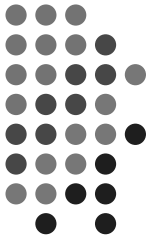
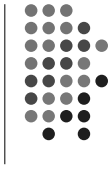
Sorting 7C

Non-comparison Sorts
Sorting in the Java API

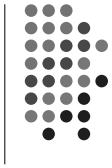


Comparison Sorts



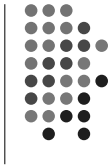
- All of the sorts we've seen so far are comparison sorts.
  - The order of the elements is determined by comparing two elements at a time.
- It has been proven that the worst-case order of complexity for comparison sorts is $\Omega(n \log n)$.
  - O gives an asymptotically upper bound on the efficiency.
  - Ω gives an asymptotically lower bound on the efficiency. (more about this in 15-211)
- But there are sorts that can sort in O(n)...
  ... they just don't use pair-wise comparisons

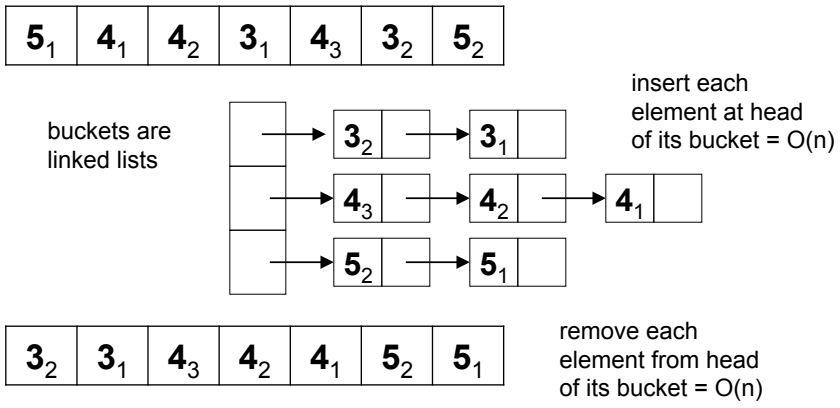
Bucket Sort



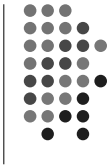
- Given an array of n elements that contain only k unique values (k < n), labeled n_1, n_2, ..., n_k such that $n_1 < n_2 < ... < n_k$.
- Create an array of k "buckets", one for each unique value.
- For each value in the array, move it into its corresponding bucket.
- Copy the data values from each bucket, n_1 to n_k, back into the array to sort the data.

Bucket Sort Example



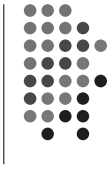


Sorts in the Java API



- In `Arrays` class:
  - `public static void sort(Object[] items)`
    - All objects must mutually Comparable.
    - Implemented with a modified merge sort in O(n log n)
    - Sort is stable.
- In `Collections` class:
  - `public static <T extends Comparable<T>> void sort(List<T> list)`
    - Same conditions as above.
    - Copies elements into an array and uses Arrays.sort

Sorts in the Java API



- In `Arrays` class:
  - `public static <T> void sort(T[] items, Comparator<? super T> comp)`
    - Another version allows a sort using a Comparator so ordering can be done on some other property other than the items' natural ordering.
    - For example: You might order strings not alphabetically, but instead by string length.
    - `comp` must be an object that implements the `Comparator` interface for type `T` or a superclass of type `T`.

`Comparator` Example

```
public class StringLengthComp
  implements Comparator<String> {
  public int compare(String s1, String s2) {
      return s1.length() - s2.length();
  }
}
```

uses String's `compareTo` to sort `s`

uses StringLengthComp's `compare` to sort `s`

Assume `s` is an array of strings.

```
Arrays.sort(s);
Arrays.sort(s, new StringLengthComp());
```


Sorting Linked Lists

- Does the efficiency change if the data is in a linked list rather than an array?
- Example:
  - Merge Sort in place