AMRITA SCHOOL OF COMPUTING

DESIGN AND ANALYSIS OF ALGORITHMS

(23CSE211)

Name: Y.V.S.Likesh

Roll No.: CH.SC.U4CSE24152

Class: BTech (CSE-B)

School: Amrita School of Computing,
Chennai Campus.

LAB-4

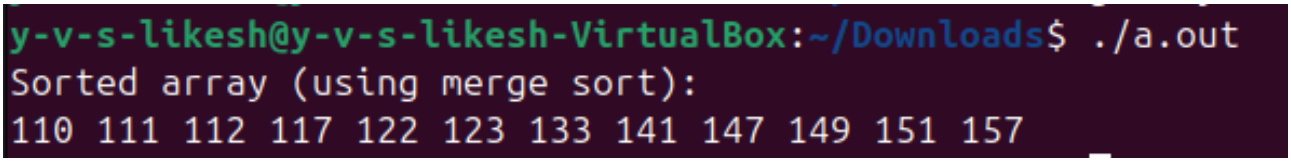
1) Solving a array of integers using Merge Sort.

Code:

```c
#include<stdio.h>
void merge(int a[], int l, int m, int r){
    int i = l, j = m + 1, k = l;
    int b[100];
    while(i <= m && j <= r){
        if (a[i] <= a[j]){
            b[k] = a[i];
            i++;
        }
        else{
            b[k] = a[j];
            j++;
        }
        k++;
    }
    while(i <= m){
        b[k] = a[i];
        i++;
        k++;
    }
    while(j <= r){
        b[k] = a[j];
        j++;
        k++;
    }
    for(i = l; i <= r; i++){
        a[i] = b[i];
    }
}
```

```c
void mergesort(int a[], int l, int r){
    if(l < r){
        int m = (l + r) / 2;
        mergesort(a, l, m);
        mergesort(a, m + 1, r);
        merge(a, l, m, r);
    }
}
int main(){
    int a[] = {157,110,147,122,111,149,151,141,123,112,117,133};
    int n = 12;
    mergesort(a, 0, n - 1);
    printf("Sorted array (using merge sort):\n");
    for(int i = 0; i < n; i++){
        printf("%d ", a[i]);
    }
    printf("\n");
    return 0;
}
```

Output:



Space Complexity:

The space complexity of this program is O(n log n). Because the function mergesort () is a recursively divides the array into two halves and merge () function uses loops for traverse.

Time Complexity:

The space complexity of this program is O(n). Because it uses an array.

2) Solving an array of integers using Quick Sort.

Code:

```c
#include <stdio.h>
int partition(int a[], int low, int high){
    int pivot = a[low];
    int i = low + 1;
    int j = high;
    int temp;
    while(i <= j){
        while(i <= high && a[i] <= pivot){
            i++;
        }
        while(a[j] > pivot){
            j--;
        }
        if(i < j){
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
    temp = a[low];
    a[low] = a[j];
    a[j] = temp;
    return j;
}
```

```c
void quicksort(int a[], int low, int high){
    if(low < high){
        int p = partition(a, low, high);
        quicksort(a, low, p - 1);
        quicksort(a, p + 1, high);
    }
}
int main(){
    int a[] = {157,110,147,122,111,149,151,141,123,112,117,133};
    int n = 12;
    quicksort(a, 0, n - 1);
    printf("Sorted array (using quick sort):\n");
    for(int i = 0; i < n; i++){
        printf("%d ", a[i]);
    }
    printf("\n");
    return 0;
}
```

Output:

```
y-v-s-likesh@y-v-s-likesh-VirtualBox:~/Downloads$ ./a.out
Sorted array (using quick sort):
110 111 112 117 122 123 133 141 147 149 151 157
```

Space Complexity:

The space complexity of this program is O(n log n) (for avg. case), O(n^2) (for worst case). Because partition () function scans the array once and quicksort () function is recursive on subarrays.

Time Complexity:

The space complexity of this program is O(log n) (for avg. case), O(n) (for worst case). Because there is no extra array used in the program and space is consumed only by recursive function calls.