Count Sort, Bucket Sort, Radix Sort
(Non-Comparison Sorting)

CSE 3318 – Algorithms and Data Structures
University of Texas at Arlington
Non-comparison sorts

- Count sort
- Radix sort
- Bucket sort (uses comparisons in managing the buckets)

- Comparison-based sorting: $\Omega(N\log N)$ lower bound
Lower-bounds on comparison-based sorting algorithms (Decision tree) – covered if time permits

- A correct sorting algorithm must be able to distinguish between any two different permutations of N items.
- If the algorithm is based on comparing elements, it can only compare one pair at a time.
- Build a binary tree where at each node you compare a different pair of elements, and branch left and right based on the result of the comparison.
  
  => each permutation must be a leaf and must be reachable

  Number of permutations for n elements: n!
  => tree will have at least n! leaves. => height ≥ lg (n!) =>

  height = Ω(nlgn) (b.c. lg(n!) = Θ(nlgn))

- The decision tree for any comparison-based algorithm will have the above properties => cannot take less than Θ(nlgn) time in the worst case.
Count Sort
Count Sort

Based on counting occurrences, not on comparisons. 
See animation.

Stable?
Adaptive?
Extra memory?
Runtime?

Does it work for ANY type of data (keys)?

Example 2: Sort an array of 10 English letters.

How big is the **Counts** array?
Θ(k)
(k = 26 possible key values letters)

Runtime: Θ(N+k)

---

<table>
<thead>
<tr>
<th>D</th>
<th>Rui</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Sam</td>
</tr>
<tr>
<td>C</td>
<td>Mike</td>
</tr>
<tr>
<td>C</td>
<td>Aaron</td>
</tr>
<tr>
<td>D</td>
<td>Sam</td>
</tr>
<tr>
<td>C</td>
<td>Tom</td>
</tr>
<tr>
<td>A</td>
<td>Jane</td>
</tr>
</tbody>
</table>

Counts ( => position range)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Sorted data
Based on counting occurrences, not on comparisons. See animation.

**Stable?** Yes

**Adaptive?** No

**Extra memory?** $\Theta(N+k)$

**Runtime?** $\Theta(N+k)$

For sorting only grades (no names), just counting is enough.

Does it work for ANY type of data (keys)?
No. E.g.: Sorting Strings, doubles

### Example 2: Sort an array of 10 English letters.

How big is the **Counts** array?

$\Theta(k)$

(k = 26 possible key values letters)

**Runtime:** $\Theta(N+k)$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st count occurrences</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

2nd cumulative sum: curr = prev+curr

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd cumulative sum: curr = prev+curr</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

(=2+0) (=2+3) (=5+2)

Sorted data; copy array, aux

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Init counts to 0

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Update with occurrences of each key

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

cumulative sum: counts[j] = counts[j-1] + counts[j];

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Copy array, aux:

- **t=6**
  - Original array, A:
    - D Rui
    - A Sam
    - C Mike
    - C Aaron
    - D Sam
    - C Tom
    - A Jane
  - A Jane

- **t=5**
  - A Jane
  - C Tom

- **t=4**
  - A Jane
  - C Tom
  - D Sam

Copy back from aux to A

- **t=6**
  - A Jane

- **t=5**
  - A Jane
  - C Tom

- **t=4**
  - A Jane
  - C Tom
  - D Sam

**Original array, A:**

<table>
<thead>
<tr>
<th>D</th>
<th>A</th>
<th>C</th>
<th>C</th>
<th>D</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rui</td>
<td>Sam</td>
<td>Mike</td>
<td>Aaron</td>
<td>Sam</td>
<td>Tom</td>
<td>Jane</td>
</tr>
</tbody>
</table>
void countSort(Records[] A, int N, int k)
{
    int[k] counts;
    Records[N] aux;
    for(j=0; j<k; j++) // init counts to 0
        counts[j]=0;
    for(t=0; t<N;t++) { // update counts
        idx = key2idx(A[t].key); //assume key2index is Θ(1)
        counts[idx]++;
    }
    for(j=1; j<k; j++) // cumulative sum
        counts[j]=counts[j]+counts[j-1];
    for(t=N-1; t>=0;t--) { // copy data in sorted order in aux array
        idx = key2idx(A[t].key); //assume key2index is Θ(1)
        counts[idx]--;
        aux[counts[idx]]=A[t]; //counts[idx] holds the index (+1) where A[t] will be in the sorted array
    }
    for(t=0; t<N;t++) // copy back in the original array
        A[t] = aux[t];
}
Count sort: comparison with Insertion sort and usage

- Compare the *time complexity* of Selection sort and Count sort for sorting
  - An array of 10 values in the range 1 to 10  vs
  - An array of 10 values in the range 501 to 1500.
  - An array of 1000 values in the range 1 to 10  vs
  - An array of 1000 values in the range 1 to 1000  vs

<table>
<thead>
<tr>
<th>Algorithm/problem</th>
<th>N = 10, k = ___ in range 1 to 10</th>
<th>N = 10, k = ____ in range 501 to 1500</th>
<th>N = 1000, k = __ in range 1 to 10</th>
<th>N = 1000, k = ____ in range 1 to 1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion sort (worst case) Θ(N²)</td>
<td>Θ(__________)</td>
<td>Θ(__________)</td>
<td>Θ(__________)</td>
<td>Θ(__________)</td>
</tr>
<tr>
<td>Count sort Θ(N+k)</td>
<td>Θ(__________)</td>
<td>Θ(__________)</td>
<td>Θ(__________)</td>
<td>Θ(__________)</td>
</tr>
</tbody>
</table>

When/for what data is count sort better?
- Is there any desired relation between k and N?
- Is there anything special (or needed) about the keys, in order for this to work?
- Can you think of data (keys) that count sort would not easily (possibly not at all) work for?
Count sort: comparison with Insertion sort

- Compare the *time complexity* of Selection sort and Count sort for sorting
  - An array of 10 values in the range 1 to 10 vs
  - An array of 10 values in the range 501 to 1500.
  - An array of 1000 values in the range 1 to 10 vs
  - An array of 1000 values in the range 1 to 1000 vs

<table>
<thead>
<tr>
<th>Algorithm/problem</th>
<th>N = 10, k = 10 in range 1 to 10</th>
<th>N = 10, k = 1000 in range 501 to 1500</th>
<th>N = 1000, k = 10 in range 1 to 10</th>
<th>N = 1000, k = 1000 in range 1 to 1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion sort (worst case)</td>
<td>$\Theta(N^2)$</td>
<td>$\Theta(10^2)$</td>
<td>$\Theta(1000^2)$</td>
<td>$\Theta(1000^2)$</td>
</tr>
<tr>
<td>Count sort $\Theta(N+k)$</td>
<td>$\Theta(10+10)$ = $\Theta(10)$</td>
<td>$\Theta(10+1000)$ = $\Theta(1000)$</td>
<td>$\Theta(1000+10)$ = $\Theta(1000)$</td>
<td>$\Theta(1000+1000)$ = $\Theta(1000)$</td>
</tr>
</tbody>
</table>

Best performing method is in red.
Note that this notation of $\Theta($number$)$ is not correct.
I am showing it like this to highlight the difference in the values of N and k.
Functions to convert key to integer

- **Function (no data structure used)**
  - **Char (the key is a char):**
    - char-'A'
    - E.g. 'D'- 'A' or grade-'A' (In C you can subtract 2 chars (it uses their ASCII code))
  - **Integer (the key is an int):**
    - index = key-min_key (index for current key is given by the formula key-min_key)
    - k = max_key-min_key+1 (possible different keys)
    - E.g. for keys (numbers) in range 501 and 1500:
      
      \[
      \text{index} = \text{key} - 501
      \]
      
      E.g. 501-501=0 so for key (number) 501, we go to index 0 in the counts array, and similar for 1500 we go to index 999 and for 700 we go to index 199 because: 1500-501=999, 700-501=199

- **Using a data structure – will require \(\Theta(k)\) extra space**
  - Unsorted array, A, with all k possible keys and linear search for a key in the array and return the index \(\rightarrow \text{TC}_{\text{key2idx}(A, k, \text{key})} = \Theta(k)\) (where \(|A|=k|\)
  - Sorted array, S, of all possible k keys, and binary search in S to find the index for a key
    \(\rightarrow \text{TC}_{\text{key2idx}(S, k, \text{key})} = \Theta(\log_2(k))\) (where \(|S|=k|\)
  - HashTable (HashMap), H, to map unique keys to indexes. HashTables will be covered later in the course. \(\rightarrow \text{TC}_{\text{key2idx}(H, k, \text{key})} = \Theta(1)\) on average
Count sort usage

- What data can count sort be applied for:
  - Values (keys) to be sorted must be integer values or chars (or be able to map to integers easily),
    - It DOES work for negative values as well. E.g. for temperatures in range [-20, 50], the formula is temp-min_temp = temp - (-20). E.g. (-20)-(-20)= (-20)+20 = 0, (-15)-(-20)=5, 50-(-20) = 70
  - What data (keys) will count sort NOT be able to handle?
    - Real numbers (float, double).
    - Strings (generates very large k and on-trivial to generate an index based on a string)
- When is count sort better than worst case insertion sort?
  - The number of all possible keys (k) should be asymptotically smaller than N^2 (written as k=O(N^2) ). Ideally k is at most proportional to N (written: k=O(N) )
- In a case when both insertion sort and count sort can be used, can you think of a reason why insertion sort would be preferred?
  - If the best case of insertion sort (data is almost sorted) is likely
  - Do not want to use the extra space of count sort
  - Want an adaptive algorithm.
- How does count sort compare to the BEST case of insertion sort?
  - Insertion sort is better: does not use extra space, and less work in general (smaller dominant term)
Least Significant Digit
Radix Sort
LSD Radix Sort

• Radix sort:
  – Addresses the problem count sort had with large range, \( k \).
  – Sorts the data by repeatedly sorting by digits
  – Versions based on what it sorts first:
    • LSD = Least Significant Digit first.
    • MSD = Most Significant Digit first – We will not cover it.

• LSD radix sort (Least Significant Digit)
  – sorts the data based on individual digits, starting at the Least Significant Digit (LSD).
  – It is somewhat counterintuitive, but:
    • It works (requires a stable sort for sorting based on the digits)
    • It is simpler to implement than the MSD version.
Sorting with radix sort

for each digit i = 0 to d-1  (0 is the least significant digit)
    count_sort A using digit i as the key

Known that values in A are in range: [0,999] => at most 3 digits
A: {708, 512, 131, 24, 742, 810, 107, 634}  (Original array)

    count_sort by units digit:
A: {810, 131, 512, 742, 24, 634, 107, 708}

    count_sort by the tens digit:
A: {107, 708, 810, 512, 24, 131, 634, 742}

    count_sort by the hundreds digit:
A: {024, 107, 131, 512, 634, 708, 742, 810}

Here (in base 10): n = 8, d = 3, k=10 (0,1,2,...9)

In base 2 (as numbers are stored): n=8, d=32 (for 4Bytes int), k=2 (bit: 0, 1)

How do you “extract” a digit from an integer in C? Use % and /.
LSD Radix Sort Complexity

• What are the quantities that affect the time & space complexity?

• What is the time and space complexity?

• Properties:
  - Stable?
  - Adaptive?
LSD Radix Sort Complexity

What are the quantities that affect the time and space complexity?

– n is the number of items
– k is radix (or the base)
– d: the number of digits in the radix-k representation of each item.

What is the time and space complexity?

• \( \Theta(d*(n+k)) \) time. (\( \Theta(nd+kd) \))
  – d * the time complexity of count sort
  – See the visualization at: https://www.cs.usfca.edu/~galles/visualization/RadixSort.html

• \( \Theta(n + k) \) space (for count sort).
  – \( \Theta(n) \) space for scratch array.
  – \( \Theta(k) \) space for counters/indices array.

• Properties (same as count sort):
  – Stable – yes (because count sort is)
  – Adaptive - no
Example 3

• Use Radix-sort to sort an array of 3-letter English words: [sun, cat, tot, ban, dog, toy, law, all, bat, rat, dot, toe, owl]
What type of data can be sorted with radix sort (that uses count sort)?

For each type of data below, say if it can be sorted with Radix sort and how you would do it.

• Integers
  – Positive ___yes________
  – Negative ___yes, but careful about the sign, reverse order of magnitude (b.c. -34 is smaller than -1 )
  – Mixed ___no____________

• Real numbers ___ no (count sort does not work for them)

• Strings ______ yes, but non trivial for different lengths _____________
  – (If sorted according to the strcmp function, where "Dog" comes before "cat", because capital letters come before lowercase letters). - yes
  – Consider “catapult” compared with “air” - careful as “cat” and “air” must be compared, not “ult” and “air”
More on RadixSort

- So far we have discussed applying Radix Sort to the data in the GIVEN representation (e.g. base 10 for numbers).

- A better performance may be achieved by changing the representation (e.g. using base 2 or base 5) of each number. Next slide gives a theorem that provides:
  - the formula for the time complexity of LSD Radix-Sort when numbers are in a different base and
  - How to choose the base to get the best time complexity of LSD_Radix sort. (But it does not discuss the cost to change from one base to another)

- The next slide is provided for completeness, but we will not go into details regarding it.
Tuning Radix Sort

Lemma 8.4 (CLRS): Given $n$ numbers, where each of them is represented using $b$-bits and any $r \leq b$, LSD Radix-sort with radix $2^r$, will correctly sort them in $\Theta( (b/r)(n+2^r) )$ if the stable sort it uses takes $\Theta(n+k)$ to run for inputs in the range 0 to $k$.

(Here the radix (or base) is $2^r$ and each new digit is represented on $r$ bits)

How to choose $r$ to optimize runtime:

• $r = \min\{b, \lfloor \log n \rfloor\}$  
  (intuition: compare $k$ with $n$ and use the log of the smaller one)
  - If $b \leq \log n$  \Rightarrow  $r = b$
  - If $b > \log n$  \Rightarrow  $r = \lfloor \log n \rfloor$

• Use as base $\min(2^u, 2^b)$, where $2^u$ is the largest power of 2 smaller than $n$ ($2^u \leq n \leq 2^{u+1}$)

What is the extra space needed for each case above?

$\Theta(n+2^r)$ (assuming it uses count sort as the stable sorting algorithm for each digit)
Bucket sort
Bucket Sort

Neither count sort nor radix sort can sort non-integer numbers. Why?

Bucket sort Idea:
- Split the RANGE of keys into smaller ranges/intervals.
  - Each interval will have a corresponding bucket.
- Copy each element in its corresponding bucket
- Sort each bucket (or maintain it sorted)
- Copy back in original array in order of buckets

Known: values in A are in range [0,1)

Array A:

<table>
<thead>
<tr>
<th>0.58</th>
<th>0.71</th>
<th>0.23</th>
<th>0.5</th>
<th>0.12</th>
<th>0.85</th>
<th>0.29</th>
<th>0.3</th>
<th>0.21</th>
<th>0.75</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.12</td>
<td>0.23</td>
<td>0.3</td>
<td>0.58</td>
<td>0.85</td>
<td>0.29</td>
<td>0.3</td>
<td>0.7</td>
<td>0.8</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Here all ‘buckets’ are shown as same size, but their size should depend on the number of items in them (e.g. linked list).
Bucket Sort – assumes numbers in [0, 1)

- Array, A, has \( n \) numbers in range [0,1).
  - If they are not in range [0,1), bring them to that range.

- Idea:
  - Make as many buckets as number of items
  - Place items in buckets
  - Sort each bucket
  - Copy from each bucket into the original array

```c
bucket_sort(int * A, int n) // assume numbers in A are in [0, 1)

Create array, B, of [linked] lists (bucket). Size of B will be n.
For each list in B:
    initialize it to be empty
For each elem in A,
    add elem to list  B[ floor(elem*n) ] // 0,1,...n-1
For each list in B:
    sort it with insertion sort
For each list in B:
    concatenate it (or copy back into A in this order).
    Destroy the list (if needed).
```

Some of the loops can be combined.
The given format makes the time complexity analysis easier.

**Time complexity:**
- Best: \( \Theta(n) \)
- Average: \( \Theta(n) \)
- Worst case: \( \Theta(n^2) \)
  (coming from worst case of insertion sort for longest list, size \( n \))
Worst case example: 
.1,.11,.1001,.15,...

**Space complexity:** \( \Theta(N) \)
\( (N \) pointes + \( N \) nodes\)

**Adaptive** – yes

**Stable** – yes (depending on where a new node is inserted in a linked list)
Bucket Sort

Example

A has 10 elements in range [0,1):
A = {0.58, 0.71, 0.23, 0.5, 0.12, 0.85, 0.29, 0.3, 0.21, 0.75}

Give both an example of the data and the time complexity for:
Best case:  A=[0.3,0.005, 0.1,0.2,]  O( N ) Explanation: one number per bucket
Worst case: A=[0.2, 0.22,0.20,0.2,0.26] O( N^2 ) Explanation: all numbers in the same bucket
Compute the bucket index

How will you compute the bucket \( idx \) for the bucket for element \( A[i] \) out of \( NB \) buckets? Let \( \text{min} = \text{min element from } A \) and \( \text{max} = \text{max element from } A \).

\[
\frac{idx}{NB} = \frac{(A[i] - \text{min})}{1 + \text{max} - \text{min}} \quad \Rightarrow \quad idx = \text{floor}\left(\frac{(A[i] - \text{min}) \times NB}{1 + \text{max} - \text{min}}\right)
\]
Range Transformations (Math review)

- Draw and show the mappings of the interval edges.

- \([0,1) \rightarrow [0,n)\]
  \[y = xn\]

- \([a,b) \rightarrow [0,1) \rightarrow [0,n)\]
  \[y = \frac{x-a}{b-a}, \quad z = yn\]

- \([a,b) \rightarrow [0,1) \rightarrow [s,t)\]
  \[z = y(t-s) + s\]

\[
\text{if } [a,b] \rightarrow [0,n): \\
z = \frac{x-a}{b-a+1}n \\
\text{As a check, see that } a- > 0 \text{ and } b- > y < n.
\]

\[
\text{Direct formula for } [a,b) \rightarrow [s,t): \\
z = \frac{x-a}{b-a}(t-s) + s \\
\text{As a check, see that } a- > s \text{ and } b- > t .
\]

- What this transformation is doing is: bring to origin \((a-0)\), scale to 1, scale up to new scale and translate to new location \(s\). The order matters! You will see this in Computer Graphics as well.
A searching problem

Extra material, if time permits
Money winning game:

- There is an array, A, with 100 items.
- The items are values in range [1,1000].
- A is sorted.
- Values in A are hidden (you cannot see them).
- You will be given a value, val, to search for in the array and need to either find it (uncover it) or report that it is not there.
- You start with $5000. For a $500 charge, you can ask the game host to flip (uncover) an item of A at a specific index (chosen by you). You win whatever money you have left after you give the correct answer. You have one free flip.

<table>
<thead>
<tr>
<th>Value, val, you are searching for.</th>
<th>What index will you flip? (this is a TV show so indexes starts from 1, not 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>524</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>99</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Money winning game – Version 2 only specific indexes can be flipped.

- There is an array, A, with 100 items.
- The items are values in range [1, 100].
- A is sorted.
- Values in A are hidden (you cannot see them).
- You will be given a value, val, to search for in the array and need to either find it (uncover it) or report that it is not there.
- You start with $5000. For a $500 charge, you can ask the game host to flip (uncover) an item of A at a specific index (chosen by you). You win whatever money you have left after you give the correct answer. You have one free flip.

<table>
<thead>
<tr>
<th>Value, val, you are searching for.</th>
<th>What index will you flip? 1?, 10?, 25?, 50?, 75?, 90?, 100?</th>
</tr>
</thead>
<tbody>
<tr>
<td>524</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>99</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Interpolated Binary Search

- Similar to binary search, but I want an ‘educated guess’.

- E.g. given sorted array, A, of 100 numbers in range \([0,1000]\), if you search in A for the value below, what index will you look at first?
  
  Assume it is a money-winning game and that for each trial/question, you loose some of the prize money. Which indexes would you pick?
  
  - 524 Look at index: 1?, 10?, 25?, 50?, 75?, 90?, 100?
  - 100 Look at index: 1?, 10?, 25?, 50?, 75?, 90?, 100?
  - 10 Look at index: 1?, 10?, 25?, 50?, 75?, 90?, 100?

Direct formula for \([a,b) \rightarrow [s,t)\):

\[
z = \frac{x-a}{b-a}(t-s) + s = (x-a)\frac{t-s}{b-a} + s
\]

As a check, see that \(a- > s\) and \(b- > t\).

Here: value range \([Mn, Mx]\), index range \([s, t]\). Use the \([Mn, Mx] \rightarrow [s, t]\) transformation and use for \(x\) the value you are searching for. The result, \(z\), is the index, \(i\), you are looking for.
Next level ...

- Let’s assume you can play the actual game.
- Can you write a program to play this game instead of you?
  - What will be the program inputs?
  - Give an algorithm for it.
  - Check that the algorithm has the same behavior as the human (do you flip the same indexes?) for specific examples.
- Check border cases
  - What border cases would you check for?
    - Value not in A / indexes cross over
    - Value at the edge (first or last in array)
  - Can you construct an example for each one of them?