Lecture 2: Concrete Models and Tight Upper and Lower Bounds

David Woodruff
Carnegie Mellon University
Theme: Tight Upper and Lower Bounds

• Number of comparisons to sort an array

• Number of exchanges to sort an array

• Number of comparisons needed to find the largest and second-largest elements in an array

• Number of probes into a graph needed to determine if the graph is connected
Formal Model

- Look at models which specify exactly which operations may be performed on the input, and what they cost
  - E.g., performing a comparison, or swapping a pair of elements

- An upper bound of $f(n)$ means the algorithm takes at most $f(n)$ steps on any input of size $n$

- A lower bound of $g(n)$ means for any algorithm there exists an input for which the algorithm takes at least $g(n)$ steps on that input
Sorting in the Comparison Model

• **Definition:** in the comparison model, we have an input consisting of n items (typically in some initial order). An algorithm may compare two items (asking is \( a_i > a_j \)) at a cost of 1. Moving the items around is free.

• No other operations are allowed, such as using the items as indices, XORing them, hashing, etc.

• **Sorting:** given an array \( a = [a_1, ..., a_n] \), the output is a permutation \( \pi(a) = [a_{\pi(1)}, ..., a_{\pi(n)}] \) in which the elements are in increasing order.
• **Theorem:** Any deterministic comparison-based sorting algorithm must perform at least $\lg(n!)$ comparisons to sort $n$ elements in the worst case, i.e., for any sorting algorithm $A$ and $n \geq 2$, there is an input $I$ of size $n$ so that $A$ makes $\geq \lg(n!) = \Omega(n \log n)$ comparisons to sort $I$.

• Need to rule out *any* possible algorithm

• Proof is information-theoretic
Sorting Lower Bound

• **Proof:** Suppose there is a problem with M possible outputs
  • For sorting $M = n!$ since for each possible output permutation $\pi$, there is an input for which the output is $\pi$

• Further, suppose for each possible output to the problem, there is an input for which that output is the only correct answer
  • For sorting there are inputs for which $\pi$ is the only correct answer

• Then there is a lower bound of $\lg M$
  • Consider a set of inputs in one-to-one correspondence with the M possible outputs. Algorithm needs to find out which of the M outputs is the right one for a given input, and each comparison can be answered in a way that removes at most half of the possible inputs remaining from consideration
Sorting Lower Bound

- Information-theoretic: need $\lg(n!)$ bits of information about the input before we can correctly decide on the output

- $\lg(n!) = \lg(n) + \lg(n - 1) + \lg(n - 2) + \ldots + \lg(1) < n \lg n$

- $\lg(n!) = \lg(n) + \lg(n - 1) + \lg(n - 2) + \ldots + \lg(1) > \left(\frac{n}{2}\right) \log\left(\frac{n}{2}\right) = \Omega(n \lg n)$

- $n! \in \left[\left(\frac{n}{e}\right)^n, n^n\right]$, so $n \lg n - n \lg e < \lg(n!) < n \lg n$
  
  - $n \lg n - 1.443n < \lg(n!) < n \lg n$

- $\lg(n!) = (n \lg n)(1 - o(1))$
Sorting Upper Bounds

• Suppose for simplicity $n$ is a power of 2

• Binary insertion sort: using binary search to insert each new element, the number of comparisons is $\sum_{k=2,\ldots,n}[\lg k] \leq n \lg n$
  
  • Note: may need to move items around a lot, but only counting comparisons

• Mergesort: merging two lists of $n/2$ elements requires at most $n-1$ comparisons
  
  • Unrolling the recurrence, total number of comparisons is
    
    $$(n - 1) + 2\left(\frac{n}{2} - 1\right) + \ldots + \frac{n}{2}(2 - 1) = n \lg n - (n - 1) < n \lg n$$
Selection in the Comparison Model

• How many comparisons are necessary and sufficient to find the maximum of n elements in the comparison model?

• **Claim:** n-1 comparisons are sufficient
• **Proof:** scan from left to right, keep track of the largest element so far

• For lower bounds, what does our earlier information-theoretic argument give?
  • Only $\Omega(\log n)$, which is too weak

• Also, we have to look at all elements, otherwise we may have not looked at the largest, but that can be done with n/2 comparisons, also not tight
Lower Bound for Finding the Maximum

• **Claim:** n-1 comparisons are needed in the worst-case to find the maximum of n elements

• **Proof:** suppose A is an algorithm which finds the maximum of n distinct elements using fewer than n-1 comparisons
  - Construct a graph G in which we join two elements by an edge if they are compared by A
  - G has at least 2 connected components $C_1$ and $C_2$
  - Suppose A outputs element $u$ as the maximum, and $u \in C_1$
  - Add a large positive number to each element in $C_2$
  - Does not change any of the comparisons made by A, so will still output $u$
  - But now $u$ is not the maximum, so A is incorrect
Lower Bound for Finding the Maximum

• **Recap**: upper and lower bounds match at n-1

• Argument different from information-theoretic bound for sorting

• Instead,
  • if algorithm makes too few comparisons on some input \( \text{In} \) and outputs \( \text{Out} \),
  • find another input \( \text{In}' \) where the algorithm makes the same comparisons and also outputs \( \text{Out} \),
  • but \( \text{Out} \) is not a correct output for \( \text{In}' \)
An Adversary Argument

• If algorithm makes “too few” comparisons, fool it into giving an incorrect answer

• Any deterministic algorithm sorting 3 elements must perform at least 3 comparisons
  • If < 2 comparisons, some element not looked at and the algorithm is incorrect
  • After first comparison, 3 elements are w, l, and z, the winner and loser of the first comparison, as well as the uninvolved item
  • If the second query is between w and z, say
    • w is larger
  • If the second query is between l and z, say
    • l is smaller
  • Algorithm needs one more comparison for correctness

• Goal: give an adversary answering comparisons so that (a) answers consistent with some input In, and (b) answers make the algorithm perform “many” comparisons
First and Second Largest of n Elements

• How many comparisons are necessary (lower bound) and sufficient (upper bound) to find the first and second largest of n distinct elements?

• **Claim:** n-1 comparisons are needed in the worst-case

• **Proof:** need to at least find the maximum
What about Upper Bounds?

• **Claim:** $2n-3$ comparisons are sufficient to find the first and second-largest of $n$ elements

• **Proof:** find the largest using $n-1$ comparisons, then find the largest of the remainder using $n-2$ comparisons, so $2n-3$ total

• Upper bound is $2n-3$, and lower bound $n-1$, both are $\Theta(n)$ but can we get tight bounds?
Second Largest of n Elements Upper Bound

- **Claim:** $n + \log n - 2$ comparisons are sufficient to find the first and second-largest of $n$ elements.
- **Proof:** find the maximum element using $n-1$ comparisons by grouping elements into pairs, finding the maximum in each pair, and recursing.

What can we say about the second maximum?
- Must have been directly compared to the maximum and lost, so $\log(n)-1$ additional comparisons suffice. Kislitsyn (1964) shows this is optimal.
Sorting in the Exchange Model

• Consider a shelf containing $n$ unordered books to be arranged alphabetically. How many swaps do we need to order them?

• **Definition:** In the exchange model, an input consists of $n$ items, and the only operation allowed on the items is to swap a pair of them at a cost of 1 step.
  • All other work is free, e.g., the items can be examined and compared

• How many exchanges are necessary and sufficient?
Sorting in the Exchange Model

- **Claim:** n-1 exchanges is sufficient
- **Proof:** here’s an algorithm:
  - In first step, swap the smallest item with the item in the first location
  - In second step, swap the second smallest item with the item in the second location
  - In k-th step, swap the k-th smallest item with the item in the k-th location
    - If no swap is necessary, just skip a given step
  - No swap ever undoes our previous work
  - At the end, the last item must already be in the correct location
Lower Bound for Sorting in Exchange Model

- **Claim:** n-1 exchanges are necessary in the worst case
- **Proof:** create a directed graph in which the edge (i,j) means the book in location i must end up in location j

![Graph for input [f c d e b a g]](image)

- **Graph is a set of cycles**
  - Indegree and Outdegree of each node is 1
Lower Bound for Sorting in Exchange Model

- What is the effect of exchanging any two elements in the same cycle?
  - Suppose we have edges \((i_1, j_1)\) and \((i_2, j_2)\) and swap elements in locations \(i_1\) and \(i_2\)
  - This replaces these edges with \((i_2, j_1)\) and \((i_1, j_2)\) since now the item in position \(i_2\) need to go to \(j_1\) and item in position \(i_1\) need to go to \(j_2\)
  - Since \(i_1\) and \(i_2\) in the same cycle, now we get two disjoint cycles
Lower Bound for Sorting in Exchange Model

• What is the effect of exchanging any two elements in different cycles?
  • If we swap elements \(i_1\) and \(i_2\) in different cycles, similar argument shows this merges two cycles into one cycle
Lower Bound for Sorting in Exchange Model

• What is the effect of exchanging any two elements in the same cycle?
  • Get two disjoint cycles

• What is the effect of exchanging any two elements in different cycles?
  • Merges two cycles into one cycle

• How many cycles are in the final sorted array?
  • n cycles

• Suppose we begin with an array \([n, 1, 2, ..., n-1]\) with one big cycle
• Each step increases the number of cycles by at most 1, so need \(n-1\) steps
Query Models and Evasiveness

- Let $G$ be the adjacency matrix of an $n$-node graph
  - $G[i,j] = 1$ if there is an edge between $i$ and $j$, else $G[i,j] = 0$
- In 1 step, we can query any element of $G$. All other computation is free
- How many queries do we need to tell if $G$ is connected?
  - Claim: $n(n-1)/2$ queries suffice
  - Proof: Just query every pair $\{i,j\}$ to learn $G$, then check if $G$ is connected

- What about lower bounds?
Connectivity is an Evasive Graph Property

- **Theorem:** \( n(n-1)/2 \) queries are necessary to determine connectivity
- **Proof:** adversary strategy: given a query \( G[u,v] \), answer 0 *unless* that would cause the graph to become disconnected
- **Invariant:** for any unasked pair \{u,v\}, the graph revealed so far has no path from u to v
- **Reason:** consider the last edge \{u’,v’\} revealed on that path. Could have answered 0 and kept same connectivity by having edge \{u,v\} be present
Connectivity is an Evasive Graph Property

• **Theorem:** $n(n-1)/2$ queries are necessary to determine connectivity

• **Proof:** adversary strategy: given a query $G[u,v]$, answer 0 unless that would cause the graph to become disconnected

• Invariant: for any unasked pair $\{u,v\}$, the graph revealed so far has no path from $u$ to $v$

• Suppose there is some unasked pair $\{u,v\}$ by the algorithm
  • If algorithm says “connected”, we place all 0s on unasked pairs
  • If algorithm says “disconnected”, we place all 1s on unasked pairs

• So algorithm needs to query every pair