CSE 241 Class 8

Jeremy Buhler

September 21, 2015

Today: QuickSort!

1 Partitioning an Array

Partitioning is the basis of algorithms for

- **sorting** (famous QuickSort algorithm)
- **order statistics** (for example, "find the third largest element in this array")

Partition Algorithm

- **input**: array $A[p \dots r]$
- **job**: rearrange A into left and right parts [**not necessarily halves**] such that every element in left part is $<$ all elements in right part
- **returns**: index of a *partition element*, placed at end of left part and $\geq$ every element on left
- In this implementation, **partition around (initially) last element** $A[r]$.

```
PARTITION(A, p, r)
  i ← p − 1
  j ← p − 1
  do
      j++
      if A[j] ≤ A[r]
          i++
          swap(A[i], A[j])
  while j < r − 1

  swap(A[i + 1], A[r])
  return i + 1
```

2 Partition Example

3 Correctness of Partition

Partition is a rather funky algorithm. It can move an element multiple times! Is it correct?

> **Claim**: After running $z = \text{Partition}(A, p, r)$, every element in $A[p \dots z]$ is $<$ any element in $A[z + 1 \dots r]$.

Proof: Will first prove following *loop invariant*: after any number of loop iterations, every element in $A[p \dots i]$ is $\leq A[r]$, and every element in $A[i + 1 \dots j]$ is $> A[r]$. (By induction on # of iterations.)

- **Base**: after 0 iterations, $i < p$ and $j < p$, so no elements in $A[p \dots i]$ or $A[i+1 \dots j]$. Invariant holds *vacuously*.
- **Ind**: Suppose invariant holds after k iterations. At this point, $j = p - 1 + k$, so every elt in $A[p \dots i]$ is $\leq$ any elt in $A[i + 1 \dots p + k - 1]$. At start of next iteration, $j \leftarrow p + k$.

  1. If $A[p + k] > A[r]$, we do *not* swap and i does not move. Only upper range $A[i + 1 \dots j]$ extended, and new elt is $> A[r]$, so invariant still holds.

  2. If $A[p + k] \leq A[r] \dots$ Let $x = A[p + k]$ and $y = A[i + 1]$ at beginning of loop. We have that $x \leq A[r]$ and (by i.h.) that $y > A[r]$. At end of loop (after i increment and swap),

$$\begin{aligned} A[i] = x &\leq A[r] \\ A[j] = y &> A[r] \end{aligned}$$

so invariant still holds for both extended ranges.

Hence, invariant holds after $k+1$ iterations, and i.h. is proven.

- Conclude that after all loop iterations, $A[p \dots i] \leq A[r]$, and $A[i+1 \dots r-1] > A[r]$.
- In last step, we swap $A[r]$ with $A[i+1]$. By construction, $A[i+1] > A[r]$ before swap. After swap, $A[1 \dots i+1]$ is $\leq$ partition elt, and $A[i+2 \dots r]$ is $>$ partition elt.
- Hence, after final swap, $A[p \dots i+1] \leq A[r]$ and $A[i+2 \dots r] > A[r]$. We return $z = i+1$, so the claim holds. QED

4 QuickSort

Partitioning can be used to sort an array. Sort occurs **in-place** – don't need extra storage as for, e.g., merge sort. Idea is divide-and-conquer:

1. partition array into high and low parts.
2. All elts in high part are now greater than any elt in low part (but parts are not yet sorted).
3. Recursively sort the two parts. (Must leave out partition elt so that both parts are always smaller than input.)

```
QUICKSORT(A, p, r)
  if p < r
     z ← PARTITION(A, p, r)
     QUICKSORT(A, p, z − 1)
     QUICKSORT(A, z + 1, r)
```

To prove correctness, check inductively that the following property holds after sorting for all array lengths:

For all pairs of indices i, j, $p \leq i, j \leq r$, if $A[i] \neq A[j]$,

$$A[i] > A[j] \text{ iff } i > j.$$

5 Efficiency of Quicksort

Let's write down a recurrence for QUICKSORT.

- Partitioning takes time $\Theta(n)$.
- But, how large are the two parts it produces? Unknown.
- In worst case, could partition A into parts of size n and 0.

- Recursive calls would be of sizes $n-1$, 0 (because partition element at end of lower part isn't passed to either call).

Assume worst case occurs on every partition. Then running time is given (for asymptotic purposes) by recurrence

$$T(n) = cn + T(n-1).$$

Sketching recursion tree, we get

$$\begin{aligned} T(n) &= \sum_{k=0}^{n-2} c(n-k) + c_0 \\ &= cn(n-1) - \frac{c(n-1)(n-2)}{2} + c_0 \\ &= \Theta(n^2). \end{aligned}$$

Can worst case actually occur? Yes!

- Suppose input is already sorted.
- For any subarray $A[x \ldots y]$, $A[y]$ is largest element.
- PARTITION splits around $A[y]$.
- Result: partitioned array is same as input, $z = r$
- Parts are of sizes $n-1$ and 0

In general, if we consistently pick largest or smallest element of A for partition, running time will be $\Theta(n^2)$.