Technische Universität München
Institut für Informatik
Scientific Computing in Computer Science

WS 2010/11
Worksheet
14.12.2010

# Fundamental Algorithms 6

## Exercise 1

Show that the second largest element of a set of $n$ elements can be determined with less than $n + \lceil \log n \rceil$ comparisons.

*Hint: start with a divide-and-conquer algorithm to find the largest element!*

Consider the following algorithm that returns the two largest elements of an array (it only works if $n$ is a power of 2):

```
Select2 (A:Array[1..n]) : Integer,Integer{

   if n<=2 then {
      if A[1] > A[2]
      then return A[1], A[2]
      else return A[2], A[1]
   }
   else
   {
      Left1, Left2 := Select2(A[1..n/2]);
      Right1, Right2 := Select2(A[n/2+1..n]);
      if Left1 > Right1
      then {
        if Left2 > Right1
        then return Left1, Left2
        else return Left1, Right1
      }
      else {
        if Left1 > Right2
        then return Right1, Left1
        else return Right1, Right2
      }
   }
}
```

The algorithm starts from a divide-and-conquer scheme to find the largest element: search the largest element separately in two halves of the array and then compare the two maxima. If we return the second largest element of each of the halves, we can still determine the maximum as the larger of the two local maxima, but we can also determine the second-largest element: the "loser" of the two local maxima has to be compared to the runner-up of the "winner".

**Problem:** the algorithm requires approx. $n + \frac{n}{2}$ comparisons, because we also determine the second largest element from the "loser" half (which we don't need to). However, we can only determine the winner and loser half after we have compared their maxima.

**BestAndLosers:**

A better idea is to look at the potential candidates for second place: as the second best will prevail against all other elements, it will finally be disregarded when compared with the winner. Hence, we need to look for second place in all element who "lost" against the largest one. The following algorithm realises this idea via lists: the function BestAndLosers will return a list that contains the best element as first list element plus all elements that were compared to this element. A following SelectTwo algorithm only need to find the maximum in the remainder of the list.

```
BestAndLosers (A:Array[r..p]) : List of Integer{

   if r=p then return A[r]
   else if r=p−1 then
      if A[r] > A[p]
      then return A[r], A[p]
      else return A[p], A[r]
   else  if
   {
      m := floor( (r+p)/2 );
      Left := BestAndLosers(A[r..m]);
      Right := BestAndLosers(A[m+1..p]);
      if first(Left) > first(Right)
      then return Left, first(Right)
      else return Right, first(Left)
   }
}

SelectTwo(A[1..n] : Integer, Integer {
   B  := BestAndLLosers(A[1..n]);
  return first(B), Maximum( rest(B) );
}
```

## Exercise 2

Analyze the time complexity of the BFPRT algorithm, if the elements of the array are subdivided into $\lceil \frac{n}{k} \rceil$ groups of $k$ elements (some groups will only have $k-1$ elements if $n \bmod k \neq 0$).

Examine the time complexity for $k \in \{3, 7, 9, 11, \ldots\}$, and find the optimal value for $k$.

**Solution:**

We simplify the solution in the sense that we do not consider the $\lceil\ \rceil$ or $\lfloor\ \rfloor$ problems, or any constant additional effort.

Then, for a division of groups of $k = 2l + 1$ elements, we obtain the following computational effort:

- $s_k(\frac{n}{k})$ to determine all $\frac{n}{k}$ medians ($s_k$ the time to determine the median of $k$ elements);
- $T\left(\frac{n}{k}\right)$ to determine the median of the medians;
- $T\left(n - \frac{l}{2k}n\right)$ for the recursive call to the (in the worst case: larger) partition.

The (simplified) recurrence equation for $T(n)$ is therefore:

$$T(n) = s_k\frac{n}{k} + T\left(\frac{n}{k}\right) + T\left(n - \frac{l}{2k}n\right)$$

Guessing the solution $T(n) = cn$ leads to the equation:

$$\begin{aligned}
cn &= s_k\frac{n}{k} + c\cdot\frac{n}{k} + c\cdot(1 - \frac{l}{2k})n \\
\Leftrightarrow \quad c(n - \frac{n}{k} - n + \frac{l}{2k}n) &= s_k\frac{n}{k} \\
\Leftrightarrow \quad cn\frac{l-2}{2k} &= s_k\frac{n}{k} \\
\Leftrightarrow \quad c &= s_k\frac{1}{k\frac{l-2}{2k}} = \frac{s_k}{\frac{l-2}{2}} = \frac{2s_k}{l-2}
\end{aligned}$$

At first glance, large values for $l$ seem to be advantageous. However, $s_k$ will also grow with $k$ and $l$. If we really do a sorting-based median selection for the $k$-element groups, we'd have $s_k \in \Theta(l \log l)$. Hence, large values for $l$ are bad.

Thus, we would have to examine closely how many comparisons are required to determine the median of $k = 2l + 1$ elements in the worst case (3 are sufficient for 3 elements, for example). In general, we can expect that $s_k \in \Omega(l)$, at least, so to really determine the best value for $k$, we would need to have a really close look at exact constants.