AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY UNDER SECTION 3 OF UGC ACT, 1956
SCHOOL OF COMPUTING

Design s Analysis of Algorithm

CSE211

Date: 6/1/2026

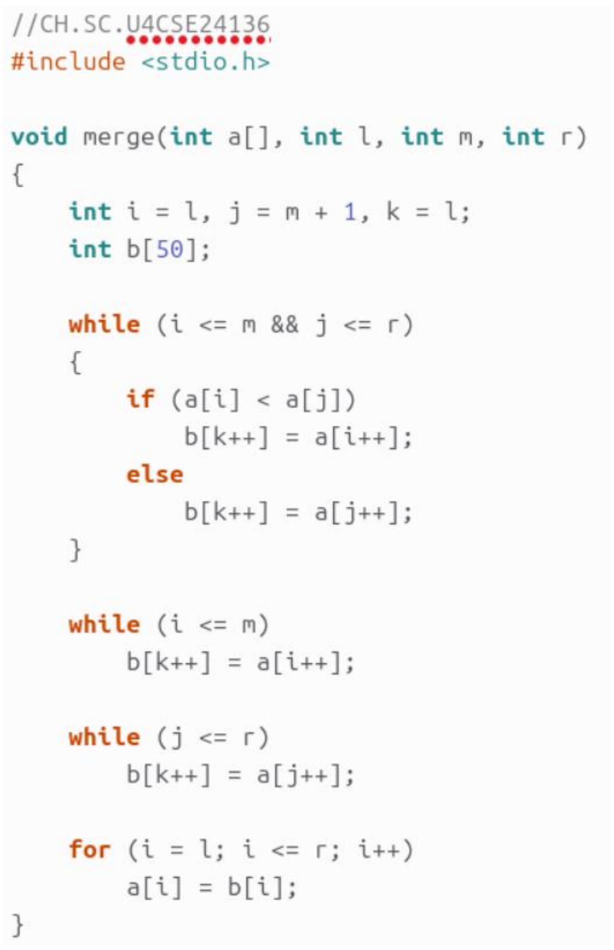
Week-3

CH.SC.U4CSE24136

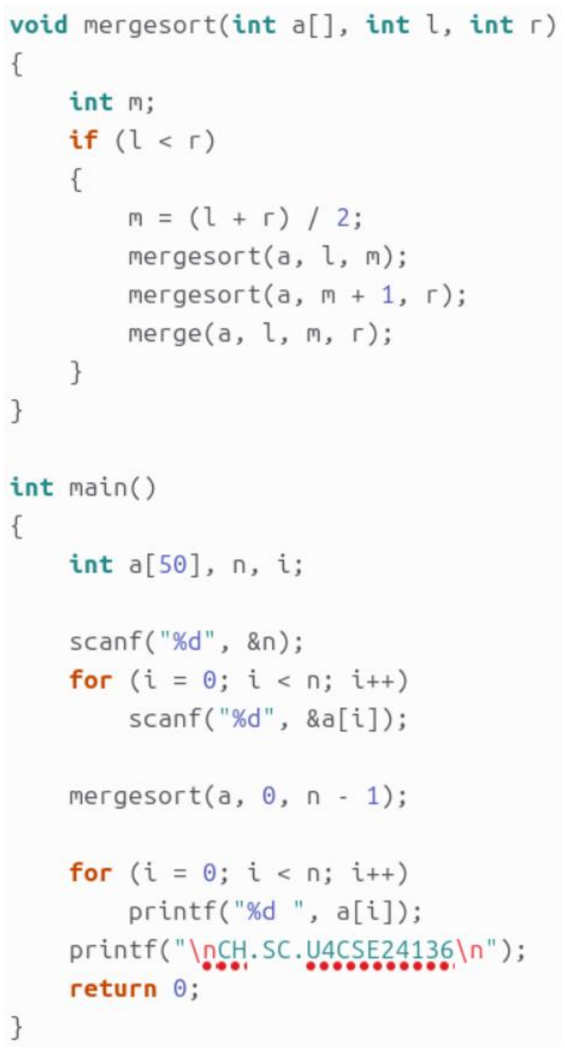
P. Geetesh

Merge Sort

Code:

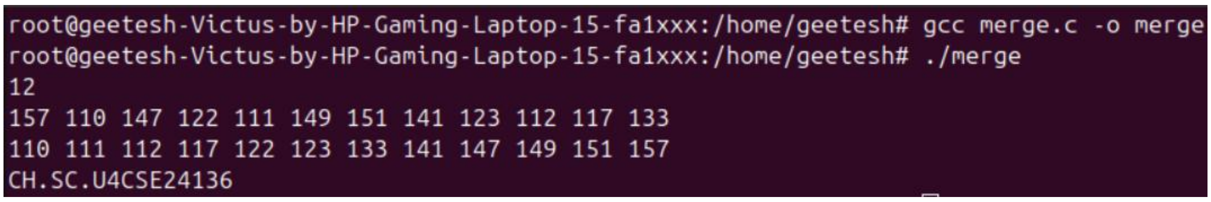


```c
//CH.SC.U4CSE24136
#include <stdio.h>

void merge(int a[], int l, int m, int r)
{
    int i = l, j = m + 1, k = l;
    int b[50];

    while (i <= m && j <= r)
    {
        if (a[i] < a[j])
            b[k++] = a[i++];
        else
            b[k++] = a[j++];
    }

    while (i <= m)
        b[k++] = a[i++];

    while (j <= r)
        b[k++] = a[j++];

    for (i = l; i <= r; i++)
        a[i] = b[i];
}
```



```c
void mergesort(int a[], int l, int r)
{
    int m;
    if (l < r)
    {
        m = (l + r) / 2;
        mergesort(a, l, m);
        mergesort(a, m + 1, r);
        merge(a, l, m, r);
    }
}

int main()
{
    int a[50], n, i;

    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

    mergesort(a, 0, n - 1);

    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\nCH.SC.U4CSE24136\n");
    return 0;
}
```

Output:



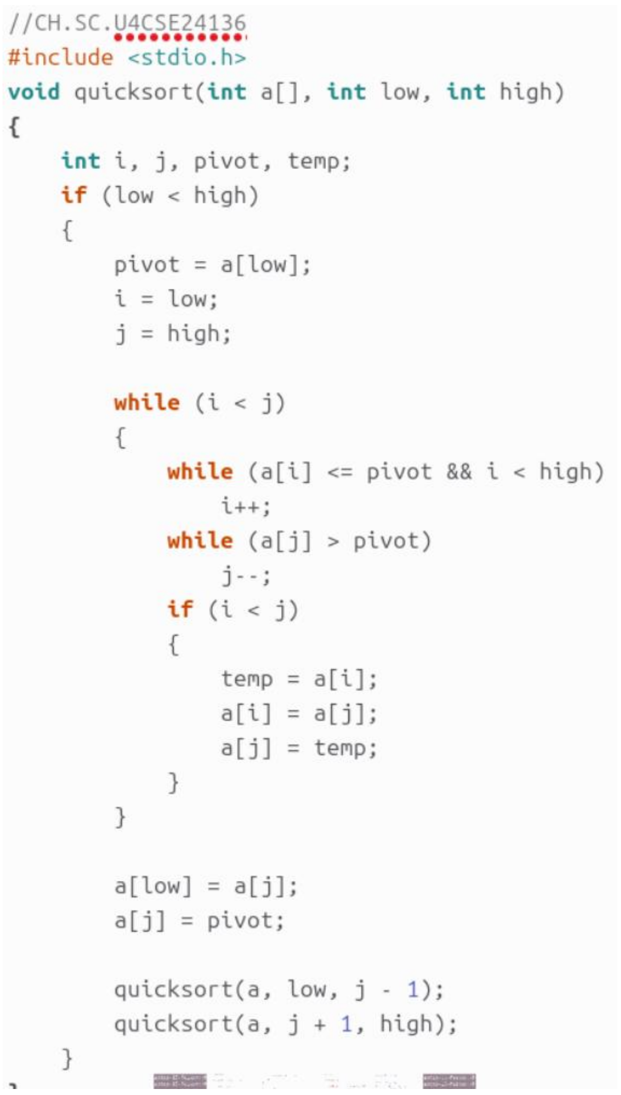
```
root@geetesh-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/geetesh# gcc merge.c -o merge
root@geetesh-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/geetesh# ./merge
12
157 110 147 122 111 149 151 141 123 112 117 133
110 111 112 117 122 123 133 141 147 149 151 157
CH.SC.U4CSE24136
```

Space Complexity and its justification:

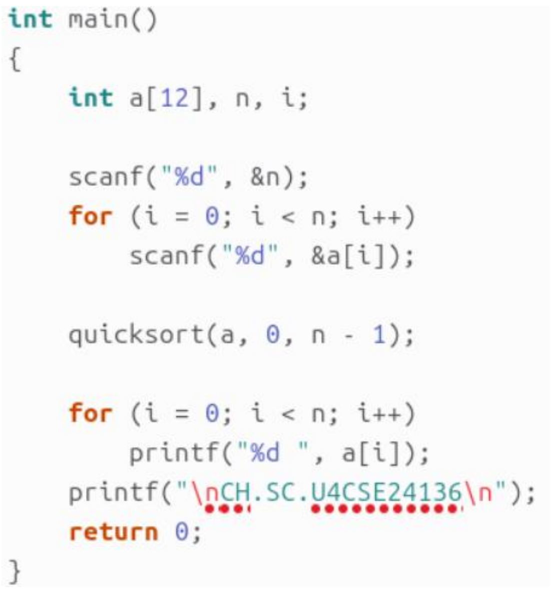
- Time Complexity (All cases): O(n log n)
- Reason: Array is divided into halves recursively → log n levels
- Merge Cost: At each level, n elements are merged → O(n) work
- Space Complexity: O(n)
- Reason: Temporary array b[] is used during merging (recursion stack O(log n) is smaller)

Quick Sort

Code:

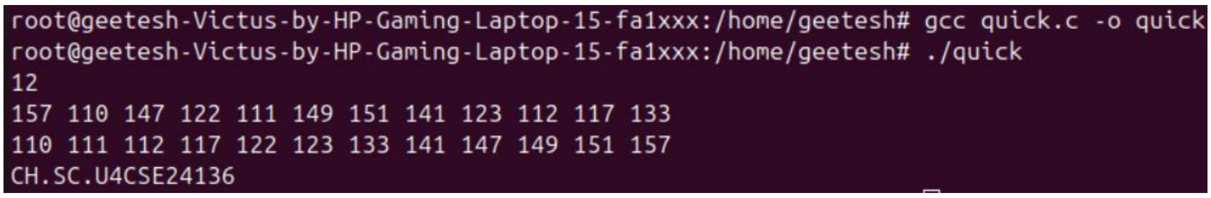


```c
//CH.SC.U4CSE24136
#include <stdio.h>
void quicksort(int a[], int low, int high)
{
    int i, j, pivot, temp;
    if (low < high)
    {
        pivot = a[low];
        i = low;
        j = high;

        while (i < j)
        {
            while (a[i] <= pivot && i < high)
                i++;
            while (a[j] > pivot)
                j--;
            if (i < j)
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }

        a[low] = a[j];
        a[j] = pivot;

        quicksort(a, low, j - 1);
        quicksort(a, j + 1, high);
    }
```



```c
int main()
{
    int a[12], n, i;

    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

    quicksort(a, 0, n - 1);

    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\nCH.SC.U4CSE24136\n");
    return 0;
}
```

Output:



```
root@geetesh-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/geetesh# gcc quick.c -o quick
root@geetesh-Victus-by-HP-Gaming-Laptop-15-fa1xxx:/home/geetesh# ./quick
12
157 110 147 122 111 149 151 141 123 112 117 133
110 111 112 117 122 123 133 141 147 149 151 157
CH.SC.U4CSE24136
```

Space Complexity and its justification:

- ✓ Best & Average Time Complexity: O(n log n)
- ✓ Worst Time Complexity: O(n^2) (when pivot is smallest/largest → unbalanced partitions)
- ✓ Reason: Partitioning takes O(n) and recursion depth is log n (best/avg) or n (worst)
- ✓ Space Complexity (Average): O(log n) due to recursion stack
- ✓ Space Complexity (Worst): O(n) when recursion becomes linear