CS 620 October 9, 2011

The Radix Sort Algorithm

Overview. Radix sort (sometimes called bucket sort or distribution sort) was one of the first sorting algorithms developed, back in the days of punched cards and card-handling machines. Today, it still can be the fastest way to sort a large number of items. However, it *does* involve considerable setup time and data structures, and works only with fixed-length sort keys. A data item could be any type of object (or pointer to object) that has one data member that acts as a sort key.

The algorithm divides the sort key into k fields, based on its binary representation and makes k passes through the entire data list. On pass 0, the least significant (rightmost) field of the key is isolated and used to assign the data item to a bucket-queue. On pass p, the *pth* field is used. At the end of each pass, all the data is collected again into a single queue, ready for re-distribution. After k passes, the data is sorted.

The number of fields in the key determines the number of passes. The number of bits in each field determines the number of buckets needed: for k fields, we need 2^k buckets. The algorithm is simpler to implement if each field is a half-byte, a byte, or two bytes, resulting in 16 or 256 or 35536 queues.

This handout illustrates an implementation of radix sort that sorts a small set of short unsigned integers, written in hex. We divide these two-byte integers into four 4-bit fields, which means we need 16 queues for our buckets.

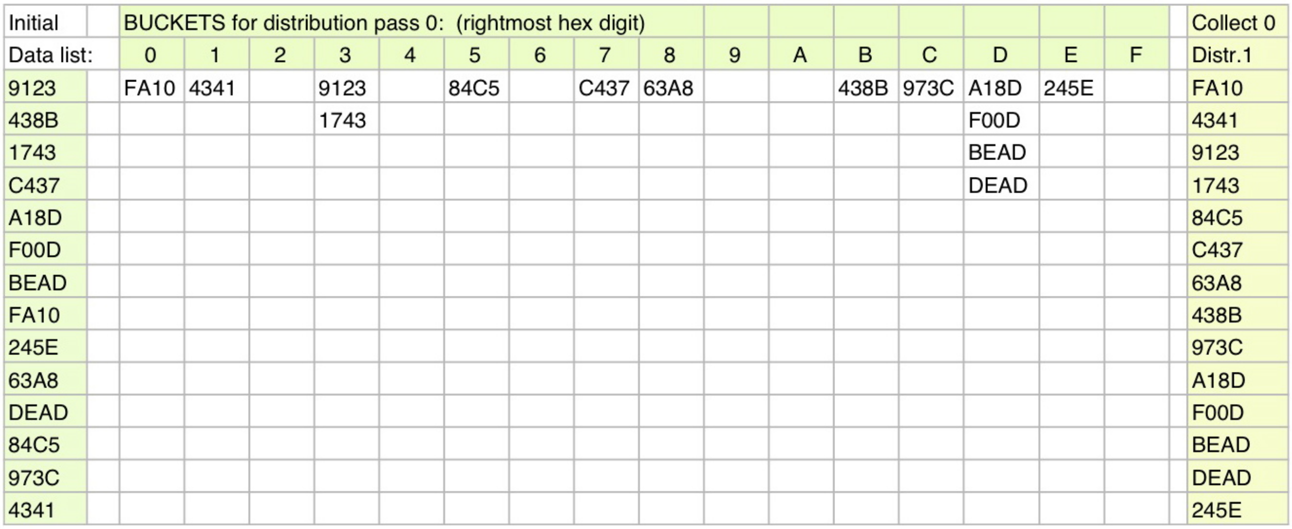
How to sort. The sorting process has these steps:

- Read the data from a file and install in Cells in a linked list. (We use a linked queue here.)
- Make k passes through the data list. On pass p:
  - Distribute all the data items into the buckets, according to the contents of the *pth* field of the key, counting from the right. Each bucket must be implemented as a queue.
  - Collect all the data from the buckets, in order, ending with the data from bucket 0 at the head of the collection queue.
- Write the sorted data out to a file.

How to find the right bucket. The heart of the algorithm is using the sort key from the data item to identify the correct bucket. Some implementations of radix sort use slow methods, such as division and modulo, to get bucket numbers. However, we want the algorithm to work efficiently, so we use instructions that are efficient on every computer: right shift and bitwise &. Given the pass number p and the number of bits in one field of the key, b, do this for every data item in the input queue:

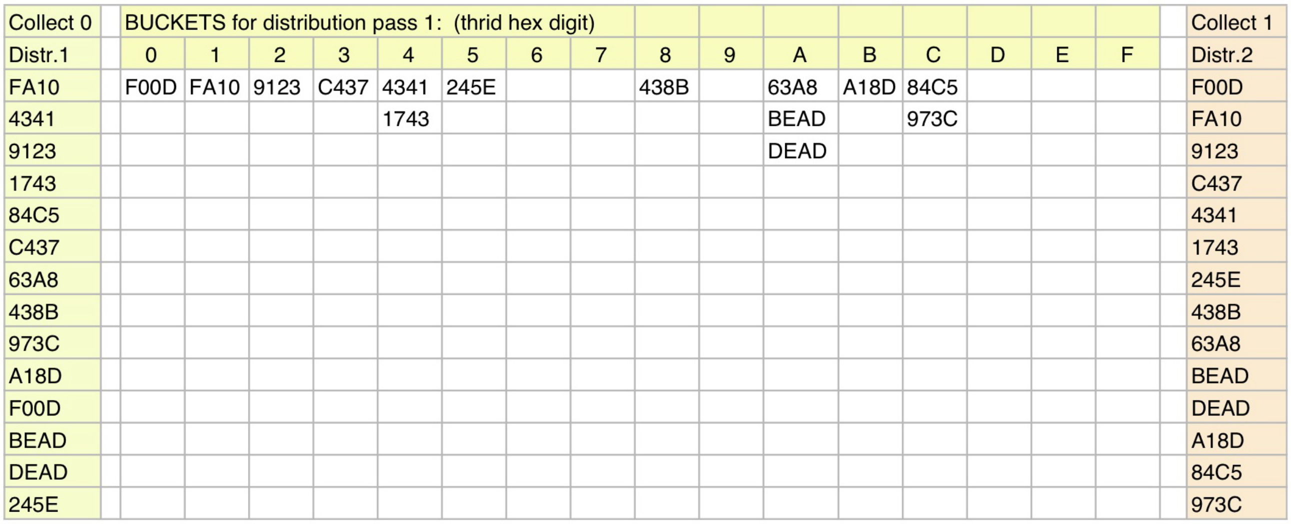
- Use `removeCell()` to get the next data item from the queue. Do not discard the Cell that contains it, since you will be appending that Cell to a new queue soon.
- Shift the item's sort key $p * b$ bits toward the right. This puts the *pth* field at the right end of the number.
- Bitwise-and the result with MASK, where MASK has b 1-bits at the right end and 0-bits elsewhere. The result is the subscript of the correct bucket.
- Append the Cell containing the data to the queue in the bucket you calculated.

After the *pth* collection step, the data is sorted.



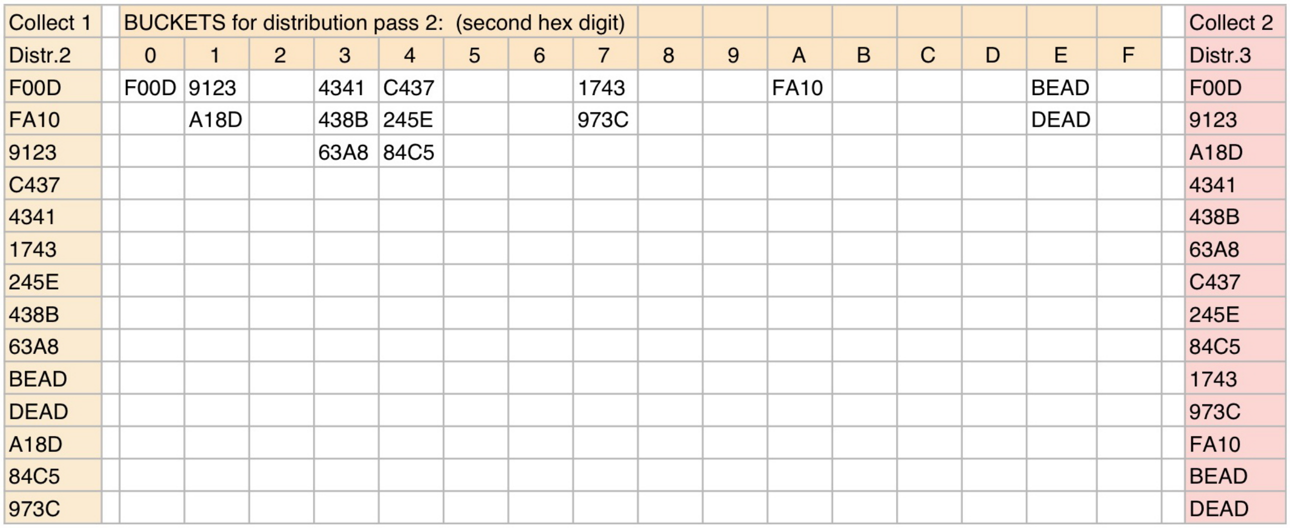
Initial		BUCKETS for distribution pass 0: (rightmost hex digit)																	Collect 0
Data list:		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		Distr.1
9123		FA10	4341		9123		84C5		C437	63A8			438B	973C	A18D	245E			FA10
438B					1743										F00D				4341
1743															BEAD				9123
C437															DEAD				1743
A18D																			84C5
F00D																			C437
BEAD																			63A8
FA10																			438B
245E																			973C
63A8																			A18D
DEAD																			F00D
84C5																			BEAD
973C																			DEAD
4341																			245E

Figure 1: Radix Sort – Pass 0



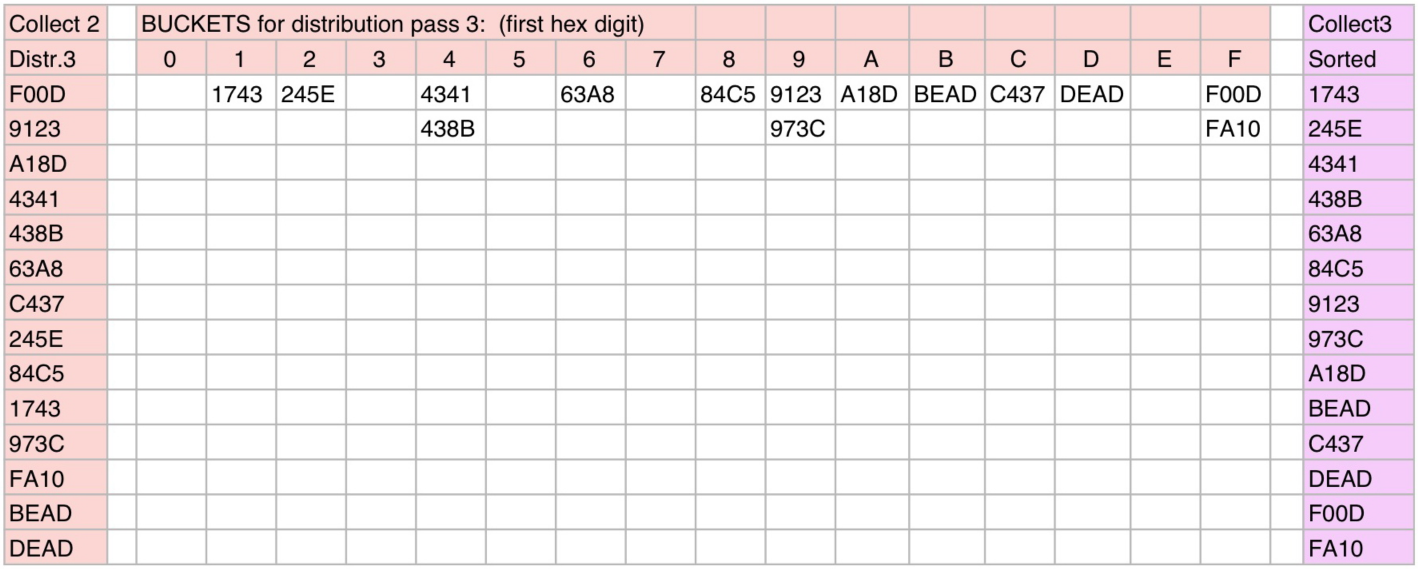
Collect 0		BUCKETS for distribution pass 1: (thrid hex digit)																	Collect 1
Distr.1		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		Distr.2
FA10		F00D	FA10	9123	C437	4341	245E			438B		63A8	A18D	84C5					F00D
4341						1743						BEAD		973C					FA10
9123												DEAD							9123
1743																			C437
84C5																			4341
C437																			1743
63A8																			245E
438B																			438B
973C																			63A8
A18D																			BEAD
F00D																			DEAD
BEAD																			A18D
DEAD																			84C5
245E																			973C

Figure 2: Radix Sort – Pass 1



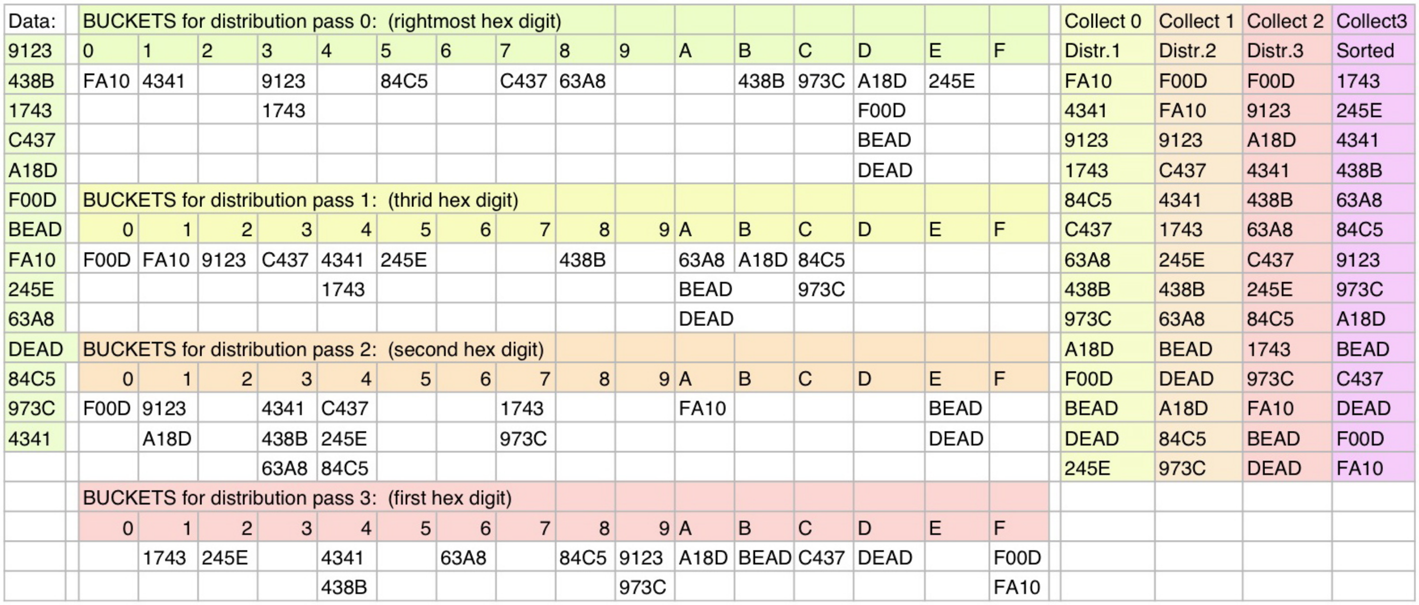
Collect 1		BUCKETS for distribution pass 2: (second hex digit)																	Collect 2
Distr.2		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		Distr.3
F00D		F00D	9123		4341	C437			1743			FA10				BEAD			F00D
FA10			A18D		438B	245E			973C							DEAD			9123
9123					63A8	84C5													A18D
C437																			4341
4341																			438B
1743																			63A8
245E																			C437
438B																			245E
63A8																			84C5
BEAD																			1743
DEAD																			973C
A18D																			FA10
84C5																			BEAD
973C																			DEAD

Figure 3: Radix Sort – Pass 2



Collect 2		BUCKETS for distribution pass 3: (first hex digit)																	Collect3
Distr.3		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		Sorted
F00D			1743	245E		4341		63A8		84C5	9123	A18D	BEAD	C437	DEAD		F00D		1743
9123						438B					973C						FA10		245E
A18D																			4341
4341																			438B
438B																			63A8
63A8																			84C5
C437																			9123
245E																			973C
84C5																			A18D
1743																			BEAD
973C																			C437
FA10																			DEAD
BEAD																			F00D
DEAD																			FA10

Figure 4: Radix Sort – Pass 3



Data:		BUCKETS for distribution pass 0: (rightmost hex digit)																	Collect 0	Collect 1	Collect 2	Collect3
9123		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		Distr.1	Distr.2	Distr.3	Sorted
438B		FA10	4341		9123		84C5		C437	63A8			438B	973C	A18D	245E			FA10	F00D	F00D	1743
1743					1743										F00D				4341	FA10	9123	245E
C437															BEAD				9123	9123	A18D	4341
A18D															DEAD				1743	C437	4341	438B
F00D		BUCKETS for distribution pass 1: (thrid hex digit)																	84C5	4341	438B	63A8
BEAD		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		C437	1743	63A8	84C5
FA10		F00D	FA10	9123	C437	4341	245E			438B		63A8	A18D	84C5					63A8	245E	C437	9123
245E						1743						BEAD		973C					438B	438B	245E	973C
63A8												DEAD							973C	63A8	84C5	A18D
DEAD		BUCKETS for distribution pass 2: (second hex digit)																	A18D	BEAD	1743	BEAD
84C5		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		F00D	DEAD	973C	C437
973C		F00D	9123		4341	C437			1743			FA10				BEAD			BEAD	A18D	FA10	DEAD
4341			A18D		438B	245E			973C							DEAD			DEAD	84C5	BEAD	F00D
					63A8	84C5													245E	973C	DEAD	FA10
		BUCKETS for distribution pass 3: (first hex digit)																				
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F					
			1743	245E		4341		63A8		84C5	9123	A18D	BEAD	C437	DEAD		F00D					
						438B					973C						FA10					

Figure 5: Radix Sort – Overview