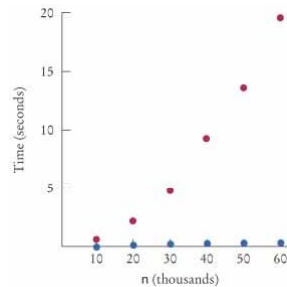


Comparing Sorting Algorithms

- Run Stopwatch to measure runtime
- Graph
 - red = SelectionSort
 - blue = MergeSort
- Result?



Runtime

- How often does SelectionSort access the array?
 - for an array with 5 elements
 - outer loop: gone through 5 times
 - 1 swap (2 accesses each)
 - inner loop: 5 times, 4 times, 3 times, 2 times, 1 time
 - total: $5 \cdot 2 + 5 + 4 + 3 + 2 + 1 = 25$
 - for an array with n elements
 - outer loop: gone through n times
 - inner loop: n, n-1, n-2, ... 1
 - total: $2n + (n(n+1))/2$
 - in the order of n^2
 - $O(n^2)$

Runtime

- How often does MergeSort access the arrays?
 - look at each level in the tree diagram during merging
 - for each element there are 5 accesses per level (copying: 2, comparing: 3)
 - how many levels are there?
 - if the array has 8 elements
 - level 1: $4 + 4$
 - level 2: $2 + 2 + 2 + 2$
 - level 3: $1 + 1 + 1 + 1 + 1 + 1 + 1 + 1$
 - if the array has n elements
 - $\lg n$ levels
 - $\lg =$ logarithm to the base 2
 - total: $5n \lg n$
 - $O(n \log n)$

Searching

- Linear search
 - in an unsorted array
- Binary search
 - in a sorted array
- Code?
- Runtime?

Sorting in Practice

- Use methods from Java library
 - Arrays. `sort(int[])`
 - Arrays. `sort(Object[])`
 - uses MergeSort
- How are objects (e.g. Strings) sorted?
 - must implement interface Comparable

```
interface Comparable< T > {
    int compareTo(T other)
}
```
 - `compareTo` returns:
 - -1 if this < other
 - 0 if this equals other
 - 1 if this > other

O-Notation

- An abbreviated notation for the runtime of algorithms
 - independent of computer speed
 - independent of implementation details
 - constant factors are ignored
 - describes how very large inputs behave
- SelectionSort: $O(n^2)$
- MergeSort: $O(n \log n)$
- Linear Search: $O(n)$
- Binary Search: $O(\log n)$