

Flipped Lecture Concepts

February 19, 2014

Focus this week

- Mergesort
 - algorithm
 - implementation
 - Analysis
 - Applications
 - Bottom-up mergesort
- Quicksort
 - algorithm
 - Average and Bad cases
 - Analysis (worst and average cases)
 - Special cases (equal elements)
 - 3-way quicksort

Mergesort

```
public class Merge
{
    private static void merge(...)
    { /* as before */ }

    private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
    {
        if (hi <= lo) return;
        int mid = lo + (hi - lo) / 2;
        sort(a, aux, lo, mid);
        sort(a, aux, mid+1, hi);
        merge(a, aux, lo, mid, hi);
    }

    public static void sort(Comparable[] a)
    {
        Comparable[] aux = new Comparable[a.length];
        sort(a, aux, 0, a.length - 1);
    }
}
```

Mergesort tree and recursion order

Merge does all the work

Merging: Java implementation

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
    assert isSorted(a, lo, mid);    // precondition: a[lo..mid] sorted
    assert isSorted(a, mid+1, hi);  // precondition: a[mid+1..hi] sorted

    for (int k = lo; k <= hi; k++)
        aux[k] = a[k];              copy

    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++)
    {
        if (i > mid)                a[k] = aux[j++];
        else if (j > hi)            a[k] = aux[i++];
        else if (less(aux[j], aux[i])) a[k] = aux[j++];
        else                        a[k] = aux[i++];
    }

    assert isSorted(a, lo, hi);    // postcondition: a[lo..hi] sorted
}
```

Challenge: prove merge code is correct

Mergesort Analysis

- Recurrence

$$C(N) \leq C(\lceil N/2 \rceil) + C(\lfloor N/2 \rfloor) + N \quad \text{for } N > 1, \text{ with } C(1) = 0.$$

↑ ↑ ↑
left half right half merge
↓ ↓ ↓

$$A(N) \leq A(\lceil N/2 \rceil) + A(\lfloor N/2 \rfloor) + 6N \quad \text{for } N > 1, \text{ with } A(1) = 0.$$

We solve the recurrence when N is a power of 2. ← result holds for all N

$$D(N) = 2 D(N/2) + N, \text{ for } N > 1, \text{ with } D(1) = 0.$$

Bottom-up mergesort

non-recursive

- Bottom-up mergesort is $n \lg n$

```
public class MergeBU
{
    private static void merge(...)
    { /* as before */ }

    public static void sort(Comparable[] a)
    {
        int N = a.length;
        Comparable[] aux = new Comparable[N];
        for (int sz = 1; sz < N; sz = sz+sz)
            for (int lo = 0; lo < N-sz; lo += sz+sz)
                merge(a, aux, lo, lo+sz-1, Math.min(lo+sz+sz-1, N-1));
    }
}
```

Applications

- Mergesort is a linearithmic stable sorting algorithm. Guaranteed $n \lg n$ performance
- Stable means...
- Java 6 system sort is implemented using mergesort with cutoff to insertion sort when sub-array size becomes < 8
- Optimizations
 - `is_sorted` test to stop recursion

Comparison based sorting

- There exists no comparison based sorting algorithm that is better than $n \lg n$. Is this true? How do you prove this?

Quicksort demo

- Distinct keys

– 12 34 56 67 10 30 20 15 68 70

Quicksort demo

- All equal keys
 - 12 12 12 12 12 12 12

3-way Quicksort

- Some/all equal keys
 - 12 34 12 67 12 30 12 15 34 70

Dijkstra 3-way partitioning demo

- Let v be partitioning item $a[10]$.
- Scan i from left to right.
 - ($a[i] < v$): exchange $a[lt]$ with $a[i]$; increment both lt and i
 - ($a[i] > v$): exchange $a[gt]$ with $a[i]$; decrement gt
 - ($a[i] == v$): increment i



Invariant



comparators