EECS3101 notes::Lower bounds

Jerry Wu

2023-02-09

Lower bounds

So far, we've only been talking about algorithm efficiency in terms of their upper bounds on its worst case time complexities. But it is also useful to talk about lower bounds to see what our limits of efficiencies are for certain problem (i.e. sorting). We want to know **at least how long** it takes to solve problems. However, keep in mind that lower bounds apply to a problem in general, rather than a specific algorithm.

> *"I'm still better than ChatGPT. Ask me your questions instead"-Larry YL Zhang 2023*

Motivation

We should strive to know our limits, but we should also approach the limits and understand why the limits exist so we know how to design effective solutions for the limit.

Lower bounds for sorting

We know a few sorting algorithms with worst case runtimes of $O(nlog(n))$. But is this the best we can do? It is, because sorting is $\Omega(nlog(n))$. $nlog(n)$ applies only to all **comparison based** sorting algorithms with **no assumptions (no precondition)** on the values of the elements. Sorting under $nlog(n)$ is possible, but it would no longer be general purpose.

Bucket sort

Suppose all keys come from a finite interval, say $[0, 1)$ and n keys. We can divide $[0, 1)$ into n equal sized sub intervals (buckets). Then we can insert the n numbers into the buckets.

- But what is the average number of keys in each bucket? It is $\frac{n_{keys}}{n_{buckets}}$

Then sort each number in each bucket using insertion sort. Finally, go throug the buckets in order, listing the elements. In the end, we get a $\Theta(n)$ runtime.

"We use bucket sort to mark exams"-Larry YL Zhang 2023

The worst case is when every element is in one bucket, whereby bubble sort is used, resulting in $O(n^2)$.

Stability of sorting algorithms (slide 15-16)

- A sorting algorithm is stable if two objects with equal keys appear in the same order in sorted output as they appear in the input array to be sorted.

Proof

So how can we prove that comparison based general purpose sorting is really $\Omega(nlog(n))$?

Define a general sorting algorithm as the following::

$$Sort(\{x, y, z\}), x \neq y \neq z$$

We constrct a decision tree for x, y, and z. Ever comparison based sorting algorithm has its own corresponding decision tree. The worst case is represented by the **longest path** from the root node to any leaf. The total possible orders for n elements would be $n!$. So this means the number of leaves in total would be $L \geq n!$.

Now what about the height?

Any binary tree with height h has at most 2^h leaves. So this means $L \leq 2^h$ as well. So $n! \leq L \leq 2^h \implies 2^h \geq n! \implies h \geq \log(n!) \approx \log(n^n) = n\log(n)$. Thus we have shown that the worst case number of comparisons needed to sort n elements is $\Omega(nlog(n))$

"Now you can convince other people about this!"-Larry YL Zhang 2023

"Is this too much math for a rainy day?"-Larry YL Zhang 2023

Proof of $\log(n!) \in \Omega(n\log(n))$

$$log(n!) = \sum_{k=1}^{n} log(k) \geq \log(\frac{n}{2}) + \ldots + \log(n) \geq \log(\frac{n}{2}) + \log(\frac{n}{2}) + \ldots + log(\frac{n}{2})$$

$$\geq \frac{n}{2}\log(\frac{n}{2}) \in \Omega(nlog(n))\blacksquare$$

Optimizing comparisons (slide 29-32)

In order to be efficient, we want to make sure that every comparison is meaningful and does something for the overall problem. We can use an idea called **reduction** tp prove and find an optimal solution for comparisons.

When using reduction, we use another problem's lower bound. If we know B can be solved by solving an instance of A i.e. A is "harder" than B. So then we know that B has a lower bound $L(n)$. So A also has a lower bound if $L(n)$.

"Have you learned something meaningful?"-Larry YL Zhang 2023

"If you uninstall OneDrive on your computer, it will reinstall itself"-Larry YL Zhang 2023

Example

Prove that $ExtractMax$ on a binary heap is lower bounded by $\Omega(\log(n))$.

Suppose that $ExtractMax$ is done faster than $\log(n)$, then $HeapSort$ can be done faste than $n\log(n)$ because $HeapSort$ is $ExtractMax$ performed n times. By way of contradiction, we have shown that $ExtractMax \in \Omega(n\log(n))$.

For reductions, the helper function is the "harder" problem compared to the base problem.