Algorithms Homework Assignment 4

Andrew Osborne

February 19, 2019

Conventions When I refer to $\mathbb{N}$, I speak of

$$\mathbb{N} = \{1, 2, 3, \dots\}$$

Problem 8.2-4

Recall our COUNTING-SORT code

```
COUNTING-SORT(A,B,k)
1  let C[0..k] be a new array
2  for i=0 to k
3      C[i] = 0
4  for j = 1 to A.length
5      C[A[j]] = C[A[j]] + 1
6  for i = 1 to k
7      C[i] = C[i] + C[i-1]
8  for j = A.length downto 1
9      B[C[A[j]]] = A[j]
10     C[A[j]] = C[A[j]] - 1
```

The solution here is to modify this code slightly.

```
MODIFIED-COUNTING-SORT(A,k)
1  let C[0..k] be a new array
2  for i=0 to k
3      C[i] = 0
4  for j = 1 to A.length // n operations
5      C[A[j]] = C[A[j]] + 1
6  for i = 1 to k // k operations
7      C[i] = C[i] + C[i-1]
8  return C
```

Clearly, this algorithm runs in $\Theta(n+k)$ time, just as `COUTNING-SORT` does. Then if given $a, b \in \mathbb{R}$, we may query the number of elements of A in $[a, b]$ by

$$|\{x \in A \ : \ a \leq x \leq b\}| = C[\lfloor b - 1 \rfloor + 1] - C[\lceil a - 1 \rceil]$$

If, however, we do not intend to include a or b in our query, the indices are slightly different. That is,

$$|\{x \in A \ : \ a < x < b\}| = C[\lceil b \rceil - 1] - C[\lfloor a - 1 \rfloor + 1]$$

We can check that $0 \leq a \leq b \leq k$ in $O(1)$ time and then evaluating a difference of array values is also an $O(1)$ operation so our query takes $O(1)$ time.

Problem 8.3-4

I propose to use the following algorith so sort n integers in the range 0 to $n^3 - 1$. Firstly, we must convert each integer into base n. We can do this using strings provided that $n < 256$, but if $n >= 256$ we could use an array of three integers for each digit in our sequence. That is, in python syntax,

```
def convert(A,n):
  B = []
  for number in A: # n iterations.. \Theta(n) time
    string = chr(number%n)
    string = string + chr((number%(n**2) - ord(string[0]))/n)
    string = string + chr((number%(n**3) - \
      ord(string[1])*n - int(string[0]))/(n**2))
    B.append(string)
  return B
```

Notice that we need only three lines of code to decompose each number into base n because we know that each number is bounded above by $n^3 - 1$, which is precisely the maximum value representable by three digits base n. That is

$$(n-1)\,n^0 + (n-1)\,n^1 + (n-1)\,n^2 = n - 1 + n^2 - n + n^3 - n^2 = n^3 - 1$$

So `convert` runs in $\Theta(n)$ time, quite clearly.

Then, knowing that counting sort runs in $\Theta(n+k)$ time where n is the number of elements sorted and k is the maxumim value of any element (assuming a lower bound of zero). Since we have a representation of our numbers in base n, $k = n-1$ and counting sort will run in $\Theta(n+n-1) = \Theta(n)$ time on each digit of our array. Then, there are at most three digits of each element in our array, so the running time of Radix sort on our array of integers is $\Theta(3\ n) = \Theta(n)$.

To put all of that succintly, we need only convert each element of our array into base n and then use Radix sort.

Problem 8.4-2

The worst–case runtime of bucket sort is $\Theta(n^2)$ because, in the worst case, when all elements are funneled into a single bucket, bucket sort reduces to INSERTION-SORT. If we wanted to impose a worst–case runtime of $O(n\,lg\,n)$ we need only replace INSERTION-SORT with a sorting algorithm which has a worst–case runtime of $O(n\,lg\,n)$ or better. Some choices that come to mind are MERGE-SORT and HEAP-SORT. I am having a hard time justifying the claim that the average runtime is still $\Theta(n)$, but I have calculated

$$E[n_i\,lg\,n_i] = \sum_{k=0}^{n}(k\,lg(k)\,\frac{n}{n-k+1}\,(\prod_{j=0}^{k}\frac{n+1-j}{2n+1-j}))$$

. At any rate, heuristically, one can imagine that, since $n\,lg\,n = o(n^2)$ then $E[n\,lg\,n] = E[o(n^2)] = o(E[n^2])$ and then our average case run time is forbidden from superlinearity by the argument in section 8.4 of the text book. Furthermore, that worst case runningtime can be only the worst case runtime of our chosen replacement algorithm (MERGE-SORT or HEAP-SORT).

In short, replace INSERTION-SORT with HEAP-SORT or MERGE-SORT to keep the worst case run time of our algorithm $O(n\,lg\,n)$ while preserving $\Theta(n)$ average case running time.