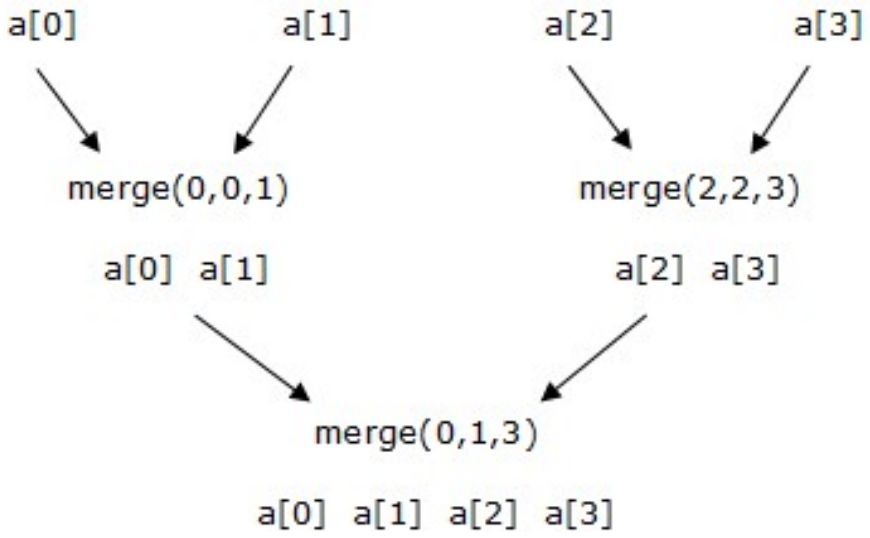
15. **MergeSort**

In dem paper **MergeSort_final.pdf** (08.03.2021) wurde die Funktion **f(n)** hergeleitet, welche in Abhängigkeit von n die Anzahl der Aufrufe der Funktion **`sort`** angibt; wegen f(n) = 2n – 1 wächst f(n) linear mit n und daher erheblich schwächer als der Aufwand A(n).

Aufgabe: Finde in entsprechender Weise einen Funktionsterm für die Funktion **g(n)**, welche die Anzahl der Aufrufe der Funktion **`merge`** in Abhängigkeit von n bestimmt.

Hinweis: Auch hier beschränke man sich auf Werte von n, die sich als Zweierpotenz schreiben lassen (n = 1, 2, 4, 8, ...). Fertige für n = 2 und n = 8 jeweils eine Baumstruktur an gemäß folgendem Beispiel (n = 4):



Die Pfeile bedeuten hier: „wird gemischt"; z. B. werden die sortierten Teillisten {a[0], a[1]} und {a[2], a[3]} vermöge merge(0,1,3) zur sortierten Liste {a[0], a[1], a[2], a[3]} gemischt.

16. Implementiere in dem in Python geschriebenen Quelltext **mergesort.py** Zählvariablen z und y, welche zur Laufzeit des Algorithmus die Anzahl der Aufrufe der Funktion **`sort`** und der Funktion **`merge`** ermitteln; bestätige auf diese Weise die Ergebnisse, die für f(n) und g(n) gefunden wurden.

Bemerkung:
Bei MergeSort hat der Rechenaufwand A(n), um eine Liste mit n Komponenten zu sortieren, wegen $A(n) \sim n \cdot \log_2(n)$ eine linear-logarithmische Komplexität; da die Anzahl der rekursiven Funktionsaufrufe linear mit n wächst, hat der zur Laufzeit des Algorithmus benötigte Speicher lineare Komplexität.