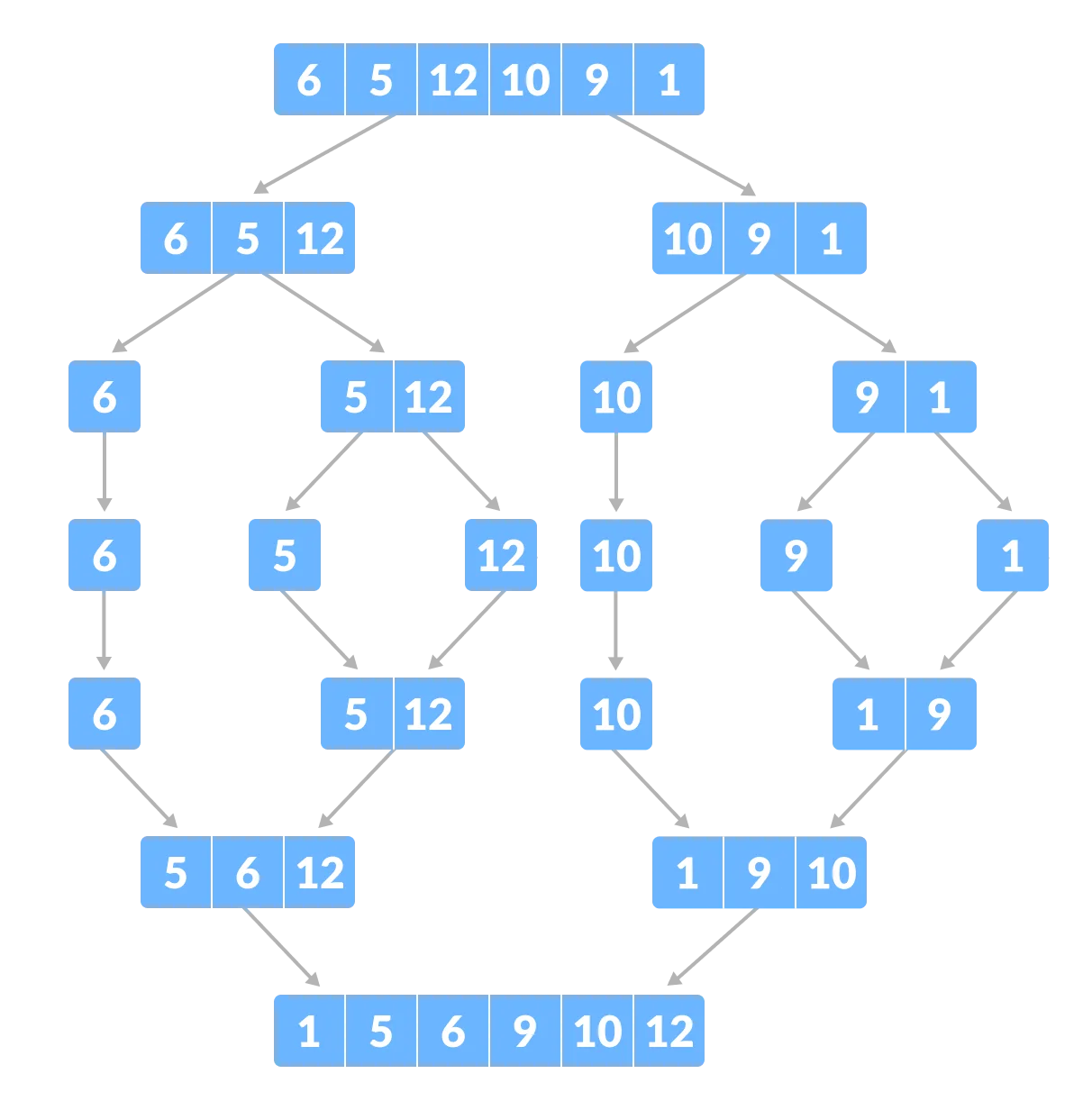
**Merge sort**

تعد خوارزمية الدمج من أشهر خوارزميات الترتيب فهي تعتمد على تقنية فرق تسد

حيث يتم تقسيم المشكلة إلى مشكلات صغيرة كل مشكلة صغيرة يتم حلها بشكل مستقل وفي النهاية يتم جمع هذه المشكلات الصغيرة لتشكيل الحل النهائي للمشكلة

****

**استراتيجية فرق تسد :**

نقوم بتقسيم المشكلة إلى مشكلات فرعية عدة وعندما يكون حل كل من هذه المشكلات الفرعية جاهز نقوم بجمع النتائج لهذه المشكلات الفرعية للوصول إلى حل للمشكلة الأساسية

لنفترض أن لدينا مصفوفة A نريد ترتيبها تتمثل المشكلة الفرعية في ترتيب قسم من المصفوفة بدءا من index (r) وانتهاء index (p) ونرمز له A[p…r]

**مرحلة divide:**

إذا كان q في منتصف المصفوفة بين p و rعندها يمكن تقسيم المصفوفة الأصلية إلى مصفوفتين فرعيتين A[p.q]

**و A[q+1,r]**

**مرحلة conquer :**

في هذه الخطوة نحول ترتيب كلا المصفوفتين الفرعتين إذا لم نصل حتى الان إلى الحالة الأساسية نقوم بتقسيم كلا المصفوفتين مرة أخرى ونحاول ترتيبهما

**مرحلة combine :**

عندما نصل مرحلة conquer إلى الحالة الأساسية نكون قد حصلنا على مصفوفتين فرعتين مرتبتين نقوم بجمع النتائج عن طريق إنشاء مصفوفة جديدة من المصفوفتين الفرعتين المرتبتين

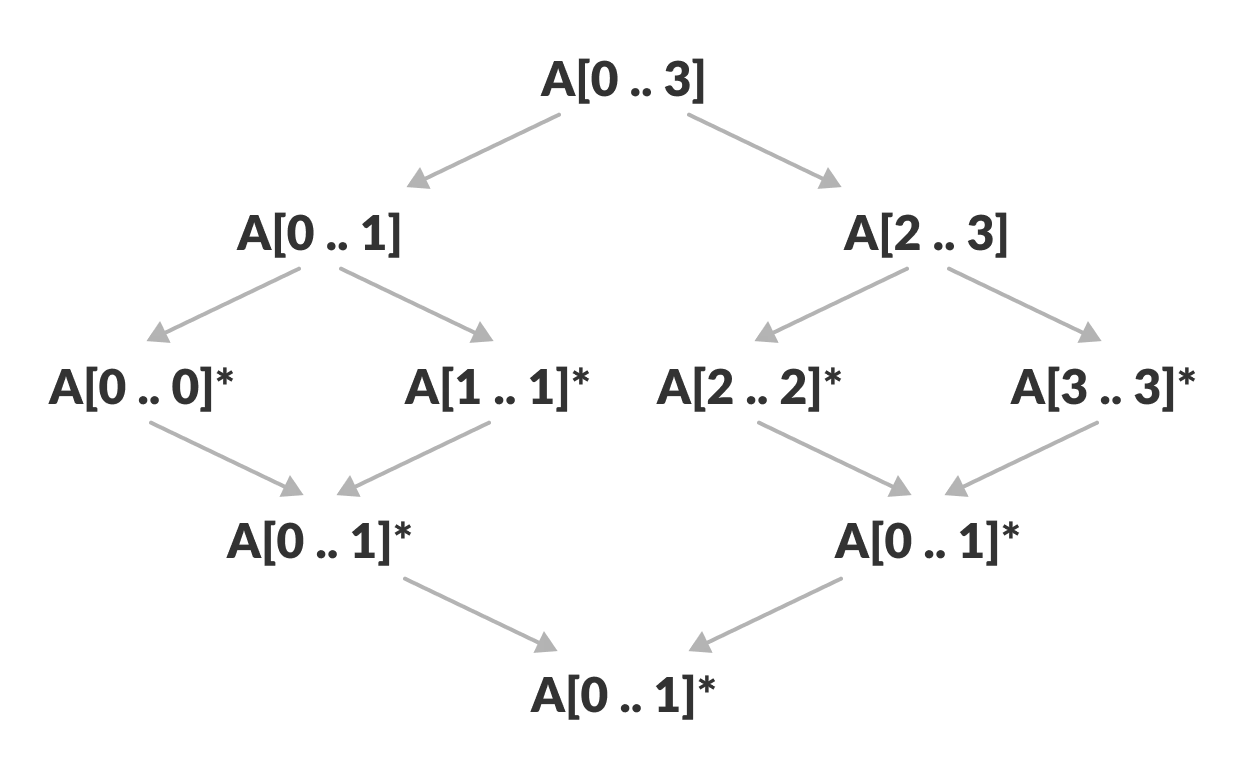
**الية العمل :**

لنعلم في البداية هناك تابعين الأول يقوم بتقسيم المصفوفة أما الثاني يقوم بعملية الدمج

يقوم التابع الأول المسمى mergesortبتقسيم المصفوفة بشكل متكرر إلى مصفوفتين فرعتين حتى نصل إلى مصفوفة مكونة من عنصر واحد

بعد ذلك يقوم تابع الدمج المسمى mergeبجمع المصفوفات المرتبة في مصفوفة أكبر حتى يتم دمج كامل المصفوفة

كما نرى في الصورة في الأسفل خوارزمية الدمج تقوم بتقسيم المصفوفة بشكل عودي حتى نصل إلى الحالة الأساسية التي تكون فيها المصفوفة مكونة من عنصر واحد بعد ذلك يقوم تابع الدمج بالتقاط المصفوفات الفرعية المرتبة وتقوم بدمجها لترتيب المصفوفة الأصلية تدريجيا

****

كل خوارزمية عودية تعتمد على حالة أساسية والمقدرة على جمع النتائج من هذه الحالات الأساسية وخوارزمية الترتيب بالدمج لا تختلف عن غيرها الشي الأكثر أهمية في هذه الخوارزمية هو الدمج

خطوة الدمج هي الحل للمشكلة البسيطة المتمثلة بدمج مصفوفتين مرتبتين لبناء مصفوفة مرتبة كبيرة

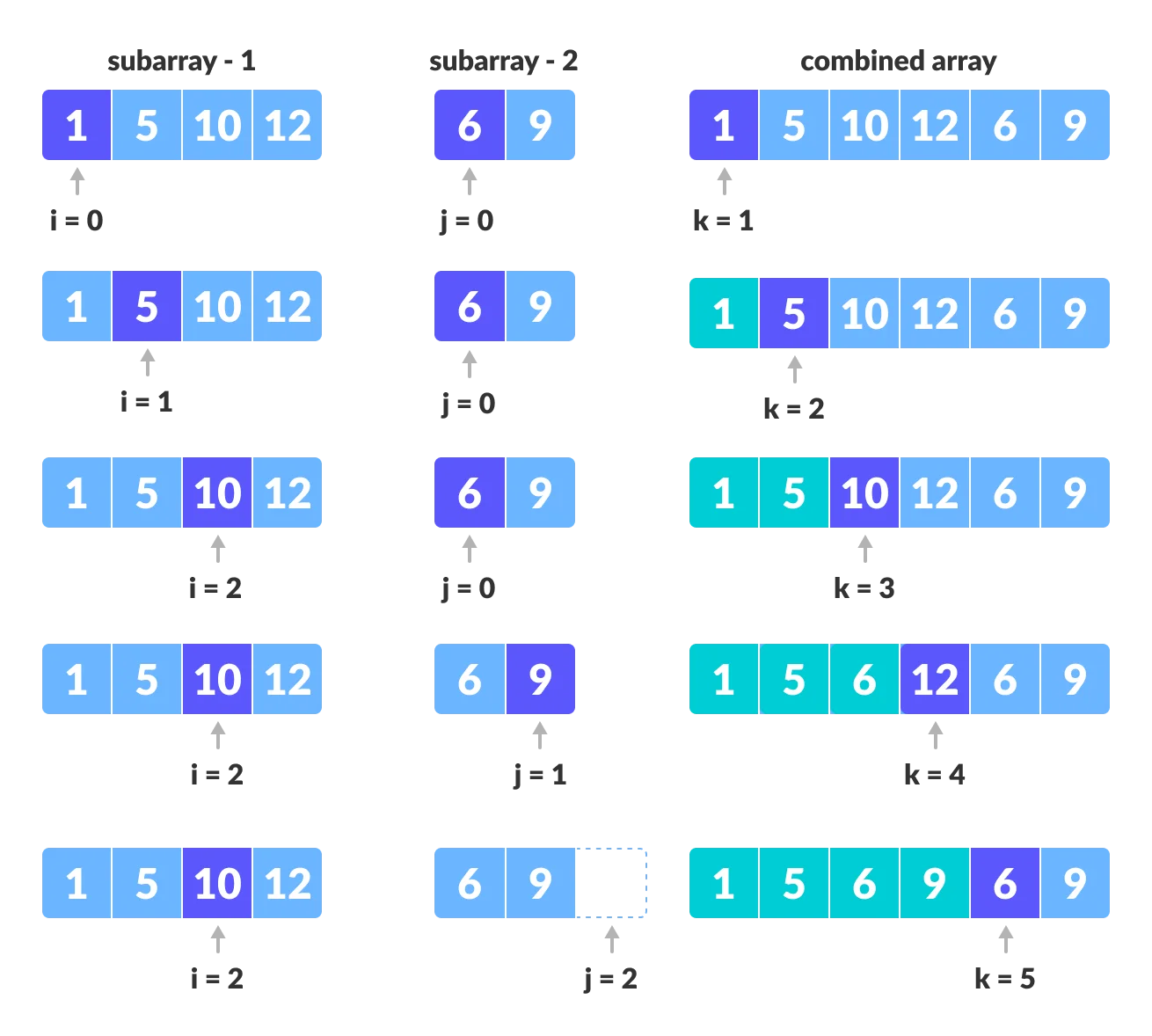
الخوارزمية تعتمد على ثلاث مؤشرات واحد لكل من المصفوفتين واحد للمصفوفة النهائية المرتبة

إذا وصلنا إلى نهاية إحدى المصفوفتين

نقوم بنسخ كل العناصر المتبقية من المصفوفة الغير فارغة

وإلا

نقوم بمقارنة العناصر من كلا المصفوفتين ونقوم بنسخ العنصر الأصغر الى المصفوفة النهائية المرتبة ونقوم بتحريك المؤشر الذي كان يشير إلى العنصر الأصغر

****

**التعقيد الزمني للخوارزمية :**

**يكون التعقيد الزمني للخوارزمية في الحالة الأفضل والأسوء هو O(nlogn)**

**البرنامج في لغة c++ :**

// C++ program for Merge Sort

#include <iostream>

using namespace std;

// Merges two subarrays of array[].

// First subarray is arr[begin..mid]

// Second subarray is arr[mid+1..end]

void merge(int array[], int const left, int const mid, int const right)

{

    auto const subArrayOne = mid - left + 1;

    auto const subArrayTwo = right - mid;

    // Create temp arrays

    auto \*leftArray = new int[subArrayOne],

         \*rightArray = new int[subArrayTwo];

    // Copy data to temp arrays leftArray[] and rightArray[]

    for (auto i = 0; i < subArrayOne; i++)

        leftArray[i] = array[left + i];

    for (auto j = 0; j < subArrayTwo; j++)

        rightArray[j] = array[mid + 1 + j];

    auto indexOfSubArrayOne = 0, // Initial index of first sub-array

        indexOfSubArrayTwo = 0; // Initial index of second sub-array

    int indexOfMergedArray = left; // Initial index of merged array

    // Merge the temp arrays back into array[left..right]

    while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo < subArrayTwo) {

        if (leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo]) {

            array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];

            indexOfSubArrayOne++;

        }

        else {

            array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];

            indexOfSubArrayTwo++;

        }

        indexOfMergedArray++;

    }

    // Copy the remaining elements of

    // left[], if there are any

    while (indexOfSubArrayOne < subArrayOne) {

        array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];

        indexOfSubArrayOne++;

        indexOfMergedArray++;

    }

    // Copy the remaining elements of

    // right[], if there are any

    while (indexOfSubArrayTwo < subArrayTwo) {

        array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];

        indexOfSubArrayTwo++;

        indexOfMergedArray++;

    }

}

// begin is for left index and end is

// right index of the sub-array

// of arr to be sorted \*/

void mergeSort(int array[], int const begin, int const end)

{

    if (begin >= end)

        return; // Returns recursivly

    auto mid = begin + (end - begin) / 2;

    mergeSort(array, begin, mid);

    mergeSort(array, mid + 1, end);

    merge(array, begin, mid, end);

}

// UTILITY FUNCTIONS

// Function to print an array

void printArray(int A[], int size)

{

    for (auto i = 0; i < size; i++)

        cout << A[i] << " ";

}

// Driver code

int main()

{

    int arr[] = { 12, 11, 13, 5, 6, 7 };

    auto arr\_size = sizeof(arr) / sizeof(arr[0]);

    cout << "Given array is \n";

    printArray(arr, arr\_size);

    mergeSort(arr, 0, arr\_size - 1);

    cout << "\nSorted array is \n";

    printArray(arr, arr\_size);

    return 0;

}