Lower-bound in comparison-based sorting

Any sorting algorithm on n-elements that uses comparisons to sort elements of unlimited range (i.e: from $-\infty$ to $+\infty$) makes no better than $\Omega(n \log n)$

Run time of $O(n)$

counting sort & radix sort

Do they violet the lower bound?

No! because they sort numbers within a range.

Counting Sort

A: array with the numbers, numbers are in range $[0 \dots k]$

Not in place as it uses auxilliary arrays but it is a stable sorting algorithm

Stable sorting

5a 4 3a 12 5b 3b 7 5c

⇒ 3a 3b 4 5a 5b 5c 7 12

$C[i] = 0 \ \forall \ i \in [0, \dots, k]$

$O(n)$
```
For j=1...length(A) do   // count how many numbers in original array A
    C[A[j]] = C[A[j]] + 1
```

$O(k)$
```
For i=1...k              // prefix sums: count how many numbers equal to / below the number
    C[i] = C[i] + C[i-1]
```

$O(n+k)$
```
For j=length(A)...1      // place number in positions
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
```

total time: $O(n+k)$. if $k = O(n)$ then $O(n+k) = O(n)$ linear time

Example:

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

⇒ range $[0 \dots 5]$

	0	1	2	3	4	5
C	2	0	2	3	0	1

⇒ count how many number there is of the index number

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

⇒ count how many number that's smaller/equal to index number

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

For j=length(A)...1

$B[C[A[j]]] = A[j]$ $C[A[j]] = C[A[j]] - 1$

$B[C[A(8)]] = A(8)$ $B[C[3]] = A[8]$ $B[7] = A[8] = 3$ $C[3] = 7-1 = 6$

$B[C[A(7)]] = A(7)$ $B[C[0]] = A(7) = 0$ $B[2] = 0$ $C[0] = 2-1 = 1$

$B[C[A(6)]] = A(6) = 3$ $B[C[3]] = 3$ $B[6] = 3$ $C[3] = 6-1 = 5$

$B[C[A(5)]] = A(5) = 2$ $B[C[2]] = B[4] = 2$ $C[2] = 4-1 = 3$

$B[C[A(4)]] = A(4) = 0$ $B[C[0]] = B[1] = 0$ $C[0] = 1-1 = 0$

$B[C[A(3)]] = A(3) = 3$ $B[C[3]] = B[5] = 3$ $C[3] = 5-1 = 4$

$B[C[A(2)]] = A(2) = 5$ $B[C[5]] = B[8] = 5$ $C[5] = 8-1 = 7$

$B[C[A(1)]] = A(1) = 2$ $B[C[2]] = B[3] = 2$

Radix Sort (Also stable sorting algorithm)

key idea: sorting from LSB → MSB (or digit)

```
Radix sort (A, d)    // d: number of digits
    For i=1...d
        stable sort A on digit-i
        use counting sort
```

Example:

	1st digit	2nd digit	3rd digit
853	153	112	112
814	112	814	153
734	814	734	654

937 →	724 →	937 →	724
152	654	152	814
654	857	654	857
112	937	857	937

n = # of numbers

k = range of the digits

d = # digit ∀ number

one pass: $\theta(n+k)$ counting sort

All passes: $d \cdot \theta(n+k) = \theta(dn+dk)$

If we assume that d.k = constants, then total time = $\theta(n)$

Motivational example:

One thousand (1.000) #s that are 64-bit each. Radix sort time?

16 bits	16 bits	16 bits	16 bits

Every "digit" is 16-bits

Radix takes "4 passes/number"

Quick sort time?

$O(n \log n) = \frac{O(1000 \cdot 10)}{1000} = 10$ passes/number