HW7

1. From Stavros Vallas

1.

Insertion sort and merge sort are stable while heapsort and quicksort are not stable. A simple scheme to make any sorting algorithm into a stable one would be if you attached the index of every value in an array to the value using an ordered pair of some sort. The first item in the ordered pair can be the index and the second can be the value. For example, [6,7,8] will become [(1,6),(2,7),(3,8)]. Then when two values with the same number are compared then they will look to see which have the smaller index and that one will go first. The above scheme does not change the running time because the comparison of the index occurs in constant time, however more space will be needed because it will be an array of ordered pairs and not just values.

$O(n \lg n)$

2-4. From Victor M Espaillat

2(1)

Initial parameters

	1	2	3	4	5	6
A	6	0	2	6	0	8

$n = 6$

	1	2	3	4	5	6
B						

$k = 8$

	0	1	2	3	4	5	6	7	8
C	2	0	1	0	0	0	2	0	1

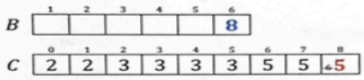
(a)
after line 5

	0	1	2	3	4	5	6	7	8
C	2	2	3	3	3	3	5	5	6

(b)
after line 7

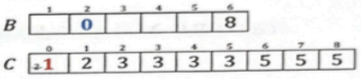
Iterations of loop in lines 8–10

$j = 6$
$B[C[A[j]]] = A[j]$
$B[C[A[6]]] = A[6]$
$B[C[8]] = 8$
$B[6] = 8$
$C[A[j]]{-}{-}$
$C[A[6]]{-}{-}$
$C[8]{-}{-}$



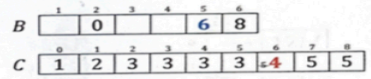
(c)

$j = 5$
$B[C[A[5]]] = A[5]$
$B[C[0]] = 0$
$B[2] = 0$
$C[A[5]]{-}{-}$
$C[0]{-}{-}$



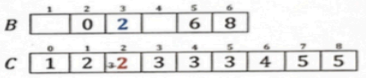
(d)

$j = 4$
$B[C[A[4]]] = A[4]$
$B[C[6]] = 6$
$B[5] = 6$
$C[A[4]]{-}{-}$
$C[6]{-}{-}$



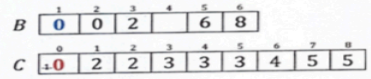
(e)

$j = 3$
$B[C[A[3]]] = A[3]$
$B[C[2]] = 2$
$B[3] = 2$
$C[A[3]]{-}{-}$
$C[2]{-}{-}$



(f)

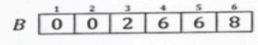
$j = 2$
$B[C[A[2]]] = A[2]$
$B[C[0]] = 0$
$B[1] = 0$
$C[A[2]]{-}{-}$
$C[0]{-}{-}$



(g)

$j = 1$
$B[C[A[1]]] = A[1]$
$B[C[6]] = 6$
$B[4] = 6$
$C[A[1]]{-}{-}$
$C[6]{-}{-}$

(h)



(i)

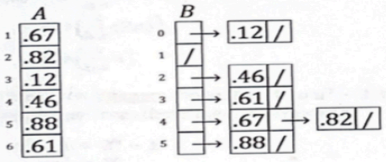
Final sorted output array B

(2) RADIX-SORT (show as in Figure 8.3): English words: PAT, CAT, CART (*right-justify this word* so that T is compared first), FAT, FIX. Padded with white spaces when needed.

(3) BUCKET-SORT (show as in Figure 8.4): $A = \langle 0.67, 0.82, 0.12, 0.46, 0.88, 0.61 \rangle$

$B[0\,..\,5]$ (6 buckets)

$\lfloor 0.67 \times 6 \rfloor = 4$
$\lfloor 0.82 \times 6 \rfloor = 4$
$\lfloor 0.12 \times 6 \rfloor = 0$
$\lfloor 0.46 \times 6 \rfloor = 2$
$\lfloor 0.88 \times 6 \rfloor = 5$
$\lfloor 0.61 \times 6 \rfloor = 3$



3(1)

Given n d-digit numbers in which each digit can take on up to k possible values, RADIX-SORT sorts the numbers in $\Theta(d(n+k))$ time using COUNTING-SORT to sort by each digit.

Each integer is in b bits. So each integer ranges from 0 to $2^b - 1$.

If we break each integer into r-bit digits, then each digit ranges from 0 to $2^r - 1$ and the total number of digits becomes $d = \lceil b/r \rceil$.

$$\begin{aligned} T(n) &= \Theta(d(n+k)) \\ &= \Theta\left(\tfrac{b}{r}(n+2^r)\right) \end{aligned}$$

If we let each digit range from 0 to $n-1$,

$$\begin{aligned} n - 1 &= 2^r - 1 \\ n &= 2^r \\ \lg n &= \lg 2^r \\ \lg n &= r \end{aligned}$$

we get r in terms on n and can see that we would need $\lg n$ bits for each digit.

$$\begin{aligned} T(n) &= \Theta\left(\tfrac{b}{\lg n}(n+n)\right) \\ &= \Theta\left(\tfrac{b}{\lg n}(2n)\right) \\ &= \Theta\left(\tfrac{b}{\lg n}n\right) \end{aligned}$$

If the given integers range from 0 to $n^3 - 1$, then we can solve for the number of bits needed to store any integer from that range, in terms of n.

$$\begin{aligned} n^3 - 1 &= 2^b - 1 \\ n^3 &= 2^b \\ \lg n^3 &= \lg 2^b \\ 3\lg n &= b \end{aligned}$$

Now we have b in terms of n.

If each integer requires $3\lg n$ bits, and each digit of these integers requires $\lg n$ bits, then we can see that the number of digits d becomes 3. That means we only need to call COUNTING-SORT three times.

$$\begin{aligned} \mathrm{T}(n) &= \Theta\left(\tfrac{3\lg n}{\lg n}n\right) \\ &= \Theta(3n) \\ &= \Theta(n) \end{aligned}$$

Therefore we can sort n integers in the range 0 to $(n^3 - 1)$ in $O(n)$ time.

(2) (5 points) What is the running time if we only use COUNTING-SORT?

$$\begin{aligned} \mathrm{T}(n) &= \Theta(n+k) \\ &= \Theta(n+n^3) \\ &= \Theta(n^3) \end{aligned}$$

*3(1) Treat the numbers as 3-digit numbers in radix n. Each digit ranges from 0 to n-1. So we can sort these 3-digit numbers with radix sort. 3 calls to counting sort, each takes linear time.

4.

The worst-case running time for BUCKET-SORT is $\Theta(n^2)$ because lists are sorted using INSERTION-SORT which has $O(n^2)$ worst-case running time, and it could be the case that every element ends up being inserted in the same list.

One simple change to preserve BUCKET-SORT's linear average-case running time and make its worst-case running time $O(n \lg n)$ is to use a more efficient sorting algorithm than INSERTION-SORT to sort its lists, such as MERGESORT, which has $O(n \lg n)$ worst-case running time.