SCHOOL OF
COMPUTING

M DHEERAJ VAMSI KRISHNA

CH.SC.U4CSE24127

WEEK 3

Design and Analysis of Algorithm(23CSE211)

DAA WEEK 3 TASK

QUICK SORT:

CODE:

```c
// CH.SC.U4CSE24127
#include <stdio.h>

void quickSort(int a[], int low, int high);
int partition(int a[], int low, int high);

int main()
{
    int a[] = {157,110,147,122,111,149,151,141,123,112,117,133};
    int n = 12, i;

    quickSort(a, 0, n - 1);

    printf("Sorted array using Quick Sort:\n");
    for(i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}

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 partition(int a[], int low, int high)
{
    int pivot = a[high];
    int i = low - 1, j, temp;

    for(j = low; j < high; j++)
    {
        if(a[j] < pivot)
        {
            i++;
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }

    temp = a[i + 1];
    a[i + 1] = a[high];
    a[high] = temp;

    return i + 1;
}
```

OUTPUT:

```
dheeraj@DheerajG15:~$ nano quicksort.c
dheeraj@DheerajG15:~$ gcc quicksort.c -o quicksort
dheeraj@DheerajG15:~$ ./quicksort
Sorted Array using Quick Sort:
110 111 112 117 122 123 133 141 147 149 151 157 dheeraj@DheerajG15:~$
```

TIME , SPACE COMPLEXITY & JUSTIFICATION

Time Complexity

Best: $O(n \log n)$

Average: $O(n \log n)$

Worst: $O(n^2)$

Space Complexity

- **Best & Average:** $O(\log n)$
- **Worst:** $O(n)$

Time Complexity Justification:

Quick sort divides the array using a pivot and sorts the two parts recursively.
If the division is balanced, it takes log n levels giving $O(n \log n)$.
If badly divided, it leads to n levels giving $O(n^2)$.

Space Complexity Justification:

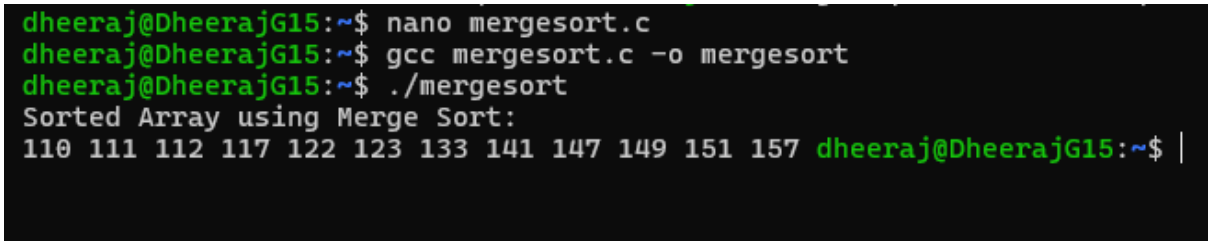
Quick sort uses only recursion stack for function calls.
Balanced recursion needs log n space, but worst case needs n.

MERGE SORT:

CODE:

```c
// CH.SC.U4CSE24127
#include <stdio.h>

void mergeSort(int a[], int l, int r);
void merge(int a[], int l, int m, int r);

int main()
{
    int a[] = {157,110,147,122,111,149,151,141,123,112,117,133};
    int n = 12, i;

    mergeSort(a, 0, n - 1);

    printf("Sorted array using Merge Sort:\n");
    for(i = 0; i < n; i++)
        printf("%d ", a[i]);

    return 0;
}

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);
    }
}

void merge(int a[], 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] = a[l + i];
    for(j = 0; j < n2; j++)
        R[j] = a[m + 1 + j];

    i = 0;
    j = 0;
    k = l;

    while(i < n1 && j < n2)
    {
        if(L[i] <= R[j])
            a[k++] = L[i++];
        else
            a[k++] = R[j++];
    }

    while(i < n1)
        a[k++] = L[i++];

    while(j < n2)
        a[k++] = R[j++];
}
```

OUTPUT:



TIME , SPACE COMPLEXITY & JUSTIFICATION

Time Complexity

Best, Average, Worst: O(n log n)

Array is always divided into halves (log n) and all n elements are merged at each level.

Space Complexity

O(n)

Time Complexity Justification:
Merge sort always divides the array into two halves and merges them. There are log n divisions and each merge processes n elements, so time is O(n log n).

Space Complexity Justification:
Extra temporary arrays are used while merging.
So additional memory of size n is required → O(n).