Merge Sort

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. **The merge() function** is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one. See following C implementation for details.

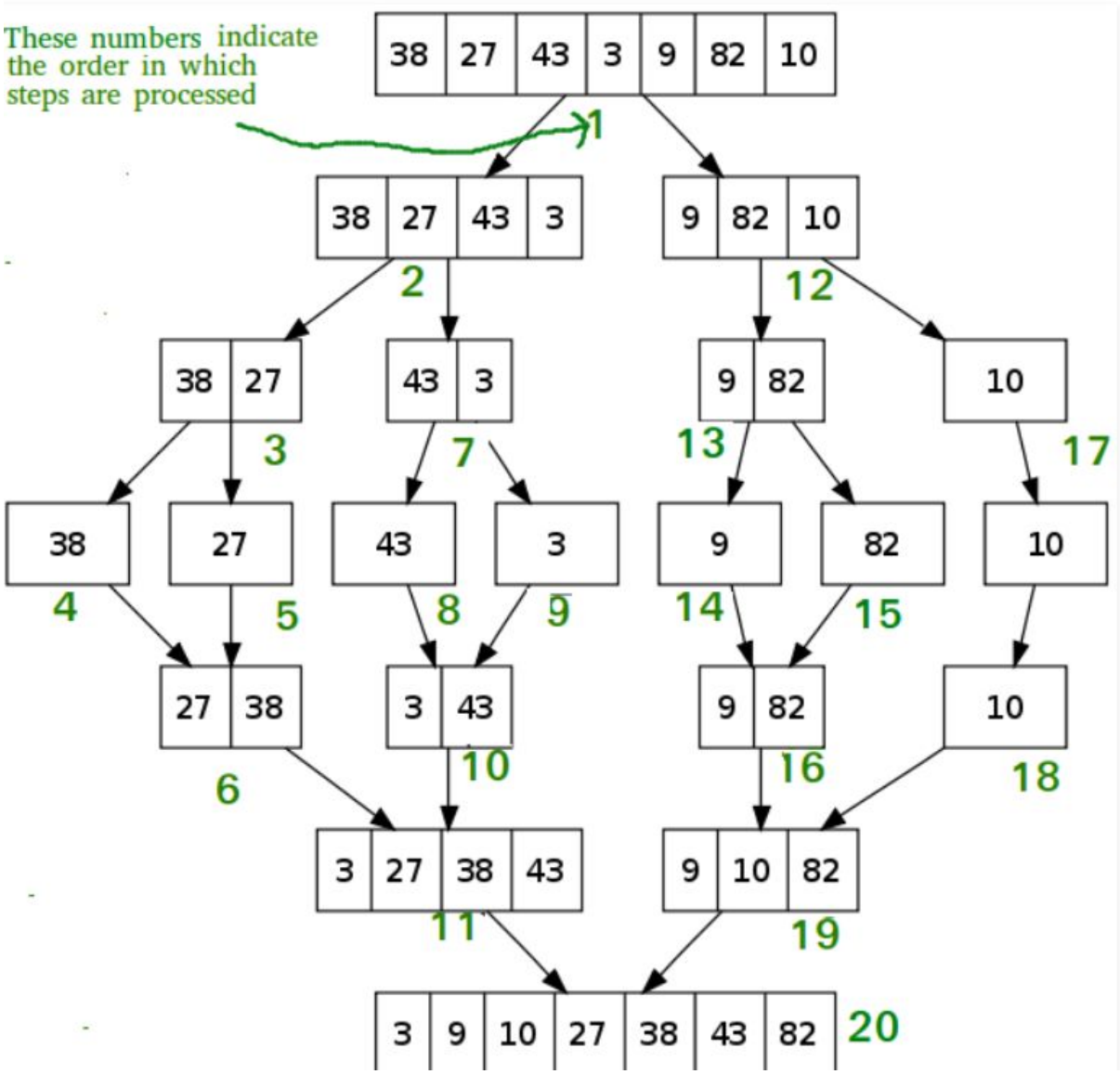
Merge sort is a sorting technique based on divide and conquer technique. With the worst-case time complexity being O(n log n), it is one of the most respected algorithms.

```
// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there
       are any */
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }
```

```
    /* Copy the remaining elements of R[], if there
       are any */
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* l is for left index and r is right index of the
   sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}




/* UTILITY FUNCTIONS */
/* Function to print an array */
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

/* Driver program to test above functions */
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;
}
```

Discussion



Quick Sort

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

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in 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.

```
#include <stdio.h>

void quick_sort(int[],int,int);
int partition(int[],int,int);

int main()
{
        int a[50],n,i;
        printf("How many elements?");
        scanf("%d",&n);
        printf("\nEnter array elements:");

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

        quick_sort(a,0,n-1);
        printf("\nArray after sorting:");

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

        return 0;
```

```
}

void quick_sort(int a[],int l,int u)
{
	int j;
	if(l<u)
	{
		j=partition(a,l,u);
		quick_sort(a,l,j-1);
		quick_sort(a,j+1,u);
	}
}

int partition(int a[],int l,int u)
{
	int v,i,j,temp;
	v=a[l];
	i=l;
	j=u+1;

	do
	{
		do
			i++;

		while(a[i]<v&&i<=u);

		do
			j--;
		while(v<a[j]);

		if(i<j)
		{
			temp=a[i];
			a[i]=a[j];
			a[j]=temp;
		}
	}while(i<j);

	a[l]=a[j];
```

```
        a[j]=v;

        return(j);
}
```

Discussion