Intro to Data Structures

Lecture #18 – Sub-quadratic Sorts
November 5, 2013

Mark Stehlik

Outline for Today

- HW4, Q4 debrief
- Quiz Thursday (HW5, Stacks/Q's, Searching)
- Sub-Quadratic – O (n log n) Sorts
- Stability
- Is there a linear sort?
- Timing Summary

Outline for Today

- Q4
  - A small disaster (6 >= 80; 9 < 60)
  - Iterator is not hard!
- HW4
  - Cases to check
    - empty, first, middle, end
    - removeAll needed to address first in separate loop
  - Updating length in adds/clone/remove/split
  - == vs .equals

Outline for Today

– An efficiency question:

```
public MyLinkedList<Anything> clone()
{
    MyLinkedList<Anything> c = new MyLinkedList<Anything>();
    Node curr = first;
    for (int i = 0; i < length; i++)
    {
        c.add(get(i));
        curr = curr.next;
    }
    return c;
}
```

Sorting (Merge sort)

- Algorithm/Illustrate (w/numbers)
  - Divide the collection in half
  - Recursively sort the two halves (by calling mergesort)
  - Merge the halves back together
  - *Invariant*: the merged "halves" are sorted

Sorting (Merge sort)

- Analysis?
  - O(log n) - the number of times you can divide in half
  - O(n) - the time to merge two halves into a whole
  - O(log n) * O(n) --> O(n log n)
  - BUT (see previous note about "but”), this algorithm needs a separate, auxiliary, array to store the halves

Sorting (Quicksort)

- Algorithm/Illustrate (w/numbers)
  - "Randomly" pick an element about which to partition the collection into two parts
  - Partition the array around that value, called the pivot, so that the partition value ends up in final position, i.e., the array looks like: < pivot, <u>pivot</u>, >= pivot
  - Recursively sort the two parts (by calling quicksort)
  - *Invariant*: After the i^{th} pass, the i^{th} <u>partition value/pivot</u> is in its final position (i.e., all values to the left are less than the partition value/pivot and all the values to the right are greater than or equal to the partition value/pivot)

Sorting (Quicksort)

- Analysis? Well…
  - O(n log n) if all goes well in choosing the pivot - which is when what is true about where the pivot ends up?
  - BUT, $O(n^2)$ in worst case - when might that be?
  - In practice, though, usually O(n log n) and faster (better constant) than merge sort (and no need for auxiliary array)

Sorting (Stable sorts)

- Definition
  - A stable sort maintains the relative position of equal elements
  - Benefit? If you were sorting students and sorted by name and then by gender, then you'd get a list that was sorted by gender, but alphabetical within gender
- Which sorts preserve stability?
  - Insertion sort
  - Merge sort
- And the others
  - Selection sort - no (why?)
  - Quicksort - not the naïve algorithm, anyway

Sorting (can we do better than n log n?)

- Bucket sort
  - Algorithm - need a bucket for each possible value
  - Analysis
    - if O(1) to insert an element into a bucket --> O(n) to insert all elements
    - If O(n) to collect all the buckets --> O(n) overall
  - Limitations - finite number of possible values (finite buckets)
- Radix sort for integers
  - 10 buckets (0 - 9); sort integers into appropriate bucket starting with least significant digit (int % 10), collect them and sort by next least significant digit until out of digits
  - O(n * k) where k is number of digits (assuming?…)
  - Since k is small (usually) --> O(n) overall

Sorting (summary)

Sort	Best case	Average	Worst case	Stable?
Selection				
Insertion				
Merge				
Quicksort				

Sorting (summary)

Sort	Best case	Average	Worst case	Stable?
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	no
Insertion	O(n)	$O(n^2)$	$O(n^2)$	yes
Merge	O(n log n)	O(n log n)	O(n log n)	yes
Quicksort	O(n log n)	O(n log n)	$O(n^2)$	??