INTRODUCTION TO DATA STRUCTURES and ALGORITHMS

Rutgers University

MERGESORT

- *mergesort*
- *bottom-up mergesort*
- *sorting complexity*
- *divide-and-conquer*

http://algs4.cs.princeton.edu



http://ds.cs.rutgers.edu

Some slides Adopted and modified from Sedgewick and Wayne

Two classic sorting algorithms: mergesort and quicksort

Critical components in the world's computational infrastructure.

- Full scientific understanding of their properties has enabled us to develop them into practical system sorts.
- Quicksort honored as one of top 10 algorithms of 20th century in science and engineering.

Mergesort. [this lecture]



Quicksort. [next lecture]



MERGESORT

- *mergesort*
- *bottom-up mergesort*
- *sorting complexity*
- *divide-and-conquer*

Mergesort

one of the best-known examples of the utility of the divide-and-conquer paradigm for efficient algorithm design

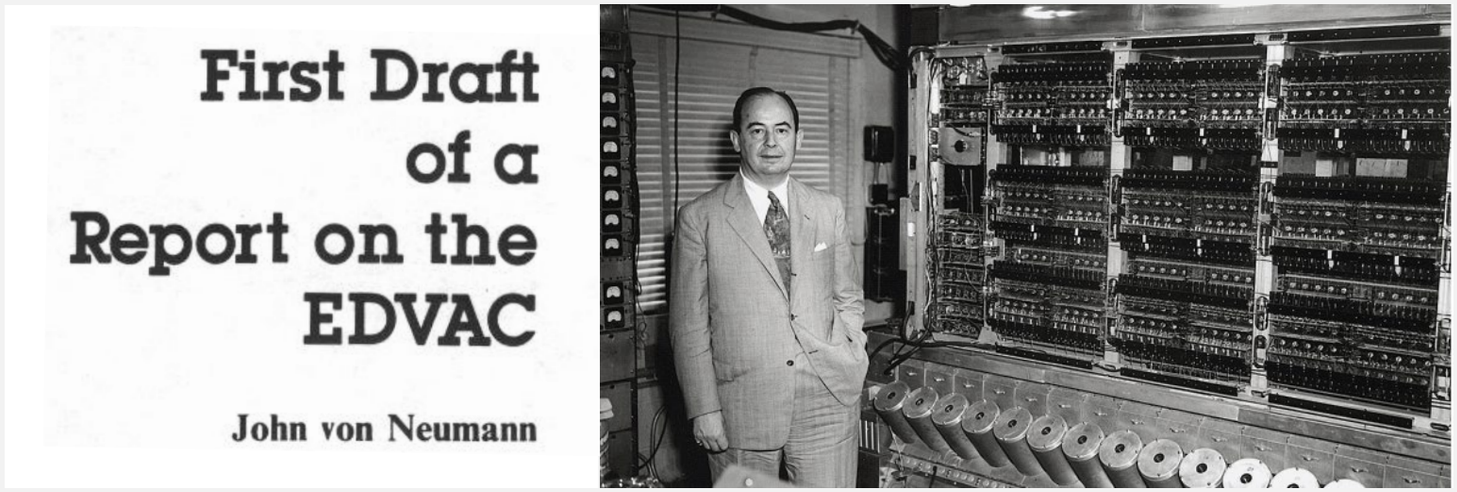
Basic plan.

- Divide array into two halves.
- Recursively sort each half. //recursively merge
- Merge two halves.

if it sorts the two subarrays,
it sorts the whole array, by merging the subarrays

input	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
sort left half	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
sort right half	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
merge results	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Mergesort overview



Abstract in-place merge demo

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].





Merging: Java implementation

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
```

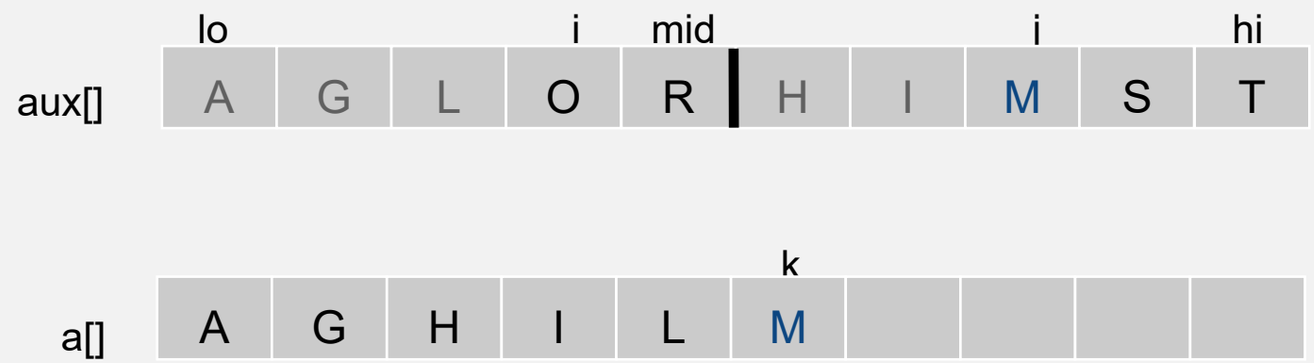
copy

```
   for (int k = lo; k <= hi; k++)
      aux[k] = a[k]; //additional array space; copy the original array to an auxiliary array
```

```
   int i = lo, j = mid+1; //index i to left subarray; j to right subarray
   for (int k = lo; k <= hi; k++)  //merge 2 subarrays into the original array space
   {
```

merge

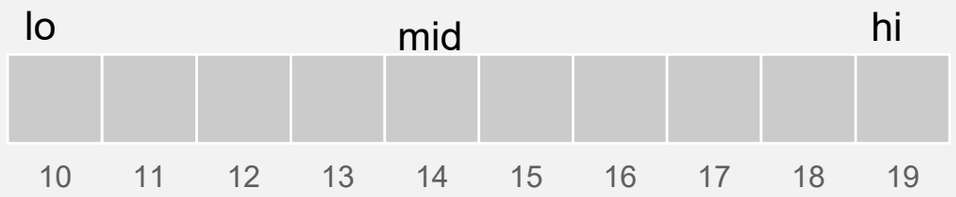
```
      if      (i > mid)              a[k] = aux[j++]; //reached the end of left subarray, move from the right subarray
      else if (j > hi)               a[k] = aux[i++]; //reached the end of right subarray, move from the left subarray
      else if (less(aux[j], aux[i])) a[k] = aux[j++]; //element in index j is strictly smaller
      else                           a[k] = aux[i++]; //element in index i is smaller or equal
   }
}
```



Mergesort: Java implementation

```
public class Merge
{
   private static void merge(...)
   {  /* as before */  }

   private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
   {
      if (hi <= lo) return; //subarray size of 1
      int mid = lo + (hi - lo) / 2;
      sort(a, aux, lo, mid); //recursively split the left subarray into 2 subarray
      sort(a, aux, mid+1, hi); //recursively split the right subarray into 2 subarray
      merge(a, aux, lo, mid, hi); //recursively merge 2 subarrays
   }
   public static void sort(Comparable[] a)
   {
      Comparable[] aux = new Comparable[a.length];
      sort(a, aux, 0, a.length - 1);
   }
}
```



```
                                  lo    hi
                                  ↑     ↑
                          sort(a, 0, 15)
sort left half              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)
                              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)
```

lo mid hi

recursively split until subarray size is 1

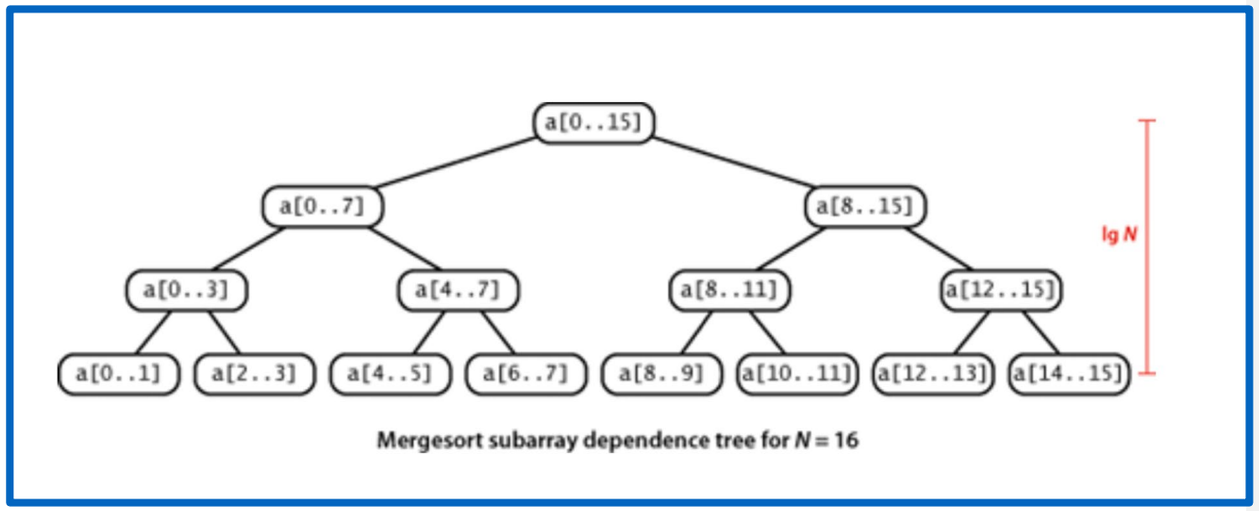
merge 2 subarray size 1

2 subarray size 2

2 subarray size 4

Top-down mergesort call trace

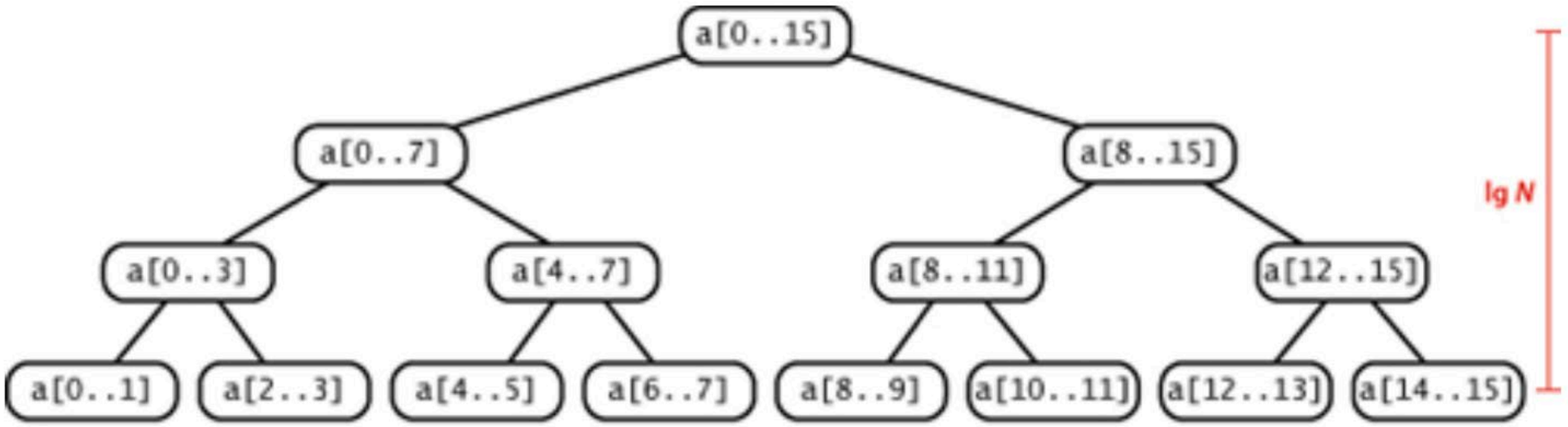
method call trace for sorting an array with index 0..15



Mergesort subarray dependence tree for $N = 16$

Mergesort

if array length is odd, the left subarray is larger by one than the right subarray.



Mergesort subarray dependence tree for $N = 16$

Mergesort quiz 2

Which of the following subarray lengths will occur when running mergesort on an array of length 12?

A. { 1, 2, 3, 4, 6, 8, 12 }

B. { 1, 2, 3, 6, 12 }

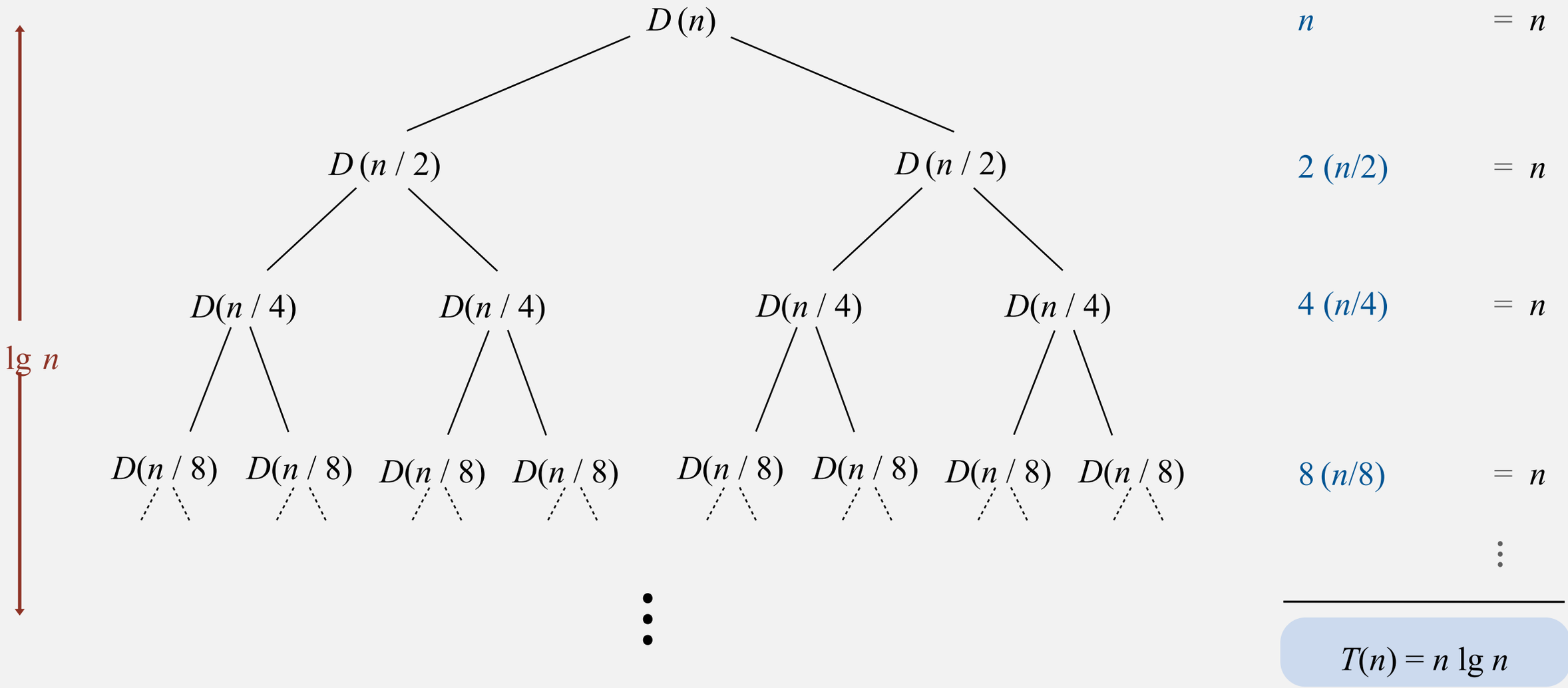
C. { 1, 2, 4, 8, 12 }

D. { 1, 3, 6, 9, 12 }

Divide-and-conquer recurrence

Proposition. If $D(n)$ satisfies $D(n) = 2\,D(n/2) + n$ for $n > 1$, with $D(1) = 0$, then $D(n) = n \lg n$.

Pf by picture. [assuming n is a power of 2] (each node depicts a subarray for which sort() does a merge)



Mergesort analysis: number of array accesses

Proposition. Mergesort uses $\leq 6\, n \lg n$ array accesses to sort an array of length n. (Each merge uses at most 6n array accesses
(2n for the copy, 2n for the move back, and at most 2n for compares)

Pf sketch. The number of array accesses $A(n)$ satisfies the recurrence:

$$A(n) \leq A(\lceil n/2 \rceil) + A(\lfloor n/2 \rfloor) + 6n \text{ for } n > 1, \text{ with } A(1) = 0.$$

Key point. Any algorithm with the following structure takes $n \log n$ time:

```
public static void f(int n)
{
   if (n == 0) return;
   f(n/2);
   f(n/2);
   linear(n);
}
```

← solve two problems of half the size

← do a linear amount of work

Notable examples. FFT, hidden-line removal, Kendall-tau distance, …

Mergesort analysis: memory

Proposition. Mergesort uses extra space proportional to n.

Pf. The array aux[] needs to be of length n for the last merge.

two sorted subarrays

A C D G H I M N U V

B E F J O P Q R S T

merged result

A B C D E F G H I J M N O P Q R S T U V

Def. A sorting algorithm is in-place if it uses $\leq c \log n$ extra memory.

Ex. Insertion sort and selection sort.

Challenge 1 (not hard). Use aux[] array of length $\sim \frac{1}{2} n$ instead of n.

Challenge 2 (very hard). In-place merge. [Kronrod 1969]

Mergesort quiz 3

Is our implementation of mergesort stable?

A. Yes.

B. No, but it can be easily modified to be stable.

C. No, mergesort is inherently unstable.

D. *I don't remember what stability means.*

a sorting algorithm is stable if it
preserves the relative order of equal keys

input	C	A_1	B	A_2	A_3
sorted	A_3	A_1	A_2	B	C

not stable

Stability: mergesort

Proposition. Mergesort is stable.

```
public class Merge
{
   private static void merge(...)
   {  /* as before */  }

   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); //data movement only happens here
   }

   public static void sort(Comparable[] a)
   {  /* as before */  }
}
```

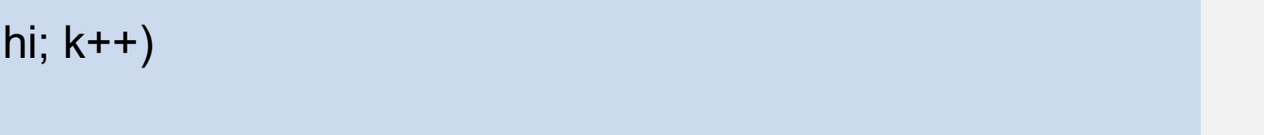
Pf. Suffices to verify that merge operation is stable.

Stability: mergesort

Proposition. Merge operation is stable.

```
private static void merge(...)
{
   for (int k = lo; k <= hi; k++)
      aux[k] = a[k];

   int i = lo, j = mid+1;
   for (int k = lo; k <= hi; k++)
   {
      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++];
   }
}
```



Pf. Takes from left subarray if equal keys.

Mergesort: practical improvements

Use insertion sort for small subarrays.

- Mergesort has too much overhead for tiny subarrays.
- Cutoff to insertion sort for ≈ 10 items.

```
private static void sort(...)
{
   if (hi <= lo + CUTOFF - 1)
   {
      Insertion.sort(a, lo, hi);
      return;
   }

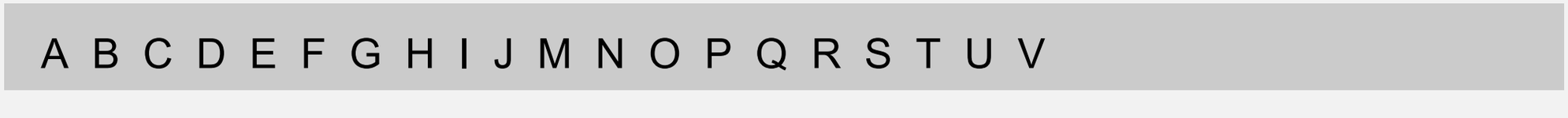
   int mid = lo + (hi - lo) / 2;
   sort (a, aux, lo, mid);
   sort (a, aux, mid+1, hi);
   merge(a, aux, lo, mid, hi);
}
```

Mergesort: practical improvements

Stop if already sorted.

- Is largest item in first half ≤ smallest item in second half?
- Helps for partially ordered arrays.






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

   if (!less(a[mid+1], a[mid])) return;

   merge(a, aux, lo, mid, hi);
}
```

Java 6 system sort

Basic algorithm in `Arrays.sort()` for sorting objects = mergesort.

- Cutoff to insertion sort = 7.
- Stop-if-already-sorted test.
- Eliminate-the-copy-to-the-auxiliary-array trick.



http://hg.openjdk.java.net/jdk6/jdk6/jdk/file/tip/src/share/classes/java/util/Arrays.java

MERGESORT

- *mergesort*
- *bottom-up mergesort*
- *sorting complexity*
- *divide-and-conquer*

Bottom-up mergesort (non-recursive version)

Basic plan.

- Pass through array, merging subarrays of size 1.
- Repeat for subarrays of size 2, 4, 8, ...

		a[i]														
		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
sz = 1																	
	merge(a, aux, 0, 0, 1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
	merge(a, aux, 2, 2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
1st iteration	merge(a, aux, 4, 4, 5)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
	merge(a, aux, 6, 6, 7)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
	merge(a, aux, 8, 8, 9)	E	M	G	R	E	S	O	R	E	T	X	A	M	P	L	E
	merge(a, aux, 10, 10, 11)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
	merge(a, aux, 12, 12, 13)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
	merge(a, aux, 14, 14, 15)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	E	L
sz = 2																	
	merge(a, aux, 0, 1, 3)	E	G	M	R	E	S	O	R	E	T	A	X	M	P	E	L
2nd iteration	merge(a, aux, 4, 5, 7)	E	G	M	R	E	O	R	S	E	T	A	X	M	P	E	L
	merge(a, aux, 8, 9, 11)	E	G	M	R	E	O	R	S	A	E	T	X	M	P	E	L
	merge(a, aux, 12, 13, 15)	E	G	M	R	E	O	R	S	A	E	T	X	E	L	M	P
sz = 4																	
3rd iteration	merge(a, aux, 0, 3, 7)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
	merge(a, aux, 8, 11, 15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
sz = 8																	
	merge(a, aux, 0, 7, 15)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Bottom-up mergesort: Java implementation

```
public class MergeBU
{
   private static void merge(...)
   {  /* as before */  }

   public static void sort(Comparable[] a)
   {
      int n = a.length;
      Comparable[] aux = new Comparable[n];
      for (int sz = 1; sz < n; sz = sz+sz)
         for (int lo = 0; lo < n-sz; lo += sz+sz)
            merge(a, aux, lo, lo+sz-1, Math.min(lo+sz+sz-1, n-1));
   }
}
```

Bottom line. Simple and non-recursive version of mergesort.

Sorting summary

	inplace?	stable?	best	average	worst	remarks
selection	✓		$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	$\frac{1}{2} n^2$	n exchanges
insertion	✓	✓	n	$\frac{1}{4} n^2$	$\frac{1}{2} n^2$	use for small n or partially ordered
shell	✓		$n \log_3 n$	?	$c\, n^{3/2}$	tight code; subquadratic
merge		✓	$\frac{1}{2} n \lg n$	$n \lg n$	$n \lg n$	$n \log n$ guarantee; stable
timsort		✓	n	$n \lg n$	$n \lg n$	improves mergesort when pre-existing order
?	✓	✓	n	$n \lg n$	$n \lg n$	holy sorting grail

INTRODUCTION TO DATA STRUCTURES and ALGORITHMS

Rutgers University

MERGE SORT

- *mergesort*
- *bottom-up mergesort*
- *sorting complexity*
- *divide-and-conquer*



http://ds.cs.rutgers.edu

Last updated on 2/11/18 12:14 PM