15-210
Parallel and Sequential Algorithms and Data Structures

Lecture 18

Quicksort Analysis and Sorting Lower Bounds

Synopsis

- Quicksort
- Work and Span Analysis of Randomized Quicksort
- Lower Bound for Comparison-based Sorting
- Lower Bound for Merging

Quicksort

- Originally invented and analyzed by Hoare in 1960's.
- I strongly urge to watch Jon Bentley on "Three beautiful Quicksorts" at
  - `www.youtube.com/watch?v=QvgYAQzg1z8`.

Sequential Quicksort

```
int i, j;
for( i = low, j = high - 1; ; )
{
    while( a[ ++i ] < pivot );
    while( pivot < a[ --j ] );
    if( i >= j )
    break;
    swap( a, i, j );
}
// Restore pivot
swap( a, i, high - 1 );
quicksort( a, low, i - 1 ); // Sort small elements
quicksort( a, i + 1, high ); // Sort large elements
```

Quicksort

```
1   fun quicksort(S) =
2     if |S| = 0 then S
3     else let
4           p = pick a pivot from S
5           S1 = ⟨ s ∈ S | s < p ⟩
6           S2 = ⟨ s ∈ S | s = p ⟩
7           S3 = ⟨ s ∈ S | s > p ⟩
8           (R1, R3) = ( quicksort(S1) || quicksort(S3) )
9       in
10        append(R1, append(S2, R3))
11      end
```

Quicksort

```
1   fun quicksort(S) =
2     if |S| = 0 then S
3     else let
4           p = pick a pivot from S
5           C = ⟨ s ∈ S | (s, compare(p, s) ⟩
6           S1 = ⟨ s | (s, LESS) ∈ C ⟩
7           S2 = ⟨ s | (s, EQUAL) ∈ C ⟩
8           S3 = ⟨ s | (s, GREATER) ∈ C ⟩
9           (R1, R3) = ( quicksort(S1) || quicksort(S3) )
10        in
11          append(R1, append(S2, R3))
12        end
```

Quicksort

- Each call to Quicksort either makes
  - No recursive calls (base case), or
  - Two recursive calls
- Call tree is a binary
- Depth the call tree determines the span of the algorithm.

Picking the Pivot

- Always pick the first element
  - Worst case $O(n^2)$ work.
  - In practice, almost sorted inputs are not uncommon.
- Pick the median of 3 elements (e.g., first, middle and last elements)
  - could possible divide evenly
  - worst case is still bad
- Pick an element at random
  - we hope this divides evenly in expectation
  - leading to expected $O(n \log n)$ work and $O(\log^2 n)$ span.

Picking the Pivot

- Pick first element
  - Worst case $O(n^2)$ work.
  - Expected $O(n \log n)$ work
    - ★ Averaged over all possible orderings.
  - Works well on the average
  - Slow on some, possibly common, cases.
- Pick a random element
  - Expected worst-case $O(n \log n)$ work.
    - ★ For input in **any** order, the expected work is $O(n \log n)$
  - No input has expected $O(n^2)$ work.
  - With a small probability, we could be unlucky and have $O(n^2)$ work.

Randomized Quicksort

- Assign a uniformly random priority to each number in $[0, 1]$.

```
1  fun quicksort(S) =
2    if |S| = 0 then S
3    else let
4          p = pick as pivot the highest priority element from S
5          S1 = ⟨ s ∈ S | s < p ⟩
6          S2 = ⟨ s ∈ S | s = p ⟩
7          S3 = ⟨ s ∈ S | s > p ⟩
8          (R1, R3) = ( quicksort(S1) || quicksort(S3) )
9        in
10         append(R1, append(S2, R3))
11       end
```

- Once the priorities are assigned, the algorithm is deterministic.

Randomized Quicksort

- Count comparisons made!
  - Almost all the work is comparisons.

$$X_n = \#\text{ of comparisons } \texttt{quicksort}$$

makes on input of size n

- Find $\mathbf{E}[X_n]$ for any input sequence S
- Notation:
  - Let $T = \texttt{sort}(S)$
  - T_i and T_j refer to elements in the final sorted order and $i < j$ and $T_i \leq T_j$.
  - p_i refers to priority chosen for T_i.
  - $A_{i,j} = 1$ if T_i and T_j were ever compared during the sort.

Analyzing Quicksort

- Crucial point is how to model $A_{i,j}$.
- In any one call to `quicksort`, there are three cases:
  - Pivot p is either T_i or $T_j \Rightarrow A_{i,j} = 1$
  - $T_i < p < T_j \Rightarrow T_i \in S_1, T_j \in S_3, A_{i,j} = 0$
  - Either $p < T_i$ or $p > T_j \Rightarrow T_i, T_j \in S_1$ or $T_i, T_j \in S_3$
- If two elements are compared in a `quicksort` call, they will never be compared again in any other call!

Analyzing Quicksort

$$X_n \leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} A_{ij}$$

- The non-optimized code compares each element to pivot 3 times, the optimized version compares once.

```
1   ...
2          C = ⟨ s ∈ S | (s, compare(p, s)) ⟩
3   ...
```

- By linearity of expectation

$$\mathbf{E}[X_n] \leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbf{E}[A_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbf{Pr}[A_{ij} = 1]$$

Analyzing Quicksort

- Consider first when the pivot is one of $T_i, T_{i+1}, \ldots, T_j$
- T_i and T_j are compared $\Leftrightarrow$ p_i or p_j is the highest priority among $\{p_i, p_{i+1}, \ldots, p_j\}$.
  - Assume $T_k, i < k < j$ has higher priority.
  - For any subdivision $\cdots, T_i, \cdots, T_k, \cdots, T_j$, T_k will become a pivot and separate T_i and T_j
  - T_i and T_j will never be compared!

Analyzing Quicksort

$$\begin{aligned}\mathbf{E}[A_{ij}] &= \mathbf{Pr}[A_{ij} = 1] \\ &= \mathbf{Pr}[p_i \text{ or } p_j \text{ is the maximum in } \{p_i, \ldots, p_j\}] \\ &= \frac{2}{j - i + 1} \text{ (Why ?)}\end{aligned}$$

- $j - i + 1$ elements between p_i and p_j and each is equally likely to be the maximum.
- We want either p_i or p_j, hence $\frac{2}{j-i+1}$
- T_i is compared to T_{i+1} with probability 1.

Analyzing Quicksort

$$\mathbf{E}[X_n] \leq \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbf{E}[A_{ij}]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k} \quad \text{(change variables)}$$

$$\leq 2 \sum_{i=1}^{n} H_n$$

$$\leq 2 \cdot n \cdot H_n \in O(n \log n)$$

Analyzing Quicksort

- Indirectly, average work for basic deterministic quicksort is $O(n \log n)$.
  - Just shuffle data randomly and apply the basic algorithm
  - $\equiv$ to picking random priorities

Alternative Analysis

- Write a recurrence for the number of comparisons:

$$X(n) = X(Y_n) + X(n - Y_n - 1) + n - 1$$

- Random variable Y_n is the size of S_1.

$$\begin{aligned}
\mathbf{E}[X(n)] &= \mathbf{E}[X(Y_n) + X(n - Y_n - 1) + n - 1] \\
&= \mathbf{E}[X(Y_n)] + \mathbf{E}[X(n - Y_n - 1)] + n - 1 \\
&= \frac{1}{n}\sum_{i=0}^{n-1}(\mathbf{E}[X(i)] + \mathbf{E}[X(n - i - 1)]) + n - 1
\end{aligned}$$

Alternative Analysis

$$\begin{aligned}\mathbf{E}[X(n)] &= \frac{1}{n}\sum_{i=0}^{n-1}(\mathbf{E}[X(i)] + \mathbf{E}[X(n-i-1)]) + n - 1 \\ &= \frac{2}{n}\sum_{i=0}^{n-1}\mathbf{E}[X(i)] + n - 1\end{aligned}$$

- With telescoping, this also solves as $O(n \log n)$

Expected Span

- S is split into $L(ess)$, $E(qual)$ and $(g)R(eater)$.
- Let $X_n = \max\{|L|, |R|\}$,
- We use `filter` to partition.

$$S(n) = S(X_n) + O(\log n)$$

Expected Span

- Let $\overline{S}(n)$ denote $\mathbf{E}[S(n)]$
- We bound $\overline{S}(n)$ by considering $\mathbf{Pr}[X_n \leq 3n/4]$ and $\mathbf{Pr}[X_n > 3n/4]$.
- $\mathbf{Pr}[X_n \leq 3n/4] = 1/2$
  - As with `SmallestK`, $1/2$ of the randomly chosen pivots results in larger partition of at most size $3n/4$ elements.

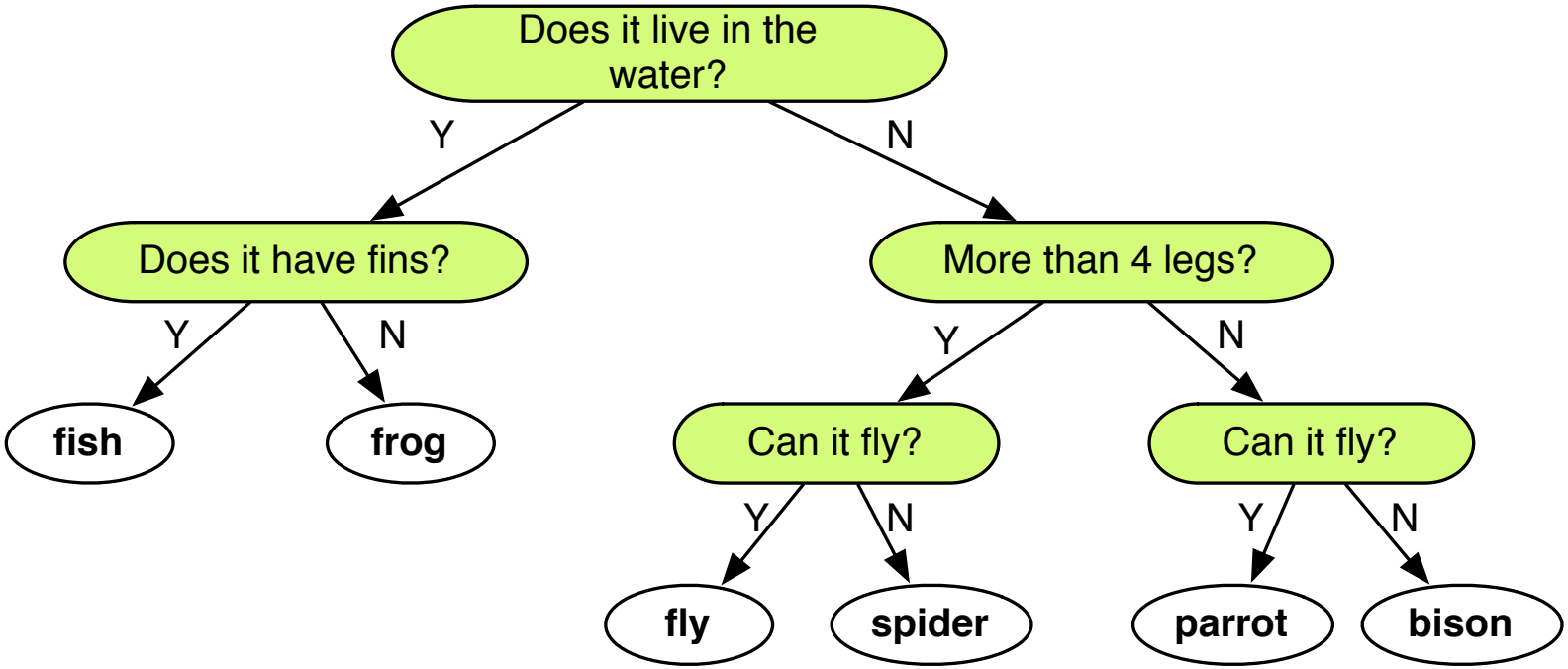
Expected Span

$$
\begin{aligned}
\overline{S}(n) &= \sum_i \mathbf{Pr}[X_n = i] \cdot \overline{S}(i) + c \log n \\
&\leq \mathbf{Pr}[X_n \leq \tfrac{3n}{4}]\overline{S}(\tfrac{3n}{4}) + \mathbf{Pr}[X_n > \tfrac{3n}{4}]\overline{S}(n) + c \cdot \log n \\
&\leq \tfrac{1}{2}\overline{S}(\tfrac{3n}{4}) + \tfrac{1}{2}\overline{S}(n) + c \cdot \log n \\
&\Longrightarrow (1 - \tfrac{1}{2})\overline{S}(n) \leq \tfrac{1}{2}\overline{S}(\tfrac{3n}{4}) + c \log n \\
&\Longrightarrow \overline{S}(n) \leq \overline{S}(\tfrac{3n}{4}) + 2c \log n \\
&\Longrightarrow \overline{S}(n) \in O(\log^2 n)
\end{aligned}
$$

Lower Bound for Sorting

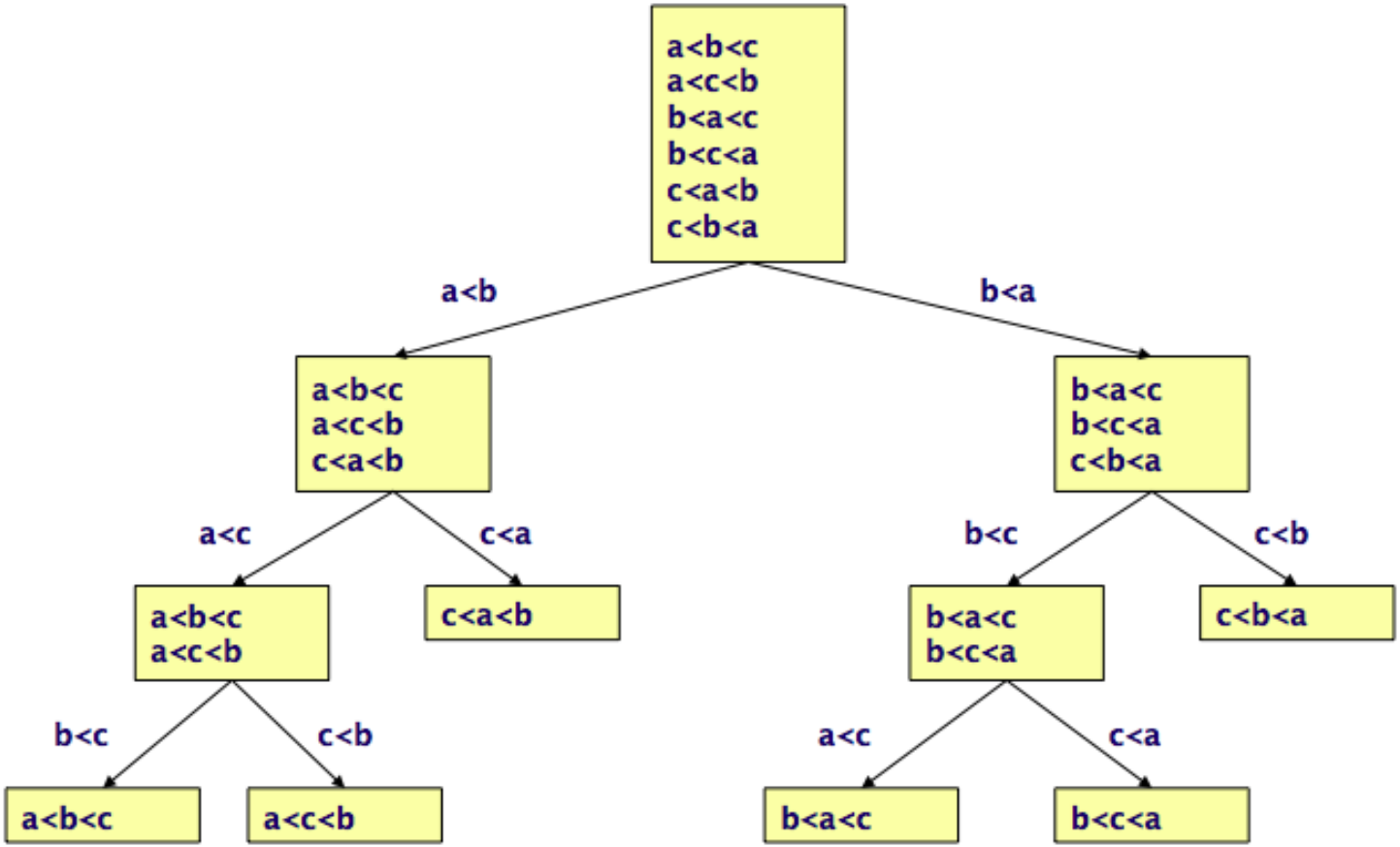
- What is asymptotically the minimum number comparisons any sorting algorithm has to make?
- Lower-bounds apply to problems not to algorithms.
  - Algorithms provide upper bounds!
- We say sorting is $\Omega(n \log n)$
- No (comparison-based) sorting algorithm has work asymptotically lower than $n \log n$.

Decision Trees



- If there are N outcomes, the number of questions is at least $\log_2 N$.

Sorting as a Decision Problem



- For n items, how many possible outcomes can there be?
  - $n! \Rightarrow$ we need at least $\log_2(n!)$ "questions".

Sorting as a Decision Problem

$$\begin{aligned}\log(n!) &= \log n + \log(n-1) + \cdots + \log(n/2) + \cdots + \log 1 \\ &\geq \log n + \log(n-1) + \cdots + \log(n/2) \\ &\geq \tfrac{n}{2} \cdot \log(n/2) \in \Omega(n \log n)\end{aligned}$$

Lower bound for Merging

- We have sorted sequences $A, |A| = n$ and $B, |B| = m$ and $m \leq n$.
  - Assume all elements are unique.
- All interleavings are possible
- We need to choose m positions out of $n + m$ to place the elements of B amongst elements of A.
- Finding the right sequence of m positions can be done with at least $\log_2 \binom{n+m}{m}$ comparisons.

Lower bound for Merging

- $\binom{n}{r} \geq \left(\frac{n}{r}\right)^r$
  - See Lemma in the notes.

$$\log_2 \binom{n+m}{m} \geq \log_2 \left(\frac{n+m}{m}\right)^m = m \log_2 \left(1 + \frac{n}{m}\right)$$