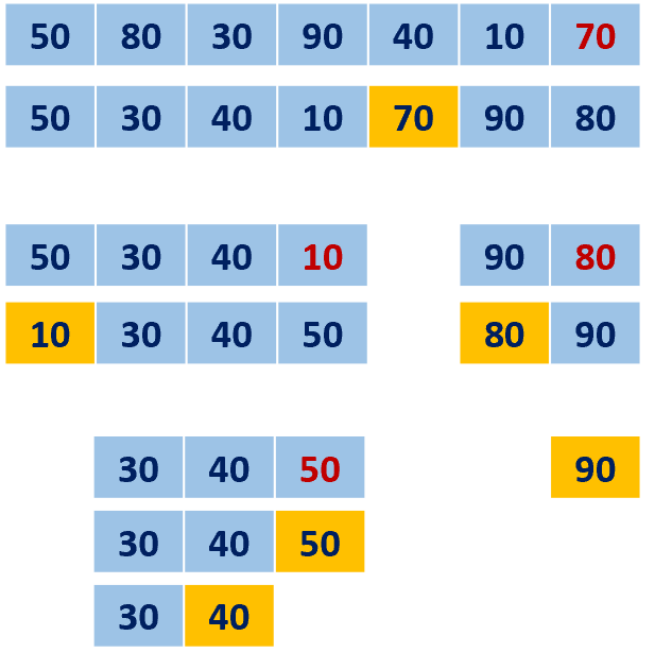
Quick Sort: Partition

```
i: index of smaller elements (i=low-1)
j: loop index (j=low) -> low to (high-1)
```




```
for j from low to high-1{
    if array[j] < pivot
        i = i + 1;
        swap array[i] and array[j]
}
swap array[i+1] and pivot
return i+1 as the pivot index
```

Using STACK in Quick Sort

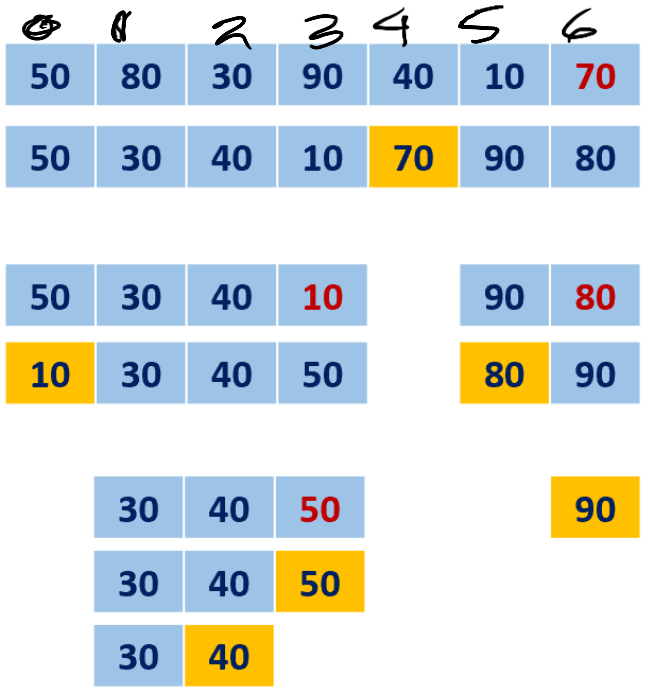



```
for j from low to high-1{
    if array[j] < pivot
        i = i + 1;
        swap array[i] and array[j]
}
swap array[i+1] and pivot
return i+1 as the pivot index
```

```
/* This function is same in both iterative and recursive*/
int partition(int arr[], int l, int h)
{
    int x = arr[h];
    int i = (l - 1);

    for (int j = l; j <= h - 1; j++) {
        if (arr[j] < x) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[h]);
    return (i + 1);
}
```

```
39  void quickSortIterative(int arr[], int l, int h)
40  {
41      // Create an auxiliary stack
42      int stack[h - l + 1];
43      // initialize top of stack
44      int top = h-l+1;
45      int tos=h-l+1;
46      // push initial values of l and h to stack
47      stack[--top] = l;
48      stack[--top] = h;
49      // Keep popping from stack while is not empty
50      while (top < tos) {
51          // Pop h and l
52          h = stack[top++];
53          l = stack[top++];
54          // Set pivot element at its correct position
55          // in sorted array
56          int p = partition(arr, l, h);
57          // If there are elements on left side of pivot,
58          // then push left side to stack
59          if (p - 1 > l) {
60              stack[--top] = l;
61              stack[--top] = p-1;
62          }
63          // If there are elements on right side of pivot,
64          // then push right side to stack
65          if (p + 1 < h) {
66              stack[--top] = p+1;
67              stack[--top] =h;
68          }
69      }
70  }
```

Using STACK in Quick Sort




```
/* This function is same in both iterative and recursive*/
int partition(int arr[], int l, int h)
{
    int x = arr[h];
    int i = (l - 1);

    for (int j = l; j <= h - 1; j++) {
        if (arr[j] < x) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[h]);
    return (i + 1);
}
```

```
39  void quickSortIterative(int arr[], int l, int h)
40  {
41      // Create an auxiliary stack
42      int stack[h - l + 1];
43      // initialize top of stack
44      int top = h-l+1;
45      int tos=h-l+1;
46      // push initial values of l and h to stack
47      stack[--top] = l;
48      stack[--top] = h;
49      // Keep popping from stack while is not empty
50      while (top < tos) {
51          // Pop h and l
52          h = stack[top++];
53          l = stack[top++];
54          // Set pivot element at its correct position
55          // in sorted array
56          int p = partition(arr, l, h);
57          // If there are elements on left side of pivot,
58          // then push left side to stack
59          if (p - 1 > l) {
60              stack[--top] = l;
61              stack[--top] = p-1;
62          }
63          // If there are elements on right side of pivot,
64          // then push right side to stack
65          if (p + 1 < h) {
66              stack[--top] = p+1;
67              stack[--top] =h;
68          }
69      }
70  }
```