Design & Analysis Algorithm Assignment – 2

Name : Jay Lapani

Roll No. : 21BCP150

Div - 3 , G - 5

-: Merge sort :-

The Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm. In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner.

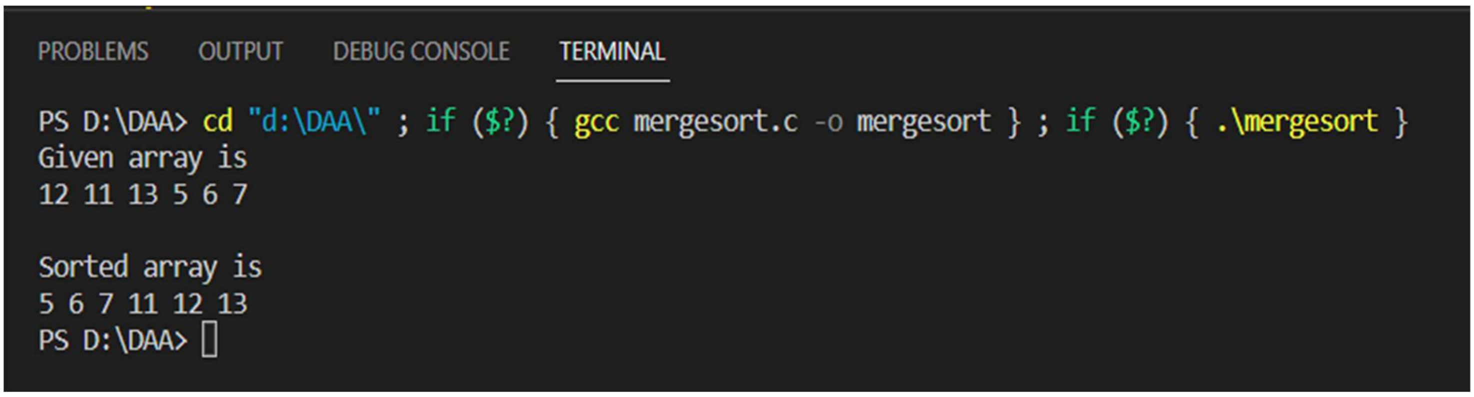
- **Merge Sort Algorithm :-**

```
1. start
2. declare array and left, right, mid variable
3.  perform merge function.
       if left > right
 return
        mid= (left+right)/2
       mergesort(array, left, mid)
       mergesort(array, mid+1,right)
       merge(array, left,mid, right)
 4. Stop
```

- **Program :-**

```
#include <stdio.h>
#include <stdlib.h>
void merge(int arr[], int l, int m, int r)
{
 int i, j, k;
 int n1 = m - l + 1;
 int n2 = r - m;
 int L[n1], R[n2];
 for (i = 0; i < n1; i++)
 L[i] = arr[l + i];
 for (j = 0; j < n2; j++)
 R[j] = arr[m + 1 + j];
 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++;
 }
 while (i < n1) {
 arr[k] = L[i];
 i++;
 k++;
 }
 while (j < n2) {
 arr[k] = R[j];
 j++;
 k++;
 }
}
void mergeSort(int arr[], int l, int r)
{
 if (l < r) {
 int m = l + (r - l) / 2;
 mergeSort(arr, l, m);
 mergeSort(arr, m + 1, r);
 merge(arr, l, m, r);
 }
}
void printArray(int A[], int size)
{
 int i;
 for (i = 0; i < size; i++)
 printf("%d ", A[i]);
 printf("\n");
}
int main()
{
 int arr[] = { 12, 11, 13, 5, 6, 7 };
 int arr_size = sizeof(arr) / sizeof(arr[0]);
 printf("Given array is \n");
 printArray(arr, arr_size);
 mergeSort(arr, 0, arr_size - 1);
 printf("\nSorted array is \n");
 printArray(arr, arr_size);
 return 0;
}
```

- **Output :-**



```
PS D:\DAA> cd "d:\DAA\" ; if ($?) { gcc mergesort.c -o mergesort } ; if ($?) { .\mergesort }
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13
PS D:\DAA>
```

-: <u>Quick sort</u> :-

Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as a pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

- Always pick the first element as a pivot.
- Always pick the last element as a pivot (implemented below)
- Pick a random element as a pivot.
- Pick median as the pivot.

The key process in quickSort is a partition(). The target of partitions is, given an array and an element x of an array as the pivot, put x at its correct position in a sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time

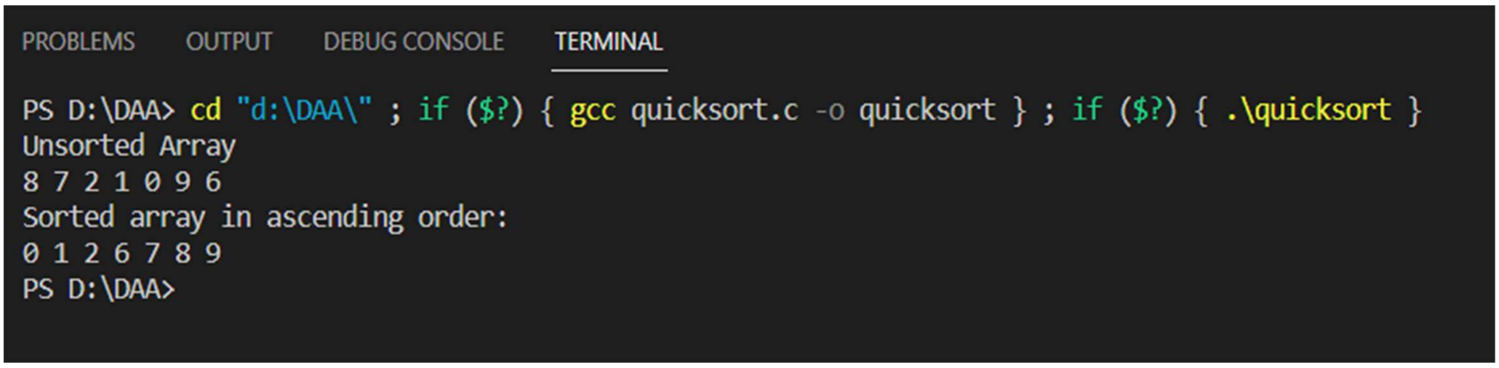
• Quick Sort Algorithm :-

```
 QUICKSORT (array A, start, end)
1. {
2. if (start < end)
3. {
4. p = partition(A, start, end)
5. QUICKSORT (A, start, p - 1)
6. QUICKSORT (A, p + 1, end)
7. }
8. }

 PARTITION (array A, start, end)
9. {
10. pivot ? A[end]
11. i ? start-1
12. for j ? start to end -1 {
13. do if (A[j] < pivot) {
14. then i ? i + 1
15. swap A[i] with A[j]
16. }}
17. swap A[i+1] with A[end]
18. return i+1
19.}
```

Program:

```
#include <stdio.h>
void swap(int *a, int *b) {
 int t = *a;
 *a = *b;
 *b = t;
}
int partition(int array[], int low, int high) {

 int pivot = array[high];

 int i = (low - 1);
 for (int j = low; j < high; j++) {
 if (array[j] <= pivot) {

 i++;

 swap(&array[i], &array[j]);
 }
 }
 swap(&array[i + 1], &array[high]);

 return (i + 1);
}
void quickSort(int array[], int low, int high) {
 if (low < high) {

 int pi = partition(array, low, high);

 quickSort(array, low, pi - 1);

 quickSort(array, pi + 1, high);
 }
}
void printArray(int array[], int size) {
 for (int i = 0; i < size; ++i) {
 printf("%d ", array[i]);
 }
 printf("\n");
}
int main() {
 int data[] = {8, 7, 2, 1, 0, 9, 6};

 int n = sizeof(data) / sizeof(data[0]);

 printf("Unsorted Array\n");
 printArray(data, n);

 quickSort(data, 0, n - 1);

 printf("Sorted array in ascending order: \n");
 printArray(data, n);
}
```

- **Output :-**



```
PS D:\DAA> cd "d:\DAA\" ; if ($?) { gcc quicksort.c -o quicksort } ; if ($?) { .\quicksort }
Unsorted Array
8 7 2 1 0 9 6
Sorted array in ascending order:
0 1 2 6 7 8 9
PS D:\DAA>
```