Quicksort: A Review

Arthur Nunes-Harwitt

Quicksort is another ‘divide and conquer’ sorting algorithm. The algorithm is pretty simple: given a sequence, pick an element of the sequence, called the pivot, and divide the sequence into three sub-sequences — one whose elements are all less than the pivot, one whose elements are all equal to the pivot, and one whose elements are greater than the pivot — then concatenate the sorted sub-sequences. Although the worst case running time is $O(n^2)$, the expected running time is $O(n \log(n))$.

Here is an implementation.

```scheme
(define (list-qsort L)
  (cond ((null? L) '())
        ((null? (cdr L)) L)
        (else
         (let* ((pivot (car L))
                (below (filter L (lambda (n) (< n pivot))))
                (same (filter L (lambda (n) (= n pivot))))
                (above (filter L (lambda (n) (> n pivot))))
                (s-below (list-qsort below))
                (s-above (list-qsort above)))
          (append s-below same s-above))))

(define (filter L p?)
  (cond ((null? L) '())
        ((p? (car L)) (cons (car L) (filter (cdr L) p?)))
        (else (filter (cdr L) p?))))
```

What many find attractive about quicksort is that it can be implemented as an ‘in place’ algorithm. The ‘in place’ version sorts arrays without doing any memory allocation. ‘In place’ partitioning can be done as follows. Assume that the left hand side of the array should contain the smaller elements, and that the right hand side of the array should contain the larger elements. Let there be two pointers; one starts on the left hand side, and one starts on the right hand side. Move the one on the left until it comes to an element that is larger than the pivot; similarly, move the one on the right until it comes to an element that is smaller than the pivot. If such elements are found, swap them, since both are on the wrong side. Continue until the pointers meet. Use one of the pointers as the point to split. After the sorting is done, no concatenation is necessary!

Here is an implementation.
(define (vector-qsort! V)
  (vector-qsort2! V 0 (- (vector-length V) 1))
V)

(define (vector-qsort2! V start finish)
  (if (>= start finish)
      'done
      (let ((middle (partition! V start finish)))
        (vector-qsort2! V start middle)
        (vector-qsort2! V (+ 1 middle) finish))))

(define (partition! V start finish)
  (let ((pivot (vector-ref V start)))
    (let loop ((left start) (right finish))
      (let ((left-val (vector-ref V left))
            (right-val (vector-ref V right)))
        (cond ((>= left right) right)
              ((< left-val pivot) (loop (+ left 1) right))
              ((< pivot right-val) (loop left (- right 1)))
              (else (vector-set! V left right-val)
                    (vector-set! V right left-val)
                    (loop left right)))))
)

That seems straightforward enough, but where did that average case time complexity $O(n \log(n))$ come from? Let’s take a look at the derivation. I am going to base my derivation on Manber’s.

How much time does a call to quicksort take? Let $T(n)$ be the time it takes to sort an array of $n$ elements. The partitioning takes $\Theta(n)$ steps, since each element of the array is visited once by one (and only one) of the pointers. Now, assume that any split point is equally likely. Thus the probability of any split point is $1/n$. If there are $p$ elements before the split point, then there are $n-p$ elements after the split, and so the time it takes to sort both sub-arrays is $T(p) + T(n-p)$. Since any split point is equally likely, we get the following recurrence equation.

$$T(n) = \Theta(n) + \frac{1}{n} \sum_{p=1}^{n-1} T(p) + T(n-p)$$

To simplify, let’s just use $n$ instead of $\Theta(n)$. Also, let’s assume that $T(1) = 1$. 

$$T(n) = n + \frac{1}{n} \sum_{p=1}^{n-1} T(p) + T(n-p)$$

2
\[= n + \frac{1}{n} \left( \sum_{p=1}^{n-1} T(p) + \sum_{p=1}^{n-1} T(n-p) \right)\]

\[= n + \frac{2}{n} \sum_{p=1}^{n-1} T(p)\]

If we multiply through by \(n\) and do some manipulation, we can get rid of the summation terms.

\[nT(n) = n^2 + 2 \sum_{p=1}^{n-1} T(p)\]

\[nT(n) - (n-1)T(n-1) = n^2 + 2 \sum_{p=1}^{n-1} T(p) - \left( (n-1)^2 + 2 \sum_{p=1}^{n-2} T(p) \right)\]

\[nT(n) - (n-1)T(n-1) = 2n - 1 + 2T(n-1)\]

\[nT(n) = 2n - 1 + (n+1)T(n-1)\]

\[T(n) = \frac{2n - 1}{n} + \frac{n+1}{n} T(n-1)\]

Note that \(\frac{2n-1}{n}\) is less than 2 if \(n > 0\), so consider \(\tilde{T}(n) = 2 + \frac{n+1}{n} \tilde{T}(n-1)\), \(\tilde{T}(1) = 2\). Since \(2 > \frac{2n-1}{n}\), \(\tilde{T}(n) > T(n)\), which will be sufficient for a bound.

\(\tilde{T}(n)\) will be now solved using the iteration method.

\[\tilde{T}(n) = 2 + \frac{n+1}{n} \tilde{T}(n-1)\]

\[= 2 + \frac{n+1}{n} \left( 2 + \frac{n}{n-1} \tilde{T}(n-2) \right)\]

\[= 2 + \frac{2(n+1)}{n} + \frac{n+1}{n-1} \tilde{T}(n-2)\]

\[= 2 + \frac{2(n+1)}{n} + \frac{n+1}{n-1} \left( 2 + \frac{n-1}{n-2} \tilde{T}(n-3) \right)\]

\[= 2 + \frac{2(n+1)}{n} + \frac{2(n+1)}{n-1} + \frac{n+1}{n-2} \tilde{T}(n-3)\]

\[= 2 \left( 1 + (n+1) \sum_{i=2}^{n} \frac{1}{i} \right)\]

However, \(\sum_{i=2}^{n} \frac{1}{i} = \Theta(\log(n))\), since \(\sum_{i=1}^{n} \frac{1}{i} = \Theta(\log(n))\).

\[\tilde{T}(n) = 2(1 + (n+1)\Theta(\log(n)))\]
\[ T(n) = 2n \Theta(\log(n)) + 2 \Theta(\log(n)) + 2 = \Theta(n \log(n)) \]

Thus it follows that \( T(n) = O(n \log(n)) \).