CSM 61B
Spring 2017

Counting Sorts & Tries

Week 14: April 25, 2017

1 Berkeley Bytes Buffet

You are the proud owner of Berkeley Bytes Buffet and business is good! You have a policy where children under 8 eat free and seniors eat 50 percent off. Since you're a savvy business owner, you keep the ages of all your customers.

1.1 Now, for taxes, you need to submit a list of the ages of your customers in sorted order. Define a procedure, ageSort, which takes an `int[] array` of all customers' ages and returns a sorted array. Assume customers are less than 150 years old.

```java
public class BerkeleyBytes {
    private static int maxAge = 150;
    public static int[] histogram(int[] ages) {
        int[] ageCounts = new int[maxAge];
        for (int age : ages) {
            ageCounts[age - 1] += 1;
        }
        return ageCounts;
    }
    public static int[] ageSort(int[] ages) {
        int[] ageCounts = histogram(ages);
        int[] result = new int[ages.length];
        int index = 0;
        for (int age = 0; age < maxAge; age++) {
            for (int count = 0; count < ageCounts[age]; count++) {
                result[index] = age + 1;
                index += 1;
            }
        }
        return result;
    }
}
```

1.2 Time passes and your restaurant is doing well. Unfortunately, our robot overlords have advanced medicine to the point where humans have become immortal.

(a) How could we extend the above algorithm to accept a list of arbitrary ages?

Radix sort. Sort the customers using the above algorithm, looking at only the last digit of their age. We would need 10 buckets, since the digit can only have 10 values. Repeat with the second to last digit, and so on, until the first digit sorted.

(b) When would we be able to use this type of sort?

The keys we want to sort must have some **base** (or radix). The type of item must be some combination of symbols.

2 Getting to Know You

2.1 Run MSD and LSD radix sort on the following DNA sequence such that the output is sorted in alphabetical order ($A < C < G < T$).

Most-significant digit.

```
ACAG ACAG ACAG ACAG ACAA
CTAG ACAA ACAA ACAA ACAG
ACAA CTAG CCTC CCTC CCTC
TGAG CCTC CTAG CTAG CTAG
CCTC GAGT GAGT GAGT GAGT
GAGT TGAG TGAG TGAG TGAG
```

Least-significant digit.

```
ACAG ACAA ACAA GAGT ACAA
CTAG CCTC ACAG ACAA ACAG
ACAA ACAG CTAG ACAG CCTC
TGAG CTAG TGAG CCTC CTAG
CCTC TGAG GAGT TGAG GAGT
GAGT GAGT CCTC CTAG TGAG
```

3 More Asymptotics Potpourri

Algorithm	Best-case	Worst-case	Stable
Counting Sort	$\Theta(N+R)$	$\Theta(N+R)$	Yes
LSD Radix Sort	$\Theta(W(N+R))$	$\Theta(W(N+R))$	Yes
MSD Radix Sort	$\Theta(N+R)$	$\Theta(W(N+R))$	Depends

Where N is the length of the list, R is the size of the alphabet (radix), and W is the length of the longest word.

MSD radix sort can be made stable with additional space for a buffer.

4 New World Order

4.1 Given a list of words (possibly repeated), devise a scheme to efficiently return a list of all the words that start with a given prefix.

Put all the names into a trie, lookup the prefix in the trie, and iterate across all the children rooted at that node.

4.2 Given a dictionary of words, describe a procedure for checking if a new word can be created out of the concatenation of two words in the dictionary. For example, if our dictionary contains the words, `"news"`, `"paper"`, `"new"`, and `"ape"`, we should be able to discover the new word, `"newspaper"`.

We can put all of the words in the dictionary into a trie. Then we can check a prefix of the word and see if the remainder of the word is in the trie as well.

5 Sorting Mechanics *Extra for Experts*

5.1 Below, the **leftmost column** is an unsorted list of strings. The **rightmost column** gives the same strings in sorted order. Each of the remaining columns gives the contents of the list during some intermediate step of one of the algorithms listed below. Match each column with its corresponding algorithm.

· Merge sort · Quicksort · Heap sort · LSD radix sort · MSD radix sort

For quicksort, choose the topmost element as the pivot. Use the recursive (or top-down) implementation for merge sort.

	A	B	C	D	E	F	G
1	4873	1876	1874	1626	9573	2212	1626
2	1874	1874	1626	1874	7121	8917	1874
3	8917	2212	1876	1876	9132	7121	1876
4	1626	1626	1897	4873	6973	1626	1897
5	4982	3492	2212	4982	4982	9132	2212
6	9132	1897	3492	8917	8917	6152	3492
7	9573	4873	4873	9132	6152	4873	4873
8	1876	9573	4982	9573	1876	9573	4982
9	6973	6973	6973	1897	1626	6973	6152
10	1897	9132	6152	3492	1897	1874	6973
11	9587	9587	7121	6973	1874	1876	7121
12	3492	4982	8917	9587	3492	9877	8917
13	9877	9877	9132	2212	4873	4982	9132
14	2212	8917	9573	6152	2212	9587	9573
15	6152	6152	9587	7121	9587	3492	9587
16	7121	7121	9877	9877	9877	1897	9877

From left to right: unsorted list, quicksort, MSD radix sort, merge sort, heap sort, LSD radix sort, completely sorted.

MSD Look at the left-most digits. They should be sorted. Mark this immediately as MSD.

LSD One of the digits should be sorted. Start by looking at the right most digit of the remaining sorts. Then check the second from right digit of the remaining sorts and so on. As soon as you find one in which at least something is sorted, mark that as LSD.

Heap Max-oriented heap so check that the bottom is in sorted order and that the top element is the next max element.

Merge Realize that the first pass of merge sort fixes items in groups of 2. Identify the passes and look for sorted runs.

Quick Run quicksort using the pivot strategy outlined above. Look for partitions and check that 4873 is in its correct final position.