
Merge Sort :شرح الخوارزمية

هي خوارزمية لفرز المصفوفات تعتمد على تقسيم وإعادة دمج العناصر بشكل متكرر. تعمل هذه Merge Sort خوارزمية حيث تقسم المصفوفة إلى نصفين وتقوم بترتيب كل ،(divide and conquer) الخوارزمية وفق مبدأ التقسيم إلى نصفين .نصف على حدة ثم دمجها معًا بشكل مرتب

الهيكل الأساسي للخوارزمية

الدالة `divide`:

هذه الدالة تقوم بتقسيم المصفوفة إلى نصفين، وتستدعي نفسها بشكل تكراري حتى يتم الوصول إلى مصفوفات تحتوي .على عنصر واحد أو أقل، حيث لا يمكن ترتيب هذه المصفوفات بشكل أكبر
.لدمج النصفين في مصفوفة واحدة مرتبة `merge` بعد تقسيم المصفوفة إلى النصفين، تقوم الدالة باستدعاء دالة
الدالة `merge`

بدمج الجزئين المرتبين في مصفوفة واحدة مرتبة. تتأكد من مقارنة `merge` بعد تقسيم المصفوفة إلى جزئين، تقوم دالة .العناصر في النصفين، ثم تدمجهم بشكل مرتب في مصفوفة واحدة

الكود

```
#include <stdio.h>

void merge(int arr[], int s, int m, int e);
void divide(int arr[], int s, int e);

int main(void)
{
    int unsorted[] = {5, 2, 1, 3, 6, 4};
    int length = sizeof(unsorted) / sizeof(unsorted[0]);

    divide(unsorted, 0, length - 1);

    // طباعة المصفوفة بعد الترتيب (اختياري)
    for (int i = 0; i < length; i++) {
        printf("%d ", unsorted[i]);
    }
    printf("\n");

    return 0;
}

void divide(int arr[], int s, int e)
{
    if (s >= e)
        return;
```

```
    int m = (s + e) / 2;

    // تقسيم المصفوفة إلى نصفين
    divide(arr, s, m);
    divide(arr, m + 1, e);

    // دمج النصفين المرتبين
    merge(arr, s, m, e);
}

void merge(int arr[], int s, int m, int e)
{
    int left_size = m - s + 1;
    int right_size = e - m;

    int left[left_size], right[right_size];

    // نسخ العناصر للنصفين
    for (int i = 0; i < left_size; i++)
        left[i] = arr[s + i];
    for (int i = 0; i < right_size; i++)
        right[i] = arr[m + 1 + i];

    int i = 0, j = 0, k = s;

    // دمج العناصر في مصفوفة واحدة مرتبة
    while (i < left_size && j < right_size) {
        if (left[i] <= right[j]) {
            arr[k] = left[i];
            i++;
        } else {
            arr[k] = right[j];
            j++;
        }
        k++;
    }

    // نسخ العناصر المتبقية في النصف الأيسر إذا كانت موجودة
    while (i < left_size) {
        arr[k] = left[i];
        i++;
        k++;
    }

    // نسخ العناصر المتبقية في النصف الأيمن إذا كانت موجودة
    while (j < right_size) {
        arr[k] = right[j];
        j++;
        k++;
    }
}
```

شرح دالة divide

الفكرة الأساسية:

هي الجزء الذي يقوم بتقسيم المصفوفة إلى جزئين متساويين أو شبه متساويين حتى يصل إلى مصفوفات divide دالة
تحتوي على عنصر واحد فقط، ثم يتم دمجها مرة أخرى باستخدام دالة merge.

خطوات عمل دالة divide:

التوقف عند الشرط الأساسي:

إذا كانت المصفوفة تحتوي على عنصر واحد أو أقل (s >= e)، وتوقف الدالة.
فلا حاجة للتقسيم.
هذا الشرط هو الأساس الذي يعتمد عليه الخوارزم في التوقف عن التكرار.

حساب المنتصف:

نبدأ بتحديد المنتصف (middle) باستخدام الصيغة:

```
int m = (s + e) / 2;
```

المنتصف يساعدنا في تقسيم المصفوفة إلى جزئين.

التكرار على الجزئين:

نقوم باستدعاء دالة divide على الجزء الأيسر:

```
divide(arr, s, m);
```

ثم نقوم باستدعاء دالة divide على الجزء الأيمن:

```
divide(arr, m + 1, e);
```

دمج الجزئين باستخدام merge:

بعد أن يتم تقسيم المصفوفة إلى عناصر فردية أو جزئين يحتوي كل منهما على عنصر واحد، يتم دمجهما معًا في دالة
merge.

الأخطاء المحتملة في دالة divide:

عدم تحديث المنتصف بشكل صحيح: إذا لم يتم حساب المنتصف بالشكل الصحيح باستخدام
(s + e) / 2
فسيحدث تقسيم غير صحيح للمصفوفة.
التوقف المبكر: إذا كانت المصفوفة تحتوي على عنصر واحد أو أقل ولم يتوقف التكرار عند الشرط
s >= e، يمكن أن تحدث أخطاء في العمليات اللاحقة.

شرح دالة merge

الفكرة الأساسية:

تقوم بدمج جزئين مرتبين في مصفوفة واحدة مرتبة. تقوم بمقارنة العناصر في الجزئين ثم تقوم بإدراج merge دالة .الأصغر في المصفوفة النهائية

خطوات عمل دالة merge:

تحديد حجم المصفوفات الفرعية:

m: نحدد حجم المصفوفات الفرعية بناءً على المنتصف

```
int left_size = m - s + 1;
int right_size = e - m;
```

نسخ العناصر إلى المصفوفات الفرعية:

right: و left يتم نسخ العناصر من المصفوفة الأصلية إلى المصفوفات الفرعية

```
for (int i = 0; i < left_size; i++)
    left[i] = arr[s + i];
for (int i = 0; i < right_size; i++)
    right[i] = arr[m + 1 + i];
```

دمج المصفوفات الفرعية:

right: و left نقوم بمقارنة العناصر في المصفوفات الفرعية

```
while (i < left_size && j < right_size) {
    if (left[i] <= right[j]) {
        arr[k] = left[i];
        i++;
    } else {
        arr[k] = right[j];
        j++;
    }
    k++;
}
```

نسخ العناصر المتبقية:

نقوم بنسخها، right أو left إذا كانت هناك عناصر متبقية في المصفوفة الفرعية:

```
while (i < left_size) {
    arr[k] = left[i];
    i++;
    k++;
}

while (j < right_size) {
    arr[k] = right[j];
    j++;
```

```
    k++;
}
```

الأخطاء المحتملة في دالة merge:

عدم التأكد من أن المصفوفات الفرعية تحتوي على عناصر: إذا كانت المصفوفات الفرعية فارغة، قد يحدث خطأ في عمليات المقارنة أو النسخ.
عدم نسخ العناصر المتبقية في المصفوفات الفرعية: في حالة وجود عناصر متبقية في أي من المصفوفات الفرعية بعد مقارنة بعض العناصر، يجب نسخها إلى المصفوفة النهائية.

أمثلة للاختبار

مثال 1: مصفوفة صغيرة

```
int unsorted[] = {5, 3, 8, 4, 2};
```

النتيجة المتوقعة بعد الفرز:

```
{2, 3, 4, 5, 8}
```

مثال 2: مصفوفة تحتوي على عناصر متكررة

```
int unsorted[] = {5, 1, 5, 2, 3, 5};
```

النتيجة المتوقعة بعد الفرز:

```
{1, 2, 3, 5, 5, 5}
```

مثال 3: مصفوفة مرتبة بالفعل

```
int unsorted[] = {1, 2, 3, 4, 5};
```

النتيجة المتوقعة بعد الفرز:

```
{1, 2, 3, 4, 5}
```

الخلاصة:

تعتمد على تقسيم المصفوفة إلى نصفين ودمجهما بشكل مرتب Merge Sort خوارزمية.
تقوم بتقسيم المصفوفة إلى أجزاء أصغر حتى نصل إلى مصفوفات تحتوي على عنصر واحد أو أقل `divide` دالة.
تقوم بدمج هذه الأجزاء المرتبة معًا `merge` دالة.
تتم بشكل صحيح لتجنب الأخطاء المحتملة `merge` و `divide` تأكد من أن الحسابات والشرطيات في