CS 4102: Algorithms
Lecture 11: Sorting Algorithms
David Wu
Fall 2020
Homework

Hashing/Sorting Exercise 1 due **September 30**
- Written (use LaTeX!)
- Randomized algorithms
- Hash functions

Hashing/Sorting Exercise 2 due October 5
- Written (use LaTeX!)
- Sorting and selection algorithms
Randomized Quicksort

**Divide:** Select a **random** pivot, and **partition** about the pivot

```
2  5  1  3  6  4  7  8  10  9  11  12
```

**Conquer:** Recursively sort left and right sublists

```
2  1  3  5  6  4  7  8  9  10  11  12
```

**Expected running time:** $\Theta(n \log n)$
Formal Argument for $n \log n$ Average

We will focus on counting the number of \textbf{comparisons}.

\textbf{For simplicity:} suppose all elements are \textbf{distinct}.

Quicksort only compares against a \textbf{pivot}.

- Element $i$ only compared to element $j$ if one of them was the pivot.
Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?

Consider the sorted version of the list

**Observation:** Adjacent elements must be compared

– **Why?** Otherwise we would not know their order

– **Every** sorting algorithm **must** compare adjacent elements

**In quicksort:** adjacent elements **always** end up in same sublist, unless one is the pivot
Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?

Consider the sorted version of the list

Pr[we compare 1 and 12] = $\frac{2}{12}$

Assuming pivot is chosen uniformly at random

Elements only compared if 1 or 12 was chosen as the first pivot since otherwise they are in different sublists
Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?

Case 1: Pivot less than $i$
Then sublist $[i, i + 1, \ldots, j]$ will be in right sublist and will be processed in future invocation of Quicksort

\[
\text{Pr[we compare } i \text{ and } j] = \text{Pr[we compare } i \text{ and } j \text{ in Quicksort([} p + 1, \ldots, n \text{])}}
\]
What is the probability of comparing two given elements?

Case 1: Pivot less than $i$
Then sublist $[i, i + 1, ..., j]$ will be processed in future invocation of Quicksort.

$$\Pr \text{[we compare } i \text{ and } j] = \Pr \text{[we compare } i \text{ and } j \text{ in Quicksort([}p + 1, ..., n\text{])}$$
What is the probability of comparing two given elements?

**Case 2:** **Pivot** greater than **j**

Then sublist \([i, i + 1, \ldots, j]\) will be in left sublist and will be processed in future invocation of Quicksort

\[
\Pr[\text{we compare } i \text{ and } j] = \Pr[\text{we compare } i \text{ and } j \text{ in Quicksort}([1, \ldots, p - 1])]\
\]
Case 3.1: Pivot contained in \([i+1, \ldots, j-1]\)

Then \(i\) and \(j\) are in different sublists and will never be compared

\[
\Pr[\text{we compare } i \text{ and } j] = 0
\]
Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
</table>

$i$        

$j$

Case 3.2: Pivot is either $i$ or $j$

Then we will **always** compare $i$ and $j$

$$\text{Pr}[\text{we compare } i \text{ and } j] = 1$$
Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td>j</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Case 1:** Pivot less than $i$

$$Pr[\text{we compare } i \text{ and } j] = Pr[\text{we compare } i \text{ and } j \text{ in Quicksort}([p + 1, \ldots, n])]$$

**Case 2:** Pivot greater than $j$

$$Pr[\text{we compare } i \text{ and } j] = Pr[\text{we compare } i \text{ and } j \text{ in Quicksort}([1, \ldots, p - 1])]$$

**Case 3:** Pivot in $[i, i + 1, \ldots, j]$

$$Pr[\text{we compare } i \text{ and } j] = Pr[i \text{ or } j \text{ is selected as pivot}] = \frac{2}{j - i + 1}$$
Formal Argument for $n \log n$ Average

Probability of comparing element $i$ with element $j$:

$$\Pr[\text{we compare } i \text{ and } j] = \frac{2}{j - i + 1}$$

Let $X_{ij}$ be an indicator random variable for the event that we compare element $i$ with element $j$

Let $X$ be a random variable for the total number of comparisons

$$\mathbb{E}[X] = \mathbb{E} \left[ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1}$$
Formal Argument for $n \log n$ Average

$$
\mathbb{E}[X] = \mathbb{E}
\left[
\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}
\right]
= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}[X_{ij}]
= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1}
$$

Expected number of comparisons:

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

Substitution: $k = j - i$
Formal Argument for \( n \log n \) Average

\[
\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k + 1} < 2 \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{1}{k} < 2 \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{1}{k}
\]

Substitution:
\[ k = j - i \]

\[
\frac{1}{k + 1} < \frac{1}{k}
\]

Useful fact:
\[
\sum_{k=1}^{n} \frac{1}{k} = \Theta(\log n)
\]

Intuition (not proof!):
\[
\sum_{k=1}^{n} \frac{1}{k} \approx \int_{1}^{n} \frac{1}{x} \, dx = \ln n
\]

Formal Argument for $n \log n$ Average

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

$$
= 2 \sum_{i=1}^{n-1} O(\log n) = O(n \log n)
$$

Useful fact: $\sum_{k=1}^{n} \frac{1}{k} = \Theta(\log n)$
Sorting Algorithms

Sorting algorithms we have discussed:
• Mergesort \( O(n \log n) \)
• Quicksort \( O(n \log n) \)

Other sorting algorithms (will discuss):
• Bubble sort \( O(n^2) \)
• Insertion sort \( O(n^2) \)
• Heapsort \( O(n \log n) \)

Can we do better than \( O(n \log n) \)?
Prove that there is no algorithm which can sort faster than $O(n \log n)$

Non-existence proof!

• Very hard to do

We will show such a lower bound for comparison sorts

Algorithm that only assumes elements can be compared (nothing about representation of the elements)
Strategy: Decision Tree

Comparison sorts use **comparisons** to determine ordering

**Strategy:** Draw tree to illustrate all possible execution paths

How do we measure running time?

Permutation of original list

[1,2,3,4,5]  [2,1,3,4,5]  ...  [5,2,4,1,3]  ...  [5,4,3,2,1]
Strategy: Decision Tree

Worst case running time is the longest execution path (measures number of comparisons) – this is the height of the decision tree.

Permutation of original list:
- [1,2,3,4,5]
- [2,1,3,4,5]
- ...
- [5,2,4,1,3]
- ...
- [5,4,3,2,1]
Strategy: Decision Tree

Worst case running time is the longest execution path (measures number of comparisons) – this is the height of the decision tree

\[
\log(n!) \quad \Omega(n \log n)
\]

Possible execution path

One comparison

Result of comparison

[1,2,3,4,5] [2,1,3,4,5] … [5,2,4,1,3] … [5,4,3,2,1]

How many such permutations do we need?

Permutation of original list

n! possible permutations
**Conclusion**: Running time of any comparison sort is $\Omega(n \log n)$
Sorting Algorithms

Sorting algorithms we have discussed:

• Mergesort  \( O(n \log n) \)  Optimal!
• Quicksort  \( O(n \log n) \)  Optimal!

Other sorting algorithms (will discuss):

• Bubble sort  \( O(n^2) \)
• Insertion sort  \( O(n^2) \)
• Heapsort  \( O(n \log n) \)  Optimal!

Can we do better than  \( O(n \log n) \)?

Not with comparison sorts...
Speed Isn’t Everything

Important properties of sorting algorithms:

**Run Time**
- Asymptotic Complexity
- Constants

**In Place**
- Only requires **constant** additional space

**Adaptive**
- Faster if list is nearly sorted

**Stable**
- Equal elements remain in original order

**Parallelizable**
- Runs faster with many processors

*Relaxed definition:* only need to copy a constant number of elements
Merge Sort

**Divide:**
- Break \( n \)-element list into two lists of \( n/2 \) elements

**Conquer:**
- If \( n > 1 \): Sort each sublist *recursively*
- If \( n = 1 \): List is already sorted (**base case**)

**Combine:**
- Merge together sorted sublists into one sorted list

**Run Time?**
\( O(n \log n) \)
Optimal!

**In Place?**  No

**Adaptive?**  No

**Stable?**  Yes*

*Technically: depends on how merge is implemented
**Combine:** Merge sorted sublists into one sorted list

We have:
- 2 sorted lists \( (L_1, L_2) \)
- 1 output list \( L_{\text{out}} \)

While \( (L_1 \text{ and } L_2 \text{ not empty}) \):

\[
\text{If } L_1[0] \leq L_2[0]: \\
L_{\text{out}}.\text{append}(L_1.\text{pop()}) \\
\text{Else:} \\
L_{\text{out}}.\text{append}(L_2.\text{pop()})
\]

\( L_{\text{out}}.\text{append}(L_1) \)
\( L_{\text{out}}.\text{append}(L_2) \)

**Stable:**
If elements are equal, leftmost comes first
## Merge Sort

### Divide:
- Break \( n \)-element list into two lists of \( n/2 \) elements

### Conquer:
- If \( n > 1 \): Sort each sublist recursively
- If \( n = 1 \): List is already sorted (base case)

### Combine:
- Merge together sorted sublists into one sorted list

### Run Time?
\( \mathcal{O}(n \log n) \)

Optimal!

<table>
<thead>
<tr>
<th>In Place?</th>
<th>Adaptive?</th>
<th>Stable?</th>
<th>Parallelizable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>No</td>
<td>Yes*</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Merge Sort

**Divide:**
- Break $n$-element list into two lists of $n/2$ elements

**Conquer:**
- If $n > 1$:
  - Sort each sublist recursively
- If $n = 1$:
  - List is already sorted (base case)

**Combine:**
- Merge together sorted sublists into one sorted list

**Parallelizable:**
Allow different processors to sort each sublist
Merge Sort (Sequential)

\[ T(n) = 2T\left(\frac{n}{2}\right) + n \]

Run Time: \( O(n \log n) \)
Merge Sort (Parallel)

$$T(n) = T(n/2) + n$$

Done in parallel

Run Time: $O(\log n)$