Sorting Practice – Count sort, Radix sort, Bucket sort - SOLUTION

Last updated: 3/16/2021

For all sorting algorithms: **Time and Space complexity. Stable? Adaptive? (Data moves)**

NCS1. (6 points) You are using count sort to sort an array of N numbers, where each number is from the range [0,M]. What is the time complexity (as Theta) of the number of data moves? (For example swapping two records requires 3 data moves.). Briefly justify your answer.

"Data" are the records in the original array, A.=> 2N moves (N to put in sorted order in the copy array and another N to copy back into A.) => Θ(N)

NCS2. (9 points**)** (Radix sort)

Show how **LSD radix sort** sorts the following numbers in the given representation (base 10). Show the numbers after each complete round of count sort.

Index:	0	1	2	3	4	5	6
Original Array:	513	145	320	235	141	433	2
	320	**141**	**2**	**513**	**433**	**145**	**235**
	2	**513**	**320**	**433**	**235**	**141**	**145**
	2	**141**	**145**	**235**	**320**	**433**	**513**

NCS3. (4 points) What is the operation you do to map/scale values from range [A,B] to range [X,Y]? You can assume that A < B and X < Y. (E.g. [47,49] -> [20,30], [5,10] -> [21,23])

$\boldsymbol{new} = \frac{curr-A}{B-A}(Y-X)+X$ ($\boldsymbol{wrong\ with}\ X-Y$: $\frac{curr-A}{B-A}\cancel{(X-Y)}+X$) See slides for more details.

NCS4. (5 pts) Assume you want to use bucket sort to sort an array A, that has integers in the range [-100, 350]. (i.e. A[i]≥-100 and A[i]≤350, for all valid i). You will use 50 buckets. Write the formula to find the index, `bucketIdx`, for the bucket where A[i] should go.
Make sure you indicate any rounding (up or down) if necessary.

$bucketIdx = \left\lfloor \frac{A[i]-(-100)}{1+350-(-100)} * 50 \right\rfloor$ where ⌊ ⌋ means rounded down

NCS5. (6 pts) Fill in the arrays to show the required processing with count sort for the data below.

	0	1	2	3	4	5	6
Original array	C, Alice	B, Jane	A, Jane	F, John	A, Matt	D, Sam	B, Tom

Counts array after part 1 (counts of each key):

Index:	0 A	1 B	2 C	3 D	4 E	5 F
Counts array:	2	2	1	1	0	1

Counts array after part 2 (after cumulative sum):

Index:	0 A	1 B	2 C	3 D	4 E	5 F
Counts array:	2	4	5	6	6	7

Show the counts array and the copy array after each of the next 2 big steps of count sort as shown in slide 6 (i.e. after a first element is placed in the copy array, and after a second element is placed in the copy array). Create columns as needed in the tables below.

Index:	0 A	1 B	2 C	3 D	4 E	5 F
Counts array:	2	**3**	5	6	6	7
Counts array:	2	3	5	**5**	6	7

Index:	0	1	2	3	4	5	6
Copy array:				**B, Tom**			
Copy array:				B, Tom		**D, Sam**	

NCS6. a) We run bucket sort (version covered in class) on the array [0.3, 0.15, 0.27, 0.8, 0.61]. How many buckets will be created? What are the elements in each bucket? ? Here we assume the numbers in the array are in the range [0,1) and we map these to the bucket indexes (like in the pseudocode). We do NOT use the min and max from the array.

When giving the elements in a bucket, give them in SORTED order, separated by commas and with no extra spaces. Say *empty* if the bucket is empty.

5 buckets (The default is to use as many as the number of elements in the array)

Formula for finding the bucket: floor(elem*buckets) (e.g. floor(0.3*5) = floor(1.5)=1, floor(0.15*5)=0, etc)

Bucket[0]: 0.15

Bucket[1]: 0.27,0.3,

Bucket[2]: empty

Bucket[3]: 0.61,~~0.8~~

Bucket[4]: 0.8 ~~empty~~

b) What if 10 buckets were created?

Bucket[0]: empty

Bucket[1]: 0.15

Bucket[2]: 0.27

Bucket[3]: 0.3

Bucket[4]: empty

Bucket[5]: empty

Bucket[6]: 0.61

Bucket[7]: empty

Bucket[8]: 0.8

Bucket[9]: empty