Please send any comments or criticisms to dwharder@alumni.uwaterloo.ca
with the subject ECE 250 Notes 8.7.
Assistances and comments will be acknowledged.

8.7 Bucket Sort

Our next goal will be to look at an algorithm that performs sorting in linear time. To break the $\Theta(n \ln(n))$ barrier, we must resort to a sorting technique that does not compare entries.

8.7.1Supporting Example

Suppose we want to sort all residential phone numbers in the 416 area code region (say, 4 000 000 different numbers). If each 7-digit phone number was interpreted as a 4-byte integer, this would require approximately 16 MiB. To sort this, we could load all the numbers into main memory and call quick sort, or a similar routine.

However, consider this alternate, strategy:

1. Create a bit-vector with 10^7 bits (this would require $10^7/8/1024^2 \approx 1.2$ MiB),
2. Set each bit to 0 (indicating false),
3. Stepping through all of the phone numbers, interpret each phone number as an index into the array and set the corresponding bit to 1 (indicating true), and
4. Finally, step through the bit-vector again but only record those entries that are flagged as true (1).

For example, Figure 1 shows the entries corresponding to numbers 685-7543 to 685-7560. The shading indicates consecutive bits occupying the same byte. Each bit that set to true indicates that the phone number corresponding to the index is a residential number. Thus, for example, 685-7550 is a valid residential phone number in the 416 area code.

...	6857543	6857544	6857545	6857546	6857547	6857548	6857549	6857550	6857551	6857552	6857553	6857554	6857555	6857556	6857557	6857558	6857559	6857560	...
...	0	1	0	0	1	0	0	1	0	1	0	0	1	1	0	1	1	1	...

If we wanted to extract the sorted residential phone numbers, we would just pass through the list and read off those that are flagged:

..., 685-7544, 685-7547, 685-7550, 685-7552, 685-7555, 685-7556, 685-7558, 685-7559, 685-7560, ...

Thus, one pass is made through all 4 000 000 residential phone numbers and one subsequent pass through the 10 000 000 entries in the array is required to extract the sorted residential numbers. Thus, because the number of potential residential phone numbers is in the same magnitude as the number of actual residential phone numbers, the run time is $\Theta(n)$.

8.7.2 Bucket Sort Algorithm

This gives us the bucket-sort algorithm. Suppose we are sorting *n* items on the range 0 through *m* – 1:

1. Create a bit-vector array of *m*buckets,
2. For each of the *n* items being sorted, set the corresponding buckets to true, and
3. Make a pass through the array recording those entries that were flagged.

The run time of this algorithm is Θ(*m*) and if *m* = Θ(*n*) (that is, they are comparable), the run time could therefore be described as Θ(*n*).

7.5.3 Example

Consider sorting the numbers

20, 1, 31, 8, 29, 28, 11, 14, 6, 16, 15, 27, 10, 4, 23, 7, 19, 18, 0, 26, 12, 22

All of these numbers are on the range 0 through 31, so create a bit vector with 32 entries:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Make one pass and set each entry corresponding to a number to 1 (true):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	1	0	0	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	0	0	1	1	1	1	0	1

Finally, make one last pass and record only those numbers that are flagged:

0, 1, 4, 6, 7, 8, 10, 11, 12, 14, 15, 16, 18, 19, 20, 22, 23, 26, 27, 28, 29, 31.

Algorithms that sort arbitrary data using comparisons must run in Ω(*n* ln(*n*)) time. In this case, however, we have constraints on the data and using those constraints, we can reduce the run time to Θ(*n*).

8.7.4 Bucket Sorting with Repetitions

If we are sorting a list with repetitions, a bit-vector will be insufficient. In this case, we have two choices. We can either use a vector where each bucket is a

1. counter (*i.e.*, an integer), or
2. a linked list.

If two objects are the same, we can either increment the counter twice or place both objects into the same linked list, respectively.

As another example, suppose we want to sort the 42digits

0 3 2 8 5 3 7 5 3 2 8 2 3 5 1 3 2 8 5 3 4 9 2 3 5 1 0 9 3 5 2 3 5 4 2 1 3

We would begin with a vector of 10 counters all set to zero:

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

Stepping through all 37 entries, we increment the bucket corresponding to each value:

0	1	2	3	4	5	6	7	8	9
2	3	7	10	2	7	0	1	3	2

Now we pass through the array and make as many copies as appears in the counters:

0 0 1 1 1 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 4 4 5 5 5 5 5 5 5 7 8 8 8 9 9

The list is now sorted in Θ(*n*) requiring Θ(*m*) memory.

8.7.5 Summary

Thus, we can summarize the run-time and memory requirements:

Case	Run Time	Memory	Comments
Worst Case	$\Theta(m)$	$\omega(n)$	$m = \omega(n)$
Average Case	$\Theta(m + n)$	$\Theta(m)$	
Best Case	$\Theta(n)$	$o(n)$	$m = o(n)$

Thus, if we are sorting *n* = 20 integers on the range from one to one million (*m* = 1 000 000), it would be absurd to use bucket sort in this case.