The purpose of this document is to introduce the idea of using recurrence relations to do average-case analysis. The average-case running time of QuickSort is obtained as an application of this idea.

1. Average-Case Analysis Using Recurrence Relations

In class we showed that when the search key $X$ is in the list $L$ of size $n$, then $\overline{c}_A(n) = \frac{n+1}{2}$ where $A$ is Linear Search. Often the analysis can be done using recurrence relations rather than evaluating a sum directly. For convenience, let $f(n) = \overline{c}_A(n)$. We can write a recurrence relation for $f(n)$ under the assumption that $X$ is in the list, namely,

$$f(n) = \begin{cases} 1, & \text{if } n = 1 \\ 1 + \frac{n-1}{n} f(n-1), & \text{if } n \geq 2 \end{cases}.$$ 

How did we arrive at this recurrence? Think about applying Linear Search to an unsorted list. We must compare the first element of the list to the search key $X$. In other words, the probability of inspecting the first element of the list is 1. How many comparisons do we make when we inspect the first element of the list? We make exactly one comparison. Now having compared the first element of the list with $X$, how many comparisons do we make on average to search the rest of the list? Well, it depends on whether the search key $X$ was found in position one of the list. The probability that we even have to search the rest of the list is $\frac{n-1}{n}$ since the probability of the search key being in the first position is $1/n$. The expression $f(n-1)$ is the average-case running time of $A$ on a list of size $n-1$. Therefore, the quantity $\frac{n-1}{n} f(n-1)$ represents the average-case running
time of $A$ on the remaining $n-1$ positions in $L$ discounted by the probability that we will even have to search the last $n-1$ positions. Putting it all together, we obtain that the average-case running time of Linear Search on a list of size $n \geq 2$ is $1 + \frac{n-1}{n} f(n-1)$. Let us now solve this recurrence by unrolling it!

$$f(n) = 1 + \frac{n-1}{n} f(n-1)$$

$$= 1 + \frac{n-1}{n} \left( 1 + \frac{n-2}{n-1} f(n-2) \right)$$

$$\cdots$$

$$= 1 + \frac{n-1}{n} + \frac{n-2}{n} + \cdots + \frac{2}{n} + \frac{1}{n}$$

$$= 1 + \frac{1}{n} (1 + 2 + \cdots + (n-1))$$

$$= 1 + \frac{1}{n} \frac{(n-1)n}{2}$$

$$= 1 + \frac{n-1}{2}$$

$$= \frac{n+1}{2}$$

Notice that we obtain the same result that we did when we evaluated the sum directly from the definition.

2. Average-Case Analysis of QuickSort

We are now ready to tackle the main task at hand, an analysis of the average-case running time of QuickSort. We will adopt the approach of the last Section, namely, to develop a recurrence relation for the average-case running time and then solve it. Let $A$ be QuickSort, and for convenience, let $f(n) = \mathbb{E}_A(n)$. We can write a recurrence relation for the average-case running time of QuickSort as

$$f(n) = \begin{cases} 0, & \text{if } n \leq 1 \\ (n-1) + \sum_{i=1}^{n-1} \frac{1}{n} (f(i-1) + f(n-i)), & \text{if } n \geq 2 \end{cases}$$

To understand this recurrence, we must consider it term by term. The term $n-1$ is needed since method Split does exactly $n-1$ comparisons. Recall that method Split returns the position (specifically, $pivotLoc$) where the list will be
partitioned. Certainly, on random data, this position could be the first one, the second one, etc. Since there are \( n \) possible positions and each one is equally likely, the probability that \( pivotLoc \) will be any one of the \( n \) possible positions is \( \frac{1}{n} \). What about the term \( f(i - 1) + f(n - i) \)? (Notice that we are multiplying this term by \( \frac{1}{n} \).) To understand what this sum means, let’s consider a specific example. Suppose \( n = 10 \), and suppose Split returns the value \( pivotLoc = 5 \). (Let’s further suppose that the array is indexed starting at 1, not 0. Therefore, the indices range from 1 to 10, not 0 to 9.) QuickSort is now called recursively, first on the left subarray of length four and then on the right subarray of length five. This situation is accounted for by taking \( i = 5 \) in the sum of the recurrence formula. That is, we will be adding in the term
\[
\frac{1}{10}(f(5 - 1) + f(10 - 5)) = \frac{1}{10}(f(4) + f(5)).
\]
to the total. The quantities \( f(4) \) and \( f(5) \) represent the average-case running time of QuickSort on arrays of size four and five respectively. However, we discount this term by multiplying it by the probability that the array will be partitioned in this particular manner! As \( i \) runs through its values from 1 to \( n \), the list is being partitioned in all the possible ways. The recurrence relation is now seen to correctly represent the average-case running time of QuickSort on lists of size \( n \).

Now for the fun part! We must solve the recurrence. The first step is to simplify as much as possible the sum in the recurrence.

\[
\sum_{i=1}^{n} \frac{1}{n}(f(i - 1) + f(n - i)) = \frac{1}{n}(f(0) + f(n - 1)) + \frac{1}{n}(f(1) + f(n - 2)) + \cdots + \frac{1}{n}(f(n - 2) + f(1)) + \frac{1}{n}(f(n - 1) + f(0)) = \frac{1}{n}(f(2) + f(2)) + \frac{1}{n}(f(3) + f(3)) + \cdots + \frac{1}{n}(f(n - 1) + f(n - 1)) = \frac{2}{n} \sum_{i=2}^{n-1} f(i)
\]

Therefore, we have simplified the sum considerably as
\[
\sum_{i=1}^{n} \frac{1}{n}(f(i - 1) + f(n - i)) = \frac{2}{n} \sum_{i=2}^{n-1} f(i)
\]
So now the recurrence has the form

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

Let’s first multiply through by \( n \) to clear fractions.

\[ nf(n) = n(n - 1) + 2 \sum_{i=2}^{n-1} f(i) \]

To have any hope of solving the recurrence, we must get rid of the sum. A trick that is often used is to subtract successive values of the recurrence. In other words, let us compute the quantity

\[ nf(n) - (n - 1)f(n - 1) = n(n - 1) + 2 \sum_{i=2}^{n-1} f(i) \]

\[ -(n - 1)(n - 2) - 2 \sum_{i=2}^{n-2} f(i) \]

\[ = 2(n - 1) + 2f(n - 1) \]

Notice that all but one of the terms of the sum cancelled out! In this way, we have gotten rid of a sum completely! So now we have

\[ nf(n) - (n - 1)f(n - 1) = 2(n - 1) + 2f(n - 1) \]

Therefore,

\[ nf(n) = (n - 1)f(n - 1) + 2f(n - 1) + 2(n - 1) \]

\[ = (n + 1)f(n - 1) + 2(n - 1) \]

So we now have

\[ nf(n) = (n + 1)f(n - 1) + 2(n - 1) \]

Dividing both sides through by \( n(n + 1) \), we get

\[ \frac{f(n)}{n + 1} = \frac{f(n - 1)}{n} + \frac{2(n - 1)}{n(n + 1)} \]
What we have actually now done is formulate a new recurrence relation! Let $g(n) = \frac{f(n)}{n+1}$. Then, clearly,

$$g(n) = \begin{cases} 
0, & \text{if } n \leq 1 \\
g(n-1) + \frac{2(n-1)}{n(n+1)}, & \text{if } n \geq 2
\end{cases}$$

Let’s now solve this recurrence relation by unrolling it.

$$g(n) = g(n-1) + \frac{2(n-1)}{n(n+1)}$$

$$= g(n-2) + \frac{2(n-2)}{(n-1)n} + \frac{2(n-1)}{n(n+1)}$$

$$\cdots$$

$$= g(1) + \frac{2 \cdot 1}{2 \cdot 3} + \cdots + \frac{2(n-1)}{n(n+1)}$$

(since $g(1) = 0$)

$$= \sum_{i=2}^{n} \frac{2(i-1)}{i(i+1)}$$

$$= 2 \sum_{i=2}^{n} \frac{i-1}{i(i+1)}$$

If we can evaluate this last sum, we would be almost done. Let us now try to do this. Computing, we obtain

$$\sum_{i=2}^{n} \frac{i-1}{i(i+1)} = \sum_{i=2}^{n} \left( -\frac{1}{i} + \frac{2}{i+1} \right) \quad \text{(we use the method of partial fractions here)}$$

$$= \sum_{i=2}^{n} -\frac{1}{i} + \sum_{i=2}^{n} \frac{2}{i+1}$$

$$= \left( -\frac{1}{2} - \frac{1}{3} - \cdots - \frac{1}{n} \right) + \left( \frac{2}{3} + \frac{2}{4} + \cdots + \frac{2}{n} + \frac{2}{n+1} \right)$$

$$= -\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} + \frac{2}{n+1}$$

$$= \frac{2}{n+1} - \frac{1}{2} + \left( \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} \right)$$

5
Let \( H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} \). The number \( H_n \) is called the \( n \)th harmonic number. (We have encountered this number before in class.) Continuing,

\[
\frac{2}{n+1} - \frac{1}{2} + \left( \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} \right) = \frac{2}{n+1} - \frac{1}{2} + H_n - 1 - \frac{1}{2} = H_n - \frac{2n}{n+1}
\]

Therefore, we have shown that

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

Getting back to \( g(n) \),

\[
g(n) = 2 \sum_{i=2}^{n} \frac{i-1}{i(i+1)} = 2 \left( H_n - \frac{2n}{n+1} \right) = 2H_n - \frac{4n}{n+1}
\]

We are now almost done! Recall that

\[
f(n) = (n+1)g(n) = (n+1) \left( 2H_n - \frac{4n}{n+1} \right) = 2(n+1)H_n - 4n = 2nH_n + 2H_n - 4n
\]

Therefore, we have shown that

\[
f(n) = 2nH_n + 2H_n - 4n
\]

Now it was proved on Test #1 (see your test) that \( H_n \in \Theta(\ln n) \). Since the logarithms all have the same growth rate, \( H_n \in \Theta(\log_2 n) \). Therefore, the expression \( 2nH_n + 2H_n - 4n \in \Theta(n \log_2 n) \). Therefore, we have proved that

\[
\tau_A(n) = f(n) \in \Theta(n \log_2 n)
\]

So, the average-case running time of QuickSort is \( \Theta(n \log_2 n) \). This running time is dramatically different from its worst-case running time which, as we know, is \( \Theta(n^2) \).
The instructor has done some computing and has produced the following table with the help of Maple©.

<table>
<thead>
<tr>
<th>(n)</th>
<th>(n \log_2 n)</th>
<th>(\tau_A(n) = 2(n + 1)H_n - 4n)</th>
<th>(\frac{\tau_A(n)}{n \log_2 n})</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>33.21928095</td>
<td>24.43730159</td>
<td>0.7356360791</td>
</tr>
<tr>
<td>50</td>
<td>282.1928095</td>
<td>258.9189445</td>
<td>0.9175249538</td>
</tr>
<tr>
<td>100</td>
<td>664.385619</td>
<td>647.8502586</td>
<td>0.9751118027</td>
</tr>
<tr>
<td>200</td>
<td>1528.771238</td>
<td>1562.968441</td>
<td>1.022369078</td>
</tr>
<tr>
<td>300</td>
<td>2468.645607</td>
<td>2582.163656</td>
<td>1.045983939</td>
</tr>
<tr>
<td>400</td>
<td>3457.542476</td>
<td>3669.083612</td>
<td>1.061182513</td>
</tr>
<tr>
<td>1000</td>
<td>9965.784285</td>
<td>10985.91266</td>
<td>1.10236308</td>
</tr>
<tr>
<td>5000</td>
<td>61438.5619</td>
<td>70963.27755</td>
<td>1.155028297</td>
</tr>
<tr>
<td>10000</td>
<td>132877.1238</td>
<td>155771.6959</td>
<td>1.172298823</td>
</tr>
<tr>
<td>50000</td>
<td>780482.0237</td>
<td>939723.1889</td>
<td>1.204029254</td>
</tr>
</tbody>
</table>

It is interesting to consider

\[
\lim_{n \to \infty} \frac{\tau_A(n)}{n \log_2 n} = \lim_{n \to \infty} \frac{2nH_n + 2H_n - 4n}{n \log_2 n} \\
= \lim_{n \to \infty} \frac{2H_n}{\log_2 n} + \lim_{n \to \infty} \frac{2H_n}{n \log_2 n} - \lim_{n \to \infty} \frac{4}{\log_2 n}
\]

On Test #1, it was shown that \(\lim_{n \to \infty} \frac{H_n}{\ln n} = 1\). Therefore, since \(\log_2 n = \log_2 e \ln n\),

\[
\lim_{n \to \infty} \frac{H_n}{\log_2 n} = \frac{1}{\log_2 e}
\]

So,

\[
\lim_{n \to \infty} \frac{2H_n}{\log_2 n} + \lim_{n \to \infty} \frac{2H_n}{n \log_2 n} - \lim_{n \to \infty} \frac{4}{\log_2 n} = \frac{2}{\log_2 e} + 0 + 0 = \frac{2}{\log_2 e}
\]

Computing, we obtain

\[
\lim_{n \to \infty} \frac{\tau_A(n)}{n \log_2 n} = \frac{2}{\log_2 e} = 1.386294361
\]

Note that our answer makes perfectly good sense given our computational results! Therefore,

\[
\tau_A(n) \leq \frac{2}{\log_2 e} (n \log_2 n)
\]