CS 677 - Analysis of Algorithm
Fall Semester
Homework 3

Dzung Bui

Due date: September 24, 2019

1. Consider the following Algorithm - *20 points*

ALGORITHM *Enigma*(A[0 .. n-1])

//Input: An array A[0 ... n-1] of integer numbers

Activities	Cost	Time
for i ← 0 to n-2 do	c_1	n
for j ← i+1 to n-1 do	c_2	$\sum_{i=0}^{n-2}$(n-i-2)
if A[i] == A[j]	c_3	$\sum_{i=0}^{n-2}$(n-i-2)
return false	c_4	1
return true	c_5	1

a) This algorithm checks whether there is any common values in the array. If two or more items in the arrays share common values, the function will return ”false”. Otherwise, it will return ”true”.
b) Compute the running time of this algorithm:

$$T(n) = c_1 n + c_2 \sum_{i=0}^{n-2}(n-i-2) + c_3 \sum_{i=0}^{n-2}(n-i-2) + c_4 + c_5$$

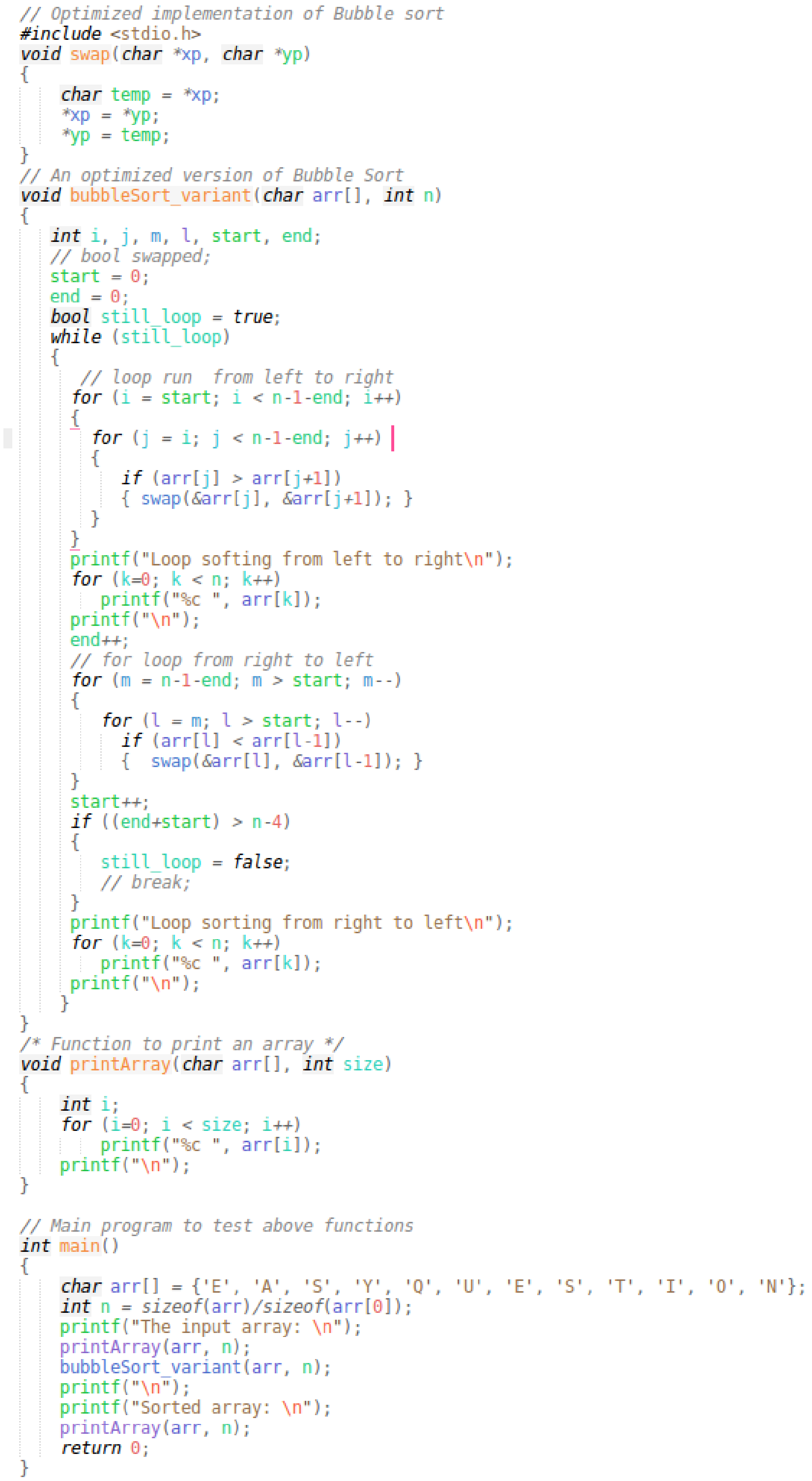
$$T(n) = c_1 n + (c_3 + c_2) \sum_{i=0}^{n-2}(n-i-2) + c_4 + c_5$$

$$\sum_{i=0}^{n-2}(n-i-2) = 1 + 2 + ... + n - 2 = \frac{(n-2)(n-1)}{2}$$

$$T(n) = c_1 n + (c_2 + c_3)\frac{(n-2)(n-1)}{2} + c_4 + c_5 \approx \Theta(n^2)$$

2. Implement a version of bubble sort - *40 points*

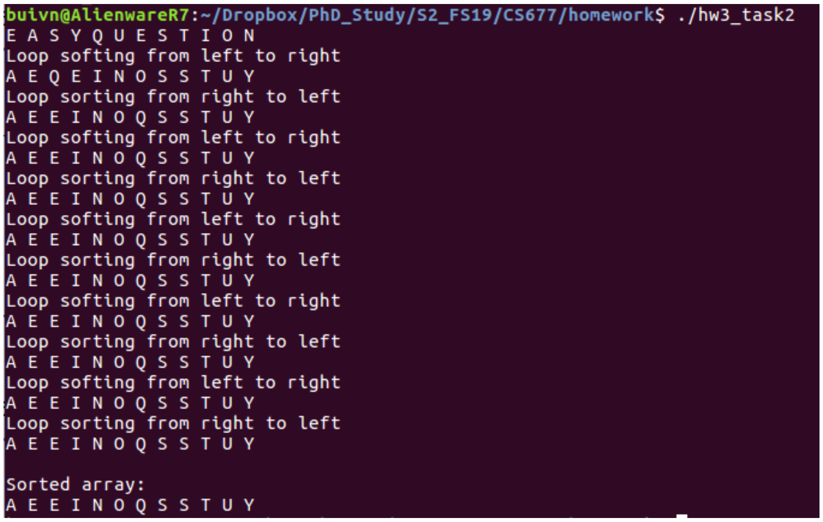
a) The code for the program is submitted by another file. Its snapshot is shown here:



```
// Optimized implementation of Bubble sort
#include <stdio.h>
void swap(char *xp, char *yp)
{
    char temp = *xp;
    *xp = *yp;
    *yp = temp;
}
// An optimized version of Bubble Sort
void bubbleSort_variant(char arr[], int n)
{
   int i, j, m, l, start, end;
   // bool swapped;
   start = 0;
   end = 0;
   bool still_loop = true;
   while (still_loop)
   {
      // loop run  from left to right
     for (i = start; i < n-1-end; i++)
     {
       for (j = i; j < n-1-end; j++)
       {
          if (arr[j] > arr[j+1])
          { swap(&arr[j], &arr[j+1]); }
       }
     }
     printf("Loop softing from left to right\n");
     for (k=0; k < n; k++)
        printf("%c ", arr[k]);
     printf("\n");
     end++;
     // for loop from right to left
     for (m = n-1-end; m > start; m--)
     {
        for (l = m; l > start; l--)
          if (arr[l] < arr[l-1])
          {  swap(&arr[l], &arr[l-1]); }
     }
     start++;
     if ((end+start) > n-4)
     {
        still_loop = false;
        // break;
     }
     printf("Loop sorting from right to left\n");
     for (k=0; k < n; k++)
        printf("%c ", arr[k]);
     printf("\n");
    }
}
/* Function to print an array */
void printArray(char arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%c ", arr[i]);
    printf("\n");
}

// Main program to test above functions
int main()
{
    char arr[] = {'E', 'A', 'S', 'Y', 'Q', 'U', 'E', 'S', 'T', 'I', 'O', 'N'};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("The input array: \n");
    printArray(arr, n);
    bubbleSort_variant(arr, n);
    printf("\n");
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Figure 1: Code of Task 2.

The result of sorting the array: ”EASYQUESTION” is shown on the figure following:



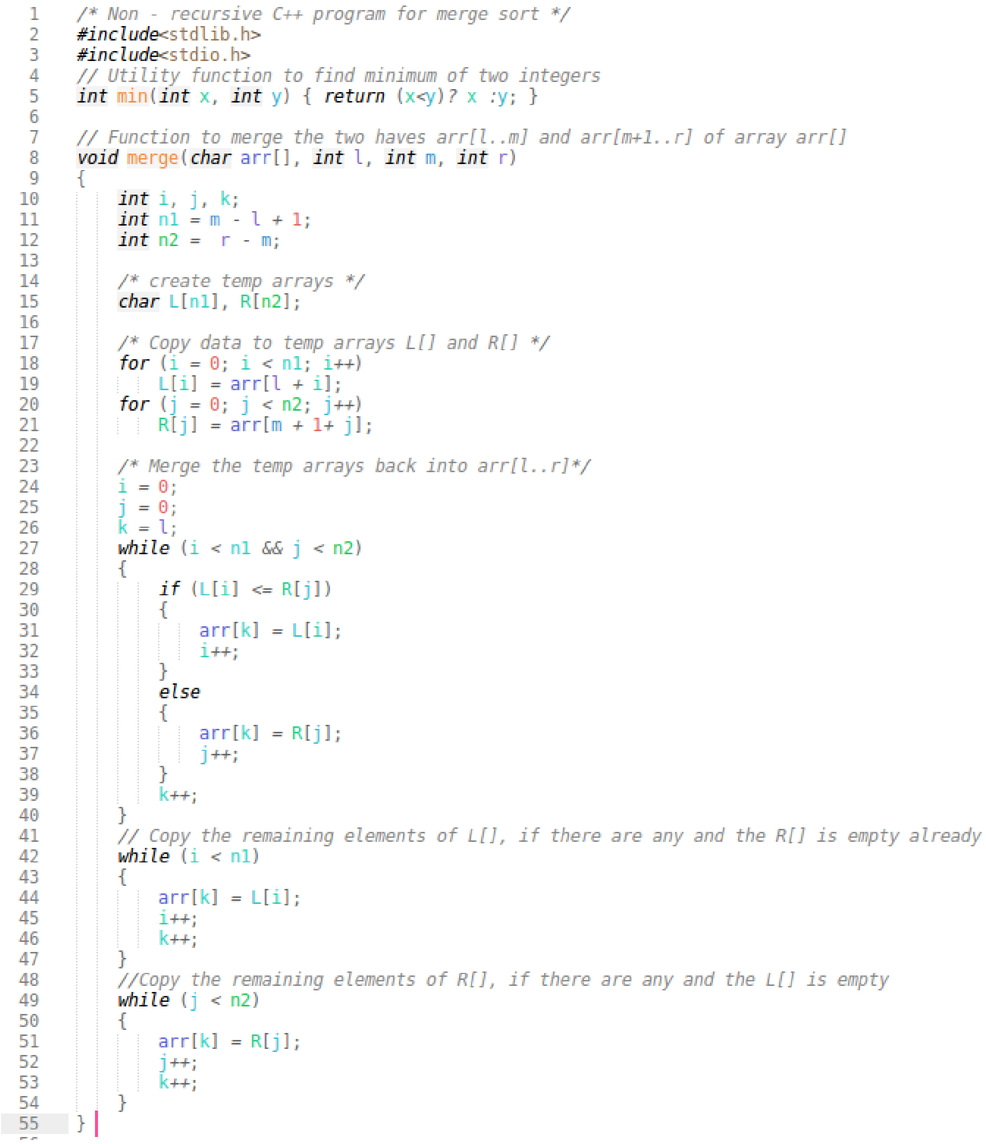
```
buivn@AlienwareR7:~/Dropbox/PhD_Study/S2_FS19/CS677/homework$ ./hw3_task2
E A S Y Q U E S T I O N
Loop softing from left to right
A E Q E I N O S S T U Y
Loop sorting from right to left
A E E I N O Q S S T U Y
Loop softing from left to right
A E E I N O Q S S T U Y
Loop sorting from right to left
A E E I N O Q S S T U Y
Loop softing from left to right
A E E I N O Q S S T U Y
Loop sorting from right to left
A E E I N O Q S S T U Y
Loop softing from left to right
A E E I N O Q S S T U Y
Loop sorting from right to left
A E E I N O Q S S T U Y
Loop softing from left to right
A E E I N O Q S S T U Y
Loop sorting from right to left
A E E I N O Q S S T U Y

Sorted array:
A E E I N O Q S S T U Y
```

Figure 2: Resulting of Task 2.

b) The number of comparison in this program:
Each time the iteration run from left to right or right-to-left, the size of processed array's length will decrease 1 cell. Thus, the total comparison is:

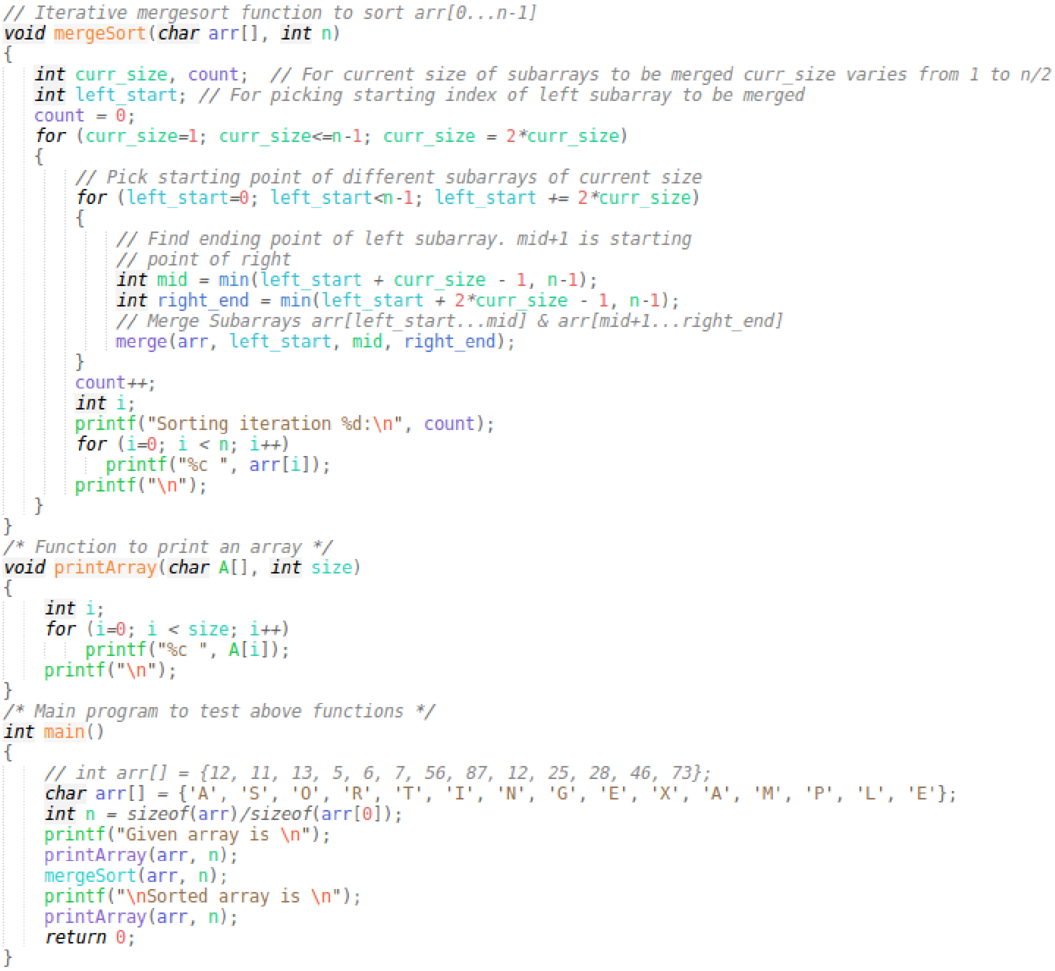
$$Comparison = (n-1) + (n-2) + ... + 1 = \frac{(n-1)n}{2} \tag{1}$$

3. Write a program to run Merge-Sort algorithm without recursion - *20 points*
The Code of the program:



```
/* Non - recursive C++ program for merge sort */
#include<stdlib.h>
#include<stdio.h>
// Utility function to find minimum of two integers
int min(int x, int y) { return (x<y)? x :y; }

// Function to merge the two haves arr[l..m] and arr[m+1..r] of array arr[]
void merge(char arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    /* create temp arrays */
    char L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    // Copy the remaining elements of L[], if there are any and the R[] is empty already
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }
    //Copy the remaining elements of R[], if there are any and the L[] is empty
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

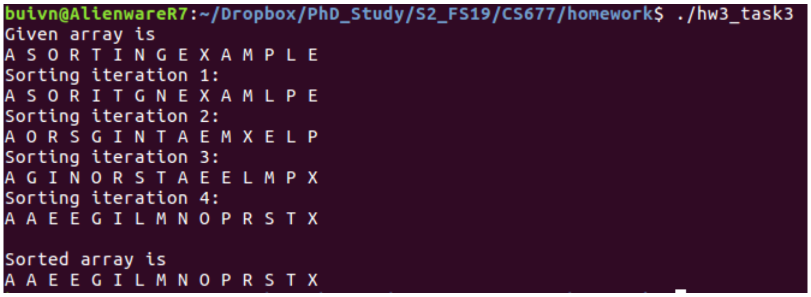
Figure 3: Code of Task 3.



```c
// Iterative mergesort function to sort arr[0...n-1]
void mergeSort(char arr[], int n)
{
    int curr_size, count;  // For current size of subarrays to be merged curr_size varies from 1 to n/2
    int left_start; // For picking starting index of left subarray to be merged
    count = 0;
    for (curr_size=1; curr_size<=n-1; curr_size = 2*curr_size)
    {
        // Pick starting point of different subarrays of current size
        for (left_start=0; left_start<n-1; left_start += 2*curr_size)
        {
            // Find ending point of left subarray. mid+1 is starting
            // point of right
            int mid = min(left_start + curr_size - 1, n-1);
            int right_end = min(left_start + 2*curr_size - 1, n-1);
            // Merge Subarrays arr[left_start...mid] & arr[mid+1...right_end]
            merge(arr, left_start, mid, right_end);
        }
        count++;
        int i;
        printf("Sorting iteration %d:\n", count);
        for (i=0; i < n; i++)
            printf("%c ", arr[i]);
        printf("\n");
    }
}
/* Function to print an array */
void printArray(char A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%c ", A[i]);
    printf("\n");
}
/* Main program to test above functions */
int main()
{
    // int arr[] = {12, 11, 13, 5, 6, 7, 56, 87, 12, 25, 28, 46, 73};
    char arr[] = {'A', 'S', 'O', 'R', 'T', 'I', 'N', 'G', 'E', 'X', 'A', 'M', 'P', 'L', 'E'};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Given array is \n");
    printArray(arr, n);
    mergeSort(arr, n);
    printf("\nSorted array is \n");
    printArray(arr, n);
    return 0;
}
```

Figure 4: Code of Task 3 - cont.

The result after sorting the sequence ”ASORTINGEXAMPLE”.

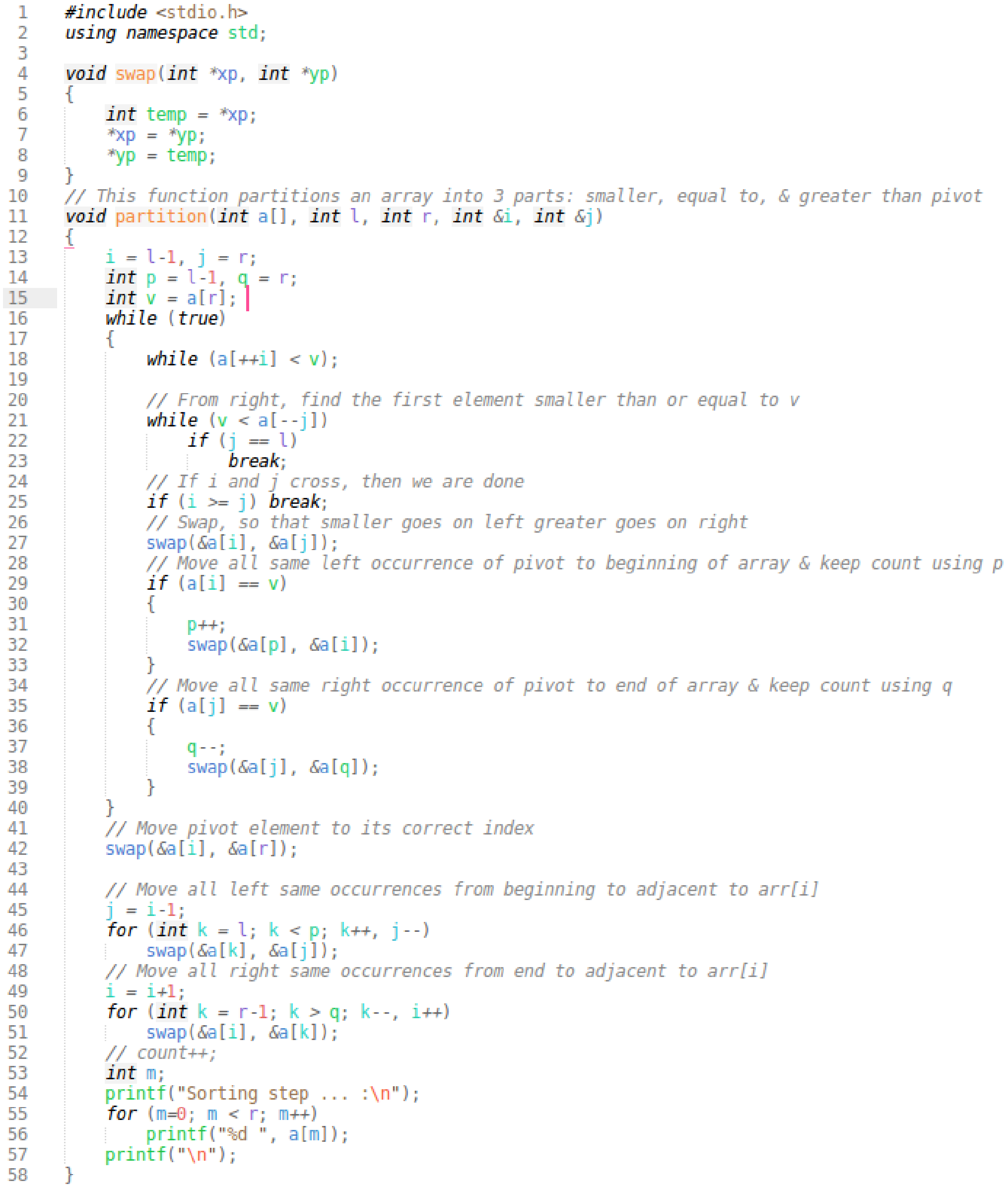


```
buivn@AlienwareR7:~/Dropbox/PhD_Study/S2_FS19/CS677/homework$ ./hw3_task3
Given array is
A S O R T I N G E X A M P L E
Sorting iteration 1:
A S O R I T G N E X A M L P E
Sorting iteration 2:
A O R S G I N T A E M X E L P
Sorting iteration 3:
A G I N O R S T A E E L M P X
Sorting iteration 4:
A A E E G I L M N O P R S T X

Sorted array is
A A E E G I L M N O P R S T X
```

Figure 5: Result of Task 3.

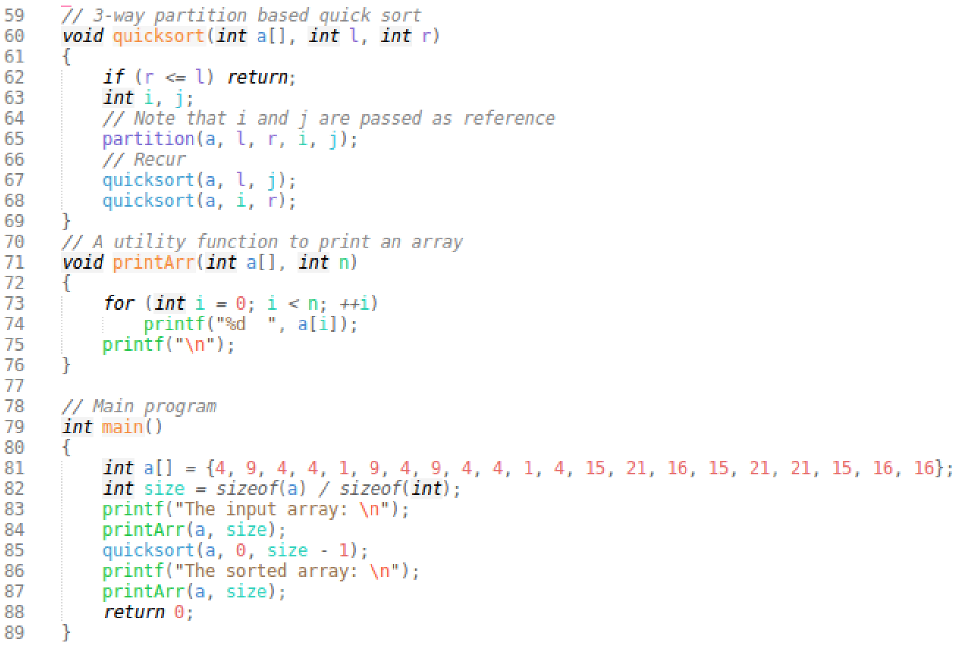
4. QUICKSORT with duplicates keys - *40 Points*

The snap shot of the program:



```c
#include <stdio.h>
using namespace std;

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
// This function partitions an array into 3 parts: smaller, equal to, & greater than pivot
void partition(int a[], int l, int r, int &i, int &j)
{
    i = l-1, j = r;
    int p = l-1, q = r;
    int v = a[r];
    while (true)
    {
        while (a[++i] < v);

        // From right, find the first element smaller than or equal to v
        while (v < a[--j])
            if (j == l)
                break;
        // If i and j cross, then we are done
        if (i >= j) break;
        // Swap, so that smaller goes on left greater goes on right
        swap(&a[i], &a[j]);
        // Move all same left occurrence of pivot to beginning of array & keep count using p
        if (a[i] == v)
        {
            p++;
            swap(&a[p], &a[i]);
        }
        // Move all same right occurrence of pivot to end of array & keep count using q
        if (a[j] == v)
        {
            q--;
            swap(&a[j], &a[q]);
        }
    }
    // Move pivot element to its correct index
    swap(&a[i], &a[r]);

    // Move all left same occurrences from beginning to adjacent to arr[i]
    j = i-1;
    for (int k = l; k < p; k++, j--)
        swap(&a[k], &a[j]);
    // Move all right same occurrences from end to adjacent to arr[i]
    i = i+1;
    for (int k = r-1; k > q; k--, i++)
        swap(&a[i], &a[k]);
    // count++;
    int m;
    printf("Sorting step ... :\n");
    for (m=0; m < r; m++)
        printf("%d ", a[m]);
    printf("\n");
}
```

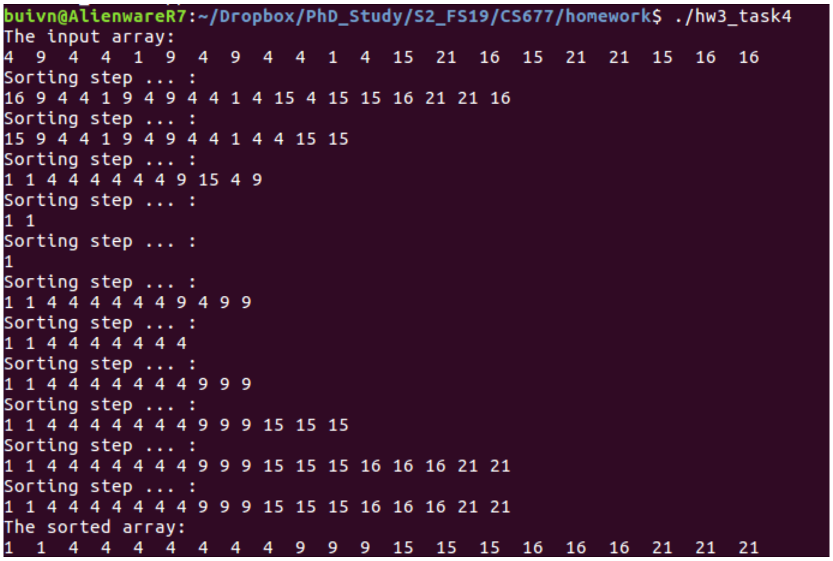
Figure 6: Code of Task 4.



```c
// 3-way partition based quick sort
void quicksort(int a[], int l, int r)
{
    if (r <= l) return;
    int i, j;
    // Note that i and j are passed as reference
    partition(a, l, r, i, j);
    // Recur
    quicksort(a, l, j);
    quicksort(a, i, r);
}
// A utility function to print an array
void printArr(int a[], int n)
{
    for (int i = 0; i < n; ++i)
        printf("%d  ", a[i]);
    printf("\n");
}

// Main program
int main()
{
    int a[] = {4, 9, 4, 4, 1, 9, 4, 9, 4, 4, 1, 4, 15, 21, 16, 15, 21, 21, 15, 16, 16};
    int size = sizeof(a) / sizeof(int);
    printf("The input array: \n");
    printArr(a, size);
    quicksort(a, 0, size - 1);
    printf("The sorted array: \n");
    printArr(a, size);
    return 0;
}
```

Figure 7: Code of Task 4 - cont.

The result of the QuickSort is shown as following:



```
buivn@AlienwareR7:~/Dropbox/PhD_Study/S2_FS19/CS677/homework$ ./hw3_task4
The input array:
4  9  4  4  1  9  4  9  4  4  1  4  15  21  16  15  21  21  15  16  16
Sorting step ... :
16 9 4 4 1 9 4 9 4 4 1 4 15 4 15 15 16 21 21 16
Sorting step ... :
15 9 4 4 1 9 4 9 4 4 1 4 4 15 15
Sorting step ... :
1 1 4 4 4 4 4 4 9 15 4 9
Sorting step ... :
1 1
Sorting step ... :
1
Sorting step ... :
1 1 4 4 4 4 4 4 9 4 9 9
Sorting step ... :
1 1 4 4 4 4 4 4 4
Sorting step ... :
1 1 4 4 4 4 4 4 4 9 9 9
Sorting step ... :
1 1 4 4 4 4 4 4 4 9 9 9 15 15 15
Sorting step ... :
1 1 4 4 4 4 4 4 4 9 9 9 15 15 15 16 16 16 21 21
Sorting step ... :
1 1 4 4 4 4 4 4 4 9 9 9 15 15 15 16 16 16 21 21
The sorted array:
1  1  4  4  4  4  4  4  4  9  9  9  15  15  15  16  16  16  21  21  21
```

Figure 8: Result of Task 4.

5. Consider the following Algorithm

```
ALGORITHM Alg2(A[left .. right])
    if $left = right$ return $A[left]$
    else:
        $temp1 \rightarrow Alg2(A[left..[(left+right)/2]])$
    temp2 $\rightarrow$ Alg2(A[[(left+right)/2]+1 .. right])
    if temp1 $\leq$ temp2 return temp1
    else return temp2
```

a) This algorithm finds the smallest value in the array.
b) From the Algorithm, the recurrence should have the form:

$$W(n) = 2W(\frac{n}{2}) + 3 \tag{2}$$

a = 2, b = 2, log_2^2 = 1, so compare n with f(n) = 3.
→ f(n) = $O(n^{1-\epsilon})$ → case 1.
So, → $T(n) = \Theta(n)$