【DS】Day8

Tags	
Date	@May 30, 2022
Summary	Merge Sort

【Week3】Merge Sort

3.1 Merge Sort

Basic Plan:

- Divide array into two halves
- Recursively sort each half
- Merge two halves

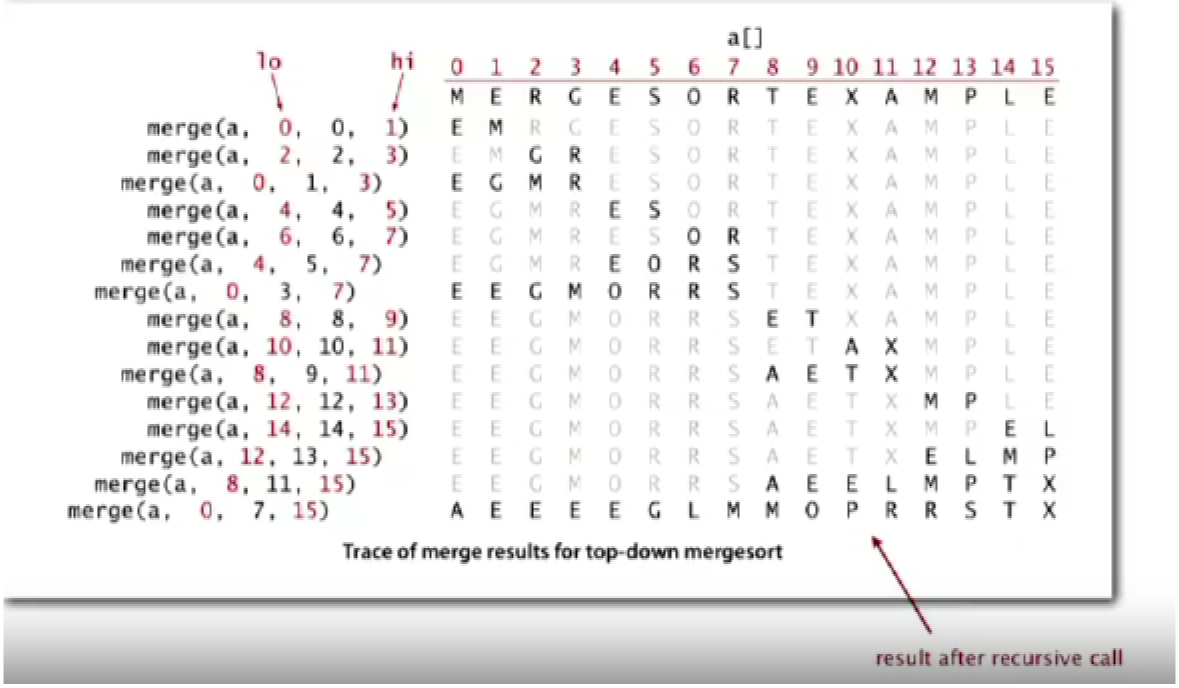
Goal: Given two sorted subarrays `a[lo]` to `a[mid]` and `a[mid+1]` to `a[hi]`, replace with sorted subarray `a[lo]` to `a[hi]`.

Java Implementation:

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

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

  int i = lo;
  int j = mid + 1;
  for (int k = lo; k <= hi; ++k) {
    // If the left array exhausts, add elements in the right array
    if (i > mid) a[k] = aux[i++];
    // If the right array exhausts, add elements in the left array
    else if (j > hi) a[k] = aux[j++];
    // Add smaller elements
    else if (less(aux[j], aux[i]) a[k] = aux[j++];
    else a[k] = aux[i++];
  }

  assert isSorted(a, lo, hi);
}
```

```
public class Merge {
  private static void merge(...) { ... }

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

  private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
      return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid + 1, hi);
    merge(a, aux, lo, mid, hi);
  }
}
```

Execution Trace



	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

result after recursive call

Proposition: Mergesort uses at most `NlgN` compares and 6NlogN array accessses to sort any array of size N

Mergesort uses extra space proportional to N.

A strong algorithm is in-place if it uses `clogN` extra memory

Improvement

User insertion sort for small subarrays:

- Mergesort has too much overhead for tiny subarrays.
- Cutoff to insertion sort for 7 times

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo + CUTOFF - 1) {
      Insertion.sort(a, lo, hi);
      return;
    }
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid + 1, hi);
    merge(a, aux, lo, mid, hi);
  }
```

Stop if already sorted:

- Is biggest item in first half ≤ smallest item in second half
- Helps for partially sorted array

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
      return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid + 1, hi);
    if (!less(arr[mid + 1], arr[mid])) return;
    merge(a, aux, lo, mid, hi);
  }
```