



AMRITA
VISHWA VIDYAPEETHAM

Amrita School of Computing

23CSE211

Design and Analysis of Algorithms

WEEK – 3 (03 January 2026)

Susendran M

CH.SC.U4CSE24154

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF COMPUTING

CHENNAI

1. Merge Sorting

Merge Sort is a Divide and Conquer sorting algorithm.

The given list is divided repeatedly into two halves until each sublist has one element.

Single elements are merged back in sorted order.

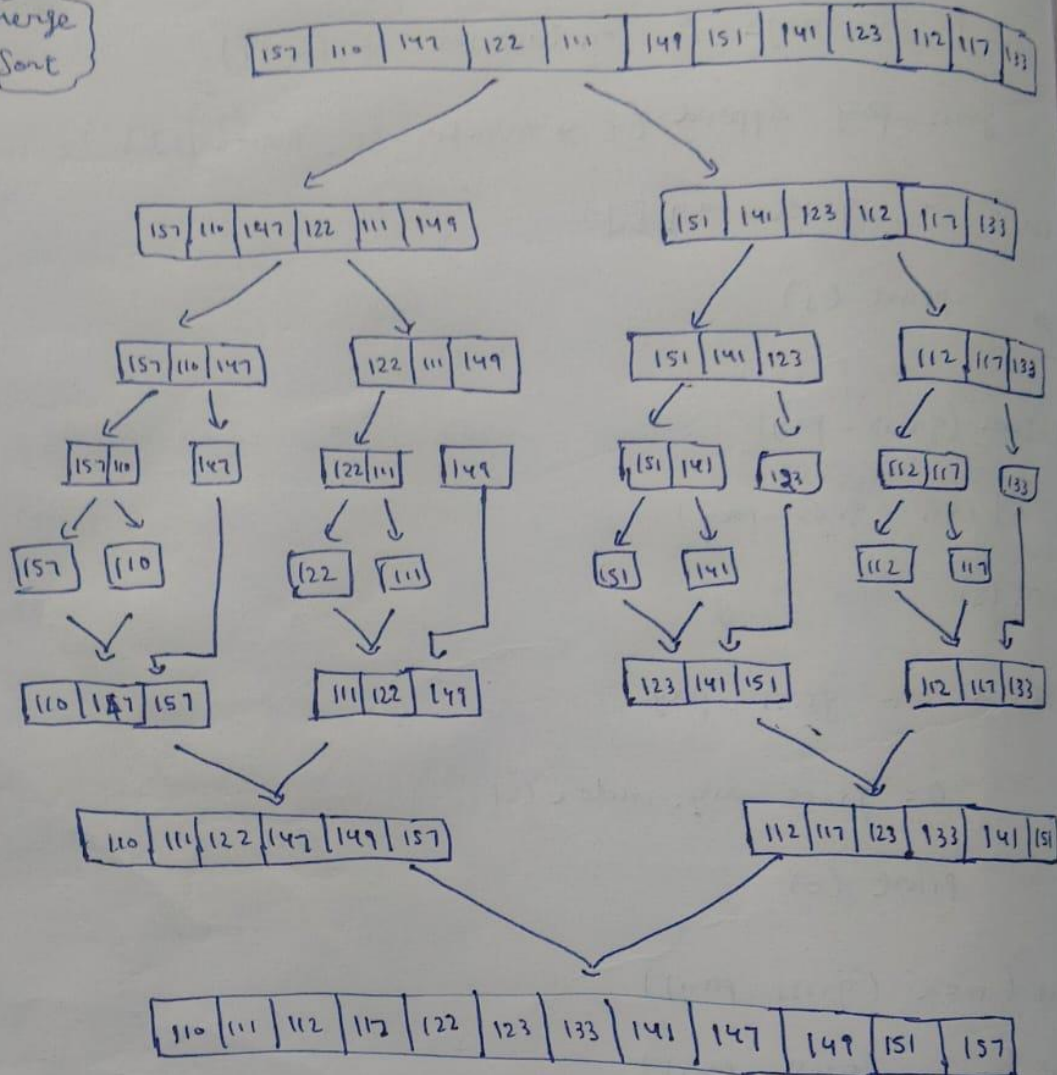
During merging, elements from two sorted sublists are compared and placed in order.

This process continues until the entire list becomes sorted.

3/26

List = [157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133]

Merge
Sort



Code:

```
#include <stdio.h>
```

```
void merge (int arr[], int low, int mid, int high)
```

```
{
```

```
    int i=low, j=mid+1, k=0;
```

```
    int temp[50];
```

```
    while (i <= low && j <= high)
```

```

while (i <= mid && j <= high)
{
    if (arr[i] < arr[j])
    {
        temp[k] = arr[i];
        i++;
    }
    else
    {
        temp[k] = arr[j];
        j++;
    }
    k++;
}

```

```

while (i <= mid)
{
    temp[k] = arr[i];
    i++;
    k++;
}

```

```

while (j <= high)
{
    temp[k] = arr[j];
    j++;
    k++;
}

```

```

for (i = low, k = 0; i <= high; i++, k++)
{
    arr[i] = temp[k];
}

```

```

void mergeSort (int arr[], int low, int high)
{
    int mid;
    if (low < high)
    {
        mid = (low + high) / 2;
    }
}

```

```

mergeSort(arr, low, mid);
mergeSort(arr, mid+1, high);
mergeSort(arr, low, mid, high);
}

int main()
{
    int arr[] = {157, 110, 147, 122, 116, 148, 151, 141, 123, 112, 147, 133};
    int i;
    mergeSort(arr, 0, 11);
    printf("Sorted Array");
    for(i=0; i<12; i++)
    {
        printf("%d ", arr[i]);
    }
    return 0;
}

```

Time Complexity = $O(n \log n)$

Space complexity = $O(1)$

2. Quick Sorting

Quick Sort is a Divide and Conquer sorting algorithm.

A pivot element is selected (can be any element).

The array is partitioned such that elements less than the pivot are placed on the left and greater than the pivot on the right.

The pivot is placed in its correct position.

The same process is recursively applied to the left and right subarrays.

Sorting continues until all subarrays have one or no elements.

list = [157, 110, 147, 122, 111, 149, 151, 141, 123, 112, 117, 133]

Quick Sort

157 110 147 122 111 149 151 141 123 112 117 133
↓
pivot

110 147 122 111 149 151 141 123 112 117 133 157
↓
pivot

110 147 122 111 149 151 141 123 112 117 133 157
↓
pivot

110 147 122 111 149 151 141 123 112 117 133 157
↓
pivot

110 147 112 117 122 141 123 133 149 157
↓
pivot

110 147 111 112 117 122 141 123 133 149 157
↓
pivot

110 147 111 112 117 122 123 133 141 149 157 151
↓
pivot

110 147 111 112 117 122 123 133 141 157 149 151
↓
pivot

110 147 111 112 117 122 123 133 141 157

110 147 111 112 117 122 123 133 141 157 149 151

110 111 112 117 122 123 133 141 147 149 151 157

Code:

```
#include <stdio.h>

int quick (int arr[], int low, int high)
{
    int pivot = arr[low];
    int i = low + 1;
    int j = high;
    int temp;

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

    temp = arr[low];
    arr[low] = arr[j];
    arr[j] = temp;

    return j;
}

void quicksort (int arr[], int low, int high)
{
    int p;

    if (low < high)
    {
        p = quick (arr, low, high);
    }
}
```

```

        quicksort (arr, low, p-1);
        quicksort (arr, p+1, high);
    }
}

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

    quicksort (arr, 0, 11);

    printf ("Sorted Array");

    for (int i=0; i<12; i++)
    {
        printf (" %d ", arr[i]);
    }

    return 0;
}

```

Time complexity = $O(n \log n)$ for Best & Average case
 $O(n^2)$ for worst case

Space complexity = $O(1)$