Comparison sorts

Goldsmiths Computing

January 13, 2019

Motivation

- sorting is a fundamental operation
- intermediate step in many other algorithms

Definition

Any kind of search algorithm using a total order relation to compare pairs of elements to decide which should precede the other.

input a sequence of objects $s_0 \dots s_{N-1}$

output a reordering of the sequence such that $s'_0 \leq s'_1 \leq s'_2 \leq \dots \leq s'_{N-1}$

Total order relations

transitivity if $a \leq b$ and $b \leq c$ then $a \leq c$

totality $a \leq b$ or $b \leq a$

Bogosort

Require: s :: sequence
  while ¬SORTED?(s) **do**
    PERMUTE(s)
  end while
  return s

Complexity analysis

Time complexity

- there are $N!$ permutations of a sequence of N elements
- in the worst case the sorted permutation will be the last one

$$\Longrightarrow \Omega(N!)$$

Insertion sort

To sort a sequence: repeatedly insert the next unsorted element into its correct place in the sorted sequence.
Properties:

- stable
- straightforward
- in-place for arrays
    - also adaptible for in-place sorting of linked lists

Insertion sort

function INSERTIONSORT(s)
    for $1 \le j <$ LENGTH(s) **do**
        key ← s[j]
        i ← j−1
        while $i \ge 0 \wedge s[i] > key$ **do**
            s[i+1] ← s[i]
            i ← i - 1
        end while
        s[i+1] ← key
    end for
end function

Complexity analysis

Time complexity

- $N - 1$ iterations;
- for iteration number j, worst-case j array writes

$$\implies \Theta(N^2)$$

Space complexity

Only constant space required for running function:

$$\implies \Theta(1)$$

Work

1. Reading
   - CLRS, sections 2.1, 2.2
2. Investigate other quadratic sorting algorithms, for example:
   - selection sort
   - bubble sort
   - odd-even sort.

   What advantages and disadvantages do they have relative to insertion sort?
3. Questions from CLRS

   2-2 Correctness of bubblesort

Merge (vector)

```
Require: a,b :: Vector
  function MERGE(a,b)
    al ← LENGTH(a); bl ← LENGTH(b); cl ← al + bl
    c ← new Vector(cl)
    ai ← bi ← ci ← 0
    while ci < cl do
      if ai = al then
        c[ci] ← b[bi]; bi ← bi + 1
      else if bi = bl ∨ a[ai] ≤ b[bi] then
        c[ci] ← a[ai]; ai ← ai + 1
      else
        c[ci] ← b[bi]; bi ← bi + 1
      end if
      ci ← ci + 1
    end while
    return c
  end function
```

Mergesort

function MERGESORT(s)
  sl $\leftarrow$ LENGTH(s)
  if sl $\leq$ 1 **then**
    return s
  else
    mid $\leftarrow \left\lfloor \frac{sl}{2} \right\rfloor$
    left $\leftarrow$ MERGESORT(s[0...mid))
    right $\leftarrow$ MERGESORT(s[mid...sl))
    return MERGE(left,right)
  end if
end function

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

$$\Longrightarrow \Theta(N)$$

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