8.1-1

What is the smallest possible depth of a leaf in a decision tree for a comparison sort?

Solution:

n – 1.

If we assume that the depth of the root is 0, then for a leaf, the best case in the sort is that we compare it with an already sorted series of numbers, we could use only n – 1 times of comparison to complete the sorting, thus the depth is n – 1.

8.1-3

Show that there is no comparison sort whose running time is linear for at least half of the n! inputs of length n. What about a fraction of 1/n of the inputs of length n? What about a fraction 1/(2^n) ?

Solution:

According to Stirling's approximation formula: we have lgn = n∗lgn - n∗lge + O(lgn),

1) For half of n! :

h ⩾ lg(n!/2) = lgn! - lg2 ⩾ n∗lgn - n∗lge – 1 = Ω(nlgn)

2) fraction of 1/n:

lg(n!/n) = lgn! - lgn ⩾ n∗lgn - n∗lge – lgn = Ω(nlgn)

3) fraction 1/2n:

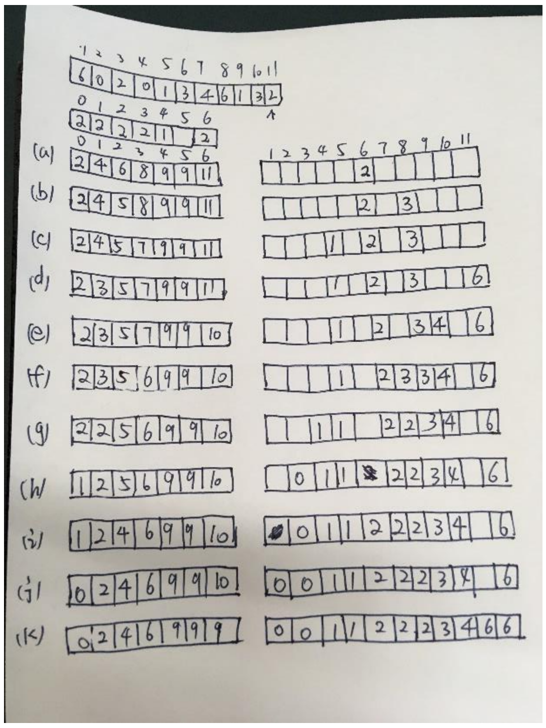
lg(n!/2n) = lgn! – n ⩾ n∗lgn - n∗lge – n = Ω(nlgn)

So there are no linear running time.

8.2-1

Using Figure 8.2 as a model, illustrate the operation of COUNTING-SORT on the array A= {6, 0 , 2, 0, 1, 3, 4, 6, 1, 3, 2}.

Solution:



8.2-2

Prove that COUNTING-SORT is stable.

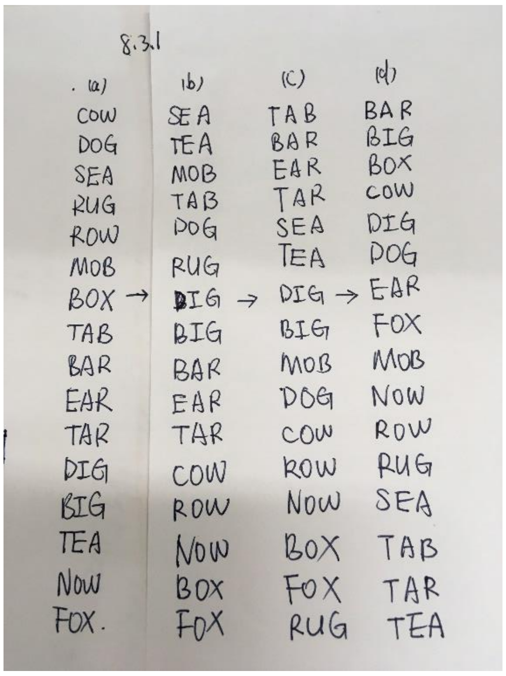
Solution:

Counting sort is stable, because in the last step, we loop over the array in the reverse order, and once we meet an item, we put it the last position among all the items who got the same key values, so the order in the result is relatively the same comparing to their initial order, in this case, the counting sort is stable.

8.3-1

Using Figure 8.3 as a model, illustrate the operation of R ADIX -S ORT on the following list of English words: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

Solution:



8.3.1

. (a)	(b)	(c)	(d)
COW	SEA	TAB	BAR
DOG	TEA	BAR	BIG
SEA	MOB	EAR	BOX
RUG	TAB	TAR	COW
ROW	DOG	SEA	DIG
MOB	RUG	TEA	DOG
BOX →	DIG →	DIG →	EAR
TAB	BIG	BIG	FOX
BAR	BAR	MOB	MOB
EAR	EAR	DOG	NOW
TAR	TAR	COW	ROW
DIG	COW	ROW	RUG
BIG	ROW	NOW	SEA
TEA	NOW	BOX	TAB
NOW	BOX	FOX	TAR
FOX.	FOX	RUG	TEA

8.3-3

Use induction to prove that radix sort works. Where does your proof need the assumption that the intermediate sort is stable?

Solution:

Loop Invariant: Before every loop, the last i – 1 digits are already sorted.

Initialization: The array is trivially sorted on the last 0 digits.

Maintenance: assume that the numbers in the array is sorted in the last i – 1 digits. So after we sort on the ith digit, the array will be sorted on the last i digits. It is obvious that elements with different digit in the ith position are ordered accordingly; in the case of the same ith digit, we still get a correct order, because we are using a stable sort and the elements were already sorted on the last i – 1 digits.

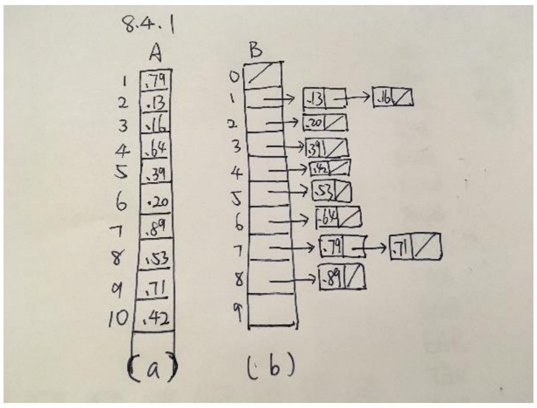
Termination: The loop terminates when i=d+1. Since the invariant holds, we have the numbers sorted on d digits.

8.4-1

Using Figure 8.4 as a model, illustrate the operation of BUCKET-SORT on the array

A = {.79, .13, .16, .64, .39, .20, .89, .53, .71, .42}

Solution:



8.4-2

Explain why the worst-case running time for bucket sort is Θ(n^2). What simple change to the algorithm preserves its linear average-case running time and makes its worst-case running time O(nlgn)?

Solution:

The worse case is that all the items are locates in the same bucket, and they are in reverse order, so for every item, we need to insert the item to the end of the same linked list, which will take Θ(n^2) time to complete.

To optimize, the we could adopt merge sort or heap sort for the sort of the linked list in the bucket, though it might be kind of complex, while it could make the time down to O(nlgn). Insertion sort was chosen, because it operates well on linked lists. If we use another sorting algorithm, we have to convert each list to an array, which might slow down the algorithm in practice.