December 12, 2021 ▪ Arrays / Data Structure

Merge two Sorted Arrays Without Extra Space

Problem statement: Given two sorted arrays **arr1[]** and **arr2[]** of sizes **n** and **m** in non-decreasing order. Merge them in sorted order. Modify arr1 so that it contains the first N elements and modify arr2 so that it contains the last M elements.

Examples:

```
Example 1:

Input:
n = 4, arr1[] = [1 4 8 10]
m = 5, arr2[] = [2 3 9]

Output:
arr1[] = [1 2 3 4]
arr2[] = [8 9 10]

Explanation:
After merging the two non-decreasing arrays, we get, 1,2,3,4,8,9,10.

Example2:

Input:
n = 4, arr1[] = [1 3 5 7]
m = 5, arr2[] = [0 2 6 8 9]

Output:
arr1[] = [0 1 2 3]
arr2[] = [5 6 7 8 9]

Explanation:
After merging the two non-decreasing arrays, we get, 0 1 2 3 5 6 7 8 9.
```

Solution:

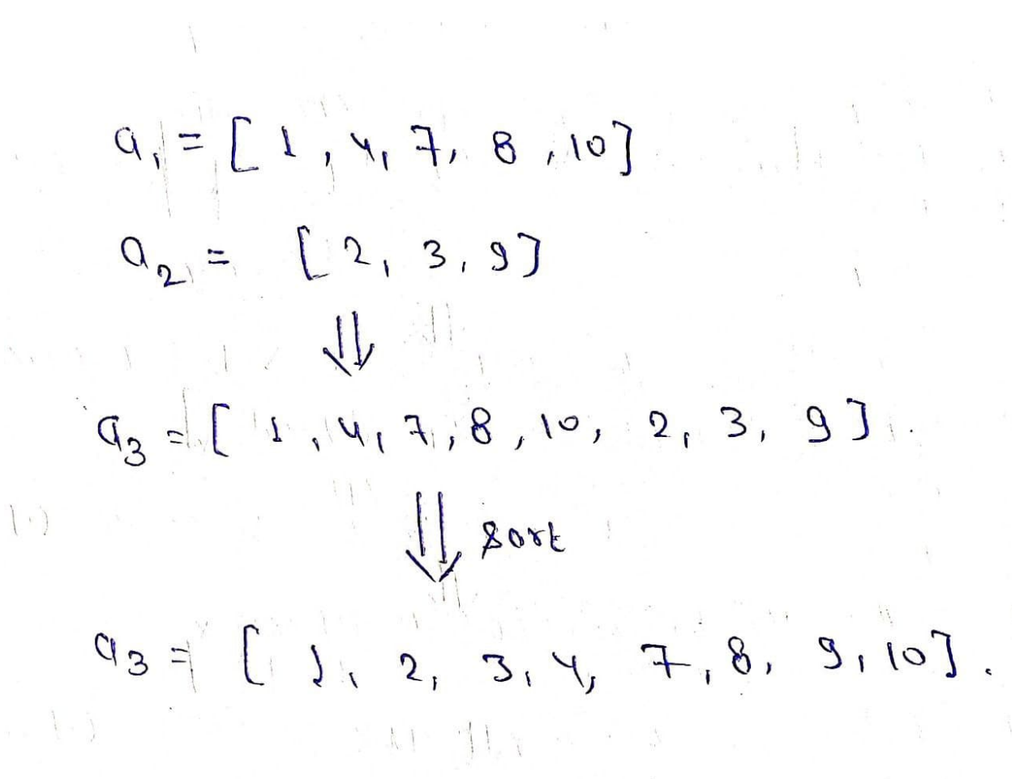
Disclaimer. Don't jump directly to the solution, try it out yourself first.

Solution1: Brute Force-

Intuition: We can use a new array of size n+m and put all elements of arr1 and arr2 in it, and sort it. After sorting it, but all the elements in arr1 and arr2.

Approach:

- Make an arr3 of size n+m.
- Put elements arr1 and arr2 in arr3.
- Sort the arr3.
- Now first fill the arr1 and then fill remaining elements in arr2.



Recent Posts

Code:

C++ Code

```
#include<bits/stdc++.h>
using namespace std;
void merge(int arr1[], int arr2[], int n, int m) {
    int arr3[n+m];
    int k = 0;
    for (int i = 0; i < n; i++) {
      arr3[k++] = arr1[i];
    }
    for (int i = 0; i < m; i++) {
      arr3[k++] = arr2[i];
    }
    sort(arr3,arr3+m+n);
    k = 0;
    for (int i = 0; i < n; i++) {
      arr1[i] = arr3[k++];
    }
    for (int i = 0; i < m; i++) {
      arr2[i] = arr3[k++];
    }

  }
int main() {
    int arr1[] = {1,4,7,8,10};
    int arr2[] = {2,3,9};
    cout<<"Before merge:"<<endl;
    for (int i = 0; i < 5; i++) {
      cout<<arr1[i]<<" ";
    }
    cout<<endl;
    for (int i = 0; i < 3; i++) {
      cout<<arr2[i]<<" ";
    }
    cout<<endl;
    merge(arr1, arr2, 5, 3);
    cout<<"After merge:"<<endl;
    for (int i = 0; i <5; i++) {
      cout<<arr1[i]<<" ";
    }
    cout<<endl;
    for (int i = 0; i < 3; i++) {
      cout<<arr2[i]<<" ";
    }

  }
```

Output:

Before merge:

1 4 7 8 10

2 3 9

After merge:

1 2 3 4 7

8 9 10

Time complexity: O(n*log(n))+O(n)+O(n)

Space Complexity: O(n)

Java Code

```
import java.util.*;

public class tuf {

  public static void main(String[] args) {
    int arr1[] = {1,4,7,8,10};
    int arr2[] = {2,3,9};
    System.out.println("Before merge:");
    for (int i = 0; i < arr1.length; i++) {
      System.out.print(arr1[i] + " ");
    }
    System.out.println();
    for (int i = 0; i < arr2.length; i++) {
      System.out.print(arr2[i] + " ");
    }
    System.out.println();
    merge(arr1, arr2, arr1.length, arr2.length);
    System.out.println("After merge:");
    for (int i = 0; i < arr1.length; i++) {
      System.out.print(arr1[i] + " ");
    }
    System.out.println();
    for (int i = 0; i < arr2.length; i++) {
      System.out.print(arr2[i] + " ");
    }

  }
```

```
  static void merge(int[] arr1, int arr2[], int n, int m) {
    int arr3[] = new int[n + m];
    int k = 0;
    for (int i = 0; i < n; i++) {
      arr3[k++] = arr1[i];
    }
    for (int i = 0; i < m; i++) {
      arr3[k++] = arr2[i];
    }
    Arrays.sort(arr3);
    k = 0;
    for (int i = 0; i < n; i++) {
      arr1[i] = arr3[k++];
    }
    for (int i = 0; i < m; i++) {
      arr2[i] = arr3[k++];
    }

  }
}
```

Output:

Before merge:

1 4 7 8 10

2 3 9

After merge:

1 2 3 4 7

8 9 10

Time complexity: O(n*log(n))+O(n)+O(n)

Space Complexity: O(n)

Python Code

```
from typing import List
def merge(arr1: List[int], arr2: List[int], n: int, m: int) -> None:
    arr3 = [0] * (n + m)
    k = 0
    for i in range(n):
        arr3[k] = arr1[i]
        k += 1
    for i in range(m):
        arr3[k] = arr2[i]
        k += 1
    arr3.sort()
    k = 0
    for i in range(n):
        arr1[i] = arr3[k]
        k += 1
    for i in range(m):
        arr2[i] = arr3[k]
        k += 1




if __name__ == "__main__":
    arr1 = [1, 4, 7, 8, 10]
    arr2 = [2, 3, 9]
    print("Before merge:")
    print(*arr1)
    print(*arr2)
    merge(arr1, arr2, 5, 3)
    print("After merge:")
    print(*arr1)
    print(*arr2)
```

Output:

Before merge:

1 4 7 8 10

2 3 9

After merge:

1 2 3 4 7

8 9 10

Time complexity: O(n*log(n))+O(n)+O(n)

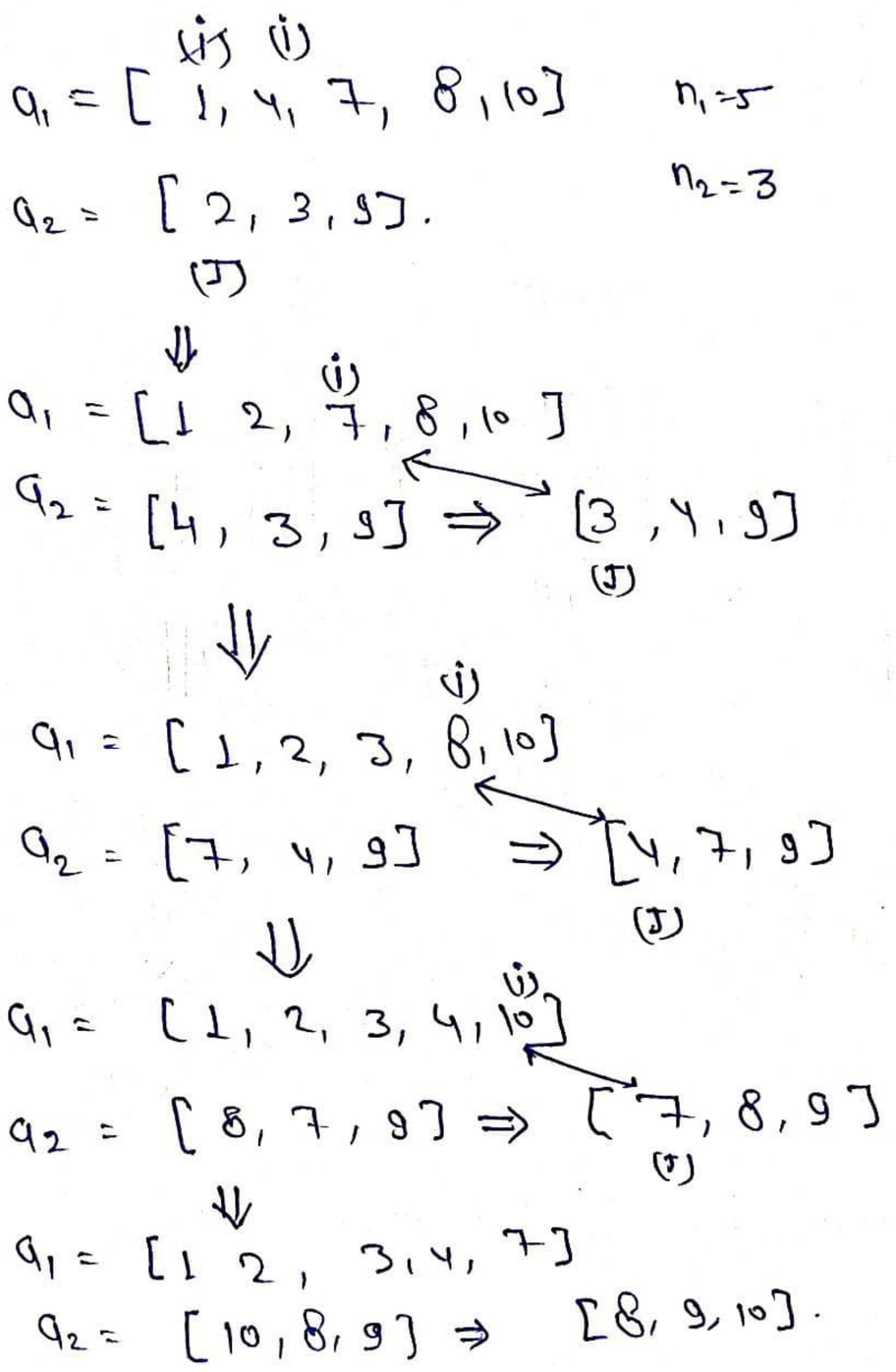
Space Complexity: O(n)

Solution 2: Without using space-

Intuition: We can think of Iterating in arr1 and whenever we encounter an element that is greater than the first element of arr2, just swap it. Now rearrange the arr2 in a sorted manner, after completion of the loop we will get elements of both the arrays in non-decreasing order.

Approach:

- Use a for loop in arr1.
- Whenever we get any element in arr1 which is greater than the first element of arr2,swap it.
- Rearrange arr2.
- Repeat the process.



Code:

C++ Code

```
#include<bits/stdc++.h>

using namespace std;
void merge(int arr1[], int arr2[], int n, int m) {
  // code here
  int i, k;
  for (i = 0; i < n; i++) {
    // take first element from arr1
    // compare it with first element of second array
    // if condition match, then swap
    if (arr1[i] > arr2[0]) {
      int temp = arr1[i];
      arr1[i] = arr2[0];
      arr2[0] = temp;
    }

    // then sort the second array
    // put the element in its correct position
    // so that next cycle can swap elements correctly
    int first = arr2[0];
    // insertion sort is used here
    for (k = 1; k < m && arr2[k] < first; k++) {
      arr2[k - 1] = arr2[k];
    }
    arr2[k - 1] = first;
  }
}
int main() {
  int arr1[] = {1,4,7,8,10};
  int arr2[] = {2,3,9};
  cout << "Before merge:" << endl;
  for (int i = 0; i < 5; i++) {
    cout << arr1[i] << " ";
  }
  cout << endl;
  for (int i = 0; i < 3; i++) {
    cout << arr2[i] << " ";
  }
  cout << endl;
  merge(arr1, arr2, 5, 3);
  cout << "After merge:" << endl;
  for (int i = 0; i < 5; i++) {
```

```
    cout << arr1[i] << " ";
  }
  cout << endl;
  for (int i = 0; i < 3; i++) {
    cout << arr2[i] << " ";
  }

}
```

Output:

Before merge:

1 4 7 8 10

2 3 9

After merge:

1 2 3 4 7

8 9 10

Time complexity: O(n*m)

Space Complexity: O(1)

Java Code

```
public class tuf {

  public static void main(String[] args) {
    int arr1[] = {1,4,7,8,10};
    int arr2[] = {2,3,9};
    System.out.println("Before merge:");
    for (int i = 0; i < arr1.length; i++) {
      System.out.print(arr1[i] + " ");
    }
    System.out.println();
    for (int i = 0; i < arr2.length; i++) {
      System.out.print(arr2[i] + " ");
    }
    System.out.println();
    merge(arr1, arr2, arr1.length, arr2.length);
    System.out.println("After merge:");
    for (int i = 0; i < arr1.length; i++) {
      System.out.print(arr1[i] + " ");
    }
    System.out.println();
    for (int i = 0; i < arr2.length; i++) {
      System.out.print(arr2[i] + " ");
    }

  }
  static void merge(int[] arr1, int arr2[], int n, int m) {
    // code here
    int i, k;
    for (i = 0; i < n; i++) {
      // take first element from arr1
      // compare it with first element of second array
      // if condition match, then swap
      if (arr1[i] > arr2[0]) {
        int temp = arr1[i];
        arr1[i] = arr2[0];
        arr2[0] = temp;
      }

      // then sort the second array
      // put the element in its correct position
      // so that next cycle can swap elements correctly
      int first = arr2[0];
      // insertion sort is used here
      for (k = 1; k < m && arr2[k] < first; k++) {
        arr2[k - 1] = arr2[k];
      }
      arr2[k - 1] = first;
    }
  }
}
```

Output:

Before merge:

1 4 7 8 10

2 3 9

After merge:

1 2 3 4 7

8 9 10

Time complexity: O(n*m)

Space Complexity: O(1)

Python Code

```python
from typing import List


def merge(arr1: List[int], arr2: List[int], n: int, m: int) -> None:
    for i in range(n):
        if arr1[i] > arr2[0]:
            arr1[i], arr2[0] = arr2[0], arr1[i]
        first = arr2[0]
        k = 1
        while k < m and arr2[k] < first:
            arr2[k - 1] = arr2[k]
            k += 1
        arr2[k - 1] = first




if __name__ == "__main__":
    arr1 = [1, 4, 7, 8, 10]
    arr2 = [2, 3, 9]
    print("Before merge:")
    print(*arr1)
    print(*arr2)
    merge(arr1, arr2, 5, 3)
    print("After merge:")
    print(*arr1)
    print(*arr2)
```

Output:

Before merge:

1 4 7 8 10

2 3 9

After merge:

1 2 3 4 7

8 9 10

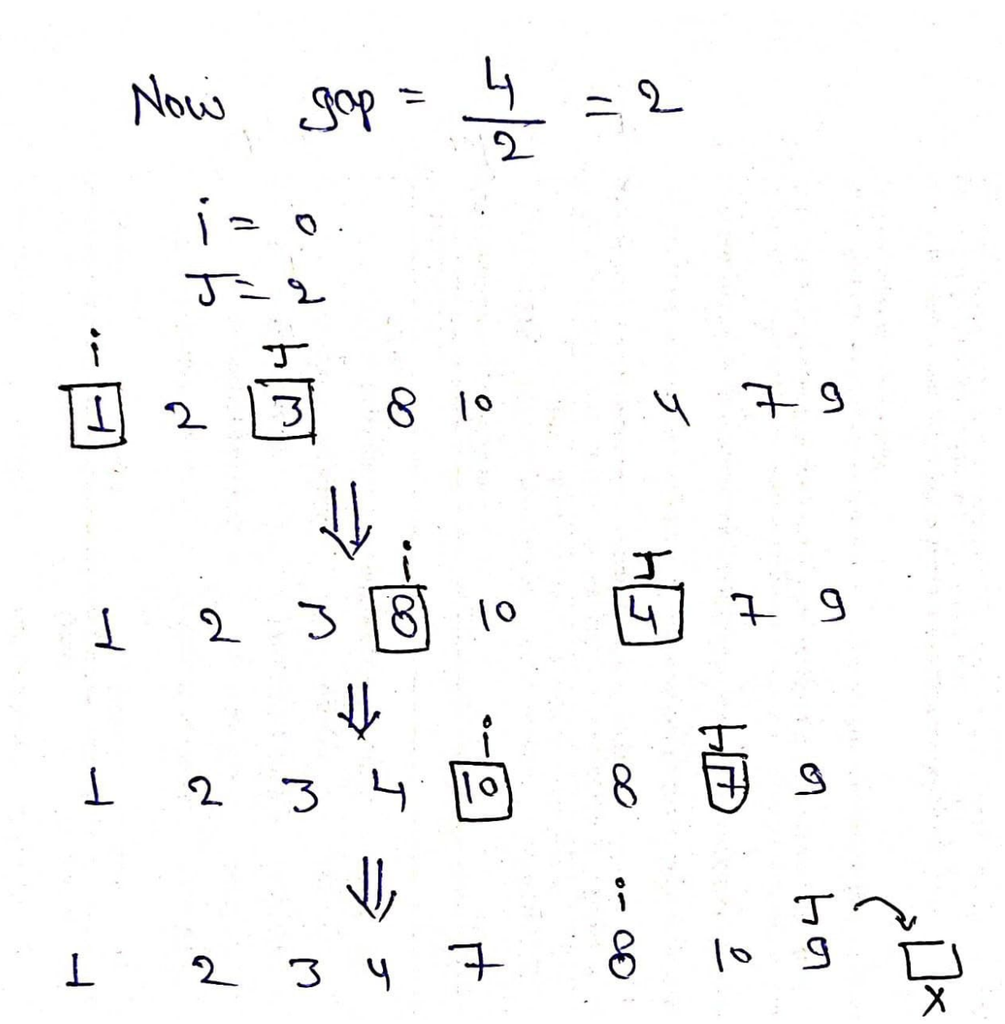
Time complexity: O(n*m)

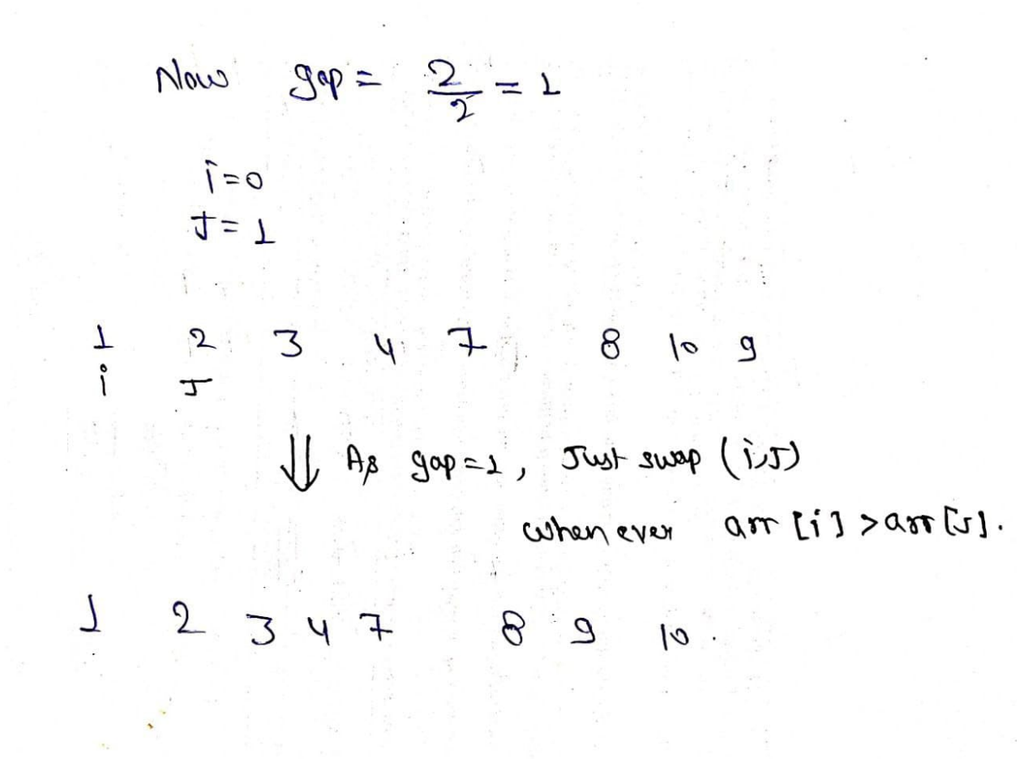
Space Complexity: O(1)

Solution 3: Gap method-

Approach:

- Initially take the gap as (m+n)/2;
- Take as a pointer1 = 0 and pointer2 = gap.
- Run a loop from pointer1 & pointer2 to m+n and whenever arr[pointer2]<arr[pointer1], just swap those.
- After completion of the loop reduce the gap as gap=gap/2.
- Repeat the process until the gap>0.





Code:

C++ Code

```
#include<bits/stdc++.h>
using namespace std;
void merge(int ar1[], int ar2[], int n, int m) {
  // code here
  int gap = ceil((float)(n + m) / 2);
  while (gap > 0) {
    int i = 0;
    int j = gap;
    while (j < (n + m)) {
      if (j < n && ar1[i] > ar1[j]) {
        swap(ar1[i], ar1[j]);
      } else if (j >= n && i < n && ar1[i] > ar2[j - n]) {
        swap(ar1[i], ar2[j - n]);
      } else if (j >= n && i >= n && ar2[i - n] > ar2[j - n]) {
        swap(ar2[i - n], ar2[j - n]);
      }
      j++;
      i++;
    }
    if (gap == 1) {
      gap = 0;
    } else {
      gap = ceil((float) gap / 2);
    }
  }
}
int main() {
  int arr1[] = {1,4,7,8,10};
  int arr2[] = {2,3,9};
  cout << "Before merge:" << endl;
  for (int i = 0; i < 5; i++) {
    cout << arr1[i] << " ";
  }
  cout << endl;
  for (int i = 0; i < 3; i++) {
    cout << arr2[i] << " ";
  }
  cout << endl;
  merge(arr1, arr2, 5, 3);
  cout << "After merge:" << endl;
  for (int i = 0; i < 5; i++) {
    cout << arr1[i] << " ";
  }
  cout << endl;
  for (int i = 0; i < 3; i++) {
    cout << arr2[i] << " ";
  }

}
```

Output:

Before merge:

1 4 7 8 10

2 3 9

After merge:

1 2 3 4 7

8 9 10

Time complexity: O(n+m)

Space Complexity: O(1)

Java Code

```java
import java.util.*;
class TUF{
    static void swap(int a,int b)
    {
        int temp=a;
        a=b;
        b=temp;
    }
   static void merge(int ar1[], int ar2[], int n, int m) {
  // code here
  int gap =(int) Math.ceil((double)(n + m) / 2.0);
  while (gap > 0) {
    int i = 0;
    int j = gap;
    while (j < (n + m)) {
      if (j < n && ar1[i] > ar1[j]) {
        swap(ar1[i], ar1[j]);
      } else if (j >= n && i < n && ar1[i] > ar2[j - n]) {
        swap(ar1[i], ar2[j - n]);
      } else if (j >= n && i >= n && ar2[i - n] > ar2[j - n]) {
        swap(ar2[i - n], ar2[j - n]);
      }
      j++;
      i++;
    }
    if (gap == 1) {
      gap = 0;
    } else {
      gap =(int) Math.ceil((double) gap / 2.0);
    }
  }
}
public static void main(String[] args) {
    int arr1[] = {1,4,7,8,10};
    int arr2[] = {2,3,9};
    System.out.println("Before merge:");
    for (int i = 0; i < arr1.length; i++) {
      System.out.print(arr1[i] + " ");
    }
    System.out.println();
    for (int i = 0; i < arr2.length; i++) {
      System.out.print(arr2[i] + " ");
    }
    System.out.println();
    merge(arr1, arr2, arr1.length, arr2.length);
    System.out.println("After merge:");
    for (int i = 0; i < arr1.length; i++) {
      System.out.print(arr1[i] + " ");
    }
    System.out.println();
    for (int i = 0; i < arr2.length; i++) {
      System.out.print(arr2[i] + " ");
    }

  }
}
```

Output:

Before merge:

1 4 7 8 10

2 3 9

After merge:

1 2 3 4 7

8 9 10

Time complexity: O(n+m)

Space Complexity: O(1)

Python Code

```python
from typing import List
def merge(arr1: List[int], arr2: List[int], n: int, m: int) -> None:
    gap = (n + m + 1) // 2
    while gap > 0:
        i = 0
        j = gap
        while j < (n + m):
            if j < n and arr1[i] > arr1[j]:
                arr1[i], arr1[j] = arr1[j], arr1[i]
            elif j >= n and i < n and arr1[i] > arr2[j - n]:
                arr1[i], arr2[j - n] = arr2[j - n], arr1[i]
            elif j >= n and i >= n and arr2[i - n] > arr2[j - n]:
                arr2[i - n], arr2[j - n] = arr2[j - n], arr2[i - n]
            j += 1
            i += 1
        if gap == 1:
            gap = 0
        else:
            gap = (gap + 1) // 2
```

```
if __name__ == "__main__":
    arr1 = [1, 4, 7, 8, 10]
    arr2 = [2, 3, 9]
    print("Before merge:")
    print(*arr1)
    print(*arr2)
    merge(arr1, arr2, 5, 3)
    print("After merge:")
    print(*arr1)
    print(*arr2)
```

Output:

Before merge:

1 4 7 8 10

2 3 9

After merge:

1 2 3 4 7

8 9 10

Time complexity: O(n+m)

Space Complexity: O(1)

> Special thanks to **Prashant Sahu** for contributing to this article on takeUforward. If you also wish to share your knowledge with the takeUforward fam, please check out this article

Merge two sorted arrays in O(1) space | GFG | C++ and Java | Brute-Better-...

« Previous Post
Implement Queue Using Array

Next Post »
Vertical Order Traversal of Binary Tree

Load Comments