Laboratory 10

1. **Questions**

   Write a C program to implement sorting of numbers using bubble sort, selection sort and quick sort techniques. Calculate the time required for each approach.

2. **Algorithm**

1.Pseudocode implementation for bubble sort

procedure bubbleSort(A : list of sortable items)

```
    n = length(A)
    repeat
        swapped = false
        for i = 1 to n-1 inclusive do
            /* if this pair is out of order */
            if A[i-1] > A[i] then
                /* swap them and remember something changed */
                swap( A[i-1], A[i] )
                swapped = true
            end if
        end for
    until not swapped
end procedure
```

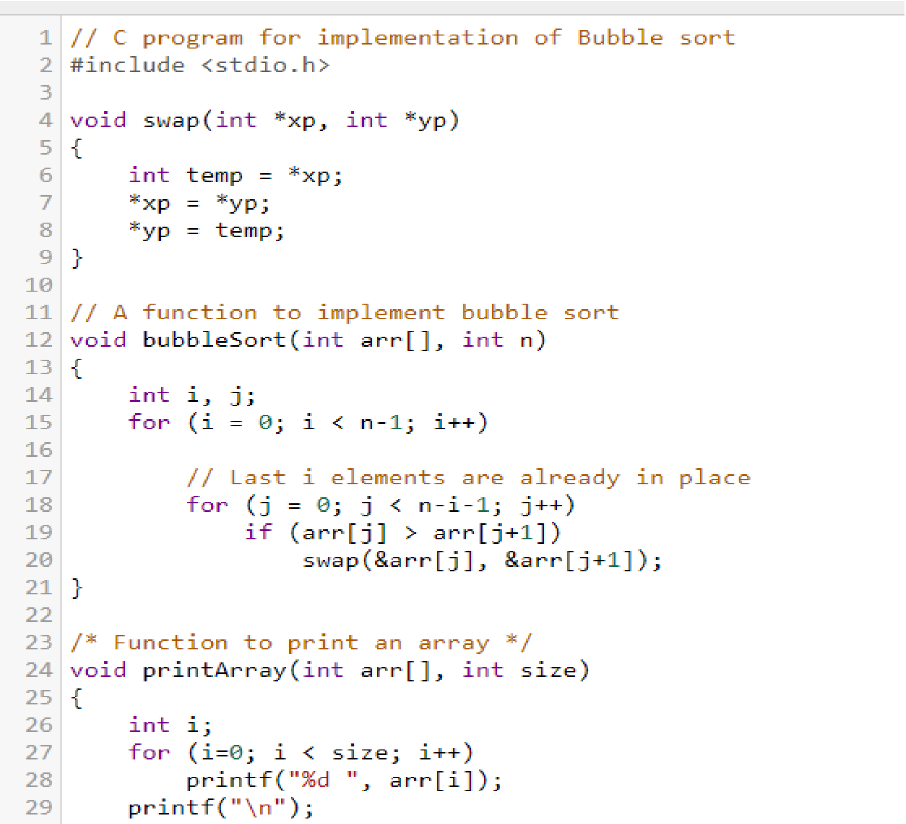
Pseudocode for selection sort

```
procedure selection sort
   list  : array of items
   n     : size of list

   for i = 1 to n - 1

      min = i

      /* check the element to be minimum */

      for j = i+1 to n
         if list[j] < list[min] then
            min = j;
         end if
      end for

      /* swap the minimum element with the current element*/
      if indexMin != i  then
         swap list[min] and list[i]
      end if
   end for

end procedure
```

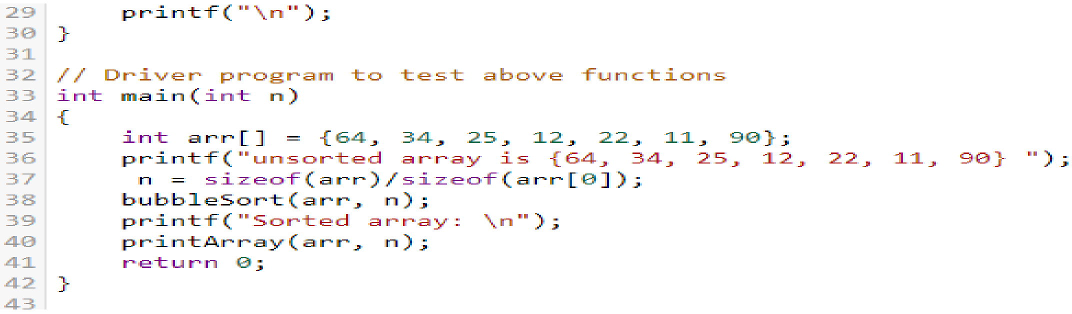
Pseudocode for Quick Sort Pivot

```
function partitionFunc(left, right, pivot)
   leftPointer = left
   rightPointer = right - 1

   while True do
      while A[++leftPointer] < pivot do
         //do-nothing
      end while

      while rightPointer > 0 && A[--rightPointer] > pivot do
         //do-nothing
      end while

      if leftPointer >= rightPointer
         break
      else
         swap leftPointer,rightPointer
      end if

   end while

   swap leftPointer,right
   return leftPointer

end function
```

3. Program

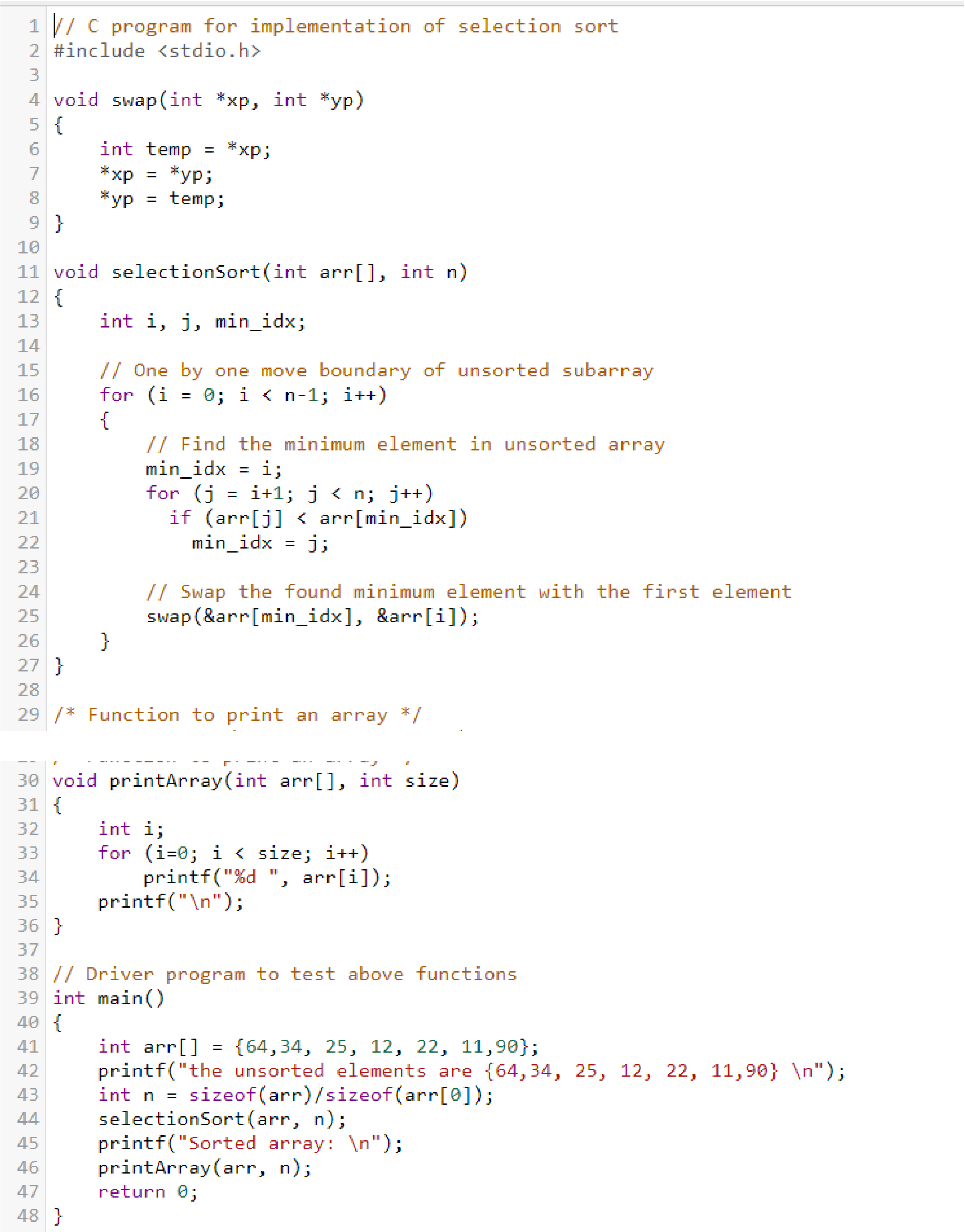


```c
// C program for implementation of Bubble sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++)

        // Last i elements are already in place
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
```



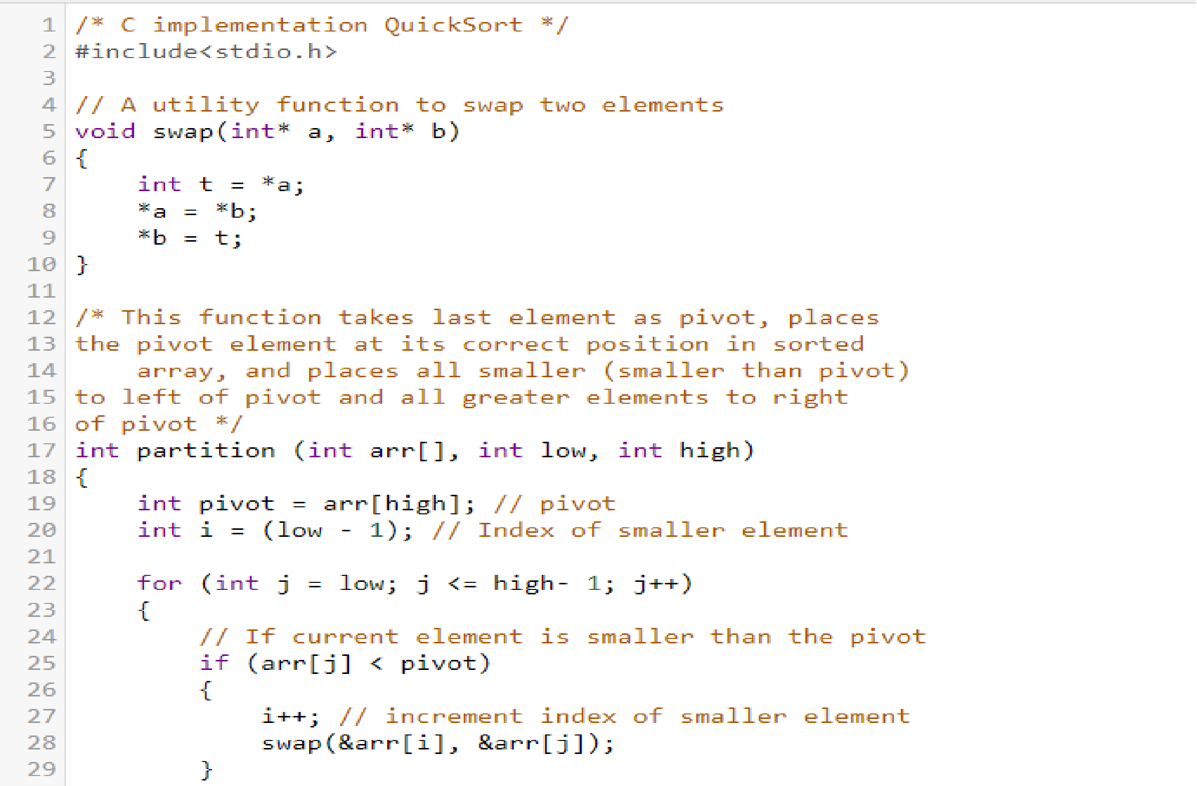
```c
    printf("\n");
}

// Driver program to test above functions
int main(int n)
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    printf("unsorted array is {64, 34, 25, 12, 22, 11, 90} ");
     n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}

```

Fig 1 program to implement bubble sort

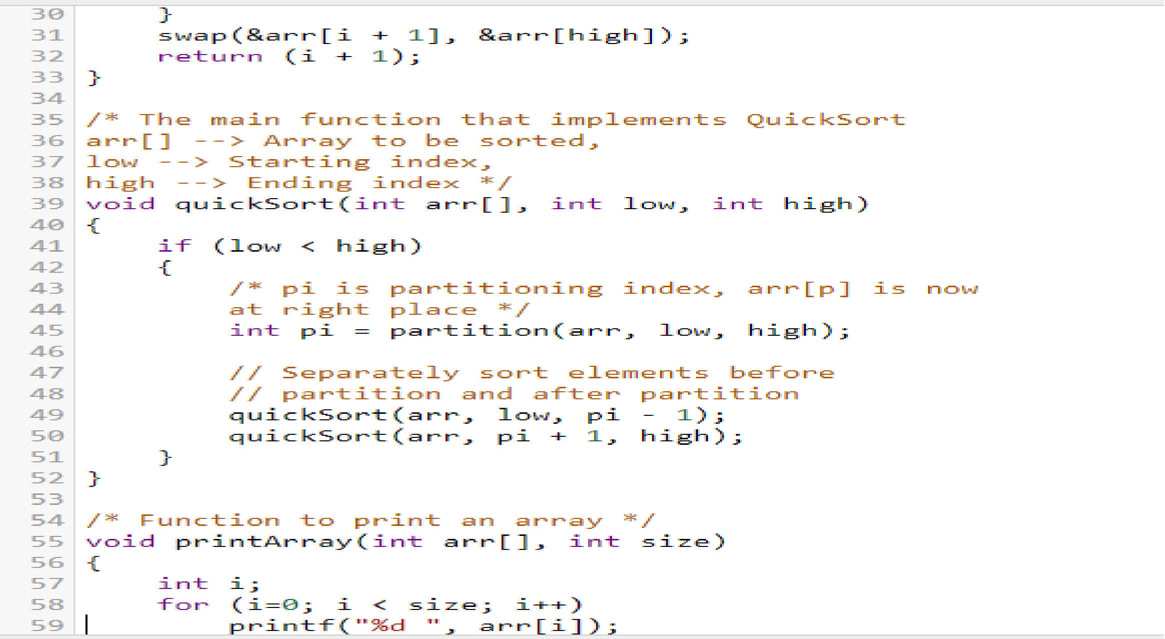


```c
// C program for implementation of selection sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
          if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {64,34, 25, 12, 22, 11,90};
    printf("the unsorted elements are {64,34, 25, 12, 22, 11,90} \n");
    int n = sizeof(arr)/sizeof(arr[0]);
    selectionSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

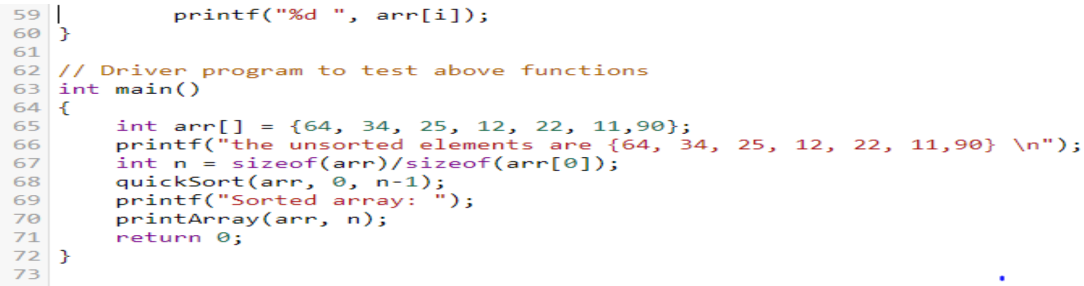
Fig 2 program to implement selection sort



```
/* C implementation QuickSort */
#include<stdio.h>

// A utility function to swap two elements
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* This function takes last element as pivot, places
the pivot element at its correct position in sorted
    array, and places all smaller (smaller than pivot)
to left of pivot and all greater elements to right
of pivot */
int partition (int arr[], int low, int high)
{
    int pivot = arr[high]; // pivot
    int i = (low - 1); // Index of smaller element

    for (int j = low; j <= high- 1; j++)
    {
        // If current element is smaller than the pivot
        if (arr[j] < pivot)
        {
            i++; // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
```



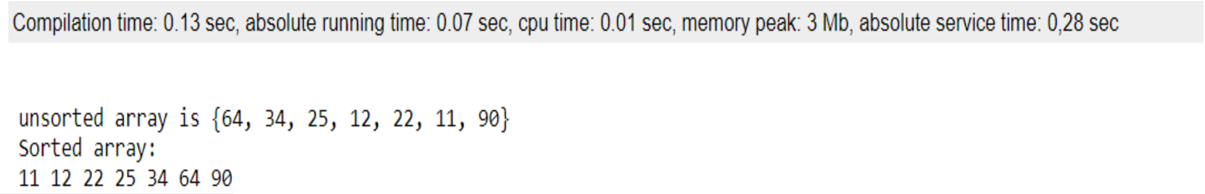
```
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
        at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
```



```
        printf("%d ", arr[i]);
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11,90};
    printf("the unsorted elements are {64, 34, 25, 12, 22, 11,90} \n");
    int n = sizeof(arr)/sizeof(arr[0]);
    quickSort(arr, 0, n-1);
    printf("Sorted array: ");
    printArray(arr, n);
    return 0;
}
```

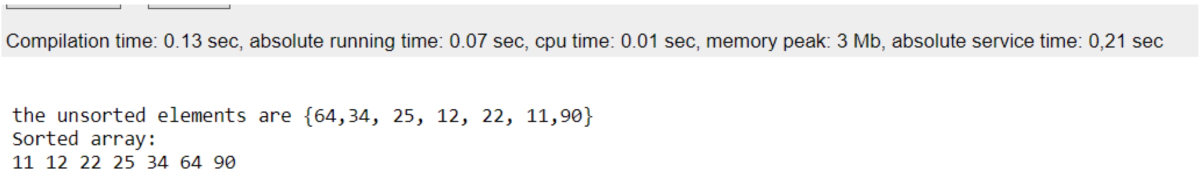
Fig 3 program for quick sort

4. Presentation of Result



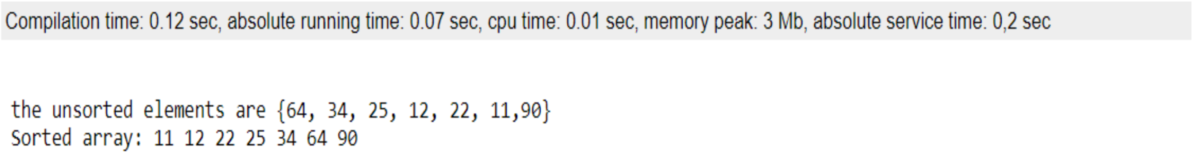
```
Compilation time: 0.13 sec, absolute running time: 0.07 sec, cpu time: 0.01 sec, memory peak: 3 Mb, absolute service time: 0,28 sec

unsorted array is {64, 34, 25, 12, 22, 11, 90}
Sorted array:
11 12 22 25 34 64 90
```

Fig 4 Result of program to implement bubble sort with execution time



```
Compilation time: 0.13 sec, absolute running time: 0.07 sec, cpu time: 0.01 sec, memory peak: 3 Mb, absolute service time: 0,21 sec

the unsorted elements are {64,34, 25, 12, 22, 11,90}
Sorted array:
11 12 22 25 34 64 90
```

Fig 5 Result of program to implement selection sort with execution time



```
Compilation time: 0.12 sec, absolute running time: 0.07 sec, cpu time: 0.01 sec, memory peak: 3 Mb, absolute service time: 0,2 sec

the unsorted elements are {64, 34, 25, 12, 22, 11,90}
Sorted array: 11 12 22 25 34 64 90
```

Fig 6 Result of program to implement quick sort with execution time

5. Conclusion

In this lab we learnt Write a C program to implement sorting of numbers using bubble sort, selection sort and quick sort techniques. And to Calculate the time required for each approach.