Revised: 5/19/2024

Sorting III

Michael C. Hackett

Assistant Professor, Computer Science

COMMUNITY COLLEGE OF PHILADELPHIA

Lecture Topics

- Counting Sort
- Radix Sort
- Bucket Sort

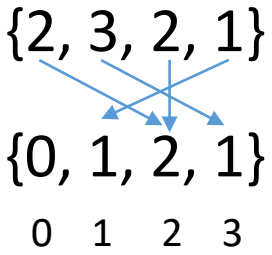
Non-Comparative Sorting Algorithms

- **Non-comparative sorting algorithms** sort the contents of a sequence using the *characters* of values to be sorted.
  - Unlike comparative sorting algorithms, which base their sorting on making relational (<, >, etc.) comparisons.

Counting Sort

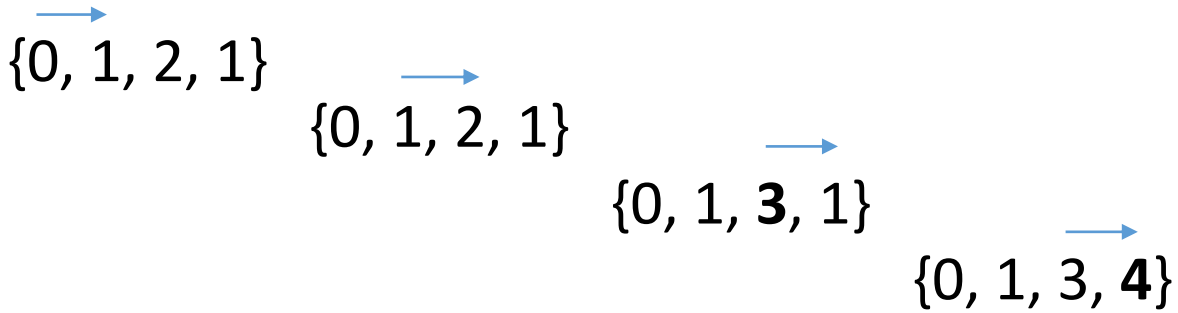
- The Counting Sort is a **non-comparative** sorting algorithm that uses a separate “counting” array for determining how many times an integer appears in an unsorted sequence.

- The counting array uses its own indexes to hold the totals of that corresponding value in the unsorted sequence.
  - For example, for the unsorted array {2, 3, 2, 1} a Counting Sort’s counting array would contain {0, 1, 2, 1}
    - Zero 0’s, One 1, Two 2’s, One 3



Counting Sort

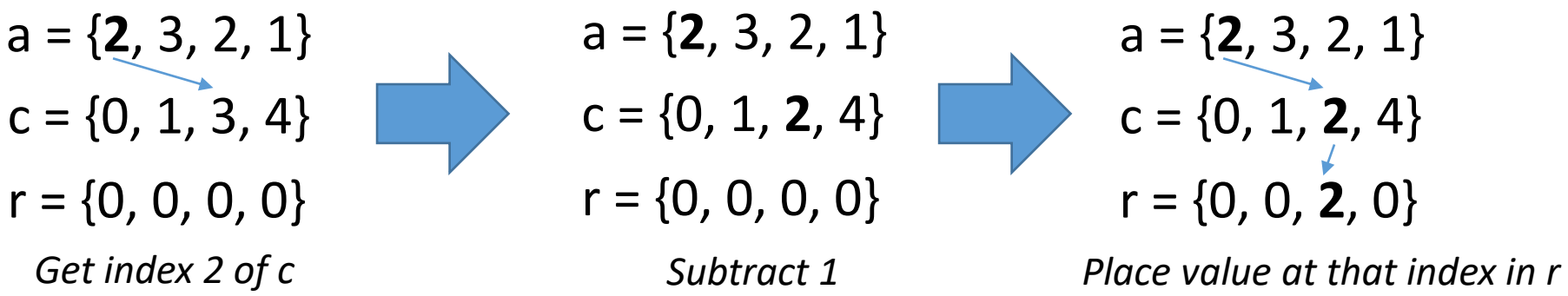
- The values in the counting array are summed linearly.
  - The counting array previously shown would become {0, 1, 3, 4}

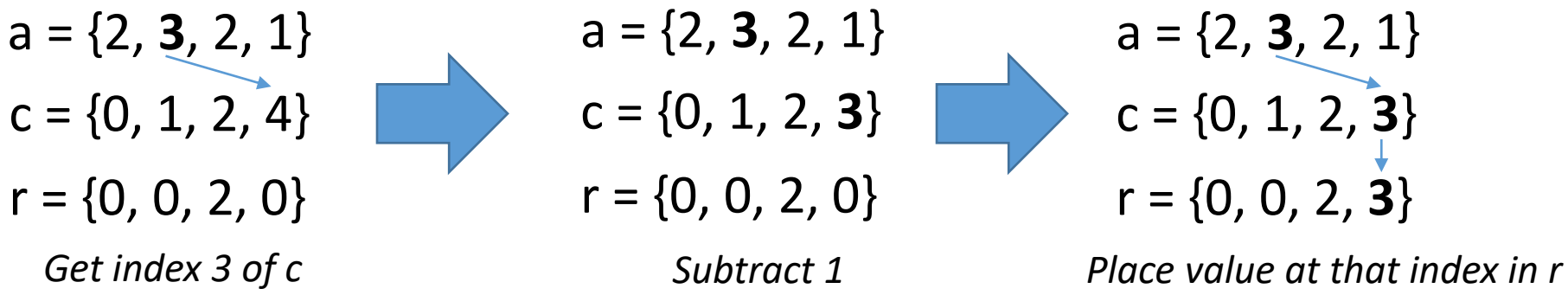


Counting Sort

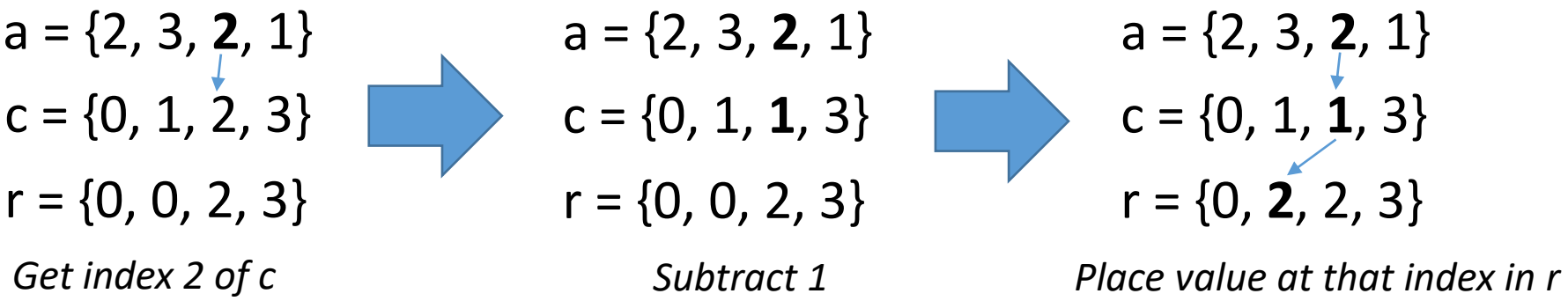
- The counting array (c) is then used to determine the placement of each unsorted element.
- Each value is retrieved from the original array.
  - 1 is subtracted from corresponding index of the counting array.
    - If 2 I retrieved from the original array, 1 is subtracted from index 2 of the counting array.
- The new value at the corresponding index is the index where the value is placed in the resulting array (r).
- The sorted array (r) is then copied over to the original array (a)

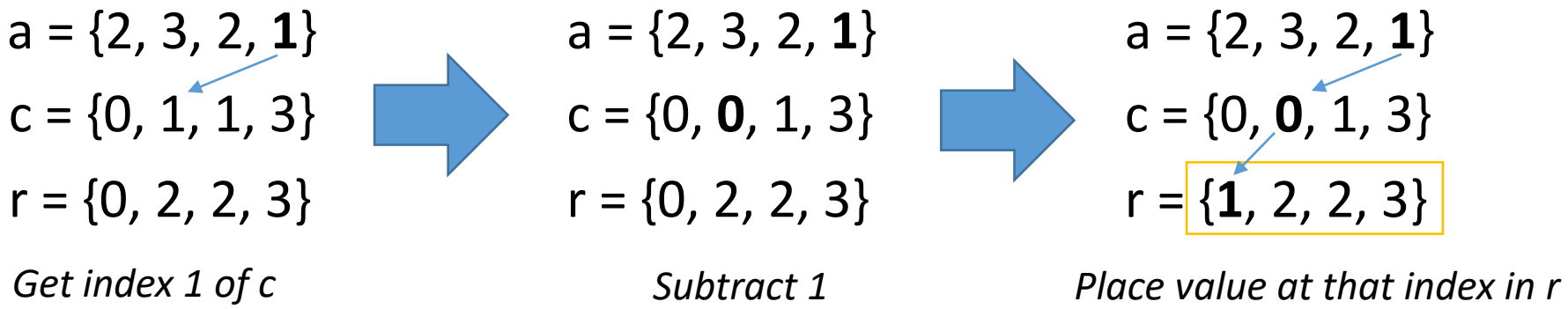
Counting Sort





Counting Sort





Counting Sort

```
public void countingSort(int[] a) {
    int[] result = new int[a.length];
    int max = a[0];
    for(int i = 1; i < a.length; i++) {
        if(a[i] > max) {
            max = a[i];
        }
    }
    int[] c = new int[max + 1];
    for(int i = 0; i < a.length; i++) {
        int value = a[i];
        c[value] += 1;
    }
    for(int i = 1; i < max + 1; i++) {
        c[i] += c[i-1];
    }
    for(int i = 0; i < a.length; i++) {
        int temp = a[i];
        c[temp] -= 1;
        result[c[temp]] = temp;
    }
    for(int i = 0; i < a.length; i++) {
        a[i] = result[i];
    }

}
```

Determine max

Increment the element at index a[i] in c

Linearly sum the elements of c

Subtract 1 from c[a[i]] and
Put a[i] in result[c[a[i]]]

Copy each element from result to a

Radix Sort

- The Radix Sort is another non-comparative sorting algorithm that is closely related to the Counting Sort algorithm.

- The Radix Sort sorts an sequence of numbers, going digit-by-digit of each value, starting with the least-significant digit to the most-significant digit.
  - The algorithm uses a modified Counting Sort to perform the actual sorting.

Radix Sort

Sorting by the 1's place → Sorting by the 10's place

a = {45, 32, 7, 19}
c = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

a = {45, 35, 7, 19}
c = {0, 0, 0, 0, 0, 2, 0, 1, 0, 1}

c = {0, 0, 0, 0, 0, 2, 2, 3, 3, 4}

a = {35, 45, 7, 19}

a = {35, 45, 7 19}
c = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

a = {35, 45, 07, 19}
c = {1, 1, 0, 1, 1, 0, 0, 0, 0, 0}

c = {1, 2, 2, 3, 4, 4, 4, 4, 4, 4}

a = {7, 19, 35, 45}

Radix Sort

```
public void radixSort(int[] a) {
    int max = getMax(a);
    for (int i = 1; max/i > 0; i *= 10) {
        countingSort(a, i);
    }
}


public int getMax(int[] a) {
    int max = a[0];
    for (int i = 1; i < a.length; i++) {
        if (a[i] > max) {
            max = a[i];
        }
    }
    return max;
}
```

```
public void countingSort(int a[], int i) {
    int[] temp = new int[a.length];
    int[] digitCount = new int[10];

    for (int j = 0; j < a.length; j++) {
        digitCount[(a[j] / i) % 10]++;
    }
    for (int j = 1; j < 10; j++) {
        digitCount[j] += digitCount[j - 1];
    }

    for (int j = length - 1; j >= 0; j--) {
        int index = digitCount[(a[j] / i) % 10] - 1;
        temp[index] = a[j];
        digitCount[(a[j] / i) % 10]--;
    }

    for (int j = 0; j < length; j++) {
        a[j] = temp[j];
    }
}
```

Radix Sort

```
public void radixSort(int[] a) {
    int max = getMax(a);
    for (int i = 1; max/i > 0; i *= 10) {
        countingSort(a, i);
    }
}
```

Find the largest number; This will determine how many times the below loop repeats

Iterates from 1, to 10, to 100, and so on… based on the largest number

Passes the array, its length, and the current position to sort to a Counting Sort algorithm

Radix Sort

```
public void countingSort(int a[], int i) {
    int[] temp = new int[a.length];
    int[] digitCount = new int[10];

    for (int j = 0; j < a.length; j++) {
        digitCount[(a[j] / i) % 10]++;
    }
    for (int j = 1; j < 10; j++) {
        digitCount[j] += digitCount[j - 1];
    }

    for (int j = length - 1; j >= 0; j--) {
        int index = digitCount[(a[j] / i) % 10] - 1;
        temp[index] = a[j];
        digitCount[(a[j] / i) % 10]--;
    }

    for (int j = 0; j < length; j++) {
        a[j] = temp[j];
    }
}
```

Counting array with a length of 10 (indexes 0-9)

Finds the digit at the i position of each number in a, and adds one at that index in the counting array

Linearly sums the values in the counting array

Subtracts 1 from the value in the counting array, for each value in a (based on the current digit/position it is sorting for). Puts the at the calculated index in a temporary array. Decrements the index by one.

Copies all values from the temporary array to the actual array (replacing the original ordering)

Bucket Sort

- The Bucket Sort is a sorting algorithm that:
  - Distributes the values of a sequence into containers or “buckets”
  - Sorts the buckets
  - Concatenates the buckets into the final, sorted result.

- Each bucket will contain the values in a certain range.
  - For example, if the range of values to be sorted is 0-100…
  - There will be a bucket for values 0-10, a bucket for values 11-20, a bucket for values 21-30, and so on.

Bucket Sort

- We'll apply the bucket sort algorithm on the following array:

{23, 16, 8, 42, 4, 15}

- First, we decide how many buckets we want.
  - In this example, we will use three.

- Next, we find the largest value in the sequence.
  - In this example, it is 42

Bucket Sort

- Now, we distribute the values into their buckets

- Before we do, let's calculate the range of each bucket (for reference)
  - Each bucket's range: $\frac{M+1}{N}$
    - M = Largest Value (42)
    - N = Number of Buckets (3)

$\frac{42+1}{3} = 14.33 \sim 15$ (Round the result up)

Bucket Sort

- Each bucket has a range of 15
  - Bucket 0: Will contain values 0 – 14
  - Bucket 1: Will contain values 15 – 29
  - Bucket 2: Will contain values 30 – 44

- We calculate the bucket number for a value with the following formula:

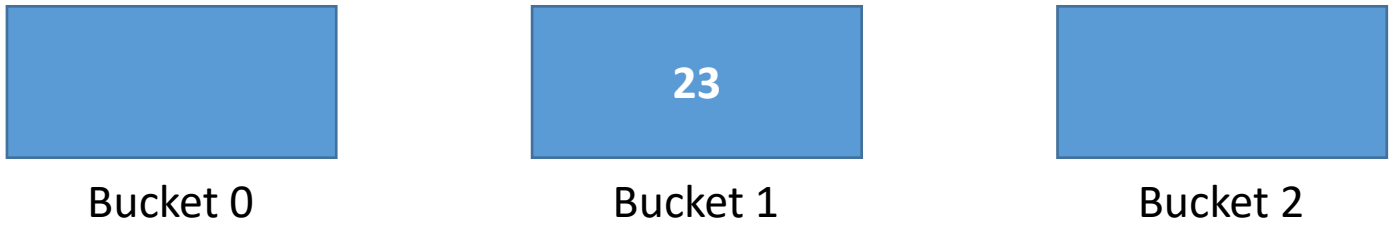
$$value * \frac{N}{M+1}$$

  - (Round the result down)

Bucket Sort

- $23 * \frac{3}{42+1} = 1.6 \sim 1$

{**23**, 16, 8, 42, 4, 15}

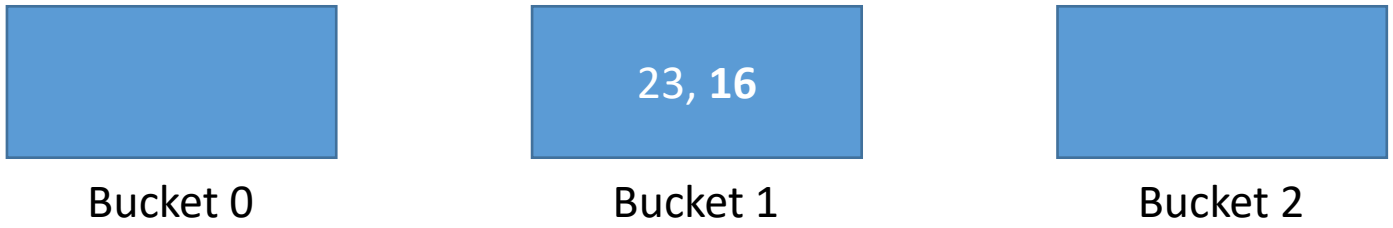


- The value 23 would be placed in bucket 1

Bucket Sort

- $16 * \frac{3}{42+1} = 1.1 \sim 1$

{23, **16**, 8, 42, 4, 15}

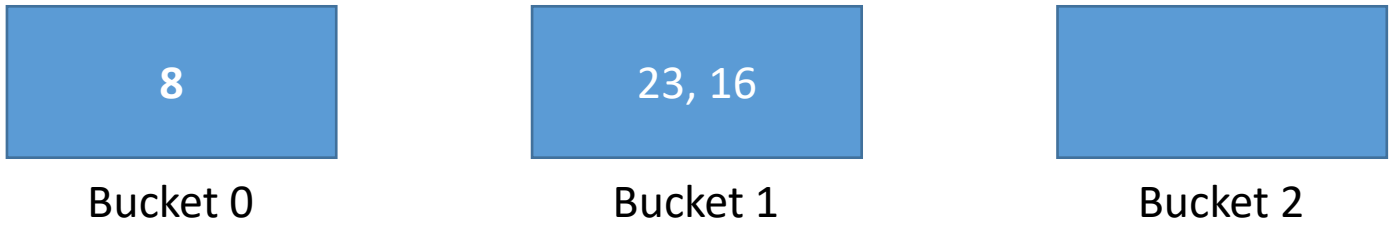


- The value 16 would be placed in bucket 1

Bucket Sort

- $8 * \frac{3}{42+1} = 0.5 \sim 0$

{23, 16, **8**, 42, 4, 15}

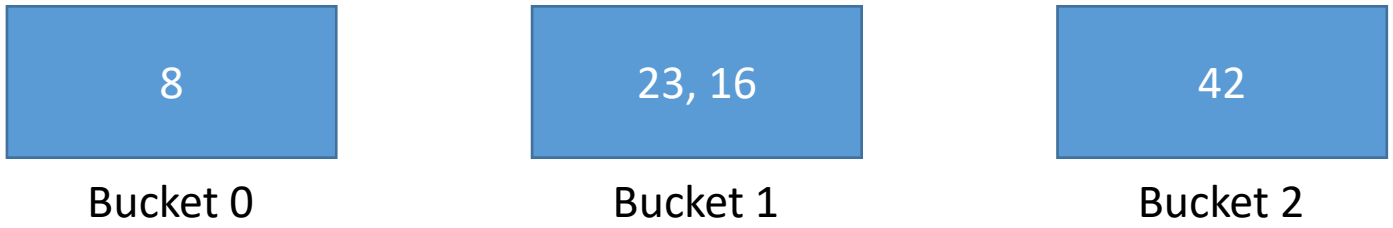


- The value 8 would be placed in bucket 0

Bucket Sort

- $42 * \frac{3}{42+1} = 2.9 \sim 2$

{23, 16, 8, **42**, 4, 15}

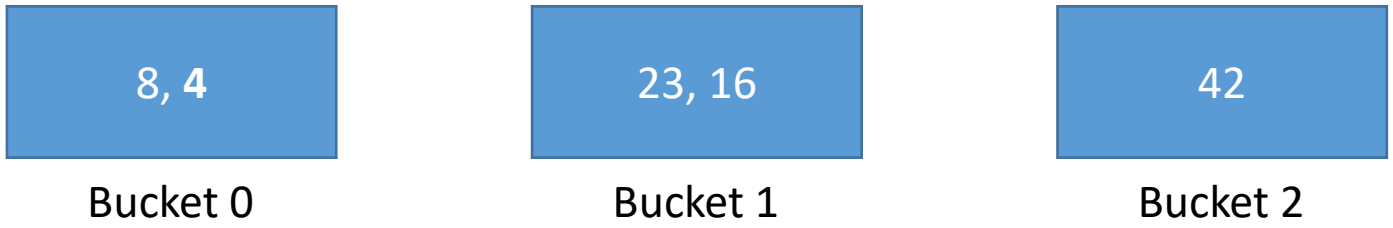


- The value 42 would be placed in bucket 2

Bucket Sort

- $4 * \frac{3}{42+1} = 0.3 \sim 0$

{23, 16, 8, 42, **4**, 15}

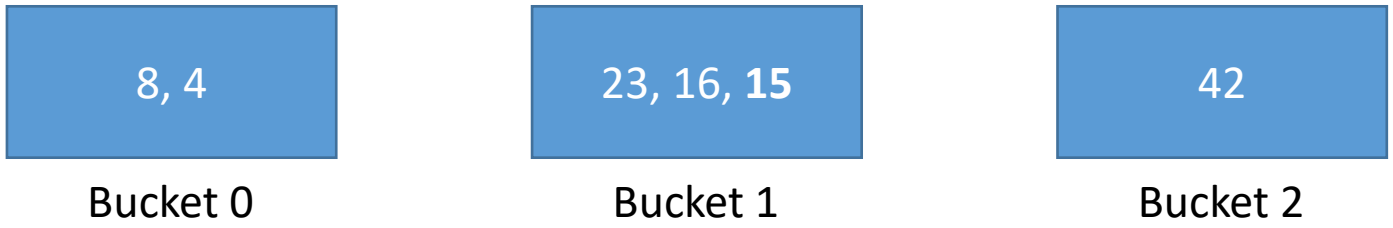


- The value 4 would be placed in bucket 0

Bucket Sort

- $15 * \frac{3}{42+1} = 1.04 \sim 1$

{23, 16, 8, 42, 4, **15**}



- The value 15 would be placed in bucket 1

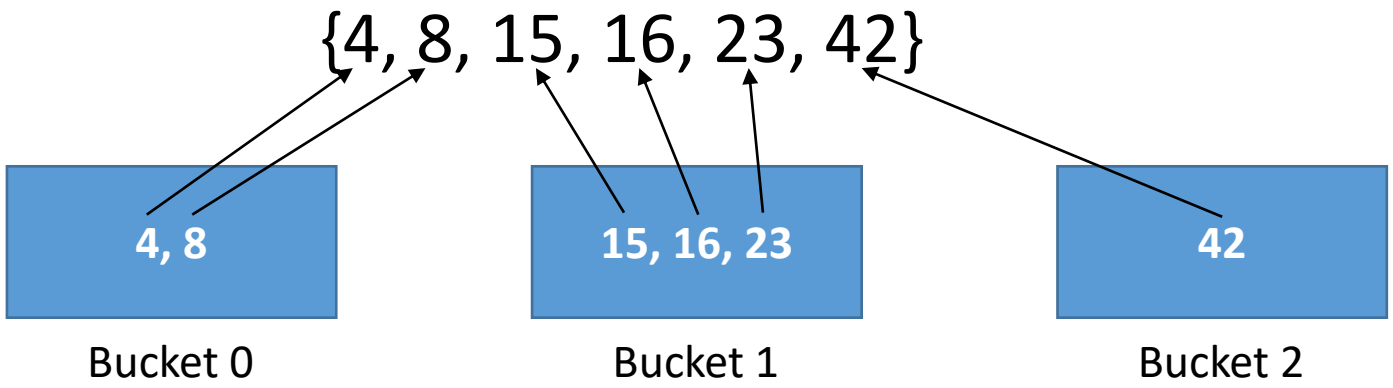
Bucket Sort

- Now, each bucket is sorted using a sorting algorithm, like insertion sort.

{23, 16, 8, 42, 4, 15}

Bucket Sort

- Finally, each value is placed back in the original array, beginning with the first bucket.



Bucket Sort

- We know the insertion sort will make, at most, $\sum_{i=1}^{n-1} i$ comparisons.
  - We'll change this sequence so it is in reverse order, which will make the algorithm to the most comparisons

{42, 23, 16, 15, 8, 4}

- To sort this sequence, the insertion sort alone will make…
  - $\sum_{i=1}^{6-1} i$ = 1 + 2 + 3 + 4 + 5 = 15 comparisons

Bucket Sort

- Now, we'll perform the bucket sort process on the same sequence.
  - The buckets and ordering of values would actually remain the same.

{42, 23, 16, 15, 8, 4}

Bucket Sort

- An insertion sort is performed on each bucket.

{42, 23, 16, 15, 8, 4}

Bucket 0 | Bucket 1 | Bucket 2

$\sum_{i=1}^{2-1} i$ = 1 comparison | $\sum_{i=1}^{3-1} i$ = 3 comparisons | $\sum_{i=1}^{1-1} i$ = 0 comparisons

- **4 Comparisons**

  (vs. 15 comparisons made by using the insertion sort alone)

Bucket Sort

```
public static void bucketSort(int[] a, int numBuckets) {
  Vector[] buckets = new Vector[numBuckets];
  for(int i = 0; i < buckets.length; i++) {
    buckets[i] = new Vector<Integer>();
  }

  int max = ModuleTools.getMax(a);

  for(int i = 0; i < a.length; i++) {
    int bIndex = a[i] * numBuckets / (max+1);
    buckets[bIndex].add(a[i]);
  }

  for(int i = 0; i < numBuckets; i++) {
    insertionSort(buckets[i]);
  }

  int index = 0;
  for (int i = 0; i < numBuckets; i++) {
    while(buckets[i].size() > 0) {
      a[index++] = (int)(buckets[i].get(0));
      buckets[i].remove(0);
    }
  }
}
```

An array of Vectors
(The Vectors will be our “buckets”)

Determine the max/largest value

Place each value into their correct bucket

Sort each bucket

Put each value from the buckets into the array