Mergesort and Analysis

•••

Cis 110 Recitation 7/27/17

Big-Oh Analysis

- Describes the growth rate of a function for input size *N*.
  - Assume that all ‘simple’ operations (such as arithmetic operation or printing) take a constant time step (O(1)).
  - Given a generalized integer *N*, how much time will it take to run an algorithm?
  - We care only about the largest term and ignore the smaller terms.
  - *i. e.* , 2N^2 + 5N + 3 = O(N^2)
- Use to compare the efficiency of two algorithms.
  - is not a specific number, therefore making the algorithm analysis general enough to do so.

Big-Oh Analysis: Selection Sort

```java
public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        int index = i;
        // find smallest element in subarray
        for (int j = i + 1; j < arr.length; j++) {
            if (arr[j] < arr[index])
                index = j;
        }
        int smallerNumber = arr[index];
        arr[index] = arr[i];
        arr[i] = smallerNumber;
    }
}
```

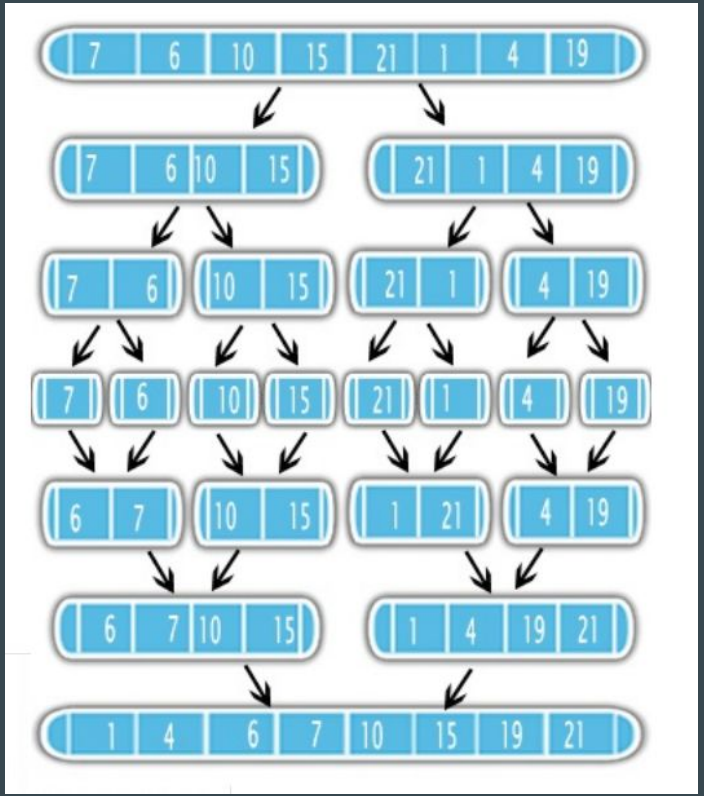
- Input: array length N
- For each time the outer for-loop runs (total: N - 1 times), the inner for-loop runs N - 2, N- 3, ..., 1 times, respectively. 1+ ... + (N-2) = O(N^2).

Mergesort

- A ‘Divide-and-Conquer’ algorithm:
  - Divide ordered list (such as an array) into two halves.
  - Recursively sort each half.
  - Merge two halves to make a whole.

- Analysis:
  - For an array of size N, merge sort is a O(N log N) algorithm.

- Takeaway:
  - The intuitive solution is not always the most effective one.



Mergesort and Big-Oh Analysis

- Selection Sort: O(N^2) (7/26 lecture slides pg. 16)

- Insertion Sort: (slides pg. 68-70)
  - Best case (array is already sorted): O(N)
  - Worst case (array elements are arranged in descending order): O(N^2)
  - Average case: O(N^2)

- MergeSort: O(N log N)
  - Since log(N) < N for all non-negative integers N, we know that N log N < N^2
  - Takeaway: The most intuitive solution is not always the most efficient solution.

Coding Exercise (Pt. 1): Mergesort on objects

Create a Person class, where the constructor takes in a firstName (String), lastName (String), and age (int) in that order.

- Make the fields (same as constructor parameters) private and write the corresponding methods to get the fields: getFirstName, getLastName, and getAge.

- void sort(Person[] arr): implement mergesort by to sort by last names.
  - You will need String's compareTo(String s) method, which compares two strings lexicographically.
  - s1.compareTo(s2) returns a int value that is:
    - less than 0 if s1 would come before s2 (i.e. ("string1".compareTo("string2")),
    - 0 if s1 and s2 are equal (i.e. ("example".compareTo("example")),
    - greater than 0 if s2 comes before s1 (i.e. ("string2".compareTo("string1")), .

Coding Exercise (Pt. 2): Mergesort on objects

Sometimes, it is not enough to sort by last name as it results in ties. Break the ties using the following rules:

- If two Person objects have the same last name, break tie using first name.
- If two Person objects have the same last and first names, break tie using age (youngest first).

Coding Exercise (Pt. 3): involved example for concept review

Try this challenge on your own. Create a Roster class with the field roster (a Person array of size capacity) and currentSize. Write the following methods:

- boolean isFull(): return true if roster is full, false otherwise.
- boolean isEmpty(): return true if roster is empty, false otherwise.
- int currentSize(): returns the number of people in roster.
- void add(Person p): add p to end of the roster if it is not already full.
- void remove(): remove the Person object at the end of the roster, if such element exists. 'Remove' an object by setting its reference to `null`.
- Person getPerson(int index): return Person at given index (starting at 0).
- void sort(Roster arr): implement mergesort by to sort by last names.