In-place merge

2022年11月12日　　18:28

Abstract in-place merge

```
public static void merge(Comparable[] a, int lo, int mid, int hi)
{  // Merge a[lo..mid] with a[mid+1..hi].
   int i = lo, j = mid+1;

   for (int k = lo; k <= hi; k++)  // Copy a[lo..hi] to aux[lo..hi].
      aux[k] = a[k];

   for (int k = lo; k <= hi; k++)  // Merge back to a[lo..hi].
      if      (i > mid)              a[k] = aux[j++];
      else if (j > hi )              a[k] = aux[i++];
      else if (less(aux[j], aux[i])) a[k] = aux[j++];
      else                           a[k] = aux[i++];
}
```

This method merges by first copying into the auxiliary array aux[] then merging back to a[]. In the merge (the second for loop), there are four conditions: left half exhausted (take from the right), right half exhausted (take from the left), current key on right less than current key on left (take from the right), and current key on right greater than or equal to current key on left (take from the left).

→ means all of the entries on the left half of aux[] has been taken.

ordered

ordered

	k	a[] 0	1	2	3	4	5	6	7	8	9	i	j	aux[] 0	1	2	3	4	5	6	7	8	9
input		E	E	G	M	R	A	C	E	R	T			-	-	-	-	-	-	-	-	-	-
copy		E	E	G	M	R	A	C	E	R	T			E	E	G	M	R	A	C	E	R	T
												0	5										
	0	A										0	6	E	E	G	M	R	A	C	E	R	T
	1	A	C									0	7	E	E	G	M	R		C	E	R	T
	2	A	C	E								1	7	E	E	G	M	R			E	R	T
	3	A	C	E	E							2	7		E	G	M	R			E	R	T
	4	A	C	E	E	E						2	8			G	M	R			E	R	T
	5	A	C	E	E	E	G					3	8			G	M	R				R	T
	6	A	C	E	E	E	G	M				4	8				M	R				R	T
	7	A	C	E	E	E	G	M	R			5	8					R				R	T
	8	A	C	E	E	E	G	M	R	R		5	9									R	T
	9	A	C	E	E	E	G	M	R	R	T	6	10										T
merged result		A	C	E	E	E	G	M	R	R	T												

Abstract in-place merge trace

Reference: [01_mergesort 归并排序_哔哩哔哩_bilibili](#) 2:23 - 6:59

Check out the code below which is a completed version of Abstract in-place mergesort. The above code is missing the two assert statements.

Merging: Java implementation

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
   assert isSorted(a, lo, mid);    // precondition: a[lo..mid]   sorted
   assert isSorted(a, mid+1, hi);  // precondition: a[mid+1..hi] sorted

   for (int k = lo; k <= hi; k++)                                          copy
      aux[k] = a[k];

   int i = lo, j = mid+1;
   for (int k = lo; k <= hi; k++)                                          merge
   {
      if      (i > mid)              a[k] = aux[j++];
      else if (j > hi)               a[k] = aux[i++];
      else if (less(aux[j], aux[i])) a[k] = aux[j++];
      else                           a[k] = aux[i++];
   }

   assert isSorted(a, lo, hi);     // postcondition: a[lo..hi] sorted
}
```