ID1020: Mergesort

Dr. Per Brand
pbrand@kth.se

kap 2.2

Slides adapted from Algoritms 4th Edition, Sedgewick.

Two classical algorithms: mergesort och quicksort

- Key components in the world’s IT-infrastructure.
  - Full scientific understanding of their properties has enabled their development into practical system sorts
  - Quicksort honored as one the top 10 algorithms of the 20th century.

- Mergesort.

…

- Quicksort.

…

Divide-and-Conquer algorithms

- Principle Divide-and-Conquer in algorithm design:
  - **Divide**: divide up the problem into smaller, independent subproblems. Subproblems are of the same type as the original problem. Continue dividing until the problem is broken down into many trivial mini-problems.
  - **Conquer**: solve the trivial subproblems. Then construct partial solutions from solved subproblems until full solution can be constructed
- Often done by recursion
- The order-of-growth can usually be solved by analysis of the recursion relation.

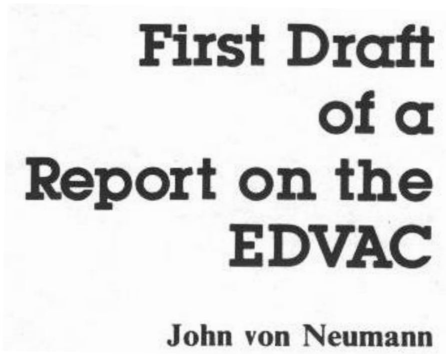
Mergesort

Mergesort

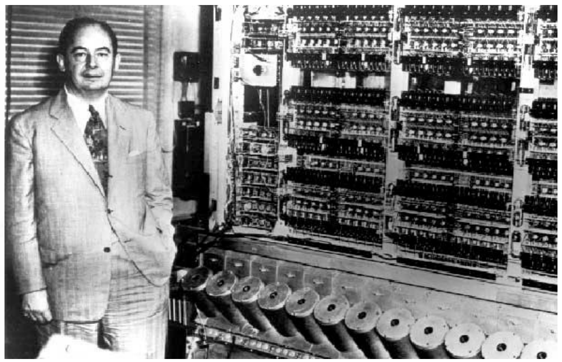
- Method.
  - Divide the array into two halves.
  - Sort the two halves recursively.
  - Merge the two halves.

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


First Draft of a Report on the EDVAC

John von Neumann



Abstract in-place merge demo

- Goal. Given two sorted subarrayer `a[lo]` to `a[mid]` and `a[mid+1]` to `a[hi]`, replace this with a sorted subarray `a[lo]` till `a[hi]`.

	lo				mid	mid+1				hi
a[]	E	E	G	M	R	A	C	E	R	T

$\underbrace{\text{E E G M R}}_{\text{sorted}}$ $\underbrace{\text{A C E R T}}_{\text{sorted}}$

Abstract in-place merge demo

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

	lo									hi
a[]	A	C	E	E	E	G	M	R	R	T

sorted

Merging

- How can we merge two sorted subarrays into one sorted array?
- Use an auxiliary array.

		a[]												aux[]									
	k	0	1	2	3	4	5	6	7	8	9	i	j	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

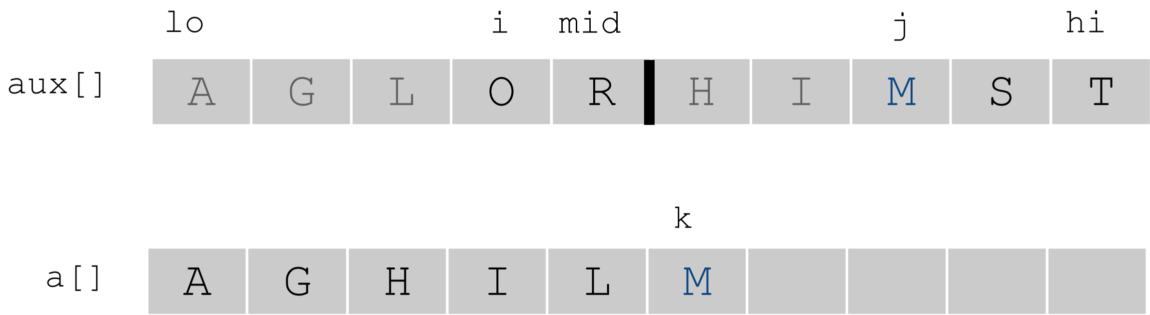
Merging: Java implementation

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

   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++];
   }
}
```

copy

merge



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

	lo			i	mid			j		hi
aux[]	A	G	L	O	R	H	I	M	S	T
						k				
a[]	A	G	H	I	L	M				

Assertions

- Assertion. A statement to check assumptions about program.
  - Assertions can be used to find bugs.
  - Assertions also serve as good documentation.

- Java assert statement. Throws an exception if the condition is false.

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

- Can be enabled or disabled during execution. ⇒ No extra overhead in production code.

```
% java -ea MyProgram   // enable assertions
% java -da MyProgram   // disable assertions
(default)
```

- Best practice. Use assertions to check *invariants*, *preconditions* and *postconditons,* for example, to an API or method.
  Assume that assertions are disabled in produktion code.

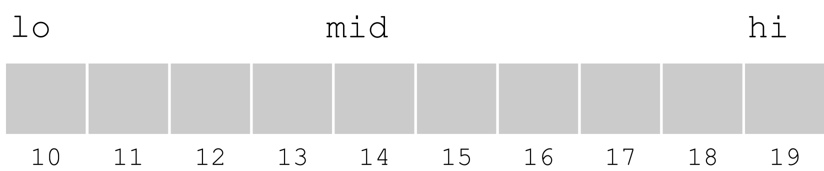
- Unit tests on the other hand check the *external behavior* of an API or method

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

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



Mergesort: trace

lo hi

a[]

	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

Mergesort: empirical analysis

- Running times:
  - A laptop executes 10^8 comparisons/second.
  - A supercomputer executes 10^{12} comparisons/second.

	insertion sort (N^2)			mergesort (N log N)		
computer	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 second	18 min
super	instant	1 second	1 week	instant	instant	instant

- Conclusion. Good algorithms are better than supercomputers.

Mergesort: comparisons

- **Theorem.** Mergesort compares ≤ $N \lg N$ to sort an array of length N.

- **Proof.** The number of comparisons $C(N)$ to run mergesort on an array of length N satisfies the recurrence relation:

$$C(N) \leq C(\lceil N/2 \rceil) + C(\lfloor N/2 \rfloor) + N \quad \text{där } N > 1, \text{ med } C(1) = 0.$$

Left half Right half merge

$$A(N) \leq A(\lceil N/2 \rceil) + A(\lfloor N/2 \rfloor) + 6N \quad \text{där } N > 1, \text{ med } A(1) = 0.$$

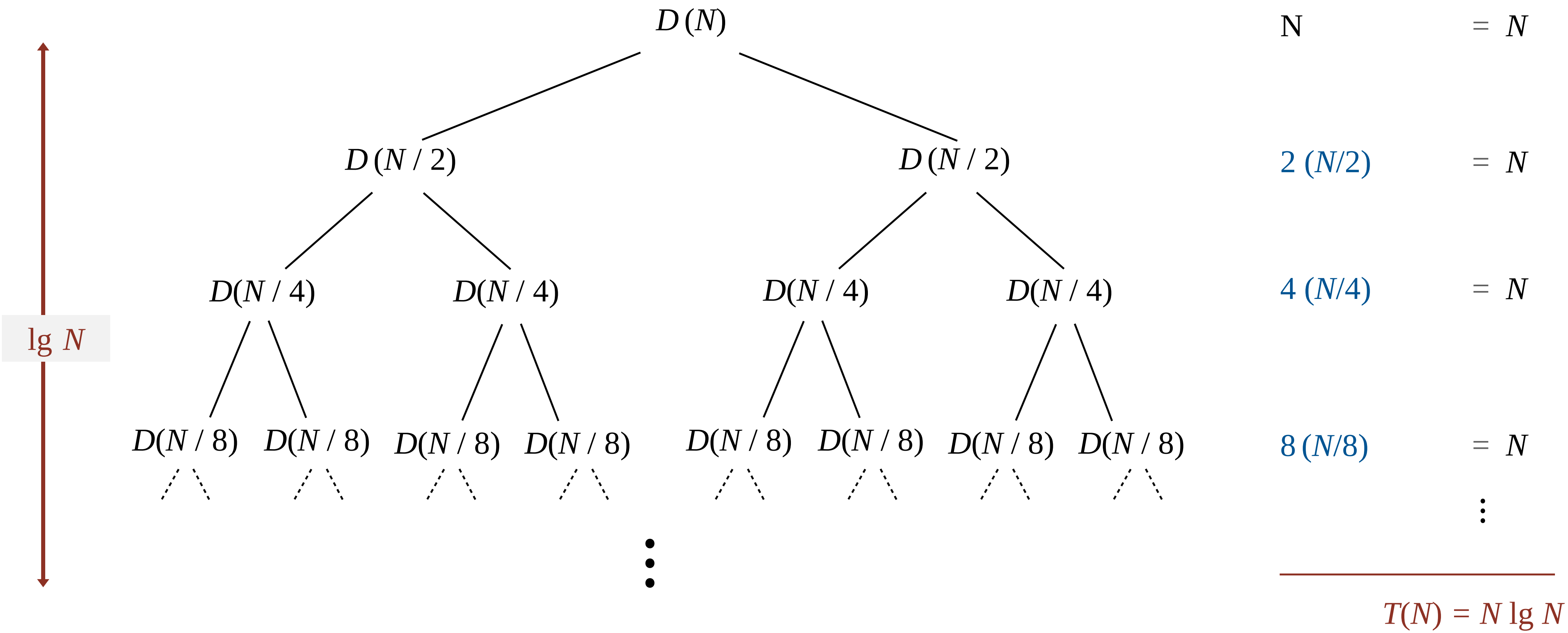
- We solve the recurrence relation when N is a power of two:

$D(N) = 2\,D(N/2) + N$, för $N > 1$, med $D(1) = 0$. Result valid for all N

Divide-och-conquer recurrence: picture proof

- **Theorem.** IF $D(N)$ satisfies $D(N) = 2\,D(N/2) + N$ for $N > 1$, with $D(1) = 0$,
  THEN $D(N) = N \lg N$.

- **Proof 1.**



Divide-och-conquer recurrence : proof with expansion

- Theorem. IF $D(N)$ satisfierar $D(N) = 2\,D(N/2) + N$ för $N > 1$, med $D(1) = 0$,
  THEN $D(N) = N \lg N$.

- Proof 2

$$
\begin{aligned}
D(N) &= 2\,D(N/2) + N && \text{given} \\
D(N)/N &= 2\,D(N/2)/N + 1 && \text{divide both sides by N} \\
&= D(N/2)/(N/2) + 1 && \text{algebra} \\
&= D(N/4)/(N/4) + 1 + 1 && \text{apply to first term} \\
&= D(N/8)/(N/8) + 1 + 1 + 1 && \text{apply to first term again} \\
&\;\;\cdots \\
&= D(N/N)/(N/N) + 1 + 1 + \ldots + 1 && \text{stop applying, D(1) = 0} \\
&= \lg N
\end{aligned}
$$

Divide-och-conquer recurrence: proof by induction

- Sats. IF $D(N)$ satisfierar $D(N) = 2\,D(N/2) + N$ för $N > 1$, med $D(1) = 0$,
  THEN $D(N) = N \lg N$.

- Proof 3.
  - Base case: $N = 1$.
  - Induktion hypothesis : $D(N) = N \lg N$.
  - Goal: show that $D(2N) = (2N) \lg (2N)$.

$$D(2N) = 2\,D(N) + 2N \qquad \text{given}$$

$$= 2\,N \lg N + 2N \qquad \text{inductive hypothesis}$$

$$= 2\,N\,(\lg(2N) - 1) + 2N \qquad \text{algebra}$$

$$= 2\,N \lg(2N) \qquad \text{QED}$$

Mergesort: array accesses

- **Theorem.** Mergesort access the array $\leq 6 N \lg N$ times to sort an array of length N.

- **Proof.** Array accesses $A(N)$ satisfies the recurrence-relation:

$$A(N) \leq A(\lceil N/2 \rceil) + A(\lfloor N/2 \rfloor) + 6N \text{ } \mathit{för} \text{ } N > 1, \text{ with } A(1) = 0$$

- **Note.** All algorithms with the following struture take $N \log N$ time:

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

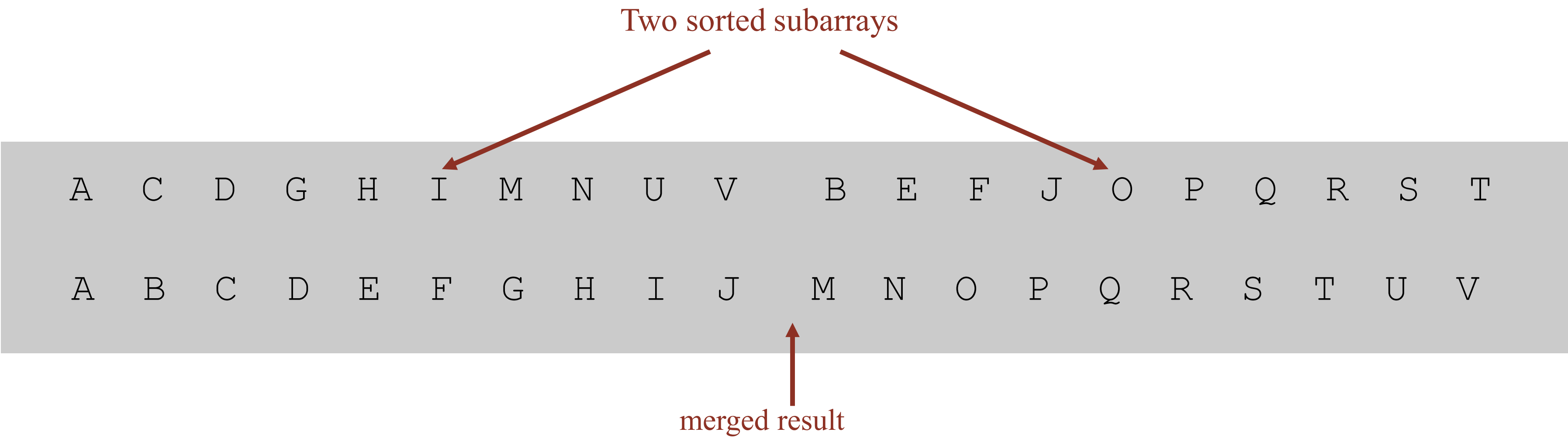
Solve two subproblems of half the size

Linear amount of work

- **Exceptions.** FFT, m.m., …

Mergesort analysis: memory usage

- Theorem: Mergesort uses extra memory proportional to N.
- Proof . The array `aux[]` needs to be of length N for the last merge.



- Def. A sorting algorithm is "in-place" if it uses $\leq c \log N$ extra memory
- T.ex. Insertion sort, selection sort, shellsort. Not this mergesort !!

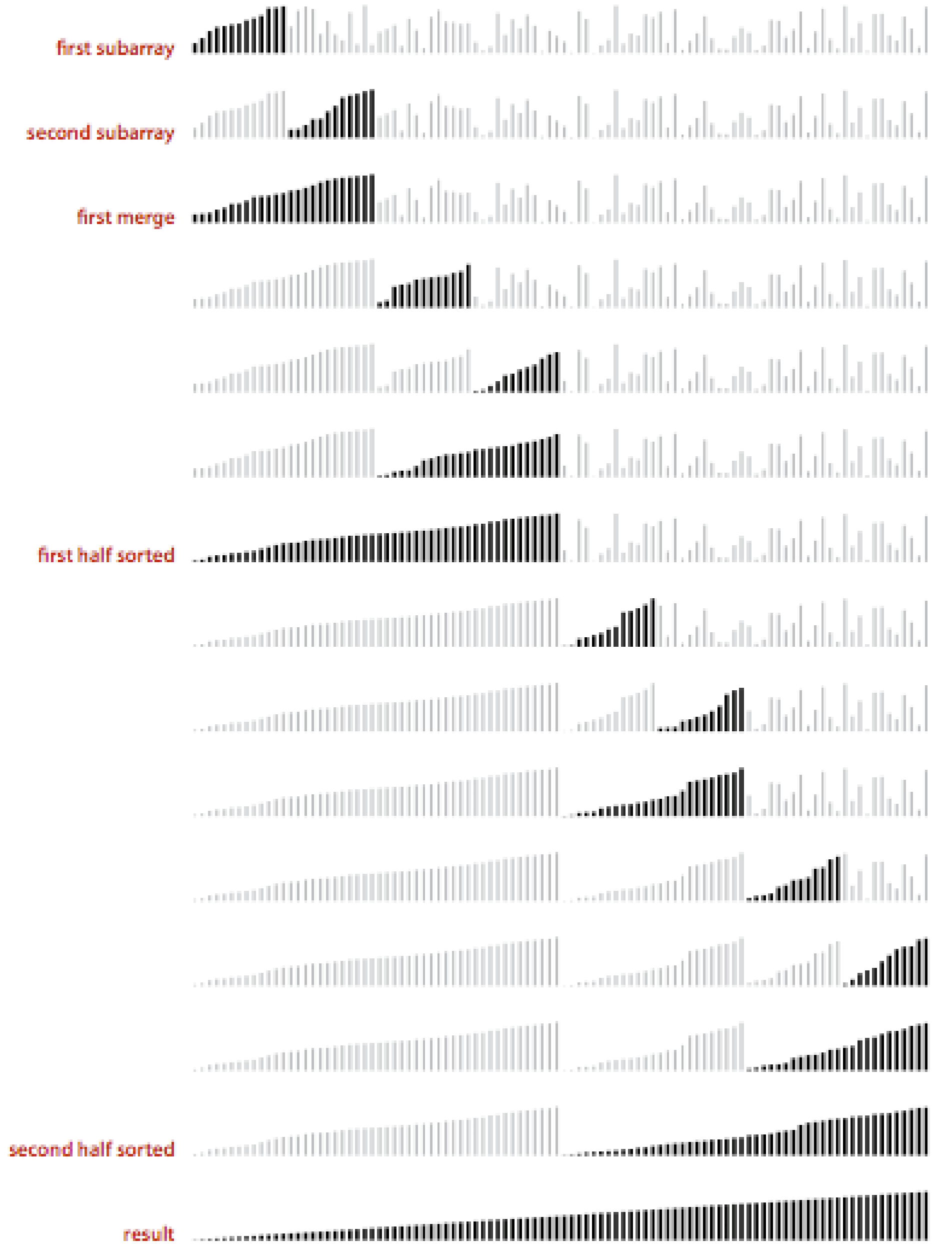
- Challenge 1 (not difficult). Use `aux[]` array of length ~ ½ N instead of N.
- Challenge 2 (extremely difficult). *In-place merge*. [Kronrod 1969]

Mergesort: improvements

- Use insertion sort for small subarrays.
  - Mergesort has too much overhead for small subarrays.
  - Cutoff – less than approx. 10 elements – use insertion sort.

```
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);
}
```

Mergesort cutoff for insertion sort: visualization



Visual trace of top-down mergesort for with cutoff for small subarrays

Mergesort: further improvement

- Stop if already sorted.
  - Stop if the largest element in first half ≤ smallest element in second half?
  - Helpful for partially ordered arrays.

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

```
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(a[mid+1], a[mid])) return;
   merge(a, aux, lo, mid, hi);
}
```

Mergesort: further improvement

- Eliminate copying to auxiliary array. Save time but not memory.
- Exchange the role of the input and auxiliary array in each recursive call.

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

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

merge from `a[]` to `aux[]`

assumes `aux[]` is initialized to `a[]` once, before recursive calls

switch roles of `aux[]` och `a[]`

Java 6 systemsort

- The chosen algorithm for sorting objects = mergesort.
  - Cutoff to insertion sort = 7.
  - Stop if already sorted test.
  - Use the no copying to auxillary array trick.

Arrays.sort(a)

http://www.java2s.com/Open-Source/Java/6.0-JDK-Modules/j2me/java/util/Arrays.java.html

Bottom-up Mergesort

- Basic method
  - Pass through the array, merging subarrays of size 1.
  - Repeat for subarrays av 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
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
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																
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

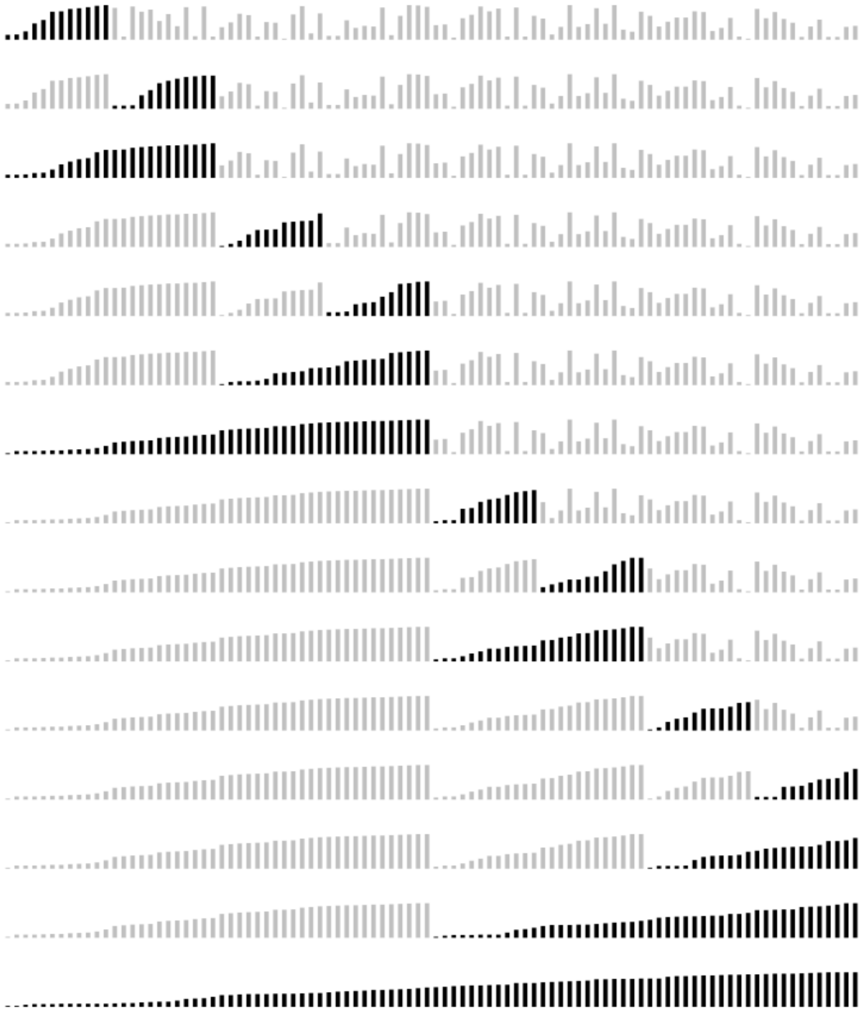
```
public class MergeBU
{
   private static void merge(...)
   {  /* as before */  }

   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));
   }
}
```

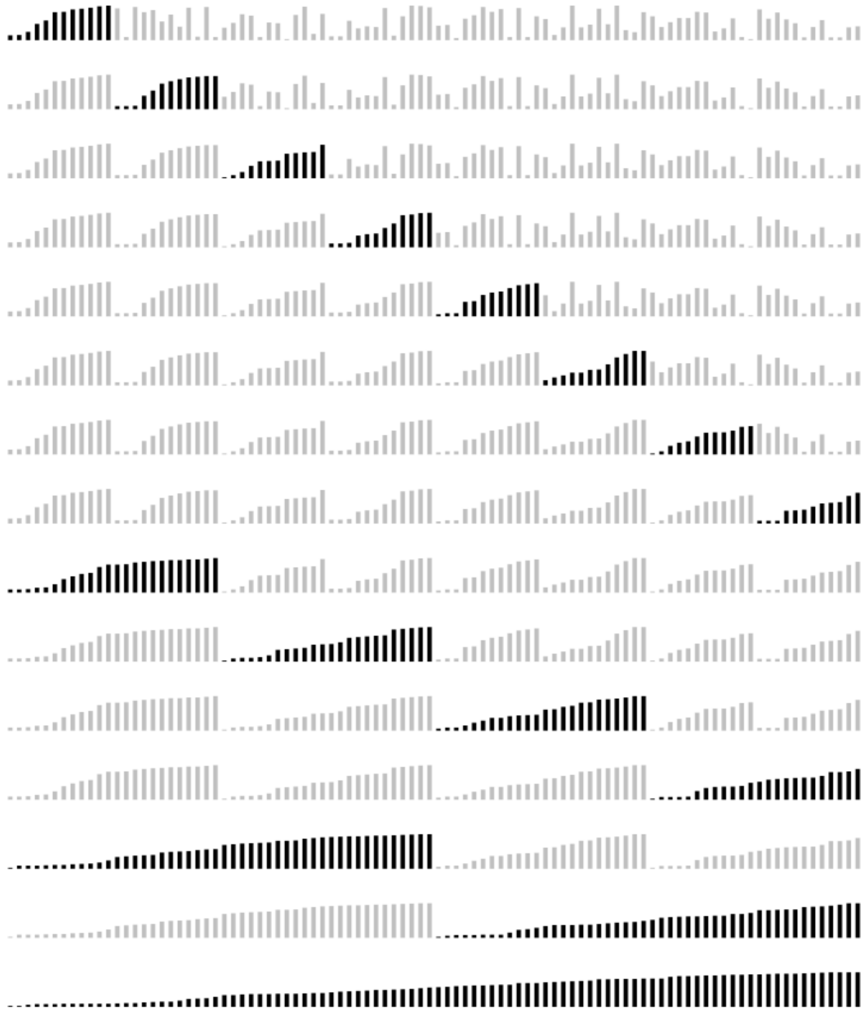
approximately 10% slower than,
top-down mergesort on most systems

Conclusion. Simple non recursive version of mergesort.

Mergesort: visualization



top-down mergesort (cutoff = 12)



bottom-up mergesort (cutoff = 12)

Computational complexity of mergesort

Computational complexity

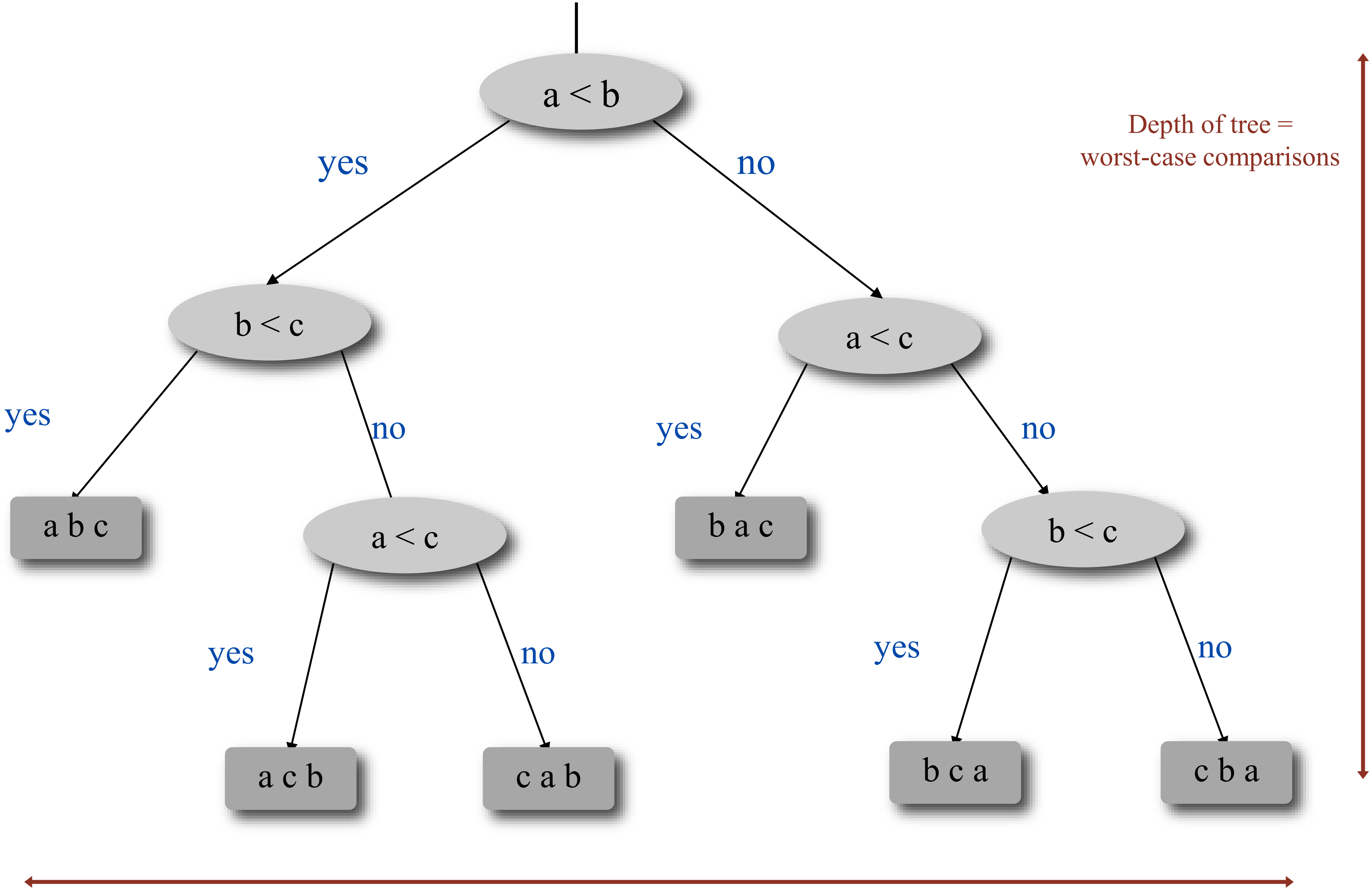
- Cost model. Number of operations
- Upped bound. Cost guarantee given by some algorithm X.
- Lower bound. No algorithm can cost less than the lower bound.
- Optimal algoritm. Algorithm with best cost guarantee för X.

Lower bound ~ upper bound

- Exempel: sortng.
  - Model of computation: decision tree.
  - Cost model: # compares
  - Upper bound: $\sim N \lg N$ från mergesort.
  - Lower bound: ?
  - Optimal algoritm: ?

Can only access by comparisons
(e.g. Java Comparable framework)

Decision tree (for 3 distinct keys a, b, och c)

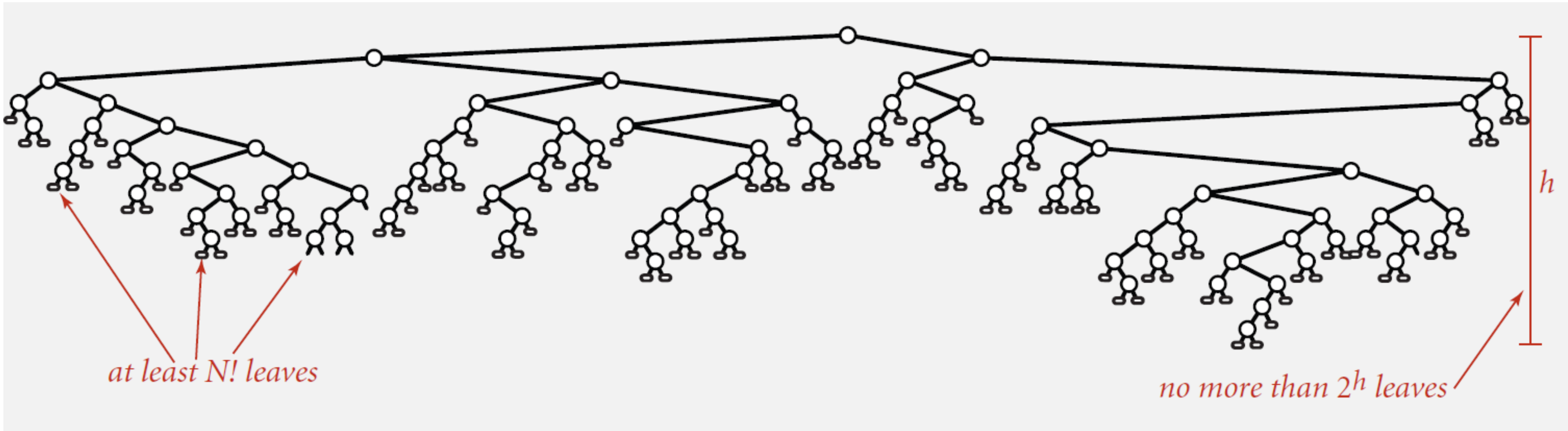


Every leaf corresponds to one distinct order,

need a leaf for every possible order.

Comparison - based lower bound for sorting

- Theorem. Alla comparison-based sorting algorithmer must compare at least $\lg(N!) \sim N \lg N$ times in the worst-case
- Proof.
  - We assume that the array consists of N distinct values a_1 to a_N.
  - Worst case is determined by the depth (height) h of the decision tree..
  - A binary tree of depth h has at most 2^h leaves.
  - $N!$ different orders $\Rightarrow$ at least $N!$ leaves.



Comparison - based lower bound for sorting

- Theorem. All comparison-based sorting algorithms must compare $\lg(N!) \sim N \lg N$ times in the worst-case.

- Proof.
  - We assume that tha array consists of N distinct values a_1 to a_N.
  - Worst case is determined by the height h of the decision tree.
  - A binary tree av height h has at most 2^h leaves.
  - $N!$ Different possible orderings $\Rightarrow$ at least $N!$ leaves.

$$2^h \geq \#\,\text{leaves} \geq N!$$
$$\Rightarrow \quad h \geq \underbrace{\lg(N!) \sim}_{\text{Stirlings formel}} N \lg N$$

Time complexity for sorting

- Cost model . Number of operations.
- Upper bound. Guarantee provided by some algorithm.
- Lower bound. Proof that no algorithm X can have a lower cost than the limit.
- Optimal algorithm. Upper bound ~lower bound.

- Example: sorting
  - Model: decision tree.
  - Cost model : # comparisons.
  - Upper bound ~ $N \lg N$ fom mergesort.
  - Lower bound: ~ $N \lg N$.
  - Optimal algorithm = mergesort.

- Primary goal of algorithm design: optimal algorithms.

Context

- Comparisons? Mergesort is optimal with respect to comparisons
  - Time complexity
- Space? Mergesort is not optimal with respect to memory
  - Auxillary array

Lesson. Theory helps us find solutions - and stops us from attempting the impossible.

Question: Is there an optimal sorting algorithm with respect to both time and space?

Time complexity in context

- Faster sorting might be achieved if the algorithmen can take advantage of:

  - Invariants in the input.
    Ex.: insertion sort need only a linear number of comparions on partially sorted array.
    Covered in previous lecture

  - Key distribution.
    Ex.: When the array contains many duplicates (i.e. if the number of distinct keys is limited). More on this later

Mergesort comparators

Need for comparators

- Forst step to generalize sorting
  - Comparable and compareTo
  - To be able to compare and sort user-defined
  - To implement generic sorting functions

- Second step
  - Work with multiple order relations on the same objects
  - Very common
  - Complex records

Sorting countries by number of gold medals

NOC	Gold	Silver	Bronze	Total
United States (USA)	46	29	29	104
China (CHN)§	38	28	22	88
Great Britain (GBR)*	29	17	19	65
Russia (RUS)§	24	25	32	81
South Korea (KOR)	13	8	7	28
Germany (GER)	11	19	14	44
France (FRA)	11	11	12	34
Italy (ITA)	8	9	11	28
Hungary (HUN)§	8	4	6	18
Australia (AUS)	7	16	12	35

Sorting countries by the total number of medals

NOC	Gold	Silver	Bronze	Total
United States (USA)	46	29	29	104
China (CHN)§	38	28	22	88
Russia (RUS)§	24	25	32	81
Great Britain (GBR)*	29	17	19	65
Germany (GER)	11	19	14	44
Japan (JPN)	7	14	17	38
Australia (AUS)	7	16	12	35
France (FRA)	11	11	12	34
South Korea (KOR)	13	8	7	28
Italy (ITA)	8	9	11	28

Sorting music by the name of the artist

	Name	Artist ▲	Time	Album
12	☑ Let It Be	The Beatles	4:03	Let It Be
13	☑ Take My Breath Away	BERLIN	4:13	Top Gun - Soundtrack
14	☑ Circle Of Friends	Better Than Ezra	3:27	Empire Records
15	☑ Dancing With Myself	Billy Idol	4:43	Don't Stop
16	☑ Rebel Yell	Billy Idol	4:49	Rebel Yell
17	☑ Piano Man	Billy Joel	5:36	Greatest Hits Vol. 1
18	☑ Pressure	Billy Joel	3:16	Greatest Hits, Vol. II (1978 - 1985) (Disc 2)
19	☑ The Longest Time	Billy Joel	3:36	Greatest Hits, Vol. II (1978 - 1985) (Disc 2)
20	☑ Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
21	☑ Sunday Girl	Blondie	3:15	Atomic: The Very Best Of Blondie
22	☑ Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
23	☑ Dreaming	Blondie	3:06	Atomic: The Very Best Of Blondie
24	☑ Hurricane	Bob Dylan	8:32	Desire
25	☑ The Times They Are A-Changin'	Bob Dylan	3:17	Greatest Hits
26	☑ Livin' On A Prayer	Bon Jovi	4:11	Cross Road
27	☑ Beds Of Roses	Bon Jovi	6:35	Cross Road
28	☑ Runaway	Bon Jovi	3:53	Cross Road
29	☑ Rasputin (Extended Mix)	Boney M	5:50	Greatest Hits
30	☑ Have You Ever Seen The Rain	Bonnie Tyler	4:10	Faster Than The Speed Of Night
31	☑ Total Eclipse Of The Heart	Bonnie Tyler	7:02	Faster Than The Speed Of Night
32	☑ Straight From The Heart	Bonnie Tyler	3:41	Faster Than The Speed Of Night
33	☑ Holding Out For A Hero	Bonny Tyler	5:49	Meat Loaf And Friends
34	☑ Dancing In The Dark	Bruce Springsteen	4:05	Born In The U.S.A.
35	☑ Thunder Road	Bruce Springsteen	4:51	Born To Run
36	☑ Born To Run	Bruce Springsteen	4:30	Born To Run
37	☑ Jungleland	Bruce Springsteen	9:34	Born To Run

Sorting music by the name of the song

	Name	Artist	Time	Album
1	Alive	Pearl Jam	5:41	Ten
2	All Over The World	Pixies	5:27	Bossanova
3	All Through The Night	Cyndi Lauper	4:30	She's So Unusual
4	Allison Road	Gin Blossoms	3:19	New Miserable Experience
5	Ama, Ama, Ama Y Ensancha El …	Extremoduro	2:34	Deltoya (1992)
6	And We Danced	Hooters	3:50	Nervous Night
7	As I Lay Me Down	Sophie B. Hawkins	4:09	Whaler
8	Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
9	Automatic Lover	Jay-Jay Johanson	4:19	Antenna
10	Baba O'Riley	The Who	5:01	Who's Better, Who's Best
11	Beautiful Life	Ace Of Base	3:40	The Bridge
12	Beds Of Roses	Bon Jovi	6:35	Cross Road
13	Black	Pearl Jam	5:44	Ten
14	Bleed American	Jimmy Eat World	3:04	Bleed American
15	Borderline	Madonna	4:00	The Immaculate Collection
16	Born To Run	Bruce Springsteen	4:30	Born To Run
17	Both Sides Of The Story	Phil Collins	6:43	Both Sides
18	Bouncing Around The Room	Phish	4:09	A Live One (Disc 1)
19	Boys Don't Cry	The Cure	2:35	Staring At The Sea: The Singles 1979-1985
20	Brat	Green Day	1:43	Insomniac
21	Breakdown	Deerheart	3:40	Deerheart
22	Bring Me To Life (Kevin Roen Mix)	Evanescence Vs. Pa…	9:48	
23	Californication	Red Hot Chili Pepp…	1:40	
24	Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
25	Can't Get You Out Of My Head	Kylie Minogue	3:50	Fever
26	Celebration	Kool & The Gang	3:45	Time Life Music Sounds Of The Seventies - C

Comparable interface: repetition

- `Comparable` interface: sorting with the natural order of a data type.

```
public class Date implements Comparable<Date> {
   private final int month, day, year;

   public Date(int m, int d, int y) {
      month = m;
      day   = d;
      year  = y;
   }
      …
   public int compareTo(Date that)   {
      if (this.year  < that.year ) return -1;
      if (this.year  > that.year ) return +1;
      if (this.month < that.month) return -1;
      if (this.month > that.month) return +1;
      if (this.day   < that.day  ) return -1;
      if (this.day   > that.day  ) return +1;
      return 0;
   }
}
```

natural order

Comparator interface

- `Comparator` interface: sorting with an alternative order.

public interface Comparator<Key>	
`int compare(Key v, Key w)`	*jämför nycklar v och w*

- Requirement. Must define a total order.

natural order	`Now is the time`
case insensitive	`is Now the time`
Spanish language	`café cafetero cuarto churro nube ñoño`
British phone book	`McKinley Mackintosh`

pre-1994 order for digraphs ch och ll och rr

Decouple comparator interface from the data type

- Can be used with Java system sort:
  - Create a `Comparator` object.
  - Send as a second argument to `Arrays.sort()`.

uses natural order

Uses and alternative order

given by Comparator<String> objekt

```
String[] a;
...
Arrays.sort(a);
...
Arrays.sort(a, String.CASE_INSENSITIVE_ORDER);
...
Arrays.sort(a, Collator.getInstance(new Locale("es")));
...
Arrays.sort(a, new BritishPhoneBookOrder());
...
```

Comparator interface:

- To used comparators in sorting:
  - Use `Object` instead of `Comparable`.
  - Send the `Comparator` to `sort()` and `less()` and use it in `less()`.

insertion sort with a Comparator

```
public static void sort(Object[] a, Comparator comparator)
{
   int N = a.length;
   for (int i = 0; i < N; i++)
      for (int j = i; j > 0 && less(comparator, a[j], a[j-1]); j--)
         exch(a, j, j-1);
}

private static boolean less(Comparator c, Object v, Object w) {
   return c.compare(v, w) < 0;
}

private static void exch(Object[] a, int i, int j){
   Object swap = a[i]; a[i] = a[j]; a[j] = swap;
}
```

To implement a Comparator interface

- Define a nested class that implements the `Comparator` interface.
- Implement the `compare()` method.

```
public class Student {
private final String name;
   private final int section;
   ...

   public static class ByName implements Comparator<Student>
   {
     public int compare(Student v, Student w)
      {  return v.name.compareTo(w.name);  }
   }

   public static class BySection implements Comparator<Student>
   {
      public int compare(Student v, Student w)
      {  return v.section - w.section;  }
   }
}
```

To implement a Comparator interface

- Define a nested class with implement the `Comparator` interface.
- Implement the `compare()` method.

`Arrays.sort(a, new Student.ByName());`

Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

`Arrays.sort(a, new Student.BySection());`

Furia	1	A	766-093-9873	101 Brown
Rohde	2	A	232-343-5555	343 Forbes
Andrews	3	A	664-480-0023	097 Little
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Kanaga	3	B	898-122-9643	22 Brown
Battle	4	C	874-088-1212	121 Whitman
Gazsi	4	B	766-093-9873	101 Brown

To implement a Comparator interface

- Definie a nested class with implement the `Comparator` interface.
- Implement the `compare()` method.

```
public class Student {
   public static final Comparator<Student> BY_NAME = new ByName();
   public static final Comparator<Student> BY_SECTION = new BySection();
   private final String name;
   private final int section;
   ...

   public static class ByName implements Comparator<Student>
   {
     public int compare(Student v, Student w)
      {  return v.name.compareTo(w.name);  }
   }

   public static class BySection implements Comparator<Student>
   {
      public int compare(Student v, Student w)
      {  return v.section - w.section;  }
   }
}
```

Mergesort and stability

A problem ?

- Common application. First, sort by namn; thereafter by ”Section”.

```
Selection.sort(a, new Student.ByName());
```

Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

```
Selection.sort(a, new Student.BySection());
```

Furia	1	A	766-093-9873	101 Brown
Rohde	2	A	232-343-5555	343 Forbes
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Andrews	3	A	664-480-0023	097 Little
Kanaga	3	B	898-122-9643	22 Brown
Gazsi	4	B	766-093-9873	101 Brown
Battle	4	C	874-088-1212	121 Whitman

- @#%&@! Students in section 3 are no longer sorted by name.

- A stable sorting maintains the relative order of records with equal keys
- The above sorting not stable.

Stability

- Which sorting algorithms are stable?
  Need to check algorithm (and implementation) to check for stability.

sorted by time		sorted by location (not stable)		sorted by location (stable)	
Chicago	09:00:00	Chicago	09:25:52	Chicago	09:00:00
Phoenix	09:00:03	Chicago	09:03:13	Chicago	09:00:59
Houston	09:00:13	Chicago	09:21:05	Chicago	09:03:13
Chicago	09:00:59	Chicago	09:19:46	Chicago	09:19:32
Houston	09:01:10	Chicago	09:19:32	Chicago	09:19:46
Chicago	09:03:13	Chicago	09:00:00	Chicago	09:21:05
Seattle	09:10:11	Chicago	09:35:21	Chicago	09:25:52
Seattle	09:10:25	Chicago	09:00:59	Chicago	09:35:21
Phoenix	09:14:25	Houston	09:01:10	Houston	09:00:13
Chicago	09:19:32	Houston	09:00:13	Houston	09:01:10
Chicago	09:19:46	Phoenix	09:37:44	Phoenix	09:00:03
Chicago	09:21:05	Phoenix	09:00:03	Phoenix	09:14:25
Seattle	09:22:43	Phoenix	09:14:25	Phoenix	09:37:44
Seattle	09:22:54	Seattle	09:10:25	Seattle	09:10:11
Chicago	09:25:52	Seattle	09:36:14	Seattle	09:10:25
Chicago	09:35:21	Seattle	09:22:43	Seattle	09:22:43
Seattle	09:36:14	Seattle	09:10:11	Seattle	09:22:54
Phoenix	09:37:44	Seattle	09:22:54	Seattle	09:36:14

no longer sorted by time

still sorted by time

Stability: insertion sort

- Theorem. Insertion sort is stable.

```
public class Insertion
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
         for (int j = i; j > 0 && less(a[j], a[j-1]); j--)
            exch(a, j, j-1);
   }
}
```

i	j	0	1	2	3	4
0	0	B_1	A_1	A_2	A_3	B_2
1	0	A_1	B_1	A_2	A_3	B_2
2	1	A_1	A_2	B_1	A_3	B_2
3	2	A_1	A_2	A_3	B_1	B_2
4	4	A_1	A_2	A_3	B_1	B_2
		A_1	A_2	A_3	B_1	B_2

Proof. If keys are equal, then insertion sort will never move one key past the other.

Stability: selection sort

- Theorem. Selection sort is not stable.

```
public class Selection
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
      {
         int min = i;
         for (int j = i+1; j < N; j++)
            if (less(a[j], a[min]))
               min = j;
         exch(a, i, min);
      }
   }
}
```

i	min	0	1	2
0	2	B_1	B_2	A
1	1	A	B_2	B_1
2	2	A	B_2	B_1
		A	B_2	B_1

- Proof by counter-example: Long distance swapping kan move an item past an item with the same key.

Stability: shellsort

- Sats. Shellsort sort is not stabile.

```
public class Shell {
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      int h = 1;
      while (h < N/3) h = 3*h + 1;
      while (h >= 1)
      {
         for (int i = h; i < N; i++)
         {
            for (int j = i; j > h && less(a[j], a[j-h]); j -= h)
               exch(a, j, j-h);
         }
         h = h/3;
      }
   }
}
```

h	0	1	2	3	4
	B_1	B_2	B_3	B_4	A_1
4	A_1	B_2	B_3	B_4	B_1
1	A_1	B_2	B_3	B_4	B_1
	A_1	B_2	B_3	B_4	B_1

- Proof by counter-example. Long distance swapping can move an element past another with the same key.

Stability: mergesort

- Theorm. 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);
   }

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

- Proof. It is enough to show that the merge operation is stable. Why?

Stability: mergesort

- Theorem. Merging 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++];
   }
}
```

0	1	2	3	4	5	6	7	8	9	10
A_1	A_2	A_3	B	D	A_4	A_5	C	E	F	G

- Proof. Take the value from the left subarray if the keys are equal.

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
mergesort		✓	$\frac{1}{2} N \lg N$	$N \lg N$	$N \lg N$	$N \log N$ guarantee; stable
?	✓	✓	N	$N \lg N$	$N \lg N$	holy sorting grail