Hash-like Sorting
Hash Tables: Sorting

- Can create very fast sort programs using hash tables

- These sorts are not ‘general purpose’ but very efficient for certain situations
  - e.g. only works on positive integers in a particular range

- Examples:
  - Counting sort
  - Radix sort
Hash Table Sorting: **Counting Sort**

- Quickly sort **positive integer values** from a **limited range**
  - Count (tally) the occurrences of each value using HT
  - Recreate sorted values according to tally

- **Example:**
  - Sort 1,000 integer elements with values between 0 and 19
  - Count (tally) the occurrences of each value:
    - 0 - 47
    - 1 - 92
    - 2 - 12
    - 3 - 14
    - 4 - 32
    - 5 - 114
    - 6 - 16
    - 7 - 37
    - 8 - 41
    - 9 - 3
    - 10 - 36
    - 11 - 92
    - 12 - 43
    - 13 - 17
    - 14 - 132
    - 15 - 93
    - 16 - 12
    - 17 - 15
    - 18 - 63
    - 19 - 89

  - Recreate sorted values according to tally:
    - 47 zeros, 92 ones, 12 twos, ...
Counting Sort: Implementation

/* Sort an array of integers, each element no larger than max. */
void countSort(int * data, int n, int max) {
    int i, j, k;
    /* Array of all possible values. – it is the hash table */
    int *cnt = malloc((max + 1) * sizeof(int));

    for (k=0; k < max; k++) cnt[i] = 0; /* initialize */

    for (i = 0; i < n; i++) /* Count the occurrences */
        cnt[data[i]]++;

    /* Cnt holds the number of occurrences of numbers from 0 to max. */
    i = 0; /* Now put values back into the array. */
    for (j = 0; j <= max; j++) /* of each value. */
        for (k = cnt[j]; k > 0; k--) data[i++] = j;
}

/* Integer itself is the hash index */

What’s the complexity of this sort?
Radix Sort

- Has historical ties to *punch cards*
• It was far too easy to drop a tray of cards, which could be a disaster

• Convention became to put a sequence number on card, typically in positions 72-80

• Could then be resequenced by sorting on these positions

• A machine called a sorter used to re-sort the cards
Mechanical Sorter: Sorts a Single Column
Mechanical Sorter

- First sort on column 80
- Then collect piles, *keeping them in order*, and sort on column 79
- Repeat for each of the columns down to 72
- At the end, the result is completely sorted
- Try it

Data:

- 624  762  852  426  197  987  269
- 146  415  301  730  78   593
Radix Sort: Example

Data: 624 762 852 426 197 987 269 146 415 301 730 78 593

<table>
<thead>
<tr>
<th>Bucket</th>
<th>Pass1</th>
<th>Pass2</th>
<th>Pass3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>730</td>
<td>301</td>
<td>78</td>
</tr>
<tr>
<td>1</td>
<td>301</td>
<td>415</td>
<td>146 - 197</td>
</tr>
<tr>
<td>2</td>
<td>762 - 852</td>
<td>624 - 426</td>
<td>269</td>
</tr>
<tr>
<td>3</td>
<td>593</td>
<td>730</td>
<td>301</td>
</tr>
<tr>
<td>4</td>
<td>624</td>
<td>146</td>
<td>415 - 426</td>
</tr>
<tr>
<td>5</td>
<td>415</td>
<td>852</td>
<td>593</td>
</tr>
<tr>
<td>6</td>
<td>426 - 146</td>
<td>762 - 269</td>
<td>624</td>
</tr>
<tr>
<td>7</td>
<td>197 - 987</td>
<td>78</td>
<td>730 - 762</td>
</tr>
<tr>
<td>8</td>
<td>78</td>
<td>987</td>
<td>852</td>
</tr>
<tr>
<td>9</td>
<td>269</td>
<td>593 - 197</td>
<td>987</td>
</tr>
</tbody>
</table>

By keeping relative order from the previous pass, ties can be broken on subsequent passes.

Collision order resolved by first digit ordering.
Hash Table Sorting: Radix Sort

- Sorts positive integer values over any range
- Hash table size of 10 (0 through 9)
- Values are hashed according to their least significant digit (the “ones” digit)
- Values then rehashed according to the next significant digit (the tens digit) while keeping their relative ordering
- Process is repeated until we run out of digits
Time Complexity

- K passes (where K is number of digits)
- Each pass puts N elements in buckets
  - \( O(KN) \)
- How does this compare to \( O(N\log N) \) sorts?
Your Turn

• Complete worksheet #39 where you will simulate radix sort on the following values:
  • 742 247 391 382 616 872 453 925 732 142 562