Algorithms for Sorting

• How long should it take to sort n values?

• Some different ways to do it:
  – Insertion Sort [Good for small n. Bad for big n. ]
  – Quick Sort     [Excellent on average. Worst case = Insertion.]

• In each case, assume values are integers in an array called data and we wish to sort the slots from lo, up to but not including hi.
Insertion Sort

• Main idea:
  – Make a series of passes over the array.
  – After the first pass, the first 1 element is sorted.
  – After the second pass, the first 2 elements are sorted.
  – After the k-th pass, the first k elements are sorted.

• Algorithm:
  – Make n passes. On pass number i, do the following:
    • Take element i and insert it into the correct position among the first i-1 elements.
    • To do this, shift the elements bigger than it up by one space to make room (using the space the i-th element came from).
At Each Stage

• Suppose we have sorted the first 4 slots:
  [2,3,8,10,5,7,1,4,6,9]

• Extract the first “unsorted” number to a variable:
  [2,3,8,10,_,7,1,4,6,9]  temp = 5

• Shuffle earlier numbers up as long they are bigger:
  [2,3,8,_,10,7,1,4,6,9]  temp = 5
  [2,3,_,8,10,7,1,4,6,9]  temp = 5

• Now entries before the space are smaller or equal and after the space are bigger.

• Put the blue number in the space.
  [2,3,5,8,10,7,1,4,6,9]
Example

- [10, 8, 2, 3, 5, 7, 1, 4, 6, 9]
- [8, 10, 2, 3, 5, 7, 1, 4, 6, 9]
- [2, 8, 10, 3, 5, 7, 1, 4, 6, 9]
- [2, 3, 8, 10, 5, 7, 1, 4, 6, 9]
- [2, 3, 5, 8, 10, 7, 1, 4, 6, 9]
- [2, 3, 5, 7, 8, 10, 1, 4, 6, 9]
- [1, 2, 3, 5, 7, 8, 10, 4, 6, 9]
- [1, 2, 3, 4, 5, 7, 8, 10, 6, 9]
- [1, 2, 3, 4, 5, 6, 7, 8, 10, 9]
- [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
void insertionSort(int[] data, int lo, int hi) {
    for (int i = lo; i < hi; i++) {
        // Start of iter’n: entries lo..i-1 are in order.
        // We will insert data[lo+i] into correct place.
        int newGuy = data[i];
        // Shuffle previous numbers up so long as bigger.
        int j = i;
        while (j > lo && data[j-1] > newGuy) {
            data[j] = data[j-1];
            j--;
        }
        // Insert new number into gap.
        data[j] = newGuy;
        // End of iter’n: entries lo..i are in order.
    }
}
Algorithm Analysis

• How does this behave if the entries are all in order?

  Time is proportional to $n$.

  (You don’t need to be able to figure this out, but you should understand what it means.)

• How does it behave if the entries are in reverse order?

  Time is proportional to $n^2$.

  (You don’t need to be able to figure this out, but you should understand what it means.)
Quick Sort

• Algorithm:
  – An array of size 0 or 1 is sorted.
  – For larger arrays do the following:
    • Pick an element in the array. Call this the “pivot.”
    • Move the elements of the array so that all elements ≤ pivot are to the left of it (smaller array index), and all elements > pivot are to the right of it (bigger array index).
    • Sort the left part.
    • Sort the right part.
Example

• Input array: \[10, 8, 2, 3, 7, 5, 1, 4, 6, 9\]
• Pick a pivot: \[10, 8, 2, 3, 7, 5, 1, 4, 6, 9\]
• Move elements: \[2, 3, 5, 1, 4, 6, 7, 10, 8, 9\]

• Sort left part (using the same method):
  – \[2, 3, 5, 1, 4, 6, 7, 10, 8, 9\]
  – \[1, 4, 2, 3, 5, 6, 7, 10, 8, 9\]
    ...
  – \[1, 2, 3, 4, 5, 6, 7, 10, 8, 9\]

• Sort right part (using the same method):
  – \[1, 2, 3, 4, 5, 6, 7, 10, 8, 9\]
  – \[1, 2, 3, 4, 5, 6, 7, 8, 9, 10\]

• Done: \[1, 2, 3, 4, 5, 6, 7, 8, 9, 10\]