Sorting

Need Comparison to Sort

- Given a set of objects, you need to know, for any two objects, a & b
  - is a<b, is a==b, or is a>b
- All sorts are built on comparison
  - Comparable interface : this.compareTo(that)
    - returns a negative integer if this < that
    - returns 0 if this==that
    - returns a positive integer if this > that
  - Comparator interface : compare(this,that)
    - Typically implemented as a separate independent concrete class
- Java sorts objects which are "comparable", or uses a "comparator" to sort objects.

Collections Sort Methods

- public static void sort(List<T> list)
  - T is any type (class) which extends the Comparable interface (e.g. supports "compareTo"
- public static void sort(List<T> list,Comparator<T> c)
  - T does not have to implement Comparable
  - "c" must be a Comparator of <T>

Selection sort

Chapter 14.1

- Algorithm:
  - Given a list of elements, find the smallest one
  - Switch the smallest element with the first element
  - Re-apply this algorithm to the remaining items (after the first)

11	9	17	5	12
5	9	17	11	12
5	9	17	11	12
5	9	11	17	12
5	9	11	12	17

Analyzing Selection sort

11	9	17	5	12	4
5	9	17	11	12	3
5	9	17	11	12	2
5	9	11	17	12	1
5	9	11	12	17	10

First Row:

Assume 11 is smallest…

compare 11 to 9, 9 is smaller, so 9 is smallest

compare 17 to 9, 9 is still smallest

compare 9 to 5, 5 is smallest

compare 5 to 12, 5 is smallest

Analyzing Selection sort

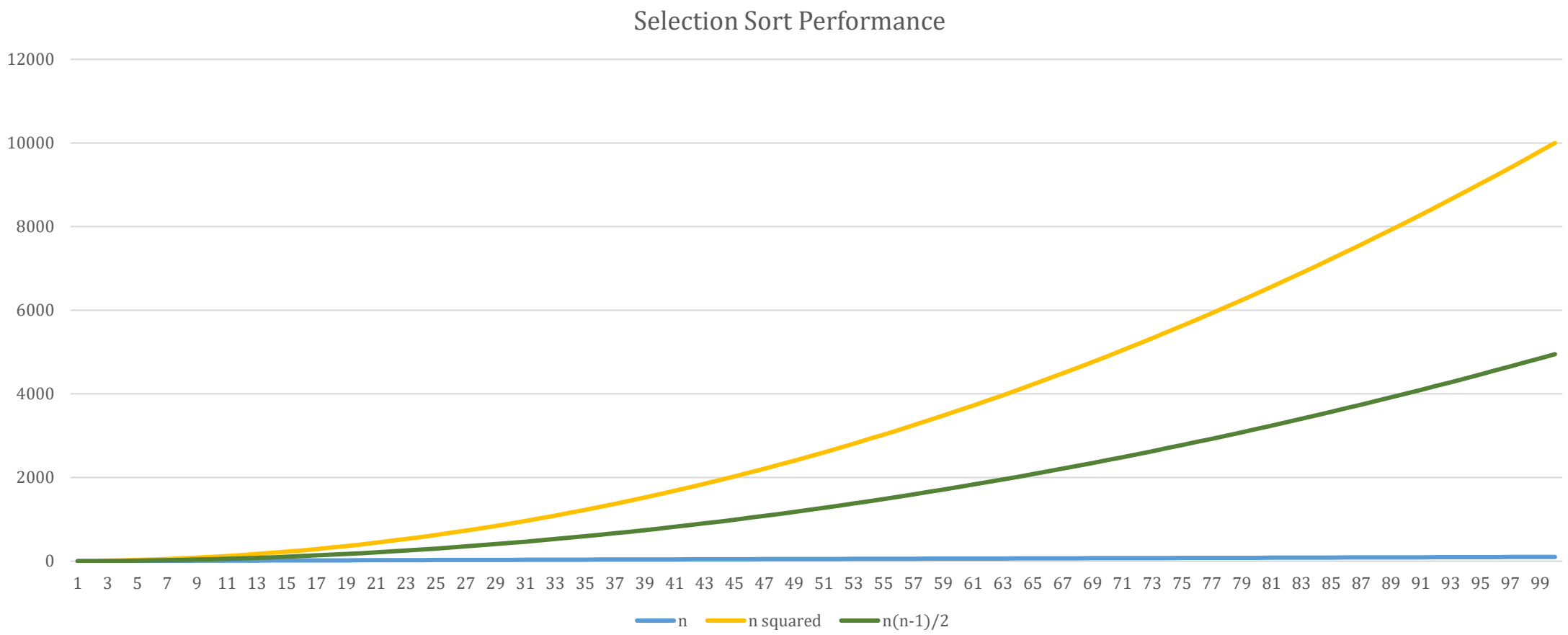
11	9	17	5	12	4
5	9	17	11	12	3
5	9	17	11	12	2
5	9	11	17	12	1
5	9	11	12	17	10

Number of compares required

Note: No Extra Memory is needed!

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$$

Selection Sort Performance $O(n^2)$



Insertion sort

Chapter 14.2

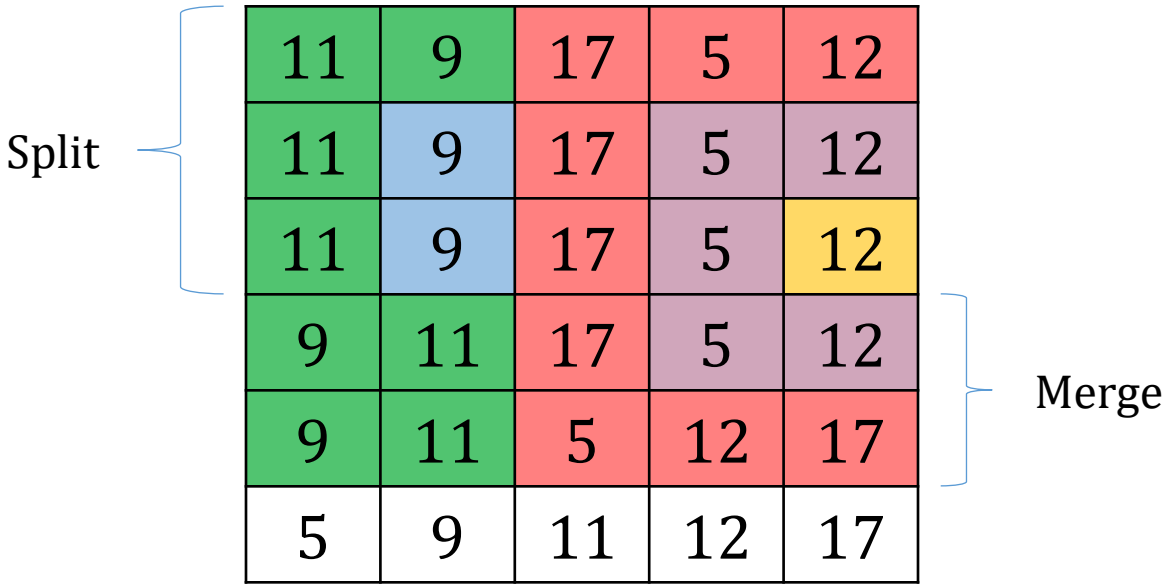
- Algorithm:
  - Given a list of elements, Assume the first is sorted
  - Insert the next element in the sorted portion at the correct place
  - Repeat with the next unsorted item until the list is sorted
  - Also $O(n^2)$

11	9	17	5	12
9	11	17	5	12
9	11	17	5	12
5	9	11	17	12
5	9	11	12	17

Merge Sort

Chapter 14.4

- Algorithm
  - If list size > 1, split list in two
  - Sort (recursively) each sub-list
  - Merge the two sorted sub-lists



- Analysis
  - Each merge takes O(n) operations
  - Need $\log_2(n)$ merges
  - Total time: n log(n)

- Merge Needs Extra Memory
  - Need an entire copy of the array