Last Time

- Recurrence Relations
Today

- Sorting in Linear Time
Sorting Algorithms we’ve seen so far

- Insertion Sort - \( \Theta(n^2) \)
- Quick Sort - \( O(n \log n) \)
- Merge Sort - \( O(n \log n) \)
Comparison Sorting - Sorting based on comparison of two elements $A[i]$ and $A[j]$

- For this discussion, we will use only $A[i] \leq A[j]$.

Is it possible to sort faster than $\Theta(n \log n)$?
Sorting faster than $\Theta(n \log n)$

- Comparison Sorting - Sorting based on comparison of two elements $A[i]$ and $A[j]$
  - For this discussion, we will use only $A[i] \leq A[j]$.

- Is it possible to sort faster than $\Theta(n \log n)$?
  - Spoiler. Yes (sometimes).
Lower bound on runtime of Comparison Sorting

- Decisions of a comparison sort can be viewed as a tree.

- nodes - comparisons, leaves - permutations of a
- # comparisons = length of the path
The runtime of any comparison algorithm is $\Omega(h)$, where $h$ is the height of the comparison tree.

Any sorting algorithm must be able to construct any permutation of $A$.

How many permutations are there of $A$?
The runtime of any comparison algorithm is $\Omega(h)$, where $h$ is the height of the comparison tree.

Any sorting algorithm must be able to construct any permutation of $A$.

How many permutations are there of $A$?

- $n = \text{size}(A)$, $n!$
Lower bound on runtime of Comparison Sort

- We have a (binary) comparison tree of height $h$, with $l$ leaves.
- Each permutation of $A$ must be reachable, so $n! \leq l$.
- Also, a binary tree can have at most $2^h$ leaves, so $l \leq 2^h$
- $2^h \geq n!$
- $\log 2^h \geq \log n!$
- $h \geq \log n!$
- $h = \Theta(n \log n)$
- Since $h$ is the maximum number of comparisons, any Comparison Sort is $\Omega(n \log n)$
Aside

Proof.

Show that $\log n! = \Theta(n \log n)$.

$$n! \approx \sqrt{2\pi n} \left( \frac{n}{e} \right)^n$$

$$\log n! \approx \log \sqrt{2\pi n} \left( \frac{n}{e} \right)^n$$

$$\approx \log \sqrt{2\pi n} + \log \left( \frac{n}{e} \right)^n$$

$$\approx \log \sqrt{2\pi n} + n \log \left( \frac{n}{e} \right)$$

$$\approx \log \left( (2\pi n)^{\frac{1}{2}} \right) + n(\log n - \log e)$$

$$\approx \frac{1}{2} \log 2\pi + \frac{1}{2} \log n + n \log n - n \log e$$

$$= \Theta(1) + \Theta(\log n) + \Theta(n \log n) + \Theta(n)$$

$$= \Theta(n \log n)$$
How can we sort faster than $\Theta(n \log n)$

- Comparison Sorting
  1. We never inspect the value of $A[i]$ – only compare to $A[j]$.
  2. The values of $A$ are unbounded.

- If we can limit the values of $A$, we can break the lower bound on sorting.
Comparison Sorting

1. We never inspect the value of \( A[i] \) – only compare to \( A[j] \)
2. The values of \( A \) are unbounded.

If we can limit the values of \( A \), we can break the lower bound on sorting.

Specifically, if \( 0 \leq A[i] < k \) for all \( A[i] \in A \) and \( k = O(n) \), we can sort in linear time.
For each input element, \( A[i] \) determine how many elements have a value less than \( A[i] \).

Use a \( k \)-element array to count the number of elements with each of the \( k \) possible values.

Reconstruct the sorted array from these counts.
Counting Sort

- For each input element, $A[i]$ determine how many elements have a value less than $A[i]$.
- Use a $k$-element array to count the number of elements with each of the $k$ possible values.
- Reconstruct the sorted array from these counts.
- Note: only works if $k$ is asymptotically smaller than $n$.
- I.e. Small vocabulary, many tokens.
- A lot of repetition in $A$. 
Counting Sort Example

- Initialize $C$
- $A = [2, 5, 3, 0, 2, 3, 0, 3]$
- $B = [x, x, x, x, x, x, x, x]$
- $C = [0, 0, 0, 0, 0, 0]$
Counting Sort Example

- Count the number of elements at each value and store in $C$
- $A = [2, 5, 3, 0, 2, 3, 0, 3]$
- $B = [x, x, x, x, x, x, x, x]$
- $C = [2, 0, 2, 3, 0, 1]$
Counting Sort Example

- Change the counts in $C$ to an index in the sorted array.
- $A = [2, 5, 3, 0, 2, 3, 0, 3]$
- $B = [x, x, x, x, x, x, x, x]$
- $C = [2, 2, 4, 7, 7, 8]$
Counting Sort Example

- Use $C$ to construct the sorted array, $B$.
- $A = [2, 5, 3, 0, 2, 3, 0, 3]$ 
- $B = [x, x, x, x, x, x, 3, x]$ 
- $C = [2, 2, 4, 6, 7, 8]$
Counting Sort Example

- Use C to construct the sorted array, B.
- \( A = [2, 5, 3, 0, 2, 3, 0, 3] \)
- \( B = [x, 0, x, x, x, x, 3, x] \)
- \( C = [1, 2, 4, 6, 7, 8] \)
Use $C$ to construct the sorted array, $B$.

$A = [2, 5, 3, 0, 2, 3, 0, 3]$

$B = [x, 0, x, x, x, 3, 3, x]$

$C = [1, 2, 4, 5, 7, 8]$
Counting Sort Example

- Use $C$ to construct the sorted array, $B$.
- $A = [2, 5, 3, 0, 2, 3, 0, 3]$
- $B = [x, 0, x, 2, x, 3, 3, x]$
- $C = [1, 2, 3, 5, 7, 8]$
Use $C$ to construct the sorted array, $B$.

- $A = [2, 5, 3, 0, 2, 3, 0, 3]$
- $B = [0, 0, x, 2, x, 3, 3, x]$
- $C = [0, 2, 3, 5, 7, 8]$
Counting Sort Example

- Use $C$ to construct the sorted array, $B$.
- $A = [2, 5, 3, 0, 2, 3, 0, 3]$
- $B = [0, 0, x, 2, 3, 3, 3, x]$
- $C = [0, 2, 3, 4, 7, 8]$
Counting Sort Example

- Use C to construct the sorted array, B.
- \( A = [2, 5, 3, 0, 2, 3, 0, 3] \)
- \( B = [0, 0, x, 2, 3, 3, 3, 5] \)
- \( C = [0, 2, 3, 4, 7, 7] \)
Use \( C \) to construct the sorted array, \( B \).

\[
A = [2, 5, 3, 0, 2, 3, 0, 3]
\]

\[
B = [0, 0, 2, 2, 3, 3, 3, 5]
\]

\[
C = [0, 2, 2, 4, 7, 7]
\]
Counting sort pseudocode

CountingSort(A,B,k)

    for i ← 0..k do
        C[i] ← 0
    end for

    for j ← 1..length(A) do
        C[A[j]] ← C[A[j]] + 1
    end for

    for i ← 1..k do
        C[i] ← C[i − 1] + C[i]
    end for

    for j ← length(A)..1 do
        B[C[A[j]]] ← A[j]
        C[A[j]] ← C[A[j]] − 1
    end for
Counting sort runtime

CountingSort(A,B,k)

for $i \leftarrow 0..k$ do
  $C[i] \leftarrow 0$
end for
for $j \leftarrow 1..\text{length}(A)$ do
  $C[A[j]] \leftarrow C[A[j]] + 1$
end for
for $i \leftarrow 1..k$ do
  $C[i] \leftarrow C[i - 1]$
end for
for $j \leftarrow \text{length}(A)..1$ do
  $B[C[A[j]]] \leftarrow A[j]$
  $C[A[j]] \leftarrow C[A[j]] - 1$
end for

Runs in $\Theta(k + n + k + n) = \Theta(n)$ since $k = O(n)$
Radix Sort

- Assume an alphabet of \( k \) elements as in CountingSort.
- Radix sort allows us to sort \( d \) length strings composed of these \( k \) elements in linear time, assuming \( d = O(n) \).
- For example: 3-digit numbers.
- 10-letter words.
- Sorting by multiple fields – date, price, inventory id.
Radix Sort Operation

radixSort(A, d)

for i ← 1..d do
    run a stable sort to sort A on digit i
end for
Stable Sorting

- Stable sort

5 3 2 3 4

2 3 3 4 5
Stable Sorting

- Unstable sort

5 3 2 3 4

2 3 3 4 5
Radix Sort Example

BEG

BAG

CAB

DOG

CAT
Radix Sort Example

BEG  CAB
BAG  DOG
CAB  BEG
DOG  BAG
CAT  CAT
Radix Sort Example

BEG  CAB  CAB
BAG  DOG  BAG
CAB  BEG  CAT
DOG  BAG  BEG
CAT  CAT  DOG
Radix Sort Example
Can we sort linearly with unbounded input?
If input falls within the range, \([0, 1)\), we can.
Can we sort linearly with unbounded input?
If input falls within the range, [0, 1), we can.
Or can be converted to this range.
Bucket Sort

- Given an input array, $A$, of length $n$.
- Divide the input range $[0, 1)$ into $n$ buckets each with width $1/n$.
  - The buckets are defined as $\left[ \frac{i}{n}, \frac{i+1}{n} \right)$ for $i \in 0..n-1$.
- For each element in $A$, examine its value, and put it in an appropriate “bucket”.
- Sort each bucket separately.
- Reconstruct the sorted array from the sorted buckets.
How does Bucket Sort work?

- Assume the elements of the array are distributed approximately evenly.
- We can expect each bucket to contain $1/n$ elements.
- Even $\Theta(n^2)$ sorting of $1/n$ elements has an expected runtime that is smaller than linear in $n$.
- In fact, $O(2 - 1/n)$.
- Derivation of this is in Section 8.4, though you are not responsible for it.
Bye

Next time (9/23)
- Binary Search Trees

For Next Class
- Homework 3 - Sorting - is due.
- Read Sections 12.1, 12.2, 12.3