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: $\Omega(N\lg N)$ lower bound

Lower-bounds on comparison-based sorting algorithms (Decision tree)

- 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.

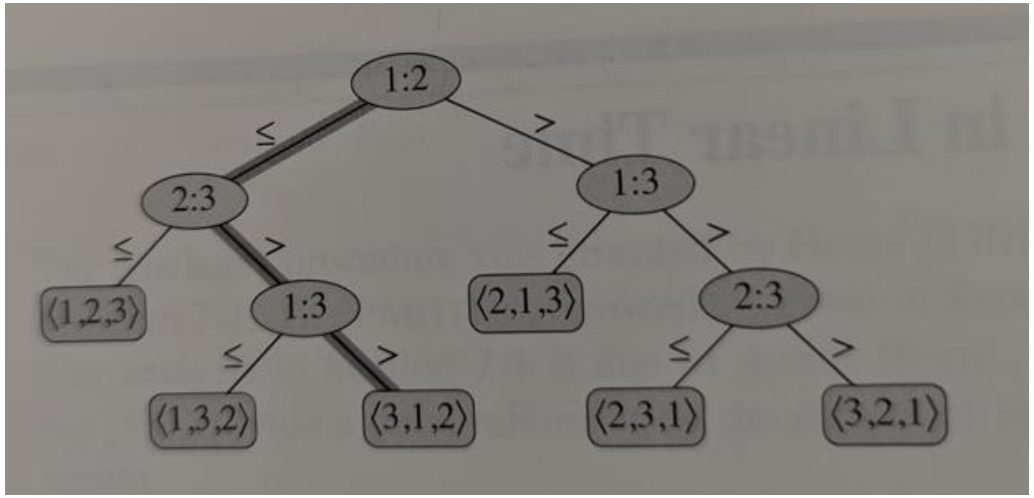
=> each permutation must be a leaf and must be reachable

Number of permutations for n elements: n!

=> tree will have at least n! leaves. => height ≥ lg (n!) =>

height = $\Omega(n\lg n)$ (b.c. $\lg(n!) = \Theta(n\lg n)$)

– The decision tree for any comparison-based algorithm will have the above properties => cannot take less than $\Theta(n\lg n)$ time in the worst case.



Count Sort

Count sort

- Used to sort when keys are integers in the range [0,k]
- E.g. Sort the given numbers by the UNITS digit (the last digit), in increasing order.

  A = {70**8**, 51**2**, 13**1**, 2**4**, 74**2**, 81**0**, 10**7**, 63**4**}

  want:

  A = {81**0**, 13**1**, 51**2**, 74**2**, 2**4**, 63**4**, 10**7**, 70**8**}

- Here
  - N = ___
  - k = ___

- Idea: if we know the count of each item, we can find the index where each item will be at in the sorted array

Count Sort

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

Stable?

Adaptive?

Extra memory?

Time Complexity?

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

3 Rui	0 Sam	2 Mike	2 Aaron	3 Sam	2 Tom	0 Jane

Counts (=> position range)

Sorted data

0	1	2	3	4	5	6

Count Sort

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

Stable? Yes

Adaptive? No

Extra memory? Θ(N+k)

Time Complexity? Θ(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

←

3 Rui	0 Sam	2 Mike	2 Aaron	3 Sam	2 Tom	0 Jane

1st count occurrences

0	1	2	3
2	0	3	2

2nd prefix sum: curr = prev+curr

0	1	2	3
2	~~0~~ **2** (=2+0)	~~3~~ **5** (=2+3)	~~2~~ **7** (=5+2)

Sorted data; copy array

0	1	2	3	4	5	6

Init counts to 0

0	1	2	3
0	0	0	0

Update with occurrences of each key

0	1	2	3
2	0	3	2

prefix sum: `counts[j]=counts[j-1]+counts[j];`

0	1	2	3
2	~~0~~ **2** (=2+0)	~~3~~ **5** (=2+3)	~~2~~ **7** (=5+2)

REPEAT

0	1	2	3
1	2	5	7

0	1	2	3
1	2	**4**	7

0	1	2	3
1	2	4	**6**

Original array, A:

0	1	2	3	4	5	6
3 Rui	0 Sam	2 Mike	2 Aaron	3 Sam	2 Tom	0 Jane

sorted_copy array:

t=6

0	*1*	2	3	4	5	6
	0 Jane					

t=5

0	1	2	3	*4*	5	6
	A Jane			***2 Tom***		

t=4

0	1	2	3	4	5	*6*
	A Jane			C Tom		***3 Sam***

Copy back from copy to A

Count sort (for an array of integers)

// Assume array A has integers in the range [0,k]

```
void countSort(int * A, int N, int k){
   int counts[k+1];
   int sorted_copy[N];  int j,t;
   for(j=0; j<=k; j++)  // init counts to 0
      counts[j]=0;
   for(t=0; t<N;t++){  // update counts
      counts[A[t]]++;
   }
   for(j=1; j<=k; j++)  // prefix sum
      counts[j]=counts[j]+counts[j-1];
   for(t=N-1; t>=0;t--){  // copy data in sorted order in sorted array
      counts[A[t]]--;
      sorted_copy[counts[A[t]]] = A[t];  //counts[A[t]] 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] = sorted_copy[t];
}
```

TC = ____

SC = ____

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 0 to 9 vs
  - An array of 10 values in the range 501 to 1500. - skip
  - An array of 1000 values in the range 0 to 9 vs
  - An array of 1000 values in the range 0 to 999 vs

Algorithm/ problem	**N = 10, k = ___ In range 0 to 9**	**N = 10, k = _____ In range 501 to 1500**	**N = 1000, k = __ In range 0 to 9**	**N = 1000, k = ____ In range 0 to 999**
Insertion sort (worst case) $\Theta(N^2)$	Θ(___________)	Θ(____________)	Θ(____________)	Θ(____________)
Count sort $\Theta(N+k)$	Θ(___________)	Θ(____________)	Θ(____________)	Θ(____________)

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, 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 0 to 9 vs
  - An array of 10 values in the range 501 to 1500. -skip
  - An array of 1000 values in the range 0 to 9 vs
  - An array of 1000 values in the range 0 to 999 vs

Algorithm/ problem	**N = 10, k = 10 In range 0 to 9**	**N = 10, k = 1000 In range 501 to 1500**	**N = 1000, k = 10 In range 0 to 9**	**N = 1000, k = 1000 In range 0 to 999**
Insertion sort (worst case) $\Theta(N^2)$	$\Theta(10^2)$	**$\Theta(10^2)$**	$\Theta(1000^2)$	$\Theta(1000^2)$
Count sort $\Theta(N+k)$	**$\Theta(10+10)$ $=\Theta(10)$**	$\Theta(10+1000)$ $=\Theta(1000)$	**$\Theta(1000+10)$ $=\Theta(1000)$**	**$\Theta(1000+1000)$ $=\Theta(1000)$**

Best performing method is in **red**.
Note that this notation of Θ(number) is not correct.
I am showing it like this to highlight the difference in the values of N and k.

Example 2

Sort an array of 10 English letters.

How big is the **counts** array?

TC: Θ(N+

Example 2

Sort an array of 10 English letters.

How big is the **counts** array?

Θ(k)

(k = 26 possible key values letters)

TC: Θ(N+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:

      index = 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 Θ(k) extra space
  - Unsorted array, L, with all k possible keys and linear search for a key in the array and return the index -> $TC_{key2idx(A,\ k,\ key)} = \Theta(k)$ (where |L|=k)
  - Sorted array, S, of all possible k keys, and binary search in S to find the index for a key

    => $TC_{key2idx(S,k,\ key)} = \Theta(\log_2(k))$ (where |S| = k)
  - HashTable (HashMap) , H, to map unique keys to indexes. HashTables will be covered later in the course. => $TC_{key2idx(H,\ k,\ key)} = \Theta(1)$ amortized
    - If can guarantee that no two different keys are hashed to the same index

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 it is non-trivial to uniquely map a string to an index)
- 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.
  - k is very big
- How does count sort compare to the BEST case of insertion sort?
  - Insertion sort is better: does not use extra space, and does less work in general (smaller dominant term)

Least Significant Digit Radix Sort

LSD Radix Sort

- LSD radix sort (Least Significant Digit)
  - Addresses the problem count sort has with large range, k.
  - sorts the data based on individual digits, starting at the Least Significant Digit (LSD).
  - requires a stable sort for sorting based on the digits

Sorting with radix sort

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

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)

sort by units

A: {81**0**, 13**1**, 51**2**, 74**2**, 2**4**, 63**4**, 10**7**, 70**8**}

sort by tens

A: {1**0**7, 7**0**8, 8**1**0, 5**1**2, **2**4, 1**3**1, 6**3**4, 7**4**2}

sort by hundreds

A: {**0**24, **1**07, **1**31, **5**12, **6**34, **7**08, **7**42, **8**10}

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)

Implementation: 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 and 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
  - All positive __yes________
  - All negative ___yes, but careful about the sign, reverse order of magnitude (b.c. -34 is smaller than -1), there will be issues with % if working in base 10
  - 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 - Extra

- 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 TC:

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

TC practice

- Analyze time complexity to place N items in an array maintaining it sorted after each item is added.

- Same question for a single linked list.

Bucket Sort - Idea

Bucket sort Idea:

- Split the RANGE of keys into smaller ranges/intervals.
  - Number of intervals = N, number of items in the array.
  - Each interval will have a corresponding bucket.
- Copy each element in its corresponding bucket in sorted order. (Maintain the bucket sorted.)
- Copy back in original array in order of buckets

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

Array A:

0.58	0.71	0.23	0.5	0.12	0.85	0.29	0.3	0.21	0.8

0	1	2	3	4	5	6	7	8	9
[0.0, 0.1)	[0.1, 0.2)	[0.2,0.3)	[0.3,0.4)	[0.4,0.5)	[0.5,0.6)	[0.6,0.7)	[0.7,0.8)	[0.8,0.9)	[0.9,1)
	0.12	0.21 0.23 0.29	0.3		0.5 0.58		0.71	0.8 0.85	

Array A:

0.12	0.21	0.23	0.29	0.3	0.5	0.58	0.71	0.8	0.85

Here all 'buckets' are shown as same size, but their size should depend on the number of items in them (e.g. linked list).
See animation : https://www.cs.usfca.edu/~galles/visualization/BucketSort.html

Index calculation

- Given an element in the array, A, how do we find the index, idx, of the bucket it should go to?
  - idx is in range 0, 1,2,…,N-1
- [0,1) case
  - when known that each element, elem, in A is in range [0,1)
  - **idx = floor(N*elem)**
- general case:
  - works when elements in A are in any range
  - find min and max values in A
  - ***idx = floor(*** $\frac{N*(elem-min\_A)}{1+max\_A-min\_A}$

Goal:

N = ________

number of buckets = _________

indexes for buckets: ____________

Map: elem -> index

min -> ___________

max -> ___________

Index calculation - special case [0,1)

N = _______

number of buckets = ________

indexes for buckets: ___________

- given: each element, elem, in A is in range [0,1)
- idx = floor(N*elem)

elem: ____ index____ calc ____________________

elem: ____ index____ calc ____________________

elem: ____ index____ calc ____________________

elem: ____ index____ calc ____________________

elem: ____ index____ calc ____________________

Exercise 1:

It is given that A has elements in range [0,1).

A = {0.9, 0.71, 0.23, 0.05}

Use formula:___________ N = ____

Index calculation - general case

- works when elements in A are in any range
- find min and max values in A
- *idx = floor(*$\frac{N*(elem - \min\_A)}{1 + \max\_A - \min\_A}$
- coding issues
  - ________________
  - ________________

Exercise 2:

A = {2, 9, 7, 1, 8}, nothing else said about A.

Use formula:___________ N = ____

N = _______

number of buckets = ________

indexes for buckets: ___________

Map:

min -> __________

max -> __________

elem: ____ index____ calc ___________________

elem: ____ index____ calc ___________________

elem: ____ index____ calc ___________________

elem: ____ index____ calc ___________________

elem: ____ index____ calc ___________________

Bucket Sort

- Array, A, has *n* numbers.
  - version in the CLRS textbook assumes numbers in A are in the range [0,1)
  - See animation: https://www.cs.usfca.edu/~galles/visualization/BucketSort.html
- Idea:
  - Make as many buckets as number of items
  - Place items in buckets . Maintain sorted buckets.
  - Copy from each bucket into the original array

*bucket_sort(int * A, int N)*

Create array, B, of linked lists (bucket). Size of B will be N.

For each list in B:

initialize it to be empty

Compute min_A, max_A

For each elem in A,

insert elem in ***sorted list*** *B[idx] where idx = floor(*$\frac{N*(elem-\min\_A)}{1+\max\_A-\min\_A}$*)*

*(if numbers in A are in [0, 1) you can use: idx = floor(elem*N)*

For each list in B:

concatenate it (or copy back into A in this order).

Destroy the list (if needed).

Exercise 3:
Give both an example of the data and the time complexity for:
Best case: A=[___, ___, ___, ___]
O( ) Explanation:
Worst case: A=[___, ___, ___, ___, ___]
O( ) Explanation:

Time complexity:
-Best: Θ(___)
-Average: Θ(___)
-Worst case : Θ(___)

Worst case example:
____

Space complexity: **Θ(____)**
(from:)

Adaptive – ____
Stable – ____

Bucket Sort - Practice

Exercise 3:

Give both an example of the data and the time complexity for:

Best case: A=[___, ___, ___, ___] O( ) Explanation:

Worst case: A=[___, ___, ___, ___, ___] O( ) Explanation:

Bucket Sort

- Array, A, has *n* numbers.
  - version in the CLRS textbook assumes numbers in A are in the range [0,1)
  - See animation: https://www.cs.usfca.edu/~galles/visualization/BucketSort.html
- Idea:
  - Make as many buckets as number of items
  - Place items in buckets . Maintain sorted buckets.
  - Copy from each bucket into the original array

*bucket_sort(int * A, int N)*

Create array, B, of linked lists (bucket). Size of B will be N.

For each list in B:

initialize it to be empty

Compute min_A, max_A

For each elem in A,

insert elem in sorted list B[idx] where idx = $floor(\frac{N*(elem-\min(A))}{1+\max(A)-\min(A)})$

*(if numbers in A are in [0, 1) you can use: idx = floor(elem*N)*

For each list in B:

concatenate it (or copy back into A in this order).

Destroy the list (if needed).

Time complexity:
-Best: Θ(N)
-Average: Θ(N)
-Worst case : $\Theta(N^2)$
(coming from worst case of insertion sort for longest list, size N)
Worst case example (for N=10):
0.1, 0.11, 0.1001, 0.15,…

Space complexity: **Θ(N)**
(from: N pointes + N nodes)

Adaptive – no
Stable – yes (depending on where a new node is inserted in a linked list)

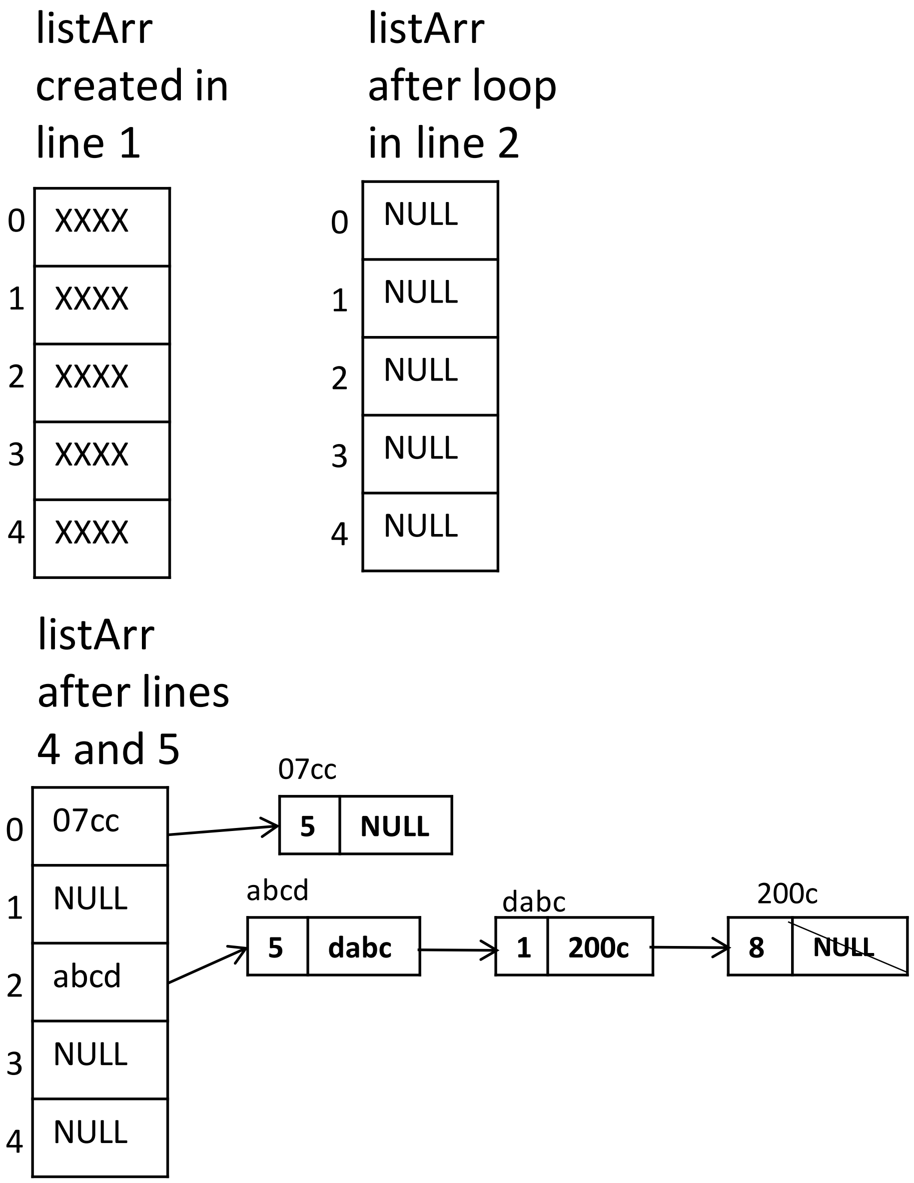
Array of linked lists – simple example

```
/* assume new_node(), array_2_list(), and
print_list_horiz() are the ones from the
provided linked list implementation. */
typedef struct node * nodePT;
truct node {
    int data;
    struct node * next;
};

int arr[] = {5,1,8};
nodePT listArr[5]; //1
// size: 5*sizeof(memory address) = 5*8B=40B
// use listArr[j] like any variable L or head (of type nodePT)

// set every pointer/list to NULL
for(j=0; j<5; j++)  {  // 2
       listArr[j]=NULL;
}
listArr[0] = new_node(5);     //4
listArr[2] = array_2_list(arr, 3); //5
print_list_horiz(listArr[0]);
  // Practice: create a new node with value 2
  // and insert it at the beginning of
  // list at index 0. Update drawing.
for(j=0; j<5; j++) {
       destroy_list(listArr[j]);   //11
}
```

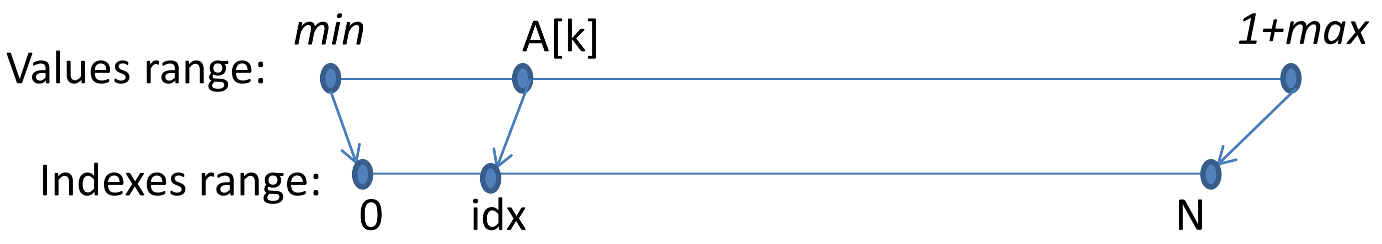
Drawings of listArr at different stages in the program.

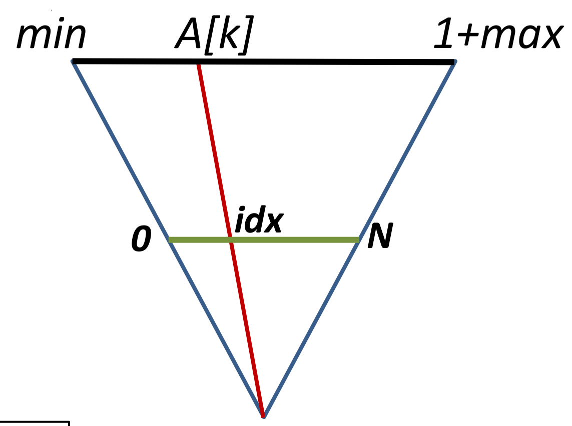


Optional: Intuition for computing the bucket index

How will you compute the index, *idx,* for the bucket for element A[k] out of N buckets?
Let
min = min element from A and
max = max element from A.

Use N buckets => indexes: 0,1,2,..., (N-1)
We want to map *min* to index 0 and *max* to index (N-1)





$$\frac{idx}{N} = \frac{(A[k]-min)}{1+max-min} \Rightarrow$$

$$idx = floor\left(\frac{(A[k]-min)*N}{1+max-min}\right)$$

How does this formula compare with the one from https://www.cs.usfca.edu/~galles/visualization/BucketSort.html
Do they make any assumptions about the data in the array?
Is there any data that that formula would not work for?

Range Transformations (Math review)

- Draw and show the mappings of the interval edges.

- [0,1) -> [0,n)

$$y = xn$$

- [a,b) -> [0,1) -> [0,n)

$$y = \frac{x-a}{b-a} \qquad z = yn$$

- [a,b) -> [0,1) -> [s,t)

$$z = y(t-s)+s$$

if $[a,b] \text{ -> } [0,n)$:

$$z = \frac{x-a}{b-a+1}n$$

As a check, see that $a\text{-}>0$ and $b\text{-}> y<n$.

Direct formula for $[a,b) \text{ -> } [s,t)$:

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

As a check, see that $a\text{-}>s$ and $b\text{-}>t$.

  - 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.

Generic count sort Algorithm

// Assume: k = number of different possible keys,
// key2idx(key) returns the index for that key (e.g. 0 for letter A)
// Records is a typename for a struct that has a 'key' field

```
void countSort(Records* A, int N, int k){
   int counts[k];
   Records copy[N];
   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 copy array
      idx = key2idx(A[t].key);  //assume key2index is Θ(1)
      counts[idx]--;
      copy[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] = copy[t];
}
```