Lower bounds on data structures

We have seen:

- Optimality of binary heap for comparison-model priority queues
  Based on the ability to sort using heaps
  Can be sidestepped by using integer arithmetic and array indexing instead of only comparisons (e.g. flat trees)

- Impossibility of nontrivial set disjointness
  Based on unproven assumption (SETH)

This time: Lower bounds for range search
Proven rigorously in a very general computational model
Are augmented search trees optimal?

We have seen that a very general class of dynamic range searching problems can be solved in time $O(\log n)$

Natural question: Is that the right time bound or can we do better?

Answer: we can prove $\Omega(\log n)$, for:

- Simple and natural range searching problem: *range sum*
  Data = ordered keys and numeric values
  Query = sum of values for key-value pairs with key in range

- A very general model of computing: *cell probe model*
  Only measure communication between CPU and memory
Prefix sum problem

Simplified version of the range sum problem
(for lower bounds, simpler problem $\implies$ stronger bound)

Maintain array $A[0] \ldots A[n - 1]$ of numbers

Update($i, x$): set $A[i]$ to new value $x$


(Can be handled by a static balanced binary search tree augmented for range sums. If these operations are hard, so are the more general operations of insertion + deletion + range sum)
Cell probe model of computing

Central processor has $O(1)$ registers, each holding one word (binary value of length $w \geq \log_2 n$); memory has up to $2^w$ words.

We count only steps that move a word between CPU and memory $\Rightarrow$ lower bound doesn’t depend on what other steps are allowed.

Measure communication between CPU and memory.
Fitting prefix sums to cell probe model

We are going to prove a lower bound for prefix sums of \( n \) \( w \)-bit binary numbers (representation size of the input values should be the same as the word size of the computer)

We will use \( n = \) a power of two (unrelated to word size)

To avoid questions of integer overflow, we will assume all arithmetic is modulo \( 2^w \) (just do binary addition and ignore overflows)

Goal: Find a sequence of prefix sum operations that forces any correct data structure to do a lot of CPU–memory communication
A special permutation of $n$

Assume $n = 2^k$

Define “bit reversal permutation” $r(i)$:

- Write $i$ as a $k$-bit binary number
- Reverse the bits
- Interpret the result as a binary number

E.g. for $k = 8$, $222_{10} = 11011110_2$ becomes $01111011_2 = 123_{10}$
Computing sequence of bit-reversals

To compute a sequence of length $2^k$, consisting of all $k$-bit numbers in bit-reversed order, compute the same sequence recursively for $k - 1$ and use it twice:

```python
def bitrev(k):
    if k == 0:
        return [0]
    L = bitrev(k-1)
    return [2*x for x in L] + [2*x+1 for x in L]
```

$[0] \rightarrow [0, 1] \rightarrow [0, 2, 1, 3] \rightarrow [0, 4, 2, 6, 1, 5, 3, 7] \rightarrow ...$

Each value in the second half of the sequence is one plus the corresponding value in the first half.
A difficult sequence of prefix-sum operations

Initialize all data values $A[i]$ to zero, then:

For each index $i$ in $\text{bitrev}[k]$:

- Set $A[i]$ to be a random $w$-bit number
- Query the prefix sum $A[0] + \cdots + A[i]

E.g. when $n = 8$, $k = 3$, we perform the operations
  Update(0, random), Query(0), Update(4, random), Query(4),
  Update(2, random), Query(2), Update(6, random), Query(6),
  Update(1, random), Query(1), Update(5, random), Query(5),
  Update(3, random), Query(3), Update(7, random), Query(7)
A binary tree on the sequence of operations

This is not a data structure! It’s just a mathematical tree that we will use in the lower bound proof.
Information transfer

For any data structure for prefix sums, and any node $x$ of this tree, define the information transfer of $x$ to be the number of times an operation in the right descendants of $x$ reads a memory cell that was last written during the operations in the left descendants of $x$.

Each memory read contributes to information transfer at $\leq 1$ node $\implies$ total number of read steps $\geq$ total information transfer.
Information transfer $\geq$ descendants/2

Information transfer = number of times an operation in node’s right descendants reads a memory cell last written on the left

Let $d = \#\text{descendants}/2 = \#\text{left updates} = \#\text{right queries}$

There are $2^{wd}$ different possible values for the updates on the left, each of which would produce different query results on the right (Independently from information derived from non-transfer reads)

$\Rightarrow$ for correct queries, information transfer $\geq d$
Finishing the lower bound

Information transfer at root node of tree: \( \geq n/2 \)

Information transfer at \( i \)th level of tree:
2\(^i\) nodes with transfer \( \geq n/2^{i+1} \), total \( \geq n/2 \)

Total over whole tree: \( \geq (n/2) \times \# \text{ levels} = (n/2) \log_2 n \)

There are 2\(n\) prefix sum operations (updates and queries together)
⇒ average number of memory reads per operation \( \geq \frac{1}{4} \log_2 n \)

Every prefix sum data structure that fits into the cell probe model of computation requires \( \Omega(\log n) \) time per operation
⇒ same is true for dynamic range sum data structures