CS624 - Analysis of Algorithms

Fall 2018 – Sorting

September 26, 2018
The midterm exam will take place on Friday, October 19, in class.

Covered material: Induction, runtime analysis, sorting (mergesort, insertion sort, quicksort, heapsort, lower bounds), heaps, medians and order statistics (maybe).

The class on October 17 will be partly a review class.

Prepare your own questions to ask me!!!
• Probably 3 questions. Assume every topic will be covered.
• Bring class notes and homework assignments.
• No books, no computers, no cellphones/smartphones/tablets, strictly no friends.
• Bags will be put in the front of the class.
• It will count towards 20% of your final grade.
A Little More on Sorting

- How well can we really do?
- Is there a sorting method whose worst case runtime is $O(n)$?
- Obviously we can’t do better than that (why?).
- For the class of algorithms we’ve seen so far the answer is no. The lower bound really is $O(n \log n)$.
- These sorting algorithms are based on comparisons and can be modeled as binary decision trees.
The run of InsertionSort on an array of 3 elements: \( \{ a_1, a_2, a_3 \} \):
Another Simple Example

The run of Quicksort on an array of elements: \( \{a_1, a_2, a_3, a_4\} \), first partition:
Theorem

In a sorting algorithm modeled by a binary decision tree, the worst-case running time is $\Omega(n \log n)$.

Proof.

The worst-case running time is bounded below by the depth of the decision tree. The number of leaves in the decision tree must be the number of possible permutations, which is $n!$. The depth of a binary tree with $L$ leaves is $\Omega(\log L)$. Therefore the depth of the decision tree is

$$\Omega(\log n!) = \Omega(n \log n)$$
Still, Can We Do Better?

- When our model is not based on comparisons we can do better.
- Example: BucketSort.
- Simple case: We have a set of integers 1..n in some random order.
- How do we sort this?
A More Complicated Example

Given a set of integers \( \{a_1, a_2, \ldots, a_n\} \).
- Each integer is in the range of 1...M where \( M \geq n \).
- Create an array \( A[1..n] \) where the elements are sets of integers.
- Each set is called a bucket.
- Each integer in the original sequence will be put in its appropriate bucket.
The largest number bucket $A[j]$ can hold is $\left\lfloor \frac{jM}{n} \right\rfloor$. Therefore the index $j$ of the bucket that we want to place the number $a_k$ in must satisfy

$$\left\lfloor \frac{(j - 1)M}{n} \right\rfloor + 1 \leq a_k \leq \left\lfloor \frac{jM}{n} \right\rfloor$$

Since we always have $x - 1 < \left\lfloor x \right\rfloor$, this yields

$$\frac{(j-1)M}{n} < a_k \leq \frac{jM}{n} \Rightarrow j - 1 < \frac{a_kn}{M} \leq j \Rightarrow j = \left\lceil \frac{a_kn}{M} \right\rceil$$
Problem: Elements are not necessarily uniformly distributed in buckets.
Some buckets are empty, some may contain several elements.
What is the average cost of sorting the buckets?
Suppose we use InsertionSort to sort each bucket (good for small buckets).
Do not assume that since the average number of elements per bucket is 1 it means that the average runtime is $O(n)$. 
Bucketsort analysis

- If bucket $i$ has $n_i$ elements, sorting it takes $O(n_i^2)$.
- We can average over all the buckets.
- Since the distribution of elements in buckets is random we can average on $n_1$ (since it doesn’t matter which bin we pick).
- The expected value of $n_1$ is

$$\sum_{j=0}^{n} (\text{the probability that } j \text{ numbers land in bucket 1}) \cdot j^2$$
The probability of $j$ items landing in bucket 1 is the probability of selecting $j$ items out of $n$, \(_{n}^{j}\).

The probability of a particular combination is: \((\frac{1}{n})^j (1 - \frac{1}{n})^{n-j}\).

The probability of any $j$ elements landing in bucket 1 is:
\[
\left(\frac{1}{n}\right)^j \left(1 - \frac{1}{n}\right)^{n-j} \binom{n}{j}.
\]

The expected runtime of sorting $n_1$ is then
\[
\sum_{j=0}^{n} \left(\frac{1}{n}\right)^j \left(1 - \frac{1}{n}\right)^{n-j} \binom{n}{j} j^2
\]
This looks like a binomial generating function that has been differentiated. So let us set:

\[ f(x) = \sum_{j=0}^{n} \left( \frac{1}{n} \right)^j \left( 1 - \frac{1}{n} \right)^{n-j} \binom{n}{j} x^j \]

Then we have

\[ f'(x) = \sum_{j=0}^{n} \left( \frac{1}{n} \right)^j \left( 1 - \frac{1}{n} \right)^{n-j} \binom{n}{j} j x^{j-1} \]

We can’t just differentiate again, because we would get \( j(j-1) \). So we multiply by \( x \) first:

\[ x f'(x) = \sum_{j=0}^{n} \left( \frac{1}{n} \right)^j \left( 1 - \frac{1}{n} \right)^{n-j} \binom{n}{j} j x^j \]

and then we can differentiate:

\[ (x f'(x))' = \sum_{j=0}^{n} \left( \frac{1}{n} \right)^j \left( 1 - \frac{1}{n} \right)^{n-j} \binom{n}{j} j^2 x^{j-1} \]
Let us set \( g(x) = (xf'(x))' \).
Then we see that the expected value of \( n_1^2 \) is just \( g(1) \).
The closed form of \( f \) follows from the binomial theorem:

\[
\begin{align*}
f(x) &= \sum_{j=0}^{n} \left( \frac{1}{n} \right)^j \left( 1 - \frac{1}{n} \right)^{n-j} \binom{n}{j} x^j \\
&= \sum_{j=0}^{n} \left( \frac{x}{n} \right)^j \left( 1 - \frac{1}{n} \right)^{n-j} \binom{n}{j} \quad = \left( 1 + \frac{x - 1}{n} \right)^n
\end{align*}
\]
Going back to $g$:

$$f'(x) = n \left(1 + \frac{x - 1}{n}\right)^{n-1} \cdot \frac{1}{n} = \left(1 + \frac{x - 1}{n}\right)^{n-1}$$

and then

$$g(x) = (xf'(x))' = \left(x \left(1 + \frac{x - 1}{n}\right)^{n-1}\right)'$$

$$= \left(1 + \frac{x - 1}{n}\right)^{n-1} + (n - 1)x \left(1 + \frac{x - 1}{n}\right)^{n-2} \cdot \frac{1}{n}$$

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

By substituting 1 for $x$, we get $g(1) = 1 + \left(1 - \frac{1}{n}\right) = 2 - \frac{1}{n}$. That is the expected value of $n_i^2$, and in fact is the expected value of $n_i^2$ for any $i$. In short – the average time for sorting each bucket in bucketsort is $O(1)$ and the overall expected runtime is $O(n)$. 
And what happened to our lower bound?
We are not using a binary decision tree!
This is only because we know something about the input.
Also – we did only average case analysis. What is the worst case?