CSCI 241

Lecture 6
Radix Sort
A bit of runtime, if time allows
Announcements

- Quiz 1 graded and available on Gradescope
- A1 is due Monday night
- If gradle test hangs, you probably have an infinite loop.
- You can run a single test with e.g.:
  - gradle test --tests SortsTest.test00Insertion
Goals:

• Be prepared to implement radix sort.

• Understand how to go from operation count to a Big-O runtime class.

• Be able to count primitive operations in the non-recursive sorting algorithms we’ve covered so far.
Comparison sorts operate by comparing pairs of elements.

How do you sort without comparing elements?

Suppose I gave you 10 sticky notes with the digits 0 through 9.

What algorithm would you use to sort them?

How many times did you need to look at each sticky note?

What if there are duplicates?
Refresher: Stacks and Queues

(Q1: What is printed?)

Stack s;
Queue q;

for i in 1..5:
    s.add(i) // push
    q.add(i) // enqueue

for i in 1..5:
    print s.remove() // pop
    print q.remove() // dequeue

(LIFO)  (FIFO)
Stability

Objects can be sorted on keys - different objects may have the same value.

- e.g., sorting on 10’s place only

A stable sort maintains the order of distinct elements with the same key.
Exercise - Q2:

• Sort the following array **stably** on the 1’s digit:

  [7, 19, 61, 11, 14, 54, 1, 8]
Exercise - Q2:

- Sort the following array **stably** on the 1’s digit:

  \[7, 19, 61, 11, 14, 54, 1, 8\]
LSD Radix Sort

/** least significant digit radix sort A */
LSDRadixSort(A):
max_digits = max # digits in any element of A
for d in 0..max_digits:
    do a stable sort of A on the dth least significant digit

// A is now sorted(!)
LSD Radix Sort

/** least significant digit radix sort A */
LSDRadixSort(A):
max_digits = max # digits in any element of A
for d in 0..max_digits:
    do a stable sort of A on the dth least significant digit

// A is now sorted(!)

Don’t believe me? https://visualgo.net/en/sorting
LSD Radix Sort
using queue buckets

Pseudocode from visualgo.net:

LSDRadixSort(A):
create 10 buckets (queues) for each digit (0 to 9)
for each digit (least- to most-significant):
  for each element in A:
    move element into its bucket based on digit
  for each bucket, starting from smallest digit
    while bucket is non-empty
      restore element to list

LSD Intuition: sort on most-significant digit last; if tied, yield to the next most significant digit, and so on. Only works because stability preserves orderings from less significant (previously sorted) digits.
LSDRadixSort(A):
create 10 buckets (queues) for each digit (0 to 9)
for each digit (least- to most-significant):
    for each element in A:
        move element into its bucket based on digit
    for each bucket, starting from smallest digit
        while bucket is non-empty
            restore element to list
Exercise - Q3: Radix sort this

[07, 19, 61, 11, 14, 54, 01, 08]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>Buckets on 1’s place:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted on 1’s place:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>Buckets on 10’s place:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted on 10’s place:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Exercise: Radix sort this**

[07, 19, 61, 11, 14, 54, 01, 08]

<table>
<thead>
<tr>
<th>Buckets on 1’s place:</th>
<th>0</th>
<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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>01</td>
<td>11</td>
<td>61</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54</td>
<td>14</td>
<td>07</td>
<td>08</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted on 1’s place:</td>
<td>61</td>
<td>11</td>
<td>01</td>
<td>14</td>
<td>54</td>
<td>07</td>
<td>08</td>
<td>19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buckets on 10’s place:</th>
<th>0</th>
<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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>08</td>
<td>19</td>
<td>07</td>
<td>14</td>
<td>01</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>54</td>
<td>61</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sorted on 10’s place:</td>
<td>01</td>
<td>07</td>
<td>08</td>
<td>11</td>
<td>14</td>
<td>19</td>
<td>54</td>
<td>61</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**LSD Radix Sort using counting sort**

```plaintext
/** least significant digit radix sort A */
LSDRadixSort(A):
max_digits = max # digits in any element of A
for d in 0..max_digits:
    counting sort A on the dth least significant digit

// A is now sorted(!)
```
Counting Sort

Formalizes what you did with the 1-9 sticky notes:

• Handles duplicates
• Stable sort

Intuition:

http://www.cs.miami.edu/home/burt/learning/Csc517.091/workbook/countingsort.html

Pseudocode in CLRS (and reproduced on the next slide).
Counting Sort - from CLRS

**Notes:**
- \( k \) is the base or radix (10 in our examples)
- \( B \) is filled with the sorted values from \( A \).
- \( C \) maintains counts for each bucket.
- The final loop **must** go back-to-front to guarantee stability.

```
COUNTING-SORT(A, B, k)
1 let C[0..k] be a new array
2 for i = 0 to k
3     C[i] = 0
4 for j = 1 to A.length
5     C[A[j]] = C[A[j]] + 1
6     // C[i] now contains the number of elements equal to i.
7 for i = 1 to k
8     C[i] = C[i] + C[i - 1]
9     // C[i] now contains the number of elements less than or equal to i.
10 for j = A.length downto 1
12     C[A[j]] = C[A[j]] - 1
```
Runtime Analysis: Overview

• Why? We want a measure of performance that is

  • Independent of what computer we run it on.
    Solution: count operations instead of clock time.

  • Dependence on problem size is made explicit.
    Solution: express runtime as a function of \( n \) (or whatever variables define problem size)

  • Simpler than a raw count of operations and focuses on performance on large problem sizes.
    Solution: ignore constants, analyze asymptotic runtime.
Runtime Analysis: Overview

• How?

1. Count the number of primitive (constant-time) operations that occur over the entire execution of the algorithm.

2. Drop constants and lower-order terms to find the asymptotic runtime class.
Comparing findMaxes

- \( \text{findMax: } 8N - 5 \)
- \( \text{sillyFindMax: } 2 + 5N + 6N^2 \)

Drop constants and lower-order terms to find the \textbf{asymptotic runtime class}. 

\text{findMin is } O(n) \\
\text{sillyFindMin is } O(n^2)
From operation count to Big-O runtime class

- findMax: $8N - 5$  
  findMax is $O(n)$

- sillyFindMax: $2 + 5N + 6N^2$  
  sillyFindMax is $O(n^2)$

How do we get from $8N-5$ to $O(N)$?

1. Drop all constant factors.

   $8N - 5$ becomes $N - 1$
From operation count to Big-O runtime class

• findMax: $8N - 5$  
  findMax is $O(n)$

• sillyFindMax: $2 + 5N + 6N^2$  
  sillyFindMax is $O(n^2)$

How do we get from $2 + 5N + 6N^2$ to $O(N^2)$?

1. Drop all constant factors.
   
   $8N - 5$ becomes $N - 1$

2. Drop all but the most-significant term
   
   $N - 1$ becomes $O(N)$
From operation count to Big-O runtime class

• findMax: 8N - 5 \hspace{1cm} \text{findMax is O(n)}

• sillyFindMax: 2 + 5N + 6N^2 \hspace{1cm} \text{sillyFindMax is O(n^2)}

How do we get from $2 + 5N + 6N^2$ to $O(N^2)$?

1. Drop all constant factors.

   $2 + 5N + 6N^2$ becomes $N + N^2$

2. Drop all but the most-significant term

   $N + N^2$ becomes $O(N^2)$
From operation count to Big-O runtime class

• findMax: $8N - 5$  \hspace{1cm} \text{findMax is } O(n)

• sillyFindMax: $2 + 5N + 6N^2$ \hspace{1cm} \text{sillyFindMax is } O(n^2)

What if the count is 8?

1. Drop all constant factor factors.
   
   8 becomes 1

2. Drop all but the most-significant term

   1 becomes $O(1) = \text{“constant time”}$
From operation count to Big-O runtime class

- findMax: $8N - 5$ 
  findMax is $O(n)$
- sillyFindMax: $2 + 5N + 6N^2$ 
  sillyFindMax is $O(n^2)$

What if the count is 800?

1. Drop all constant factor factors.
   
   $800$ becomes 1
2. Drop all but the most-significant term
   
   1 becomes $O(1)$ = “constant time”
Counting Operations

What’s a constant-time operation?

• Anything that doesn’t depend on the input size:
  • Reading/writing from/to a variable or array location.
  • Evaluating an arithmetic or boolean expression.
  • Returning from a method.
Counting Operations

What’s a constant-time operation?

• Anything that doesn’t depend on the input size:
  • Reading/writing from/to a variable or array location.
    \[
    \text{int } i = 2; \text{ int } b = 4; \text{ a}[i] = b;
    \]
  • Evaluating an arithmetic or boolean expression.
    \[
    \text{int } i = 0; \text{ int } j = i+4; \text{ int } k = i*j;
    \]
  • Returning from a method. \text{return } k;

Key intuition:
• These don’t take identical amounts of time, but the times are within a constant factor of each other.
• Same for running the same operation on a different computer.