Better than $N \lg N$?

- Can prove that if all you can do to keys is compare them then sorting must take $\Omega(N \lg N)$.
- Basic idea: there are $N!$ possible ways the input data could be scrambled.
- Therefore, your program must be prepared to do $N!$ different combinations of move operations.
- Therefore, there must be $N!$ possible combinations of outcomes of all the $if$ tests in your program (we’re assuming that comparisons are 2-way).

**Height $\propto$ Sorting time**

```
  T  a < b
     F  a < c
         b < c
```

Necessary Choices

- Since each $if$ test goes two ways, number of possible different outcomes for $k$ $if$ tests is $2^k$.
- Thus, need enough tests so that $2^k > N!$, which means $k \in \Omega(\lg N!)$.
- Using Stirling’s approximation,
  \[ m! \in \sqrt{2\pi m} \left(\frac{m}{e}\right)^m \left(1 + \Theta\left(\frac{1}{m}\right)\right), \]
  this tells us that $k \in \Omega(N \lg N)$.

Beyond Comparison: Distribution

- But suppose can do more than compare keys?
  - For example, how can we sort a set of $N$ integer keys whose values range from 0 to $kN$, for some small constant $k$?
  - One technique: put the integers into $N$ buckets, with an integer $p$ going to bucket $p/k$.
  - At most $k$ keys per bucket, so catenate and use insertion sort, which will now be fast.
  - E.g., $k = 2, N = 10$ :

    Start:
    
    `14  3  10  13  4  2  19  17  0  9`

    In buckets:
    
    `| 0  | 3  | 2  | 4  | 9  | 10 | 13 | 14 | 17 | 19 |`

- Now insertion sort is fast. For fixed $k$, $\Theta(N)$.
**Distribution Counting**

- Another technique: count the number of items \(< 1, < 2, \text{ etc.}\).
- If \(M_p = \#\text{items with value} < p\), then in sorted order, the \(j^\text{th}\) item with value \(p\) must be \(\#M_p + j\).
- Gives linear-time algorithm.

**Distribution Counting Example**

- Suppose all items are between 0 and 9 as in this example:

  \[
  7040919195373167420
  \]

  \[
  3312211303
  \]

- "Counts" line gives \# occurrences of each key.
- "Running sum" gives cumulative count of keys \(\leq\) each value . . .
- . . . which tells us where to put each key:
- The first instance of key \(k\) goes into slot \(m\), where \(m\) is the number of key instances that are \(< k\).

**Radix Sort**

**Idea:** Sort keys one character at a time.

- Can use distribution counting for each digit.
- Can work either right to left (LSD radix sort) or left to right (MSD radix sort).
- LSD radix sort is venerable: used for punched cards.

  Initial: set, cat, cad, con, bat, can, be, let, bet

  Pass 1 (by char \#2)
  be, cad, con, set
  bet, bat

  Pass 2 (by char \#1)
  be, cad, con, set
  bat, be, bet, bet

  Pass 3 (by char \#0)
  bat, be, bet, bet

- MSD Radix Sort

  A bit more complicated: must keep lists from each step separate
  - But, can stop processing 1-element lists

  \[
  \begin{array}{c|c}
  A & \text{posn} \\
  \hline
  & 0 \\
  + & 1 \\
  + & 2 \\
  + & 1 \\
  + & 2 \\
  \end{array}
  \]

  | set, cat, cad, con, bat, can, be, let, bet | 0 |
  | bat, be, bet / cat, cad, con, can / let / set | 1 |
  | bat / + be, bet / cat, cad, con, can / let / set | 2 |
  | bat / be / bet / + cat, cad, con / can / let / set | 1 |
  | bat / be / bet / + cat, cad, con / can / let / set | 2 |
  | bat / be / bet / + cat, cad, con / can / let / set | 3 |
Performance of Radix Sort

- Radix sort takes $\Theta(B)$ time where $B$ is total size of the key data.
- Have measured other sorts as function of #records.
- How to compare?
- To have $N$ different records, must have keys at least $\Theta(\lg N)$ long [why?]
- Furthermore, comparison actually takes time $\Theta(K)$ where $K$ is size of key in worst case [why?]
- So $N \lg N$ comparisons really means $N(\lg N)^2$ operations.
- While radix sort takes $B = N \lg N$ time.
- On the other hand, must work to get good constant factors with radix sort.

And Don’t Forget Search Trees

Idea: A search tree is in sorted order, when read in inorder.

- Need balance to really use for sorting [next topic].
- Given balance, same performance as heapsort: $N$ insertions in time $\lg N$ each, plus $\Theta(N)$ to traverse, gives

$$\Theta(N + N \lg N) = \Theta(N \lg N)$$

Summary

- Insertion sort: $\Theta(Nk)$ comparisons and moves, where $k$ is maximum amount data is displaced from final position.
  - Good for small datasets or almost ordered data sets.
- Quicksort: $\Theta(N \lg N)$ with good constant factor if data is not pathological. Worst case $O(N^2)$.
- Merge sort: $\Theta(N \lg N)$ guaranteed. Good for external sorting.
- Heapsort, treesort with guaranteed balance: $\Theta(N \lg N)$ guaranteed.
- Radix sort, distribution sort: $\Theta(B)$ (number of bytes). Also good for external sorting.