

Drawing 1: Show the progression of one sort as it sorts and processes duplicates

WordRec

Input File

token
count

```
Whale Pony Cat
Cat
Bear Dog
Pony Llama Pony
```

Data: WordRec wordlist[5];

First, read until capacity

wordList					numWords: 5
0	1	2	3	4	
Whale	Pony	Cat	Cat	Bear	
1	1	1	1	1	

Sort:

spot: 0

idxMin: 4

Swap Whale with Bear

wordList					numWords: 5
Whale	Bear	Pony	Cat	Cat	Bear Whale
1	1	1	1	1	1

Bear is settled.

spot: 1

idxMin: 2

Swap first Cat and Pony (Cat goes next)

wordList					numWords: 5
0	1	2	3	4	
Bear	Cat	Pony	Cat	Whale	
1	1	1	1	1	

Cat in index 3 is next. But, it is a duplicate

spot: 2

idxMin: 3

This is determined by $wordlist[idxMin] == wordlist[spot-1]$. This must be checked **before** checking $wordList[idxMin]$ against $wordList[spot]$.

Swap the duplicate with the last in-use element (Whale in index $numWords-1$), then decrement $numWords$. Finally, increment the duplicate word's counter.

wordList					numWords: 5
Bear	Cat	Pony	Cat-Whale	Whale Cat	
1	2	1	1	1	

In Use
Not In Use

Remains

spot: 2

idxMin: 2

$wordList[idxMin] != wordList[spot-1]$, so no duplicate. $idxMin == spot$, so no swap. Pony stays.

wordList					numWords: 4
Bear	Cat	Pony	Whale	Cat	
1	2	1	1	1	

In Use
Not In Use

Spot becomes 3, which is items -1. The loop is done. At this point, all but the last element has been processed, which means the list is ordered. Check for last element duplicate finds none. This sort is done. What would be the final result after processing this entire data file?

This Example Demonstrates a Special Case of the Duplicate at the End

WordRec

Input File

token
count

Whale Pony Cat
Bear
Pony Dog
Pony Llama Pony

Data: WordRec wordlist[5];
 First, read until capacity

wordList					numWords: 5
0	1	2	3	4	
Whale	Pony	Cat	Bear	Pony	
1	1	1	1	1	

Sort:

spot: 0

idxMin: 3

Swap Whale with Bear

wordList					numWords: 5
Whale	Bear	Pony	Cat	Bear	Whale
1	1	1	1	1	1

Bear is settled.

spot: 1

idxMin: 2

Swap Cat and Pony (Cat goes next)

wordList					numWords: 5
Bear	Cat	Pony	Whale	Pony	
1	1	1	1	1	

No swap. Pony (1st one) is in place

spot: 2

idxMin: 2

wordList					numWords: 5
Bear	Cat	Pony	Whale	Pony	
1	1	1	1	1	

spot: 3

idxMin: 4

Next item is Pony. But $wordlist[idxMin] == wordlist[spot-1]$ → we have a duplicate
 Rather than swap, we increment $list[spot-1]$'s count, keep spot at 3, increment $list[spot-1].count$,
 and decrement numWords

wordList					numWords: 4
Bear	Cat	Pony	Whale	Pony	
1	1	2	1	1	

In Use
Not In Use

Spot is now 3, which is equal to items -1. The loop is done. At this point, all but the last element has been processed, which means the list is ordered.

You must add in a check if the last item is a duplicate (see next example).

This Example Demonstrates Another Special Case

WordRec

Input File – Updated

token
count

Horse Pony Cat
Pony
Bear Dog
Pony Llama

Data: WordRec wordlist[5];

First, read until capacity

wordList					numWords: 5
0	1	2	3	4	
Horse	Pony	Cat	Pony	Bear	
1	1	1	1	1	

Sort:

Swap Horse with Bear

wordList					numWords: 5
Hor	Bear	Pony	Cat	Pony	BearHorse
1	1	1	1	1	

Bear is settled.

Swap Cat and Pony (Cat goes next)

wordList					numWords: 5
Bear	Cat	Pony	Pony	Horse	
1	1	1	1	1	

Swap Horse and Pony

wordList					numWords: 5
Bear	Cat	Horse	Pony	Pony	
1	1	1	1	1	

No swap. Pony (1st one) is in place

wordList					numWords: 5
Bear	Cat	Horse	Pony	Pony	
1	1	1	1	1	

*At this point, the list is **definitely** ordered. You must add in a check if the last item is a duplicate.*

Pony is a duplicate (it is the same as the item before it). Increment Pony's count & decrement numWords

wordList					numWords: 4
Bear	Cat	Horse	Pony	Pony	
1	1	1	2	1	

In Use
Not In Use

Drawing 2: Show the Progression of Processing an Entire Data File

Read first five words

wordList					numWords: 5
0	1	2	3	4	
Horse	Pony	Bear	Pony	Bear	
1	1	1	1	1	

Horse Pony Bear
Pony
Bear Pony
Dog Cat Llama
Giraffe Horse Llama

1st Sort: Note that the two duplicates were found at index numWords-1; must check for that

wordList					numWords: 3
0	1	2	3	4	
Bear	Horse	Pony	Pony	Bear	
2	1	2	1	1	
In Use			Not In Use		

Read until capacity. Pony goes in index 3. Dog goes in index 4. They overwrote the previous contents;
 Note: Pony overwrote Pony! numWords is back to CAPACITY

wordList					numWords: 5
0	1	2	3	4	
Bear	Horse	Pony	Pony	Dog	
2	1	2	1	1	

Next sort/process duplicates. Only pony is a duplicate.

wordList					numWords: 4
0	1	2	3	4	
Bear	Dog	Horse	Pony	Pony	
2	1	1	3	1	

Again, read until capacity. Cat got read.

wordList					numWords: 5
0	1	2	3	4	
Bear	Dog	Horse	Pony	Cat	
2	1	1	3	1	

Sort. No duplicates. numWords==CAPACITY after sort means no new words will be added.

wordList					numWords: 5
0	1	2	3	4	
Bear	Cat	Dog	Horse	Pony	
2	1	1	1	3	

Rest of file is read. Only duplicate of word already in array is Horse. Llama and Giraffe can't be added.

wordList					numWords: 5
0	1	2	3	4	
Bear	Cat	Dog	Horse	Pony	
2	1	1	2	3	

CSc 136

Project 2

Drawings

One more: What if the array doesn't end up full? What if the file ends first?

Read first five words

wordList			numWords: 5	
0	1	2	3	4
Horse	Pony	Bear	Pony	Bear
1	1	1	1	1

Horse Pony Bear
Pony
Bear
Horse Pony Bear

1st Sort: Duplicates handled

wordList			numWords: 3	
0	1	2	3	4
Bear	Horse	Pony	Pony	Bear
2	1	2	1	1
In Use			Not In Use	

Read next two words

wordList			numWords: 5	
0	1	2	3	4
Bear	Horse	Pony	Horse	Pony
2	1	2	1	1
In Use				

Both values read are duplicates. Why do Horse and Pony change places in the unused section?

wordList			numWords: 3	
0	1	2	3	4
Bear	Horse	Pony	Pony	Horse
2	2	3	1	1
In Use			Not In Use	

Read in final word, Bear

wordList			numWords: 4	
0	1	2	3	4
Bear	Horse	Pony	Bear	Horse
2	2	3	1	1
In Use			Not In Use	

File is done. Since file ended first, instead of processing remaining words in file, run sort/process one last time.

Bear is found when you check after the normal sort, for a duplicate with the last element. Decrement numWords and increment Bear's counter.

wordList			numWords: 3	
0	1	2	3	4
Bear	Horse	Pony	Bear	Horse
3	2	3	1	1
In Use			Not In Use	