Collections, Part Three
Outline for Today

- **Lexicon**
  - Storing a collection of words.

- **HashSet**
  - Storing a group of whatever you’d like.

- **HashMap**
  - A powerful, fundamental container.
Recap from Last Time
Parameter Flowchart

1. Pass by reference!
   - Yes!
     - Need to change the argument?
   - No!
     - Pass by value!
     - Primitive type!
       - What kind of argument?
     - Object!
       - Pass by const reference!

2. Start!
New Stuff!
Lexicon

• A **Lexicon** is a container that stores a collection of words.

• The **Lexicon** is designed to answer the following question efficiently:

  *Given a word, is it contained in the **Lexicon**?*

• The **Lexicon** does *not* support access by index. You can’t, for example, ask what the 137th English word is.

• However, it *does* support questions of the form “does this word exist?” or “do any words have this as a prefix?”
Tautonyms

• A *tautonym* is a word formed by repeating the same string twice.
  • For example: murmur, couscous, papa, etc.
• What English words are tautonyms?
Some Aa

http://upload.wikimedia.org/wikipedia/commons/f/f1/Aa_large.jpg
One Bulbul
More than One Caracara

http://www.greglasley.net/images/CO/Crested-Caracara-F3.jpg
Introducing the Dikdik
And a Music Recommendation
Time-Out for Announcements!
Assignment 2

- Assignment 2 (Fun with Collections) goes out today. It’s due next Friday.
  - Explore the impact of sea level rise.
  - Build a personality quiz!
- We’ve provided a suggested timetable for completing this assignment on the front page of the handout. Aim to stick to this timeline; you’ve got plenty of time to complete things if you start early.
- You must complete this assignment individually. Working in pairs is not permitted on this assignment.
LaIR Closure

• The LaIR will be closed on Sunday in observance of Dr. Martin Luther King, Jr. Day.

• The LaIR will, however, be open on Monday during the usual 7PM – 11PM time slot.
The CS department is putting on an undergraduate research panel next Wednesday at 5:30PM in Gates 415.

Curious to hear what it’s like to do research in CS? Stop on by! An RSVP is requested using this link.
HashSet
HashSet

- The **HashSet** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.
HashSet

• The **HashSet** represents an unordered collection of distinct elements.

• Elements can be added and removed, and you can check whether or not an element exists.
HashSet

- The **HashSet** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.
HashSet

- The **HashSet** represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.
HashSet

- The HashSet represents an unordered collection of distinct elements.
- Elements can be added and removed, and you can check whether or not an element exists.
Operations on HashSets

• You can add a value to a HashSet by writing
  \[
  \text{hashSet} += \text{value};
  \]

• You can remove a value from a HashSet by writing
  \[
  \text{hashSet} -= \text{value};
  \]

• You can check if a value exists in a HashSet by writing
  \[
  \text{hashSet}.contains(\text{value})
  \]

• Many more operations are available (union, intersection, difference, subset, etc.), so be sure to check the documentation.
HashMap
HashMap

- The HashMap class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.
The `HashMap` class represents a set of key/value pairs.

- Each key is associated with a unique value.
- Given a key, can look up the associated value.
HashMap

- The **HashMap** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CS106B</td>
<td>Hello!</td>
</tr>
<tr>
<td>Dikdik</td>
<td>Cute!</td>
</tr>
</tbody>
</table>
HashMap

- The **HashMap** class represents a set of key/value pairs.
- Each key is associated with a unique value.
- Given a key, can look up the associated value.

<table>
<thead>
<tr>
<th>CS106B</th>
<th>Hello!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dikdik</td>
<td>Cute!</td>
</tr>
<tr>
<td>This Slide</td>
<td>Self Referential</td>
</tr>
</tbody>
</table>
HashMap

• The **HashMap** class represents a set of key/value pairs.
• Each key is associated with a unique value.
• Given a key, can look up the associated value.

<table>
<thead>
<tr>
<th>CS106B</th>
<th>Hello!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dikdik</td>
<td>Very Cute!</td>
</tr>
<tr>
<td>This Slide</td>
<td>Self Referential</td>
</tr>
</tbody>
</table>
Using the Map

- You can create a map by writing
  
  `HashMap<KeyType, ValueType> hashMap;`
  
- You can add or change a key/value pair by writing
  
  `hashMap[key] = value;`
  
  If the key doesn't already exist, it is added.

- You can read the value associated with a key by writing
  
  `hashMap[key]`
  
  If the key doesn't exist, it is added and associated with a default value. (More on that later.)

- You can check whether a key exists by calling
  
  `hashMap.containsKey(key)`
Using the Map

You can create a map by writing

```java
HashMap<KeyType, ValueType> hashMap;
```

You can add or change a key/value pair by writing

```java
hashMap[key] = value;
```

If the key doesn't already exist, it is added.

- You can read the value associated with a key by writing

```java
hashMap[key]
```

If the key doesn't exist, it is added and associated with a default value. (More on that later.)

You can check whether a key exists by calling

```java
hashMap.containsKey(key)
```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
Map Autoinsertion

```cpp
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```
HashMap<string, int> freqMap;

while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap
Map Autoinsertion

HashMap&lt;string, int&gt; freqMap;
while (true) {
  string text = getline("Enter some text: ");
  cout &lt;&lt; "Times seen: " &lt;&lt; freqMap[text] &lt;&lt; endl;
  freqMap[text]++;
}

freqMap
Map Autoinsertion

```cpp
HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```
Map Autoinsertion

```cpp
HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```
Map Autoinsertion

```
HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

```
"Hello"
```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap

Oh no! I don't know what that is!
Map Autoinsertion

HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

Let's pretend I already had that key here.
Map Autoinsertion

HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

The values are all ints, so I'll pick zero.
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

Phew! Crisis averted!
HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
Map Autoinsertion

```cpp
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

- **"Hello"**: 0

```cpp
 freqMap["Hello"]++;
```
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

"Hello" 0

Cool as a cucumber.

C(⊂(▀¯▀⊂)
Map Autoinsertion

```cpp
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

Cool as a cucumber.
HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap

"Hello" 1

text

"Hello"
HashMap<string, int> freqMap;

while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap = {
    "Hello" : 1
}
Map Autoinsertion

```cpp
HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

freqMap

```
"Hello" 1
```
Map Autoinsertion

```cpp
HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

```
freqMap {
    "Hello" 1
}
```
HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap = {
    "Hello": 1
}

text = "Goodbye"
HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap {
    "Hello" 1
}

text "Goodbye"
Map Autoinsertion

```cpp
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

Oh no, not again!
Map Autoinsertion

HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap

"Hello" 1
"Goodbye" 0

I'll pretend I already had that key.
Map Autoinsertion

```cpp
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

 freqMap

```
<table>
<thead>
<tr>
<th>Text</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello</td>
<td>1</td>
</tr>
<tr>
<td>Goodbye</td>
<td>0</td>
</tr>
</tbody>
</table>
```

"Goodbye"
Map Autoinsertion

HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap

<table>
<thead>
<tr>
<th>text</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Hello&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;Goodbye&quot;</td>
<td>0</td>
</tr>
</tbody>
</table>
Map Autoinsertion

```cpp
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

Chillin' like a villain.

$c(\text{[\hspace{2pt}][\hspace{2pt}]}\hspace{2pt}c)$
Map Autoinsertion

```cpp
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

Chillin' like a villain.

$c(\bullet\bullet\bullet\bullet\bullet)$
Map Autoinsertion

```cpp
HashMap<string, int> freqMap;
while (true) {
    string text = getLine("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}
```

(freqMap

- "Hello" 1
- "Goodbye" 1)
HashMap<string, int> freqMap;
while (true) {
    string text = getline("Enter some text: ");
    cout << "Times seen: " << freqMap[text] << endl;
    freqMap[text]++;
}

freqMap = {
    "Hello" : 1,
    "Goodbye": 1
}
Sorting by First Letters
Map Autoinsertion

Lexicon english("EnglishWords.txt");

HashMap<
char,
Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
Lexicon english("EnglishWords.txt");

HashMap<
  char,
  Lexicon>
  wordsByFirstLetter;

for (string word: english) {
  wordsByFirstLetter[word[0]].add(word);
}
Map Autoinsertion

Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter
Map Autoinsertion

Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;

for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

Map Autoinsertion

```java
Lexicon english("EnglishWords.txt");

HashMap<Character, Lexicon> wordsByFirstLetter;
for (string word : english) {
    wordsByFirstLetter[word[0]].add(word);
}
```

`wordsByFirstLetter`
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

word "first"
Map Autoinsertion

Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

word  "first"
Map Autoinsertion

Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word : english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

Oops, no f's here.
Map Autoinsertion

Lexicon english("EnglishWords.txt");

HashMap<
  char,
  Lexicon>
  wordsByFirstLetter;

for (string word: english) {
  wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

'f'

word "first"

Let's insert that key.
Map Autoinsertion

```java
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}
```

I'll give you a blank Lexicon.
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter
{
    'f'
    {
    }
}

word "first"
Map Autoinsertion

Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

{ "first" }

"first"
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
  wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

'f' { "first" }

word "first"
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter['f'] = { "first" }
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;

for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

\{'f'\: \{ "first" \} }
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;

for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter['f'] = { "first" }

"foremost"
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter
    'f' { "first" }

word "foremost"
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

'f' {
    "first"
}

"foremost"
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter['f'] { "first" }

"foremost"

Easy peasy.

Easy peasy.
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

\{ 'f' \}

\{ "first" \}

word

"foremost"
Lexicon english("EnglishWords.txt");

HashMap<
  char,
  Lexicon
> wordsByFirstLetter;

for (string word: english) {
  wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter
{
  'f'
  { "first", "foremost" }
}

word
"foremost"
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter['f'] { "first", "foremost" }
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter
{ 'f' : { "first", "foremost" } }
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

\{ 'f' : { "first", "foremost" } \}
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;

for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter['f'] = { "first", "foremost" }

word = "initial"
Map Autoinsertion

Lexicon english("EnglishWords.txt");

HashMap<\texttt{char}, \texttt{Lexicon}> wordsByFirstLetter;
for (\texttt{string} word : english) {
    wordsByFirstLetter[word[0]].add(word);
}

\begin{verbatim}
wordsByFirstLetter
\end{verbatim}

\begin{verbatim}
'f' { "first", "foremost" }
\end{verbatim}

\begin{verbatim}
word "initial"
\end{verbatim}
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

<table>
<thead>
<tr>
<th>'f'</th>
<th>{ &quot;first&quot;, &quot;foremost&quot; }</th>
</tr>
</thead>
<tbody>
<tr>
<td>'i'</td>
<td>{}</td>
</tr>
</tbody>
</table>

word "initial"
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

wordsByFirstLetter

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>'f'</td>
<td>{ &quot;first&quot;, &quot;foremost&quot; }</td>
</tr>
<tr>
<td>'i'</td>
<td>{}</td>
</tr>
</tbody>
</table>

word
"initial"
Lexicon english("EnglishWords.txt");

HashMap<char, Lexicon> wordsByFirstLetter;
for (string word: english) {
    wordsByFirstLetter[word[0]].add(word);
}

- wordsByFirstLetter
  - 'f': { "first", "foremost" }
  - 'i': { "initial" }

word: "initial"
Quokka
Quokka Quincunx
Quarter Quokka Quincunx
Anagrams
Anagrams

• Two words are **anagrams** of one another if the letters in one can be rearranged into the other.

• Some examples:
  • “Praising” and “aspiring.”
  • “Arrogant” and “tarragon.”

• **Question for you:** does this concept exist in other languages? If so, please send me examples!
Anagrams

- **Nifty fact:** two words are anagrams if you get the same string when you write the letters in those words in sorted order.
- For example, “praising” and “aspiring” are anagrams because, in both cases, you get the string “aiignprs” if you sort the letters.
Anagram Clusters

• Let’s group all words in English into “clusters” of words that are all anagrams of one another.

• We’ll use a HashMap<string, Lexicon>.
  • Each key is a string of letters in sorted order.
  • Each value is the collection of English words that have those letters in that order.
Your Action Items

• **Read Chapter 5.**
  • It’s all about container types, and it’ll fill in any remaining gaps from this week.

• **Start Assignment 2.**
  • Make slow and steady progress here, if you can. Aim to complete Rising Tides and to have started You Got Hufflepuff!

• **Enjoy MLK Weekend**
  • Read “Letter from Birmingham City Jail.” Like, seriously.
  • Watch the actual “I Have a Dream” speech. It’s truly amazing.
  • Watch “The Two Americas,” a speech MLK delivered here on Stanford campus.
Next Time

- **Thinking Recursively**
  - How can you best solve problems using recursion?
  - What techniques are necessary to do so?
  - And what problems yield easily to a recursive solution?
Appendix: How to Sort a String
Counting Sort
Counting Sort

banana

b a n a n a
Counting Sort

banana

letterFreq
Counting Sort

for (char ch: input) {
    letterFreq[ch]++;
}

letterFreq
Counting Sort

```
for (char ch: input) {
    letterFreq[ch]++;
}
```
Counting Sort

```java
for (char ch: input) {
    letterFreq[ch]++;
}
```
Counting Sort

`banana` for `char ch: input` {
    letterFreq[ch]++;
}

letterFreq

for (char ch: input) {
    letterFreq[ch]++;
}
Counting Sort

for (char ch: input) {
    letterFreq[ch]++;
}

letterFreq

banana

a 1
b 1
Counting Sort

for (char ch: input) {
    letterFreq[ch]++;
}

letterFreq
Counting Sort

for (char ch: input) {
    letterFreq[ch]++;
}

letterFreq

banana
Counting Sort

for (char ch: input) {
    letterFreq[ch]++;
}

letterFreq

banana

<table>
<thead>
<tr>
<th>a</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
</tr>
</tbody>
</table>
Counting Sort

for (char ch: input) {
    letterFreq[ch]++;
}

letterFreq = 
\begin{array}{c|c|c|c|c|c}
  a & b & n & a & n & a \\
  \hline
  2 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
Counting Sort

for (char ch: input) {
    letterFreq[ch]++;
}

letterFreq

banana

| a | 2 |
| b | 1 |
| n | 1 |
Counting Sort

```
for (char ch: input) {
    letterFreq[ch]++;
}
```
Counting Sort

```java
for (char ch: input) {
    letterFreq[ch]++;
}
```
Counting Sort

```
for (char ch: input) {
    letterFreq[ch]++;
}
```
Counting Sort

banana

letterFreq

| a | 3 |
| b | 1 |
| n | 2 |
Counting Sort

for (char ch = 'a'; ch <= 'z'; ch++) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
Counting Sort

for (char ch = 'a'; ch <= 'z'; ch++) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
Counting Sort

for (char ch = 'a'; ch <= 'z'; ch++) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
Counting Sort

for (char ch = 'a'; ch <= 'z'; ch++) {
  for (int i = 0; i < letterFreq[ch]; i++) {
    result += ch;
  }
}

letterFreq

banana

<table>
<thead>
<tr>
<th>letterFreq</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>n</td>
</tr>
</tbody>
</table>

a a a a a
Counting Sort

banana

for (char ch = 'a'; ch <= 'z'; ch++) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}

letterFreq

<table>
<thead>
<tr>
<th>a</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
</tr>
</tbody>
</table>

aaaab
Counting Sort

for (char ch = 'a'; ch <= 'z'; ch++) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}

letterFreq
Counting Sort

```java
for (char ch = 'a'; ch <= 'z'; ch++) {
    for (int i = 0; i < letterFreq[ch]; i++) {
        result += ch;
    }
}
```
Counting Sort

banana

letterFreq

<table>
<thead>
<tr>
<th>a</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
</tr>
</tbody>
</table>

aaaabbbbnnn