Lecture 28:
Problem Complexity

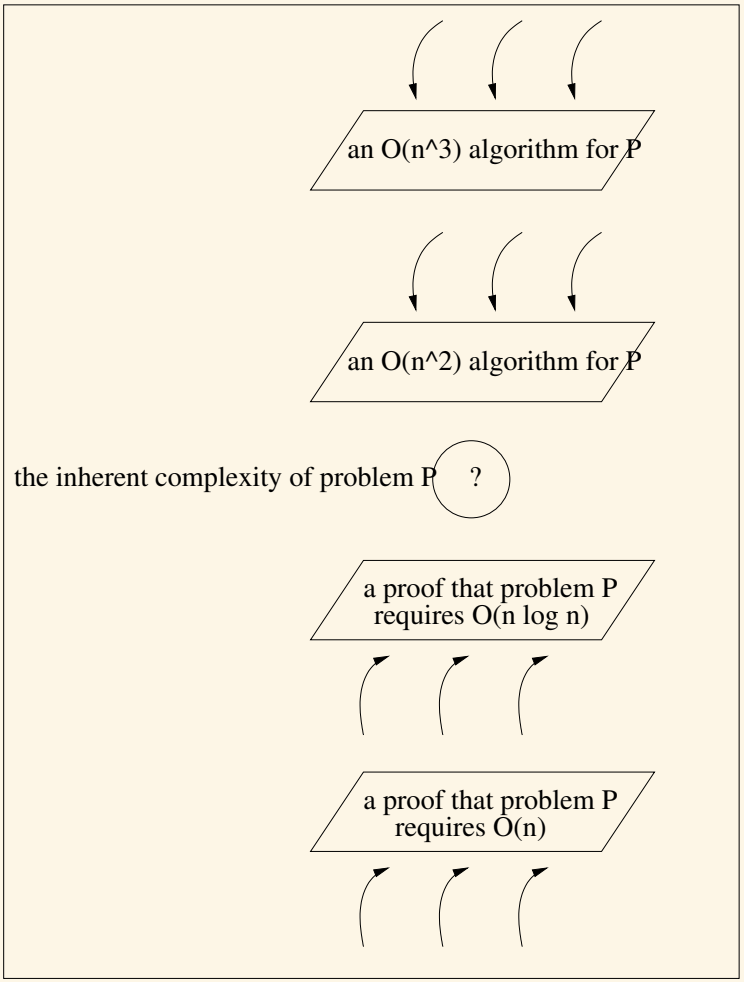
Aims:

- To look at the idea of problem complexity and to contrast it with algorithm complexity;
- To discuss bounds on problem complexity.

28.1. Problem Complexity

- We have been studying *algorithm complexity*.
  - E.g. the complexity of bubble-sort; of merge-sort; of quick-sort; of heap-sort; etc.
- But we can also study *problem complexity*.
  - E.g. the complexity of sorting in general;
  - the inherent worst-case complexity.
- For a lot of problems, we don't know the problem complexity.
- But we have bounds on it.
- We know it falls between
  - an upper bound and
  - a lower bound.
- Discovery of better algorithms brings the upper bound on the worst-case time complexity down.
  - Suppose we have problem P and someone comes along with algorithm A of complexity $O(n^3)$.
  - So we know the problem complexity cannot be higher than $O(n^3)$.
  - Later, someone discovers a better algorithm, whose complexity is $O(n^2)$.
  - So the problem complexity cannot be higher than $O(n^2)$.

- Each better algorithm brings the upper bound downwards.
- Computer Scientists prove statements about the lower bound.
  - Suppose they prove, for example, that the problem simply cannot be solved in less than linear time, $O(n)$. In other words, they prove that no algorithm could possibly exist that would solve this problem in less than $O(n)$ time.
  - Note they are not coming up with algorithms (unlike the story with the upper bound). They prove that no such algorithm could exist so the problem complexity cannot be lower than $O(n)$.
  - Later someone finds a proof that the problem cannot be solved in less than $O(n \times \log n)$ time.
  - They prove no such algorithm could exist so the problem complexity cannot be lower than $O(n \times \log n)$ time.
- Each better proof brings the lower bound upwards.
- By the way, finding such proofs can be hard! You have to prove something that will hold true for all possible algorithms that solve the problem (not just algorithms that exist but ones that could be discovered in the future). You have to prove that none of these possible algorithms could ever do better than, e.g., $O(n \times \log n)$.
- And in some sense finding better proofs gets harder and harder. It might be easy to show that no algorithm could possibly solve P in $O(1)$ time; it might not be much harder to show that no algorithm can solve P in $O(n)$ time. But to show that the lower bound is $O(n \times \log n)$ or $O(n^2)$ or... might get harder and harder.


an O(n^3) algorithm for P
an O(n^2) algorithm for P
the inherent complexity of problem P
?
a proof that problem P requires O(n log n)
a proof that problem P requires O(n)

- If we're lucky, the lower bound (discovered by proof) and the upper bound (exhibited by an algorithm) will coincide.
- Then we know the inherent complexity of problem P.
- We say that the problem is *closed* as far as big-Oh time estimates are concerned.
- E.g. If we have a proof that no algorithm can solve P in less than $O(n \times \log n)$ time (lower bound) and we have an algorithm that takes $O(n \times \log n)$ time (upper bound), then $O(n \times \log n)$ is P's complexity. (Of course, we then might continue to search for algorithms whose constant factors are smaller.)

- More often there's a gap.
- E.g. we have a proof that no algorithm can solve P in less than $O(n \times \log n)$ time and the best algorithm we know takes $O(n^2)$ time.
- Where does this leave P?
- Maybe we've got the best algorithm, and it's the proof that falls short: we should seek a proof for a higher lower bound!
- Maybe the proof shows the best we can do, in which case our algorithm falls short: we should seek a better algorithm!
- Maybe both fall short.
- We don't know which of these three is the case. When there's a gap, we should seek both better proofs and better algorithms.

> *"... if a problem gives rise to a... gap, the deficiency is not in the problem but in our knowledge about it. We have failed either in finding the best algorithm for it or in proving that a better one does not exist, or in both".* [Har92], p.148

Here are some examples of problems and their bounds.

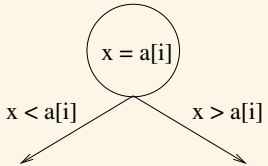
- **Problem:** Searching an unordered list of n items.
  - Upper bound: $O(n)$ comparisons (from linear search).
  - Lower bound: $O(n)$ comparisons.
  - No gap; so we know the problem complexity: $O(n)$.
- **Problem:** Searching an ordered list of n items.
  - Upper bound: $O(\log n)$ comparisons (from binary search).
  - Lower bound: $O(\log n)$ comparisons (see proof later).
  - No gap; so we know the problem complexity: $O(\log n)$.
- **Problem:** Sorting n arbitrary elements.
  - Upper bound: $O(n \times \log n)$ comparisons (from, e.g., merge sort).
  - Lower bound: $O(n \times \log n)$ comparisons.
  - No gap; so we know the problem complexity: $O(n \times \log n)$.
- **Problem:** Towers of Hanoi for n disks ($n > 1$).
  - Upper bound: $O(2^n)$ moves.
  - Lower bound: $O(2^n)$ moves.
  - No gap; so we know the problem complexity: $O(2^n)$.
- **Problem:** Multiplication of two integers, where n is the number of digits.

  - Upper bound: $O(n \log n \log \log n)$.
  - Lower bound: $O(n)$.
  - There's a gap (but only a small one)!

- **Problem:** Finding the minimum spanning tree in a graph of n edges.
  - Upper bound: $O(n \times f(n))$ (where $f(n)$ is a very slow growing function, whose definition I will not give you here).
  - Lower bound: $O(n)$.
  - There's a gap (but only a small one)!
- Do not be fooled by the last two examples into thinking that all gaps are small!

28.2. Proofs of Lower Bounds

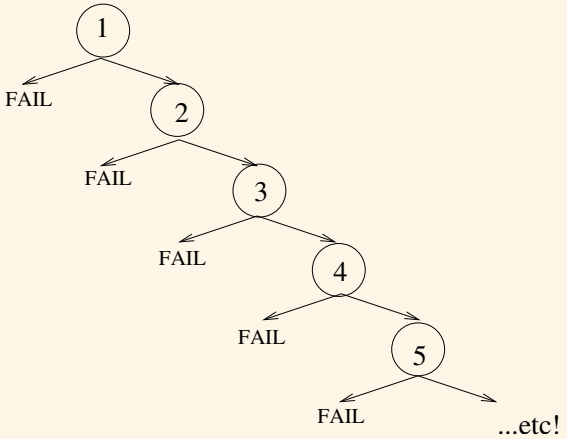
- Proofs of lower bounds are hard to come up with. (That's one of the main reasons why there are gaps for so many problems.)
- I thought we should look at one such proof, in very high level terms. But we are not going to look at any others in this module.
- Let's (informally) prove that $O(\log n)$ is a lower bound for searching for an item x in an ordered list or array a of n items.
- Any such algorithm will carry out comparisons between x and elements of the array, $a[i]$.
- For each comparison, there are three possible outcomes:
  - $x < a[i]$
  - $x = a[i]$
  - $x > a[i]$
- We can depict the behaviour of any such algorithm as a binary tree, where each node represents a comparison.
  - If $x < a[i]$, follow the left branch;
  - If $x = a[i]$, terminate;
  - If $x > a[i]$, follow the right branch.



- So the worst-case time depends on the height of the tree.

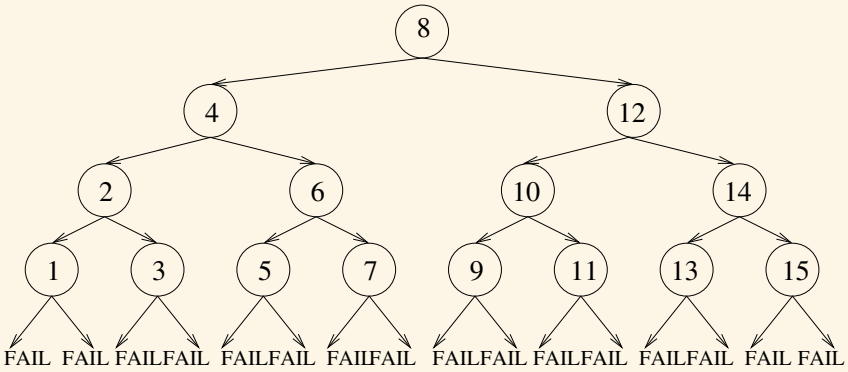
Compare this tree...

100	110	120	130	140	150	160	170	180	190	200	210	220	230	240
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



...with this tree

100	110	120	130	140	150	160	170	180	190	200	210	220	230	240
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



- Think about all the possible binary trees you could build with these kinds of nodes. Think about their height.
- The first diagram above and ones like it will be the tallest.
- The second diagram above will be the shortest.
- The height of the second diagram is $O(\log n)$.
- So the best performance we can hope for on this problem comes from algorithms that carry out these comparisons.
- This is therefore our lower bound.
- No algorithm could ever do better.

- And, of course, as we saw in a previous lecture, these are actually the comparisons that binary search does. So we are in the happy position of having an algorithm whose performance is as good as the lower bound.

Acknowledgements

The diagram showing upper and lower bounds is based on one in [Har92].

The proof of the lower bound on searching for an item in an ordered list is based on treatments given in [Har92] and [HSR96].

Clip Art (of head with bomb) licensed from the Clip Art Gallery on DiscoverySchool.com.

References

[Har92] D. Harel. *Algorithmics: The Spirit of Computing.* Addison-Wesley, 2nd edition, 1992.

[HSR96] E. Horowitz, S. Sahni, and S. Rajasekaran. *Computer Algorithms/C++.* W.H. Freeman, 1996.