Empirical and Average Case Analysis

We have discussed theoretical analysis of algorithms in a number of ways
- Worst case big O complexities
- Recurrence relations

What we often want to know is what will our typical or average case complexity be for our real problems of interest

Can do empirical analysis – Test the algorithm multiple times on representative data and average the results

Can also do average case analysis where we use probabilities to derive an average case complexity class for our algorithm, given certain probability distribution assumptions about the data
Quicksort vs Mergesort

Let's revisit Quicksort and Mergesort for our example

Mergesort recursively divides the unsorted list in half and then merges in linear time as we go back up the tree

Quicksort algorithm:
- Randomly pick a pivot within the unsorted list
- Swap everything smaller than the pivot to the left
- Swap everything greater than the pivot to the right
- Recursively quicksort each side in place

What are complexities of Mergesort and Quicksort?
Quicksort vs Mergesort

Let's revisit Quicksort and Mergesort for our example.

Mergesort recursively divides the unsorted list in half and then merges in linear time as we go back up the tree.

Quicksort algorithm:
- Randomly pick a pivot within the unsorted list
- Swap everything smaller than the pivot to the left
- Swap everything greater than the pivot to the right
- Recursively quicksort each side in place.

Mergesort is always $O(n \log n)$ independent of the initial list.

Quicksort is usually $O(n \log n)$ but is worst-case $O(n^2)$ depending on the initial list and how we choose the pivot.
- Why?
Choosing a Good Pivot

Choosing a good pivot is critical. If we choose a pivot which is the highest or lowest element in the list then the larger side of the partition decreases by only one element each call giving a tree of depth $n$.

The instances that become worst or best cases depend on the method used for choosing a pivot.

What would best pivot be?

Book’s pivot routine chooses first element in the list
  - Worst case is an already sorted list (!)
  - This would be bad for applications where the list may be already partially sorted

Alternatively could choose
  - A random element
  - The middle element
  - Median of first, last, and middle elements – fix worst case problem?
  - Median of a few random elements
Empirical Analysis

- In Empirical Analysis we simply run the problem against a large number of *representative* cases and then average the complexity results.

- **Key** is testing on data which is truly representative of what the algorithm will be actually be used on.
  - Thus if we are going to use it later on different distributions of data, we would need a new empirical analysis of our algorithm.
  - For sorting, we would want to test with lists which are as similar as possible to the types of lists (initial order, size, etc.) that we expect to encounter in actual usage.
Empirical Analysis

- For our sorting case, let's assume that the types of lists we will be sorting are uniformly randomly distributed.
- As such we can just always pick the first element of the list as our random pivot.
  - Why?
Empirical Comparison

Averages computed from 50 random samples for each list size

![Graph showing comparison between MergeSort and QuickSort](image-url)
Add \( n \log n \) to the Graph

Averages computed from 50 random samples

Size of List
Log Scale shows that we are $n \log n$

Can Approximate our Constants?

Size of List

CS 312 – Empirical and Average Case Analysis
Pick Suitable Constants

Averages computed from 50 random samples

![Graph showing comparisons of sorting algorithms]

- **Miliseconds** vs. **Size of List**
  - MergeSort
  - QuickSort
  - $n/67 \log n$
  - $n/40 \log n$
Worst Case

- So for average case, Quicksort is almost twice as fast
- But, what about worst case, where we know Quicksort might struggle compared to mergesort
- We use the same 50 tests for each list size, but only record the worst case of the 50 for each size and then see which of the two approaches is best for the worst case
  - Thus we are not computing the theoretical worst case, but the empirical average worst case for 50 runs at each list size
Worst Case in 50

Computed from 50 random samples

Size of List

<table>
<thead>
<tr>
<th>Miliseconds</th>
<th>MergeSort</th>
<th>QuickSort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>80</td>
<td>40</td>
</tr>
<tr>
<td>2000</td>
<td>120</td>
<td>60</td>
</tr>
<tr>
<td>3000</td>
<td>160</td>
<td>80</td>
</tr>
<tr>
<td>4000</td>
<td>200</td>
<td>100</td>
</tr>
<tr>
<td>5000</td>
<td>240</td>
<td>120</td>
</tr>
<tr>
<td>6000</td>
<td>280</td>
<td>140</td>
</tr>
<tr>
<td>7000</td>
<td>320</td>
<td>160</td>
</tr>
<tr>
<td>8000</td>
<td>360</td>
<td>180</td>
</tr>
<tr>
<td>9000</td>
<td>400</td>
<td>200</td>
</tr>
<tr>
<td>10000</td>
<td>440</td>
<td>220</td>
</tr>
</tbody>
</table>
Empirical Analysis

- Where would Quicksort 3-pivot sit on the graph?
- Which approach would be best if lists are more arbitrary
  - Some already sorted
  - Some with skewed distributions, etc.
  - How would you know?
  - Mergesort for these?
Empirical Comparison

Averages computed from 50 random samples for each list size

Size of List

Miliseconds
Average Case Analysis

- Average case analysis lets us derive a complexity class for the average case (rather than the normal worst case) using probabilities rather than actual trials.
- Just as in empirical analysis where it was critical to use representative data, it is critical that we use representative probability distributions of the data.
- Given these distributions we can algebraically derive the average case complexity class for the algorithm given the type of data represented by the assumed distributions.
- Requires review of probability, etc., so given time constraints we will just walk through a brief example here.
QuickSort Average Case Analysis

- We'll assume again that the lists are uniformly randomly distributed and still pick the first element as pivot
  - If not then distributions become more complex to work with

- Then there is a $1/n$ chance of picking any element $p$ as the pivot

- For a chosen $p$, the time do quicksort is
  - $t(|\text{left of pivot}|) + t(|\text{right of pivot}|) + \text{linear stuff}$
  - Where $t(m)$ is the average time to sort an array of size $m$

- Our random variable is $X$ (the time it takes to run quicksort for a particular list)

- Our goal is to find $E[X]$, the expected value (mean) of $X$
Quicksort Average Case Analysis

Assume list of size $n = 10$

What is probability of 2 items in left list and 8 in right?
Quicksort Average Case Analysis

\[ E[X] = \sum_x x p(x) \]

- Assume list of size \( n = 10 \)
- What is probability of 2 items in left list and 8 in right?
- \( n \) equi-probable scenarios

<table>
<thead>
<tr>
<th>1</th>
<th>9</th>
<th>1/n</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>1/n</td>
</tr>
</tbody>
</table>
| ... | ... | ...
| 9 | 1 | 1/n |
| 10 | 1/n |
Quicksort Average Case Analysis

Where \( l \) is the length of the left list and \( g(n) \) is the time to do the linear overhead part of quicksort which is independent of the partition. Note that there is a \( 1/n \) probability of each possible length for the left list (assuming unique list elements).

Some fun algebra (which we skip here) shows that the right side reduces to approximately

\[
2(n + 1) \ln n \in O(n \log n)
\]

Thus proving that average case for Quicksort is \( O(n \log n) \)