Class 6: Sorting Algorithms

Betül Demirkaya

August 13, 2015

Complexity

Sorting Algorithms

Recursion

Complexity

- The amount of time / the number of operations necessary to complete a task.

Complexity

- The amount of time / the number of operations necessary to complete a task.
- $O(n)$ notation
  - Gives complexity in terms of the size of the input.
  - eg. How many comparisons/swaps do I need to sort a list of n items?
  - Consider only the term with the highest order of n.

Complexity

- The amount of time / the number of operations necessary to complete a task.
- $O(n)$ notation
  - Gives complexity in terms of the size of the input.
  - eg. How many comparisons/swaps do I need to sort a list of n items?
  - Consider only the term with the highest order of n.
- Check the best case, the average case and the worst case.

Insertion Sort

- Start with the element in the second position.
- Insert it to the appropriate position among the numbers to its left.
    - Check whether it is greater than the last element to its left.
    - If not, check the second to last element to its left.
    - ...
- Continue with the element in the third position.

Selection Sort

- Go over the unsorted list to find the minimum and place it as your first element of your sorted list.
- Repeat.

Bubble Sort

- Compare - swap stage
  - Compare the first two elements and swap them if necessary.
  - Compare the second and third elements and swap them if necessary.
  - Repeat until the end of the list.
- If you did any swaps in the first stage, repeat it with the first n-1 elements.
- Repeat.

Merge Sort

- Divide the list into sublists - each with one element.
- Merge the sublists to create new sublists - each with two elements.
- Repeat until you have a single list

Recursion

- Function calls itself.
- You need to know:
    - the base case
    - when to call the function
    - when to stop