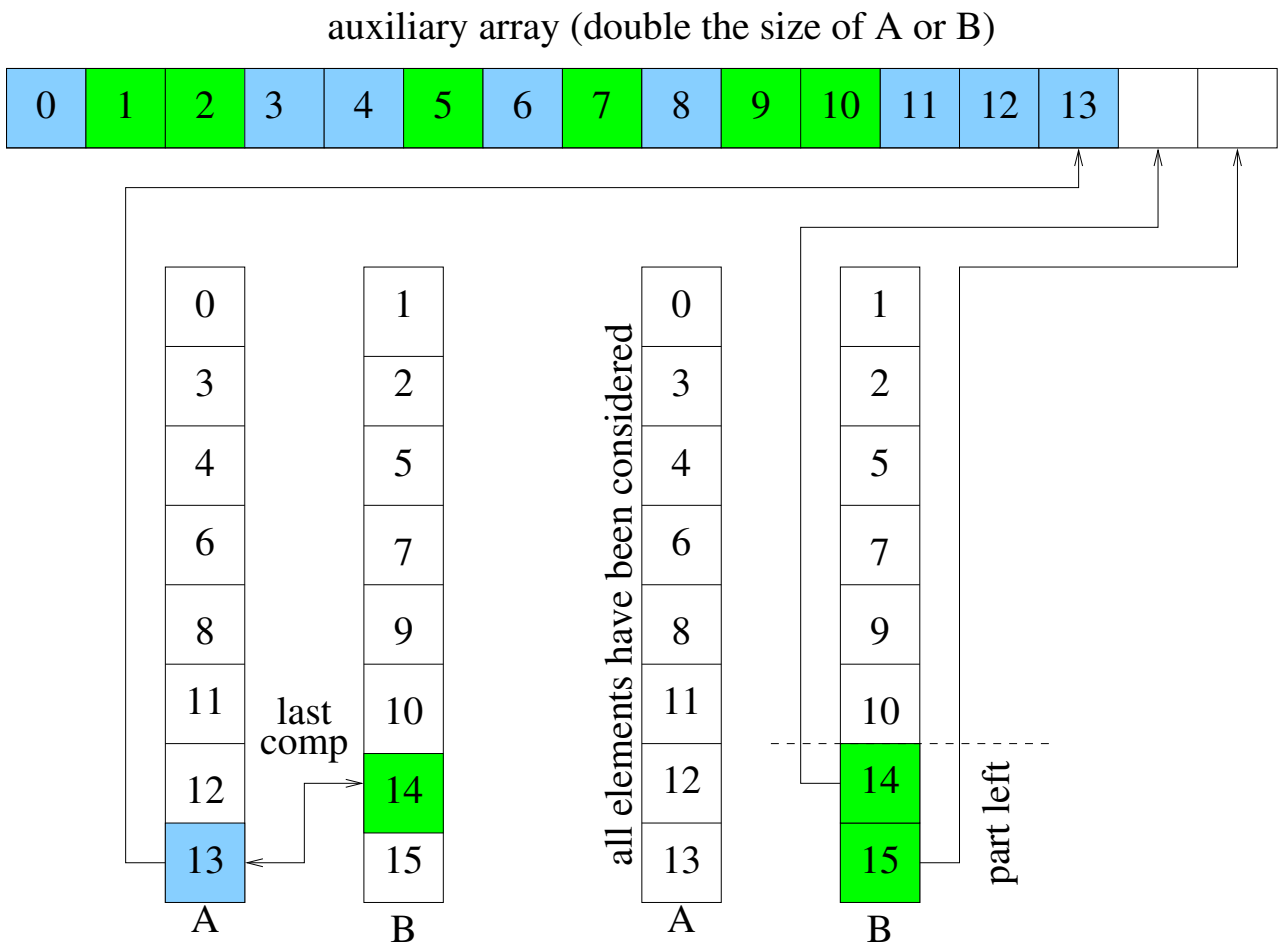
Merge Sort

Merging



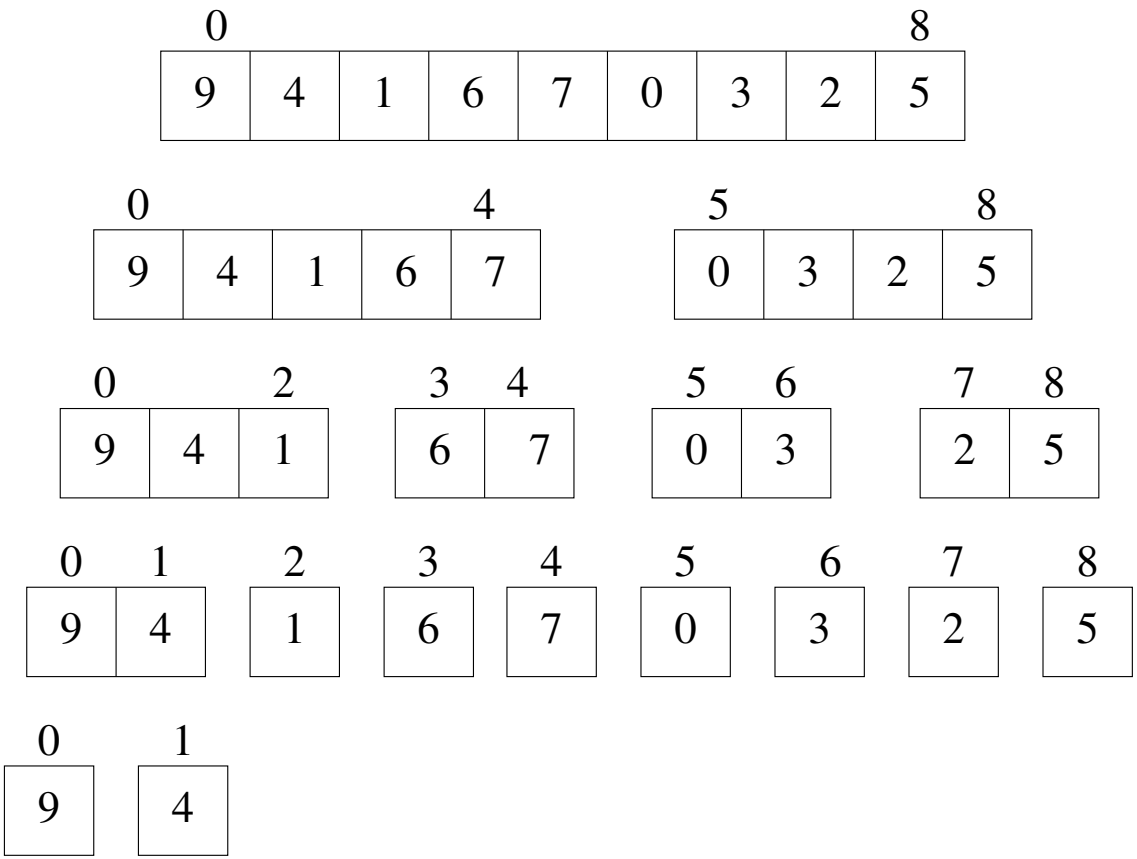
Merge Sort

Merging

- Merge sort required merging of a pair of sorted arrays.
- Merging is simple
  - Take two elements one from each array A and B.
  - Compare them and place smaller of two (say from A) in sorted list.
  - Take next element from A and compare with element in hand (from B).
  - Repeat until one of the array is exhausted.
  - Now place all remaining elements of non-empty array one by one.

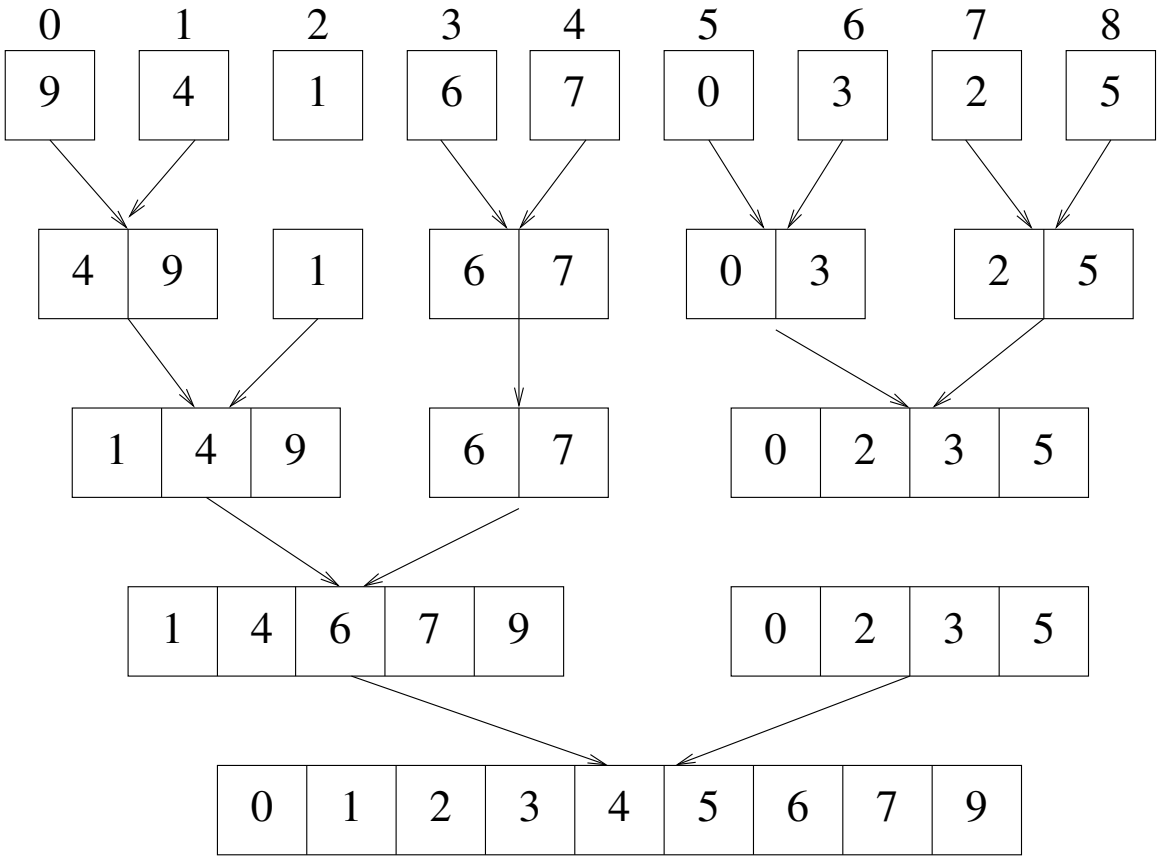
Merge Sort

Merge Sort (Splitting)



Merge Sort

Merge Sort (Merging)



Merge Sort

Merging Sort (Code for Merging)

```
void merge(int a[], int l, int m, int h)
{
    int i, j, k;
    int b[l + h + 1]; // Auxiliary array

    for (i = l; i <= h; i++)
        b[i] = a[i];

    /* Rest of the program appear in next slide */
}
```

Merge Sort

Merge Sort (Code for Merging)

```
/* Copy back the greater of the two elements of    *
 * two sorted halves into next position of array a */

for (i = k = l, j = m + 1; i <= m && j <= h; k++) {
    if (b[i] <= b[j]) {
        a[k] = b[i]; // a[k] overwritten by b[i]
        i++;
    }
    else {
        a[k] = b[j]; // a[k] overwritten by b[j]
        j++;
    }
}

// Copy back rest of the elements (if any)

for (; i <= m; i++, k++)
    a[k] = b[i];
```

Merge Sort

Explanation for Merging

- Why copy back is required for left half of array only?
  - If every element from left half of array is smaller than the smallest element of right half (no exchange takes place).
  - `i > m` on exit of first `for` imply the above.
- But if some elements from right half are smaller than certain elements of left half then (exchange will take place)
  - Implying `i <= m` on exit of first `for` loop.
  - In other words, some larger elements from left half are left out.
  - Whereas all elements of right half have been accounted for.
  - The left out elements of left half should go to positions from `k` onward.

Merge Sort

Merge Sort (Recursion)

```
void mergeSort(int a[], int l, int h) {
    int m;

    if (l < h) {
        m = (l+h)/2;              // Find the middle of the array
        mergeSort(a, l, m);       // Sort the first half
        mergeSort(a, m+1, h);     // Sort the next half
        merge(a, l, m, h);        // Merge the sorted halves
    }
    return;
}
```

Merge Sort

Main Function

```
#include <stdio.h>
#define N 10
int main() {
    int a[N];
    int n = sizeof(a)/sizeof(a[0]);

    generateNumbers(a, n);
    printf("Unsorted input \n");
    printArray(a, n);

    mergeSort(a, 0, n-1);
    printf("Sorted output \n");
    printArray(a, n);
}
```

Quick Sort

General Idea

- It also based on principle of divide and conquer.
- The unsorted array is first partitioned using an element of the array.
- For simplicity use the first element for partitioning.
- Array is partitioned such that
  - All elements larger than partitioning element appear to its right.
  - All elements smaller than the partitioning element appear to its left.
- So, partitioning element gets into correct place.

Quick Sort

Splitting