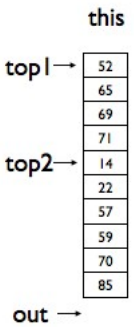
Merging & Sorting (Better)

Merging

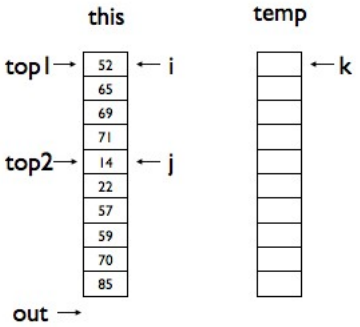
- given two adjacent sorted segments of an array, merge them into a single sorted segment
- merge should take linear time — i.e., O(n)

```
public class Array {
    private Comparable[] val;
    private int count;

    public Array (int max) {
        this.val = new Comparable[max];
        this.count = 0;
    }

    public void swap (int i, int j) {
        Comparable temp = this.val[i];
        this.val[i] = this.val[j];
        this.val[j] = temp;
    }
.
.
.
}
```

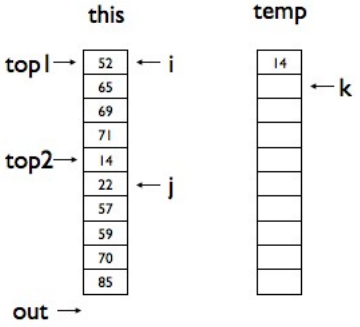
Merging Two Adjacent Sorted Segments



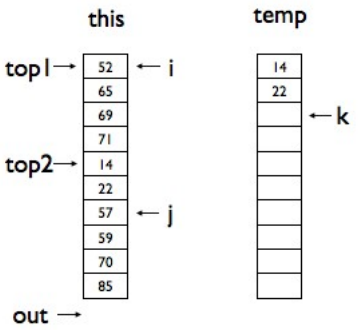
Merging Two Adjacent Sorted Segments



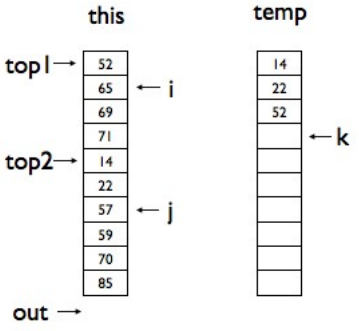
Merging Two Adjacent Sorted Segments



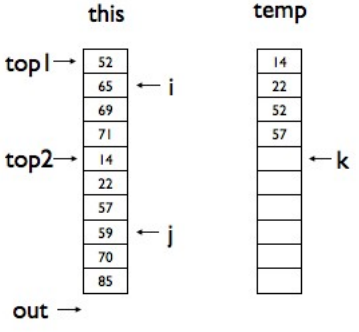
Merging Two Adjacent Sorted Segments



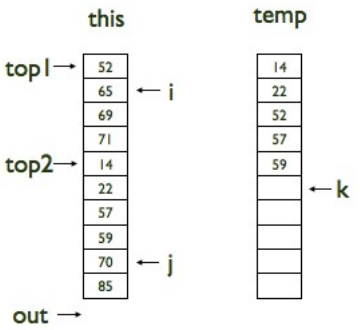
Merging Two Adjacent Sorted Segments



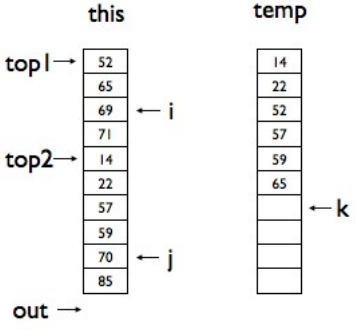
Merging Two Adjacent Sorted Segments



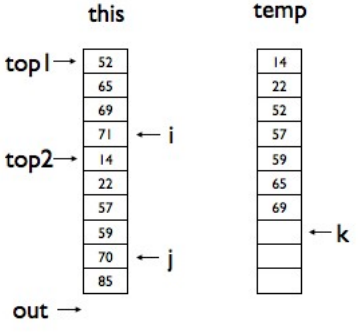
Merging Two Adjacent Sorted Segments



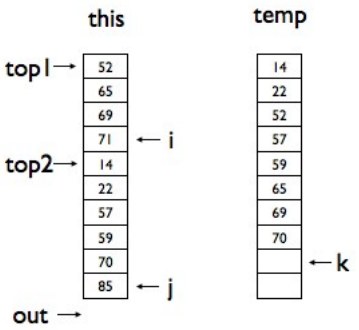
Merging Two Adjacent Sorted Segments



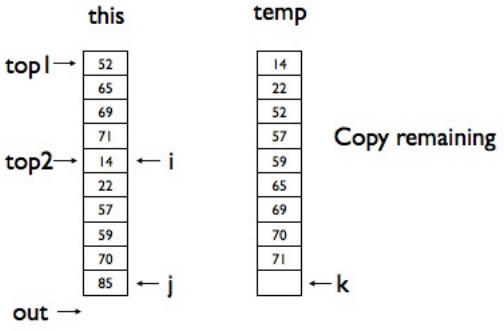
Merging Two Adjacent Sorted Segments



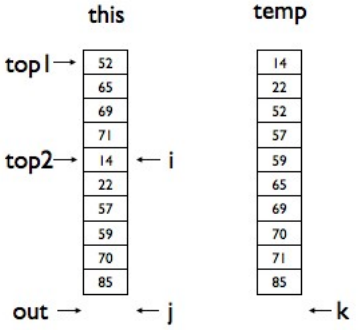
Merging Two Adjacent Sorted Segments



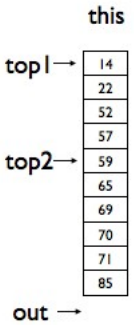
Merging Two Adjacent Sorted Segments



Merging Two Adjacent Sorted Segments



Merging Two Adjacent Sorted Segments



Copy from temp back to this

```
public class Array {
.
.
    public void merge (int top1, int top2, int out) {
        Array temp = new Array(out - top1);
        int i = top1;
        int j = top2;
        int k = 0;

        while (i < top2 && j < out)
            temp.val[k++] =
               this.val[i].compareTo(this.val[j]) < 0 ?
               this.val[i++] : this.val[j++];

        while (i < top2) temp.val[k++] = this.val[i++];
        while (j < out) temp.val[k++] = this.val[j++];

        for (k = 0; k < temp.val.size; k++)
            this.val[top1++] = temp.val[k];
    }
.
.
.
}
```

Mergesort

- general sorting problem — given an unsorted array, sort it
- base case: arrays of size zero or one are already sorted — do nothing
- otherwise, first divide the array in half
- then, recursively mergesort each half
- then, merge the two sorted halves

```
public class Array {
.
.
.
    public void mergesort () {
        mergesort(0,count);
    }
.
.
.
}
```

```
public class Array {
.
.
.
    private void mergesort (int top, int out) {
        if (top < out - 1) {
            int mid = (top + out) / 2;
            mergesort(top, mid);
            mergesort(mid, out);
            merge(top, mid, out);
        }
    }
.
.
.
}
```

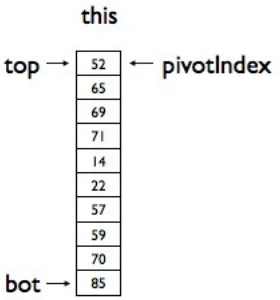
The Problem with Mergesort

- time complexity is good — O(nlogn)
- but, you need the temp array
- can you sort in O(nlogn) in place?

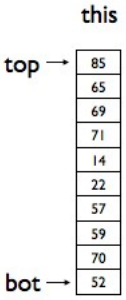
Quicksort

- general sorting problem — given an unsorted array, sort it
- base case: arrays of size zero or one are already sorted — do nothing
- otherwise, select a pivot value and divide the array into two parts, swapping values so that first part has values less than pivot and second part has values greater than or equal to pivot
- then, recursively quicksort each part

Dividing the Array by the Pivot Value

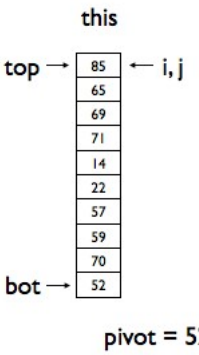


Dividing the Array by the Pivot Value

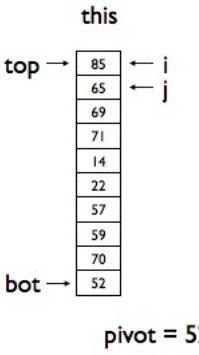


pivot = 52

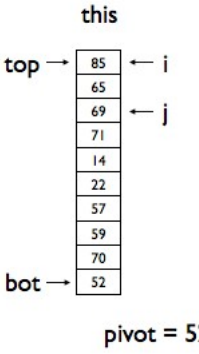
Dividing the Array by the Pivot Value



Dividing the Array by the Pivot Value

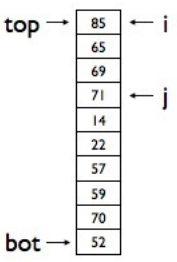


Dividing the Array by the Pivot Value



Dividing the Array by the Pivot Value

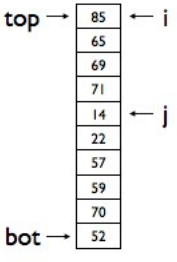
this



pivot = 52

Dividing the Array by the Pivot Value

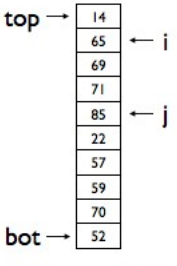
this



pivot = 52

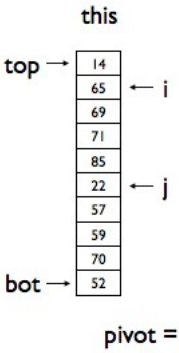
Dividing the Array by the Pivot Value

this



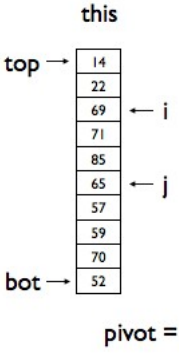
pivot = 52

Dividing the Array by the Pivot Value



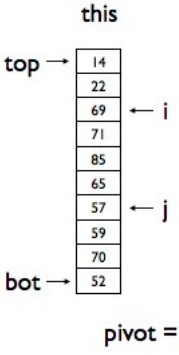
pivot = 52

Dividing the Array by the Pivot Value



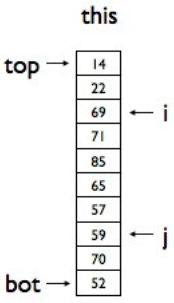
pivot = 52

Dividing the Array by the Pivot Value



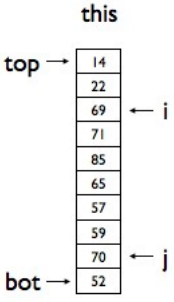
pivot = 52

Dividing the Array by the Pivot Value



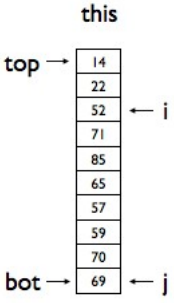
pivot = 52

Dividing the Array by the Pivot Value



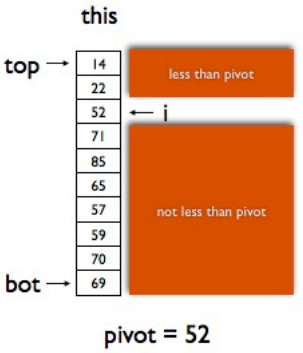
pivot = 52

Dividing the Array by the Pivot Value



pivot = 52

Dividing the Array by the Pivot Value




```
public class Array {
.
.
.
    private int pivotIndex (int top, int out) {
        return top;
    }
.
.
.
}
```

```
public class Array {
.
.
.
    private int split (int top, int bot, int pivotIndex) {
        Comparable pivot = this.val[pivotIndex];
        swap(pivotIndex, bot);
        int i = top;
        for (int j = top; j < bot; j++)
            if (this.val[j].compareTo(pivot) < 0)
                swap(j, i++);
        swap(i, bot);
        return i;
    }
.
.
.
}
```

```
public class Array {
.
.
.
    public void quicksort () {
        quicksort(0, count);
    }
.
.
.
}
```

```
public class Array {
.
.
.
    private void quicksort (int top, int out) {
       if (top < out - 1) {
            int mid = this.split(top, out - 1,
                                 this.pivotIndex(top, out));
            this.quicksort(top, mid);
            this.quicksort(mid + 1, out);
       }
    }
.
.
.
}
```