Sorting & Searching Lower Bounds

Pomona College - Spring 2016
Michael J Bannister

Unsorted Search

- **The unsorted search problem**
  Input: An unsorted array *A* of len *n* and a value *v*
  Output: The index of *v* in *A* or -1 if *v* is not in *A*

- **Lower bound**
  If an algorithm *P* solves this problem, then *P* must read every cell of *A*, and therefore must have a runtime ≥ *C n*.

Sorted Search

- **The sorted search problem**
  Input: A sorted array *A* of length *n* and a value *v*
  Output: The index of *v* in *A* or -1 if *v* is not in *A*

- **Lower bound**
  If a comparison based algorithm *P* solves this problem, then it must make at least *log n* comparisons, and therefore must have a runtime ≥ *C log n.*

Sorting Problem

- **The sorting problem**
  Input: An array *A* of length *n*
  Output: An array *J* of len *n* containing exactly the values 0,1,…,n-1 in some order such that
  A[J[0]] ≤ A[J[1]] ≤ … ≤ A[J[n-2]] ≤ A[J[n-1]]

- **Lower bound**
  If comparison based algorithm *P* solves this problem, then *P* must make at least *C n log n* comparisons, and therefore must have a runtime ≥ C n log n.

Proof Strategies

- **Adversarial Construction**
  Come up with a method that an adversary could use to construct "bad" inputs for any possible algorithm.
- **Information Theoretic**
  Show that each primitive step of an algorithm (comparison for sorting) can only learn a small about of information about the input.

We will only be working with non-randomized algorithms!

Unsorted Search

Adversarial Strategy

- Every time the algorithm looks in an array cell return $v+1$, except when there is only one cell left, then return v.
- This strategy builds an array that will always look in every array cell.

Sorted Search

Information theoretic

Each comparison can eliminate at most half of the remaining indexes. Therefore we must make at least $\log n$ comparisons to find the value.

This process can be illustrated with a decision tree like the animal game uses.

Sorting Problem

Information theoretic

There are $n!$ possible out puts, and only one is valid. Furthermore, each comparison can eliminate at most half of the valid outputs. Thus, we must make at least $\log (n!) \geq C\, n \log n$ comparisons.

This process can be illustrated with a decision tree like the animal game uses.

Conclusion

The sorting algorithms we have learned in class are asymptotically optimal!