Worst Case Analysis of QuickSort

Recall that QuickSort works by calling a method called `split` which divides the array in two. Method `split` makes \( n - 1 \) comparisons when processing an array of size \( n \). QuickSort is then called recursively on the subarrays \( A[0..\text{pivotLoc} - 1] \) and \( A[\text{pivotLoc} + 1..n - 1] \). The challenge in analyzing QuickSort is that we do not know the value of `pivotLoc` and so we do not know exactly how the array is divided.

Let \( f(n) \) denote the most comparisons that QuickSort makes on any instance of size \( n \). The function \( f(n) \) satisfies the following recurrence relation.

\[
f(n) = \begin{cases} 
0, & \text{if } n \leq 1 \\
(n - 1) + \max_{0 \leq i \leq \left\lfloor \frac{n-1}{2} \right\rfloor} \{f(i) + f(n - i - 1)\}, & \text{if } n \geq 2 \end{cases}
\]

The justification for this recurrence relation is as follows. As stated, method `split` does \( n - 1 \) comparisons when processing an array of size \( n \). But then QuickSort is called recursively on the subarrays \( A[0..\text{pivotLoc} - 1] \) and \( A[\text{pivotLoc} + 1..n - 1] \). One of these subarrays could have size anywhere between \( 0 \) and \( \left\lfloor \frac{n-1}{2} \right\rfloor \).

We will solve the recurrence relation by guessing at the solution and then proving our guess by induction. Computing the first few values of \( f(n) \) we obtain the following table.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( f(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3 = 1 + 2</td>
</tr>
<tr>
<td>4</td>
<td>6 = 1 + 2 + 3</td>
</tr>
<tr>
<td>5</td>
<td>10 = 1 + 2 + 3 + 4</td>
</tr>
</tbody>
</table>

The first few values of \( f(n) \)

It looks as though

\[
f(n) = \begin{cases} 
0, & \text{if } n \leq 1 \\
\frac{(n-1)n}{2}, & \text{if } n \geq 2 \end{cases}
\]

We prove our observation using the technique of weak induction.

**BASE CASE:** Clearly, the two formulas for \( f(n) \) agree for \( n \leq 1 \).

**INDUCTION HYPOTHESIS:** Assume that the two formulas for \( f(n) \) agree for all \( k = 0, 1, \ldots, n - 1 \) where \( n \geq 2 \).

**INDUCTION STEP:** Prove that the two formulas for \( f(n) \) agree for \( k = n \).
To do the proof, start with the original formula for \( f(n) \) and try to derive the second formula.

\[
\begin{align*}
 f(n) &= (n-1) + \max_{0 \leq i \leq \lfloor \frac{n-1}{2} \rfloor} \{ f(i) + f(n-i-1) \} \\
 &= (n-1) + \max_{0 \leq i \leq \lfloor \frac{n-1}{2} \rfloor} \left\{ \frac{i(i-1)}{2} + \frac{(n-i-2)(n-i-1)}{2} \right\}.
\end{align*}
\]

The second equality follow from the induction hypothesis. After some algebra we obtain that

\[
\begin{align*}
 f(n) &= (n-1) + \max_{0 \leq i \leq \lfloor \frac{n-1}{2} \rfloor} \left\{ \frac{(n-1)(n-2)}{2} + i^2 + (1-n)i \right\}.
\end{align*}
\]

In order to maximize the quantity in the brackets, all we need to do is maximize the quantity \( i^2 + (1-n)i \) over the interval \( 0 \leq i \leq \lfloor \frac{n-1}{2} \rfloor \). Let \( g(i) = i^2 + (1-n)i \). Taking derivatives, we obtain that \( g'(i) = 2i + 1 - n \). Clearly, \( g'(i) \leq 0 \) if and only if \( 2i \leq n - 1 \) and so \( i \leq \frac{n-1}{2} \). Since \( i \) is an integer, \( i \leq \lfloor \frac{n-1}{2} \rfloor \). Since the derivative is nonpositive, \( g(i) \) is nonincreasing on the interval \( 0 \leq i \leq \lfloor \frac{n-1}{2} \rfloor \). Therefore the maximum occurs for the value \( i = 0 \). Thus,

\[
\max_{0 \leq i \leq \lfloor \frac{n-1}{2} \rfloor} \left\{ \frac{(n-1)(n-2)}{2} + i^2 + (1-n)i \right\} = \frac{(n-1)(n-2)}{2}.
\]

So, we have

\[
\begin{align*}
 f(n) &= (n-1) + \max_{0 \leq i \leq \lfloor \frac{n-1}{2} \rfloor} \left\{ \frac{(n-1)(n-2)}{2} + i^2 + (1-n)i \right\} \\
 &= (n-1) + \frac{(n-1)(n-2)}{2} \\
 &= \frac{(n-1)n}{2}.
\end{align*}
\]

The proof is complete. \( \blacksquare \)

It should be noted that QuickSort performs exactly \( f(n) \) comparisons on an array of size \( n \) that is sorted in ascending order.

What about array assignment statements? The function \( f(n) \) only gives us the number of comparisons that QuickSort does in the worst case. Counting array assignment statements is harder than counting comparisons since method \texttt{split} does a variable number of these operations depending on the data. Therefore, we will be satisfied with obtaining an upper bound on the number of array assignment statements that QuickSort performs in the worst case. We know that QuickSort performs exactly \( \frac{(n-1)n}{2} \) comparisons in the worst case. Suppose that every comparison is accompanied by a swap. In this case, QuickSort would do \( \frac{3(n-1)n}{2} \) array assignment statements. (Recall that swapping requires three array assignment statements.) Remember that in method \texttt{split}, there
is a final swap between $A[0]$ and $A[pivotLoc]$. Therefore, each call of method
\texttt{split} results in yet another swap. Method \texttt{split} is called at most $n - 1$ times
on an array of size $n$. Therefore, at most $3(n - 1)$ extra array assignment
statements are performed. Adding these quantities together, we now obtain an
upper bound on the number of array assignment statements that are performed,
namely
$$\frac{3(n - 1)n}{2} + 3(n - 1) = \frac{3(n - 1)(n + 2)}{2}.$$ 
Therefore, we have obtained an upper bound on $c_{QS}(n)$, the worst case cost of
QuickSort:
$$c_{QS}(n) \leq \frac{(n - 1)n}{2} + \frac{3(n - 1)(n + 2)}{2} = (n - 1)(2n - 3).$$