Sorted by Frequency, Web Backend Application

—

Andrew Gordon

What is sorting by frequency?

Assume you have some text and want to know what words show up the most (alphabetic order on ties). We can sort by:

1. Words with the most occurrences (largest at the top)

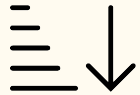
2. Words with the same amount of occurrences show up in alphabetical order (‘a’ higher than ‘I’ if tied)

Ways to sort by frequency...

- Count number of occurrences of each word, use built in sorting tools to order output
- Binary Tree
- Min/Max Heap
- Find *correct* place in the list to put words as they come in

Ways to sort by frequency…

- **Count number of occurrences for each word, use built in sorting tools to order output**



Pros	Cons
Can utilize built in tools	Costly if words added one by one
Ease of retrieving and passing the sorted list	Two rounds of sorting (freq, alphabetically)

Example Input

"this is a test, stay calm it is just a test"

Count occurrences

1 1 1 1 1 1 1 2 1 2 2

this is a test, stay calm it is just a test

List words and frequencies

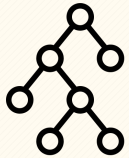
this: 1, is: 2, a: 2, test: 2, stay: 1, calm: 1, it: 1, just: 1

Sort by freq, sort alphabetically on ties

a: 2, is: 2, test: 2, calm: 1, it: 1, just: 1, stay: 1, this: 1

Ways to sort by frequency...

- **Binary Tree**
  - Each entry is a *leaf* on the tree
  - Each *leaf* to the left of *root* is less than *root*, each *leaf* to the right of *root* is greater than *root*



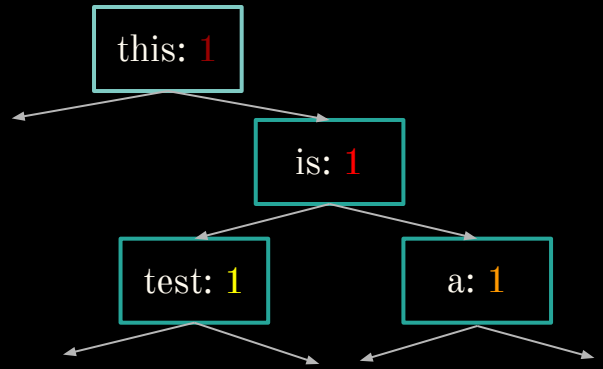
Pros	Cons
Insertions do not require full sort	Tree can become unbalanced
Ease of retrieving ordered list (inorder traversal)	Custom implementation of binary tree functions

Example Input

"this is a test, stay calm it is just a test"

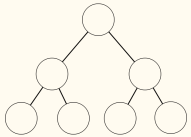
Insert each word one by one. When a duplicate is found, add to its frequency, remove then re-insert it to tree.

[this, is, a, test, ...]



Ways to sort by frequency…

- **Min/Max Heap**
  - Min/max entry exists at the top node
  - All node's *children* guaranteed to be lesser



Pros	Cons
Insertions do not require full sort	Finding element in heap costs O(n), build new heap each time
Heaps are maximally balanced binary trees	Heaps not ordered, pop all elements to view list

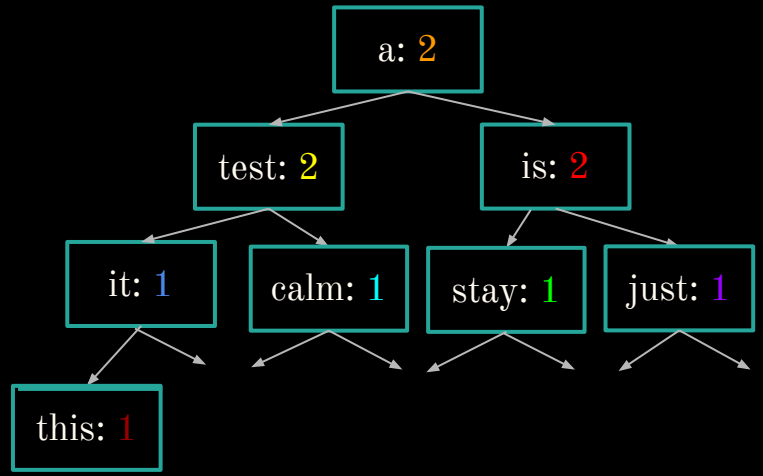
Example Input

"this is a test, stay calm it is just a test"

Generate set of words and frequencies

this: 1, is: 2, a: 2, test: 2, stay: 1, calm: 1, it: 1, just: 1

Build heap from these elements (max heap below)



Ways to sort by frequency…

- **Find *correct* place in the list to put words as they come in**
  - Build sorted version of list one by one
  - Maintain dictionary of previously seen words and their frequencies

Pros	Cons
Insertions do not require full sort and have strict time bound	Requires custom implementation of binary search
Ease of retrieving and passing the sorted list	Requires maintenance of dictionary with list

Example Input

"this is a test, stay calm it is just a test"

Process words one by one. Find correct place for each word to reside as they come in.

Let's say we've processed example input up to the last word. Here is how we would handle the second 'test'.

this: 1, test: 1, stay: 1, just: 1, it: 1, calm: 1, is: 2, a: 2

binarySearch(list, [test: 1]) = 1, remove it from list

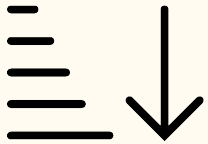
this: 1, stay: 1, just: 1, it: 1, calm: 1, is: 2, a: 2

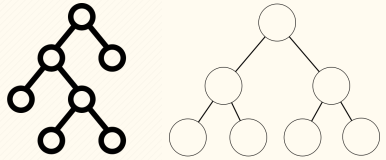
bisect(list, [test: 2]) = 5, insert between calm and is

this: 1, stay: 1, just: 1, it: 1, calm: 1, test: 1, is: 2, a: 2

Runtime of approaches, N = number of words

- Count number of occurrences of each word, use built in sorting tools to order output
  - Naive: **O(N*NlogN)** time, **O(N)** space
- Binary Tree and Min/Max Heap
  - **O(NlogN)** time, **O(N)** space
- Find *correct* place in the list to put words as they come in
  - **O(NlogN)** time, **O(N)** space

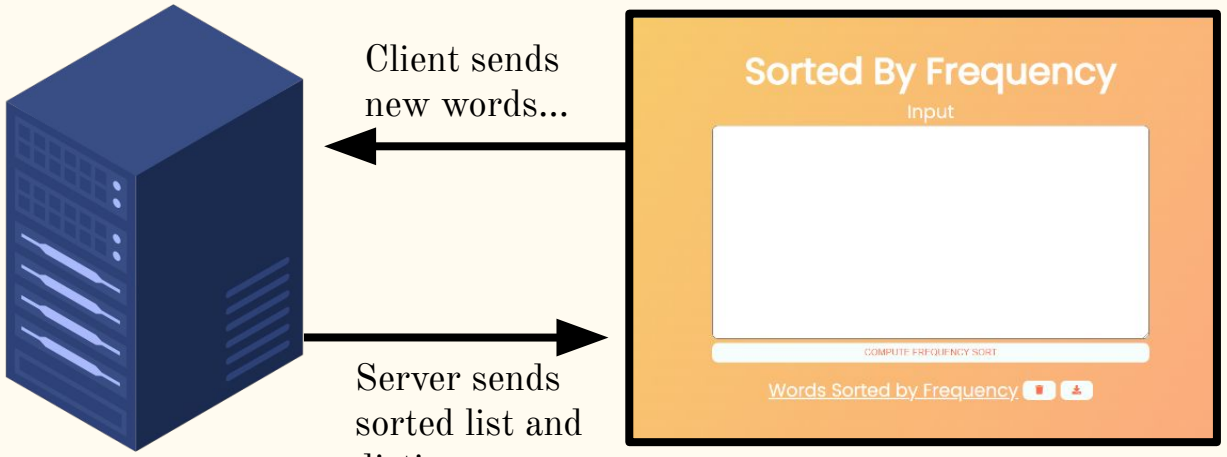




System

REST API

- **/api/processWords**
  - POST
  - Request body
    - newWords
    - wordsList (optional)
    - wordsListDict (optional)
  - Successful Response
    - wordsList
    - wordsListDict
  - Bad Response
    - Bad request
      - No new words or
      - New words is not array of strings

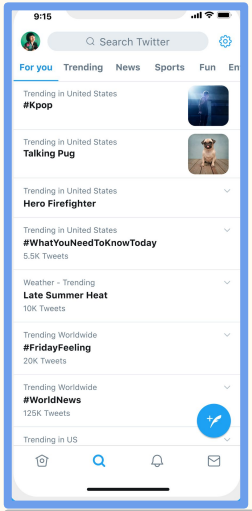


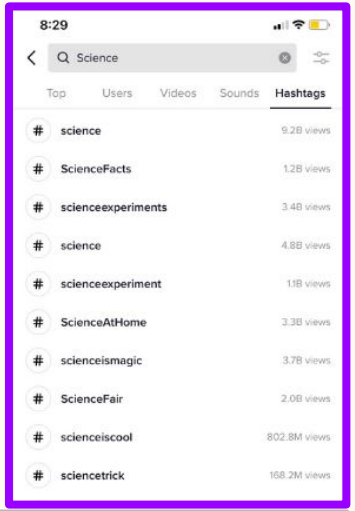
Server
Express, Node.js

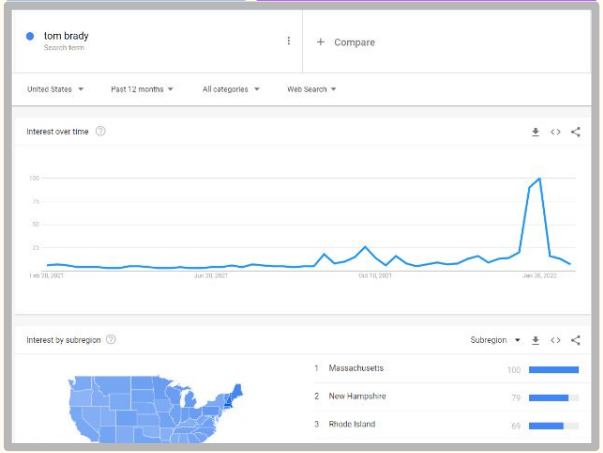
Client
React, JavaScript

Use cases

- Tracking trends in social media
  - Maintain list of popular topics/tags
  - Track emerging words or slang (e.g. Twitter, TikTok comments...)
  - Google Trends, frequency of search terms over time
- Preparing data/training for Machine Learning
  - Compute word lists sorted by frequency for various texts, feed/train machine learning model to guess what topic the text is about based on word frequency
  - Train machine learning model to predict next word in sentence
- Tracking other frequencies
  - Taking sensor data and tracking the frequency of values over time







Demo

https://sorting-by-frequency.herokuapp.com/

Code repository

https://github.com/andrew-d-gordon/sorting_by_frequency