Quicksort

Goldsmiths Computing

January 13, 2019

Quicksort

To sort a sequence: choose a pivot element, and generate subsequences of elements smaller and larger than that pivot element; sort those subsequences, and combine with the pivot.

Properties:

- in-place sort
- no extra heap storage required (and low stack space requirement)
- (only works on arrays)

Quicksort

```
function PARTITION(s,low,high)
    pivot ← s[high-1]
    loc ← low
    for 0 ≤ j < high-1 do
        if s[j] ≤ pivot then
            SWAP(s[i],s[j])
            i ← i + 1
        end if
    end for
    SWAP(s[hi],s[i])
    return i
end function
```

Quicksort

```
function QUICKSORT(s,low,high)
    if low < high then
        p ← PARTITION(s,low,high)
        QUICKSORT(s,low,p)
        QUICKSORT(s,p+1,high)
    end if
end function
```

Complexity analysis

Time complexity: partition

- $N - 1$ iterations, each with (worst-case) one SWAP
- final SWAP at the loop epilogue

$$\implies \Theta(N)$$

Time complexity: quicksort

$$T(N) = T(N - p) + T(p - 1) + \Theta(N)$$

- depends on value of p!
- (we'll come back to this)

Complexity bounds

How efficient can comparison sorts be?

- how many possible permutations are there of a sequence of N distinct elements?
- how many of those possible permutations are sorted?
- how much information does a single comparison give?

Work

1. Reading
   - CLRS, section 2.3; CLRS, chapter 7
   - Jon Bentley, *Programming Pearls*, Column 11: sorting
2. Questions from CLRS

   Exercises 2.1-1, 2.1-2, 2.2-2, 2.3-1