Comparision-based sorting

Stephen Huan

https://cgdct.moe

Theory club 2024-01-19

Overview

Introduction

Counting

Information theory

Game theory

Sorting

Given a list $(\ell_i)_{i=1}^{n}$ and an ordering $\prec$, find permutation π s.t.

$$\ell_{\pi_i} \prec \ell_{\pi_{i+1}}, \quad \forall i \in \mathbb{N},\, 1 \leq i \leq n-1.$$

Sorting

Given a list $(\ell_i)_{i=1}^n$ and an ordering $\prec$, find permutation π s.t.

$$\ell_{\pi_i} \prec \ell_{\pi_{i+1}}, \quad \forall i \in \mathbb{N}, 1 \leq i \leq n-1.$$

Comparison-based: only access ℓ through queries against oracle

$$\prec (\ell_i, \ell_j), \quad \forall i, j \in \mathbb{N}, 1 \leq i, j \leq n.$$

Sorting

Given a list $(\ell_i)_{i=1}^n$ and an ordering $\prec$, find permutation π s.t.

$$\ell_{\pi_i} \prec \ell_{\pi_{i+1}}, \quad \forall i \in \mathbb{N}, 1 \leq i \leq n-1.$$

Comparison-based: only access ℓ through queries against oracle

$$\prec (\ell_i, \ell_j), \quad \forall i, j \in \mathbb{N}, 1 \leq i, j \leq n.$$

Assume algorithm *functionally pure*, i.e. deterministic.

A simple counting argument

Suppose algorithm makes k queries.

A simple counting argument

Suppose algorithm makes k queries.

Oracle only has 2^k possible responses.

A simple counting argument

Suppose algorithm makes k queries.

Oracle only has 2^k possible responses.

But there are $n!$ possible permutations.

A simple counting argument

Suppose algorithm makes k queries.

Oracle only has 2^k possible responses.

But there are $n!$ possible permutations.

So $2^k \geq n!$, or $k \geq \Theta(n \log n)$.

A fatal flaw

Interpretation depends on which *indices* were queried!

A fatal flaw

Interpretation depends on which *indices* were queried!

Possible responses still 2^k, but possible *queries* $\binom{n}{2}^k$ for

$$2^k \binom{n}{2}^k = [n(n-1)]^k \leq n^{2k},$$

resulting in the disappointing trivial bound $k \geq \Theta(n)$.

An information-theoretic perspective?

Distribution X over permutations

An information-theoretic perspective?

Distribution X over permutations

Uniform is the worst-case distribution over n possibilities

$$\mathbb{H}[X] \leq \mathbb{H}[\mathsf{Unif}(\{x_i\}_{i=1}^{n})] = \Theta(n \log n).$$

An information-theoretic perspective?

Distribution X over permutations

Uniform is the worst-case distribution over n possibilities

$$\mathbb{H}[X] \leq \mathbb{H}[\mathsf{Unif}(\{x_i\}_{i=1}^{n})] = \Theta(n \log n).$$

But only receive at most 1 bit of information from the oracle.

An information-theoretic perspective?

Distribution X over permutations

Uniform is the worst-case distribution over n possibilities

$$\mathbb{H}[X] \leq \mathbb{H}[\mathsf{Unif}(\{x_i\}_{i=1}^n)] = \Theta(n \log n).$$

But only receive at most 1 bit of information from the oracle.

So we need $\Theta(n \log n)$ queries.

More precisely

But how do we compute the entropy given new information?

More precisely

But how do we compute the entropy given new information?

Conditional entropy $\mathbb{H}[X \mid y]$ for oracle response y,

$$\mathbb{H}[X \mid y] \coloneqq \mathbb{H}[X] - \mathbb{I}[X, y].$$

More precisely

But how do we compute the entropy given new information?

Conditional entropy $\mathbb{H}[X \mid y]$ for oracle response y,

$$\mathbb{H}[X \mid y] \coloneqq \mathbb{H}[X] - \mathbb{I}[X, y].$$

We have $\mathbb{I}[X, y] \leq \min(\mathbb{H}[X], \mathbb{H}[y])) \leq 1$.

More precisely

But how do we compute the entropy given new information?

Conditional entropy $\mathbb{H}[X \mid y]$ for oracle response y,

$$\mathbb{H}[X \mid y] \coloneqq \mathbb{H}[X] - \mathbb{I}[X, y].$$

We have $\mathbb{I}[X, y] \leq \min(\mathbb{H}[X], \mathbb{H}[y])) \leq 1$.

But this is only *in expectation*.

More precisely

But how do we compute the entropy given new information?

Conditional entropy $\mathbb{H}[X \mid y]$ for oracle response y,

$$\mathbb{H}[X \mid y] \coloneqq \mathbb{H}[X] - \mathbb{I}[X, y].$$

We have $\mathbb{I}[X, y] \leq \min(\mathbb{H}[X], \mathbb{H}[y])) \leq 1$.

But this is only *in expectation*.

Even so, this is sufficient to show no algorithm is $o(n \log n)$.

A different perspective

We can view finding a *lower bound* on the *worst-case* as

$$\min_{\textsf{alg } \mathcal{A}} \max_{(\ell_i)_{i=1}^n} (\#\ \textsf{steps } \mathcal{A} \textsf{ takes on } \ell).$$

A different perspective

We can view finding a *lower bound* on the *worst-case* as

$$\min_{\text{alg } \mathcal{A}} \max_{(\ell_i)_{i=1}^n} (\#\ \text{steps } \mathcal{A} \text{ takes on } \ell).$$

Each *player* controls a variable.

A different perspective

We can view finding a *lower bound* on the *worst-case* as

$$\min_{\text{alg } \mathcal{A}} \max_{(\ell_i)_{i=1}^n} (\#\text{ steps } \mathcal{A} \text{ takes on } \ell).$$

Each *player* controls a variable.

This perspective suggests a constructive proof.

A different perspective

We can view finding a *lower bound* on the *worst-case* as

$$\min_{\text{alg } \mathcal{A}} \max_{(\ell_i)_{i=1}^n} (\#\text{ steps } \mathcal{A} \text{ takes on } \ell).$$

Each *player* controls a variable.

This perspective suggests a constructive proof.

Normally, we think of ℓ as fixed in advance.

A different perspective

We can view finding a *lower bound* on the *worst-case* as

$$\min_{\text{alg } \mathcal{A}} \max_{(\ell_i)_{i=1}^n} (\#\text{ steps } \mathcal{A} \text{ takes on } \ell).$$

Each *player* controls a variable.

This perspective suggests a constructive proof.

Normally, we think of ℓ as fixed in advance.

Instead, *adaptively* choose ℓ dynamically.

A constructive proof

Maintain a set of all possible permutations.

A constructive proof

Maintain a set of all possible permutations.

For every query, pick the result that maximizes the size.

A constructive proof

Maintain a set of all possible permutations.

For every query, pick the result that maximizes the size.

The best $\mathcal{A}$ can possibly do is split the set evenly.

A constructive proof

Maintain a set of all possible permutations.

For every query, pick the result that maximizes the size.

The best $\mathcal{A}$ can possibly do is split the set evenly.

This means $\mathcal{A}$ must do $\Omega(n \log n)$ queries on this list.