CMSC 351: RadixSort

Justin Wyss-Gallifent

October 25, 2023

1 What it Does

Consider the following three situations:

- We are sorting a list of n decimal numbers all between 000 and 999 inclusive.
- We are sorting a list of n binary numbers all between 00000000 and 11111111 inclusive.
- We are sorting a list of n 5-digit strings, each string is made up of the letters A through Z inclusive.

In each of these situations we are sorting a collection of items where each item is a string of either numbers or letters. Radix sort often works especially well for such situations.

2 How it Works

Radix Sort works as follows:

1. Stable sort the list by the least significant digit.
2. Stable sort the list by the next most significant digit.
3. Etc.

Let's see how this works with an example.

Example 2.1. For example consider the following list. Observe that we have preprended `0`s as necessary to equalize the lengths.

$$\texttt{170,045,075,090,802,024,002,066}$$

First we identify each right-most (least significant) digit:

$$\texttt{17}\underline{\texttt{0}}\texttt{, 04}\underline{\texttt{5}}\texttt{, 07}\underline{\texttt{5}}\texttt{, 09}\underline{\texttt{0}}\texttt{, 80}\underline{\texttt{2}}\texttt{, 02}\underline{\texttt{4}}\texttt{, 00}\underline{\texttt{2}}\texttt{, 06}\underline{\texttt{6}}$$

We sort the list according to that digit, preserving order within that group:

$$\texttt{17}\underline{\texttt{0}}\texttt{, 09}\underline{\texttt{0}}\texttt{, 80}\underline{\texttt{2}}\texttt{, 00}\underline{\texttt{2}}\texttt{, 02}\underline{\texttt{4}}\texttt{, 04}\underline{\texttt{5}}\texttt{, 07}\underline{\texttt{5}}\texttt{, 06}\underline{\texttt{6}}$$

Then we identify the next digit, moving leftwards:

$$\texttt{1}\underline{\texttt{7}}\texttt{0, 0}\underline{\texttt{9}}\texttt{0, 8}\underline{\texttt{0}}\texttt{2, 0}\underline{\texttt{0}}\texttt{2, 0}\underline{\texttt{2}}\texttt{4, 0}\underline{\texttt{4}}\texttt{5, 0}\underline{\texttt{7}}\texttt{5, 0}\underline{\texttt{6}}\texttt{6}$$

We sort the list according to that digit, preserving order within that group:

$$\texttt{8}\underline{\texttt{0}}\texttt{2, 0}\underline{\texttt{0}}\texttt{2, 0}\underline{\texttt{2}}\texttt{4, 0}\underline{\texttt{4}}\texttt{5, 0}\underline{\texttt{6}}\texttt{6, 1}\underline{\texttt{7}}\texttt{0, 0}\underline{\texttt{7}}\texttt{5, 0}\underline{\texttt{9}}\texttt{0}$$

Then we identify the next digit, moving leftwards:

$$\underline{\texttt{8}}\texttt{02, }\underline{\texttt{0}}\texttt{02, }\underline{\texttt{0}}\texttt{24, }\underline{\texttt{0}}\texttt{45, }\underline{\texttt{0}}\texttt{66, }\underline{\texttt{1}}\texttt{70, }\underline{\texttt{0}}\texttt{75, }\underline{\texttt{0}}\texttt{90}$$

We sort the list according to that digit, preserving order within that group:

$$\underline{\texttt{0}}\texttt{02, }\underline{\texttt{0}}\texttt{24, }\underline{\texttt{0}}\texttt{45, }\underline{\texttt{0}}\texttt{66, }\underline{\texttt{0}}\texttt{75, }\underline{\texttt{0}}\texttt{90, }\underline{\texttt{1}}\texttt{70, }\underline{\texttt{8}}\texttt{02,}$$

Now we are done.

3 Details

3.1 Some Variables

Our list has n items in it. Let d be the number of digits (or characters) in each item in the list and let b be the base, or number of different characters available.

Example 3.1. If we're sorting n decimal numbers between 000 and 999 inclusive then $d = 3$ and $b = 10$.

Example 3.2. If we're sorting n binary numbers between 00000000 and 11111111 inclusive then $d = 8$ and $b = 2$.

Example 3.3. If we're sorting n strings between $AAAAA$ and $ZZZZZ$ inclusive then $d = 5$ and $b = 26$. Here technically we're saying $A = 0$ and $Z = 25$.

3.2 Using Counting Sort for the Stable Sort

We commented early that we "Stable sort the list..." but we didn't say how so let's visit that now.

In essentially all cases the base b is independent of n and so we are effectively doing d separate stable sorts and for each stable sort we are sorting by a single digit or character in each item. Essentially that means for each stable sort we are treating each item as if it were just that digit or character with the other digits or characters just going along for the ride.

If b is the base then this means each stable sort is sorting n items where each item is essentially a number between 0 and $b-1$. Assuming b is independent of n then we might as well use Counting Sort to do the stable sort part.

If this is the case then each counting sort is $\Theta(n + (b - 1))$ and we are doing d of those so the time complexity is $\Theta(d(n + (b - 1)))$. Again assuming b is independent of n then this is $\Theta(dn)$.

Of course if d is independent of n then this becomes $\Theta(n)$.

Example 3.4. If we're sorting n decimal numbers between 000 and 999 inclusive then $d = 3$ and $b = 10$ and the time complexity is $\Theta(3(n + (10 - 1))) = \Theta(n)$.

Example 3.5. If we're sorting n binary numbers between 00000000 and 11111111 inclusive then $d = 8$ and $b = 2$ and the time complexity is $\Theta(8(n + (2 - 1))) = \Theta(n)$.

Example 3.6. If we're sorting n strings between $AAAAA$ and $ZZZZZ$ inclusive then $d = 5$ and $b = 26$. Here technically we're saying $A = 0$ and $Z = 25$ and the time complexity is $\Theta(5(n + (26 - 1))) = \Theta(n)$.

It might be tempting to conclude that we always get $\Theta(n)$ and this is true when both b and d are independent of n, but certainly we can fabricate ideas where this is not true.

It's possible that the number of digits d might change with n.

> **Example 3.7.** If we're sorting n numbers between 0 and $2^n - 1$ represented in binary then $d = n$ and $b = 2$ and the time complexity is $\Theta(n(n+(2-1))) = \Theta(n^2)$.

It's less likely that the base may change with n.

As far as auxiliary space, if we're using Counting Sort then that underlying counting sort is $\Theta(n+(b-1))$. We repeat this d times but this is in series so the memory of each sorting step is freed up before the next one and so th auxiliary space is $\Theta(n + (b - 1))$.

As far as stability, Radix Sort is stable.

As far as in-place, if we're using Counting Sort then the underlying sort itself is not in-place and so Radix Sort will not be.

3.3 Using Another Sort of Sort for the Stable Sort

Because b is essentially always fixed it makes sense to use Counting Sort for the underlying stable sort but this is not really mandatory. We could use any other stable sorting method such as Bubble Sort, Merge Sort, etc.

The main issue with this of course is that the underlying sort will not be best-case $\Theta(n)$ as counting sort is and this would affect Radix Sort.

> **Example 3.8.** If we're sorting n decimal numbers between 000 and 999 inclusive then $d = 3$ and $b = 10$. If we use Merge Sort as the underlying sort then each Merge Sort pass is $\Theta(n \lg n)$ and so the time complexity of Radix Sort would be $\Theta(d(n \lg n))$.

4 Thoughts, Problems, Ideas

1. Suppose M is a fixed positive integer and b is a fixed base. Suppose we have a list of length n which contains integers between 0 and M inclusive. Explain why, if RadixSort+CountingSort is used to sort the list, the time complexity is the same.

2. Suppose $b = 2$ is the fixed base. Suppose that we have a list of length n which contains integers between 0 and n inclusive. Show that the RadixSort+CountingSort time complexity will be $\Theta(n \lg n)$.

   Hint: How many digits are needed to represent the integers 0 through n inclusive in base 2? Does the fixed base matter?

3. Suppose that we have a list of length n which contains integers between 0 and $n-1$ inclusive. Show that the RadixSort+CountingSort time complexity can in fact be made to be $\Theta(n)$. Hint: What can we make b equal to?

4. Explain how RadixSort+CountingSort can sort n integers between 0 and $n^2 - 1$ inclusive in $\Theta(n)$ time with two iterations of CountingSort.

5. Explain how RadixSort+CountingSort can sort n integers between 0 and $n^3 - 1$ inclusive in $\Theta(n)$ time with three iterations of CountingSort.

6. Explain how RadixSort+CountingSort can sort n integers between 0 and $n^n - 1$ inclusive in $\Theta(n^2)$ time with n iterations of CountingSort.

7. Suppose that M is a fixed positive integer and suppose we have a list of length n which contains integers between 0 and M inclusive. Show that the RadixSort+CountingSort time complexity will be $\Theta\left(\frac{n \lg M}{\lg n}\right)$.

8. Radix sort is the method used for alphabetizing whereby the underlying sort simply sorts by character in some way. Don't worry about how this underlying sort works. Show how RadixSort works on the list of words:

   `ANTE,ANTI,AMEX,BITE,BARK,INTO,INIT,`
   `LARK,PARK,PACK,RITE,UNTO,UNIT,ZOOT`

   You only need to show the result of each RadixSort loop iteration.

9. Suppose a list A contains integers between 0 and 999 inclusive. For indices i, j we wish to compare $A[i]$ and $A[j]$ and return the minimum but we are not permitted to use any sort of comparison. Explain how we could use RadixSort to do this. What would the Θ time complexity be? Explain.

5 Python Test

We assume arrays are global and we use Counting Sort for the underlying sort. The Radix here is 10.

```
import random
# This unstable version of counting sort
# sorts by the digit passed to it.
# digit = 1,10,100,etc.
def countingsort(A,digit):
    n = len(A)
    ANEW = [0] * n
    C = [0] * 10
    for i in range(0,n):
        sdigit = int((A[i]/digit)%10)
        C[sdigit] = C[sdigit] + 1
    for i in range(1,10):
        C[i] = C[i] + C[i-1]
    for i in range(n-1,-1,-1):
        sdigit = int((A[i]/digit)%10)
        ANEW[C[sdigit]-1] = A[i]
        C[sdigit] = C[sdigit] - 1
    for i in range(0,n):
        A[i] = ANEW[i]
# The radixsort function sorts by increasing digit.
def radixsort(A):
    maxval = max(A)
    d = 1
    while d < maxval:
        print('Sorting by radix: '+str(d))
        countingsort(A,d)
        print('Result: '+str(A))
        d = 10*d
A = []
for i in range(0,15):
    A.append(random.randint(0,1000))
n = len(A)
print(A)
radixsort(A)
print(A)
```

Output:

```
[91, 546, 43, 88, 954, 731, 975, 285, 737, 519, 830, 686, 744, 637, 322]
Sorting by radix: 1
Result: [830, 91, 731, 322, 43, 954, 744, 975, 285, 546, 686, 737, 637, 88,
Sorting by radix: 10
Result: [519, 322, 830, 731, 737, 637, 43, 744, 546, 954, 975, 285, 686, 88,
Sorting by radix: 100
Result: [43, 88, 91, 285, 322, 519, 546, 637, 686, 731, 737, 744, 830, 954,
[43, 88, 91, 285, 322, 519, 546, 637, 686, 731, 737, 744, 830, 954, 975]
```