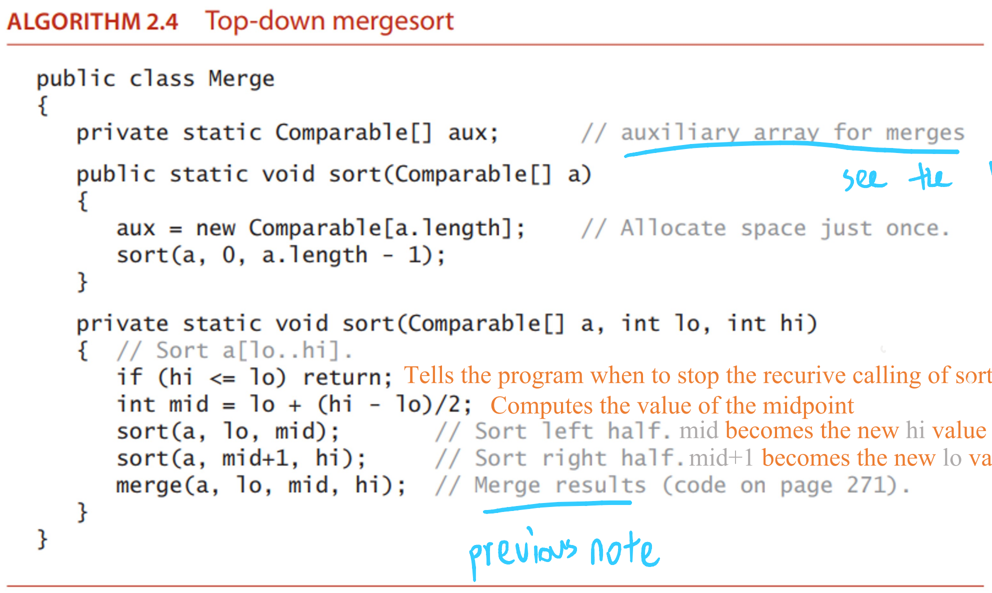
Top-down mergesort

2022年11月12日 19:08

Recursive mergesort implementation:

Being call recursively until



ALGORITHM 2.4 Top-down mergesort

```
public class Merge
{
   private static Comparable[] aux;      // auxiliary array for merges

   public static void sort(Comparable[] a)
   {
      aux = new Comparable[a.length];    // Allocate space just once.
      sort(a, 0, a.length - 1);
   }

   private static void sort(Comparable[] a, int lo, int hi)
   {  // Sort a[lo..hi].
      if (hi <= lo) return;
      int mid = lo + (hi - lo)/2;
      sort(a, lo, mid);       // Sort left half.
      sort(a, mid+1, hi);     // Sort right half.
      merge(a, lo, mid, hi);  // Merge results (code on page 271).
   }
}
```

see the previous note

Tells the program when to stop the recurive calling of sort method

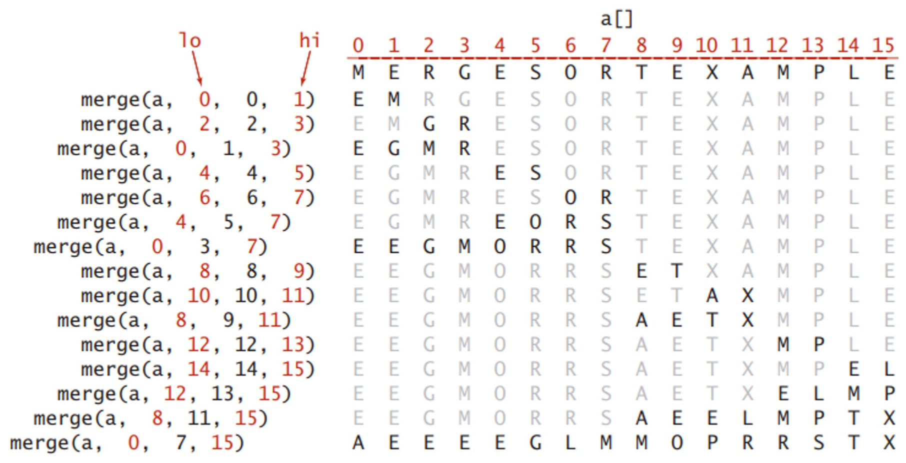
Computes the value of the midpoint

mid becomes the new hi value

mid+1 becomes the new lo value

previous note

To sort a subarray a[lo..hi] we divide it into two parts: a[lo..mid] and a[mid+1..hi], sort them independently (via recursive calls), and merge the resulting ordered subarrays to produce the result.

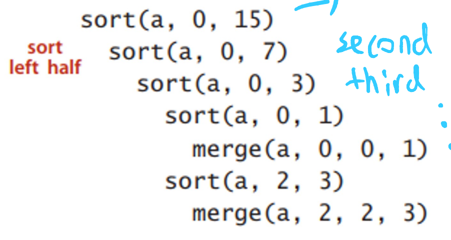


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 0, 0, 1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 2, 2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 0, 1, 3)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 4, 4, 5)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 6, 6, 7)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 4, 5, 7)	E	G	M	R	E	O	R	S	T	E	X	A	M	P	L	E
merge(a, 0, 3, 7)	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
merge(a, 8, 8, 9)	E	E	G	M	O	R	R	S	E	T	X	A	M	P	L	E
merge(a, 10, 10, 11)	E	E	G	M	O	R	R	S	E	T	A	X	M	P	L	E
merge(a, 8, 9, 11)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a, 12, 12, 13)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a, 14, 14, 15)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L
merge(a, 12, 13, 15)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a, 8, 11, 15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
merge(a, 0, 7, 15)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Trace of merge results for top-down mergesort

The method call trace:

the first call



```
sort(a, 0, 15)
  sort(a, 0, 7)
    sort(a, 0, 3)
      sort(a, 0, 1)
        merge(a, 0, 0, 1)
      sort(a, 2, 3)
        merge(a, 2, 2, 3)
```

sort
left half

second call

third call

:

```
               sort(a, 0, 15)  →
     sort        sort(a, 0, 7)      second call
  left half        sort(a, 0, 3)    third call
                     sort(a, 0, 1)          ⋮
                       merge(a, 0, 0, 1)
                     sort(a, 2, 3)
                       merge(a, 2, 2, 3)
                     merge(a, 0, 1, 3)
                   sort(a, 4, 7)
                     sort(a, 4, 5)
                       merge(a, 4, 4, 5)
                     sort(a, 6, 7)
                       merge(a, 6, 6, 7)
                     merge(a, 4, 5, 7)
                   merge(a, 0, 3, 7)
     sort
  right half     sort(a, 8, 15)
                   sort(a, 8, 11)
                     sort(a, 8, 9)
                       merge(a, 8, 8, 9)
                     sort(a, 10, 11)
                       merge(a, 10, 10, 11)
                     merge(a, 8, 9, 11)
                   sort(a, 12, 15)
                     sort(a, 12, 13)
                       merge(a, 12, 12, 13)
                     sort(a, 14, 15)
                       merge(a, 14, 14,15)
                     merge(a, 12, 13, 15)
                   merge(a, 8, 11, 15)
    merge
   results       merge(a, 0, 7, 15)
```

Top-down mergesort call trace