CSM 61B Spring 2019

Sorting Algorithms

Mentoring 12: 22 April, 2019

1 QuickCo

Malicious Mallory has been hired by a competitor to break into QuickCo, the world leader in sorting algorithms, and tamper with its Quicksort algorithms by making them as slow as possible. Mallory succeeded in unlocking the mainframe, but now she needs your help in slowing QuickCo's algorithms down to a halt!

For this question, assume that the Quicksort algorithm used has three steps.

1. Iterate through the array, to count how many elements are smaller than the pivot, larger than the pivot, and equal to the pivot.
2. Create 3 arrays for smaller, larger and equal elements of the correct size and then, in a second run through the array, place the appropriate elements in these arrays.
3. Finally, recurse on the smaller and larger arrays.

```
int[] data = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

1.1 Mallory decides to change the way QuickCo chooses a pivot for Quicksort.

Given the `int[] data` of size `n`, what choice of pivot would cause the worst-case runtime for Quicksort? Expand out the first few steps of the quicksort algorithms in such a case.

Pivot: Pick the first element, last element, or alternate picking the first and last elements.

Picking the first element, the Quicksort algorithm will continue in the following fashion:

```
{ 1 }, { 2, 3, 4, 5, 6, 7, 8 }
{ 1 }, { 2 }, { 3, 4, 5, 6, 7, 8 }
{ 1 }, { 2 }, { 3 }, { 4, 5, 6, 7, 8 }
```

...

Since the size of our problem is decreasing only by one at each step, this will end up doing $O(N)$ work in the first step, $O(N-1)$ in the second step, $O(N-3)$ in the next, and so on. This will take $O(N^2)$ work.

1.2 Mallory finds an algorithm which always selects the middle element but she is unable to gain write access to it. If the array has an even number of elements, the algorithm picks the element to the left. However, Mallory discovers a way to intercept the incoming data and rearrange it before the algorithm runs.

Given the `int[] data` of size `n`, rearrange the numbers such that the algorithm will run in its worst-case time. Expand out the first few steps of the quicksort algorithms in such a case.

There are multiple solutions to this problem.

```
Case 1: { 8, 6, 4, 2, 1, 3, 5, 7, 9 }
The pivot starts as the smallest element and increases: 1, 2, 3, ...

Case 2: { 2, 4, 6, 8, 9, 7, 5, 3, 1 }
The pivot starts as the largest element and decreases: 9, 8, 7, ...

Case 3: { 6, 7, 8, 9, 1, 2, 3, 4, 5 }
The pivot alternates between the smallest and largest element: 1, 9, 2, 8, 3, 7...
```

In the third case, for example, the Quicksort algorithm will proceed as follows:

```
Pivot: { 1 }; Greater: { 6, 7, 8, 9, 2, 3, 4, 5 };
{ 1 }, { 6, 7, 8, 2, 3, 4, 5 }, { 9 }
{ 1 }, { 2 }, { 6, 7, 8, 3, 4, 5 }, { 9 }
```

...

With a similar reasoning as the worst case pivot choice runtime, this will also take $O(N^2)$ work.

1.3 Does the worst-case runtime of Quicksort depend on the array order, pivot choice, or both? Why?

The worst-case runtime of Quicksort depends on both the array order and the choice of pivots. The worst-case always occurs when the pivot's final position is on an end of the array, which means it was either the smallest or the largest element.

1.4 If the worst-case runtime of Quicksort is $O(N^2)$ while the worst-case runtime of Mergesort is $O(N \log N)$, why do we ever use Quicksort?

In most cases, Quicksort actually performs very well, with an average runtime of $O(N \log N)$. The probability that Quicksort ends up running in $O(N^2)$ is, in reality, very, very low. There is a mathematical discussion about this probability here.

Most implementations of MergeSort requires extra space (more arrays to be created). However, there are more complicated ways to write a MergeSort algorithm that doesn't require any extra space, but Quicksort is still often preferred when the number of elements being sorted isn't extremely large.

The general idea for this is that if all the data being sorted fits inside your RAM, then Quicksort is faster, since it doesn't need to access data stored in your Harddisk, which is slower. For reference, in 1GB RAM, we can hold an array of 32 million integers. This means, that for almost all regular purposes, Quicksort is faster. More information about this is available on the first and second answer here.

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
```

2.2 Performing Radix Sort seems to be a fast sorting algorithm. Why don't we always use it?

Radix sort is only possible when the elements being compared have some radix or base. For strings, they can be broken up into individual characters and separately compared, as we did above. We can also do the same for integers. However, what if we wanted to compare 10 `Cat` objects? We cannot compare `Cat` objects by breaking them up into any smaller pieces or radices.

2.3 Why does Least-Significant Digit radix sort work?

The key is to maintain stability during sorting. If we make sure that at each step we sort stably, then the following proof explains why LSD sort works.

Consider the final step of the algorithm, where we are sorting the first digit.

If two strings differ on the first character, then their ordering will now be fixed, and since the previous steps of the algorithm relatively sorted the rest of the string, the ordering will be fully correct.

If two strings do not differ on the first character, e.g., we having CAT and CBX, we know that CAT would appear before CBX because of the previous steps. All we have to ensure is that this final step doesn't change the ordering of CAX and CBX. This will be ensured, since we make sure we sort stably.

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	$\Theta(N+R)$	$\Theta(W(N+R))$	Yes

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.

Extra: MSD radix sort is stable when implemented with additional space for a buffer.

4 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.

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

```
public class BerkeleyBytes {
    private static int maxAge = 149;
    public static int[] histogram(int[] ages) {
        int[] ageCounts = new int[maxAge + 1];
        for (int age : ages) {
            ageCounts[age] += 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;
                index += 1;
            }
        }
        return result;
    }
}
```

4.2 Time passes and your restaurant is doing well. Unfortunately, our robot overlords advanced medicine to the point where humans are now immortal.

(a) How could we extend the algorithm to accept a list of any 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.