CS32 Week 5: Sorting

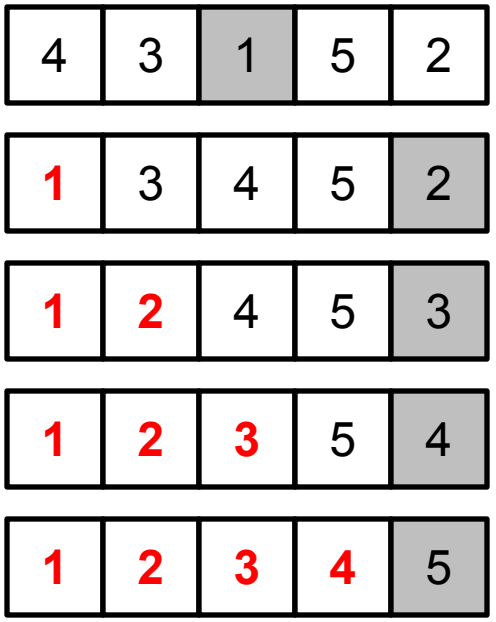
Doga Kisa

Sorting

- Sorting is useful.
  - Better retrieval efficiency.
  - Better data organization.
    - Sorting midterm papers by last name, first name.

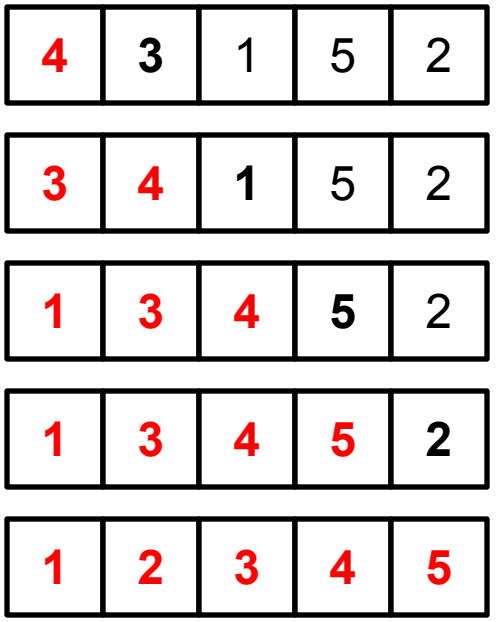
Sorting

- Selection Sort
  - Find the smallest item in the unsorted portion, then place it in the front.
  - Time complexity:
    - Worst case?
      - O(n^2)
    - Best case?
      - O(n^2)
    - Average case?
      - O(n^2)



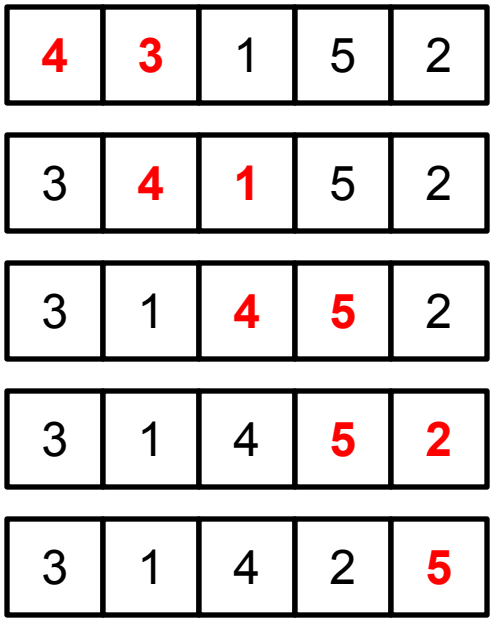
Sorting

- Insertion Sort
  - Pick one from unsorted portion, and place it in the right position in the sorted part.
  - Time complexity:
    - Worst case?
      - O(n^2)
    - Best case?
      - O(n)
    - Average case?
      - O(n^2)



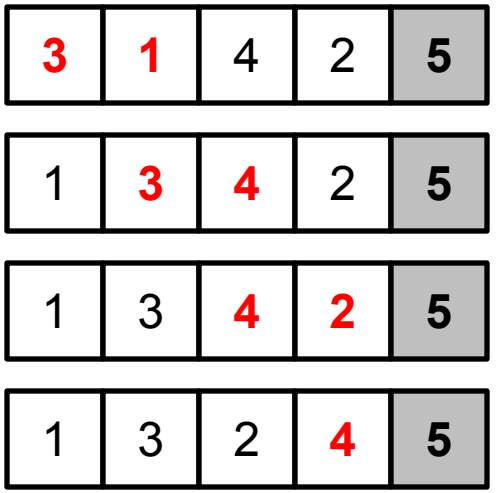
Sorting

- Bubble Sort
  - Bubble up the largest item to the end of unsorted part, until all is sorted. Loop through the array, if not in the correct order, swap.



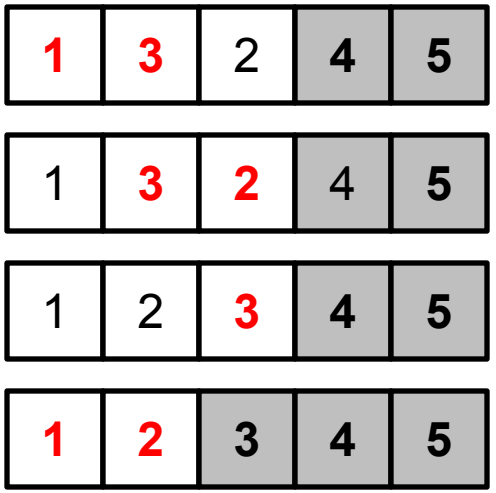
Sorting

- Bubble Sort
  - Bubble up the largest item to the end of unsorted part, until all is sorted. Loop through the array, if not in the correct order, swap.



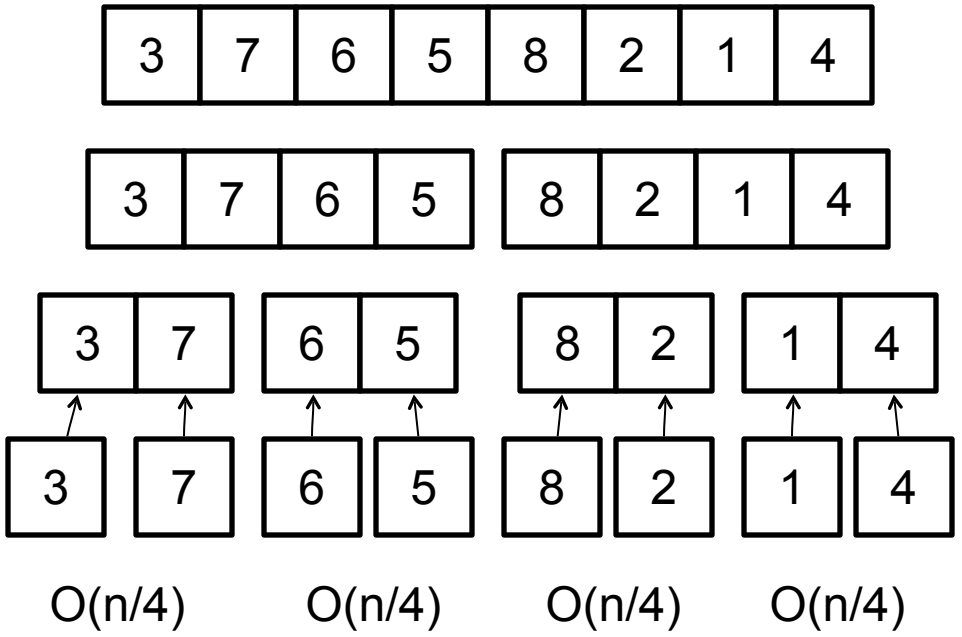
Sorting

- Bubble Sort
  - Bubble up the largest item to the end of unsorted part, until all is sorted. Loop through the array, if not in the correct order, swap.
  - Time complexity:
    - Worst case?
      - O(n^2)
    - Best case?
      - O(n)
    - Average case?
      - O(n^2)



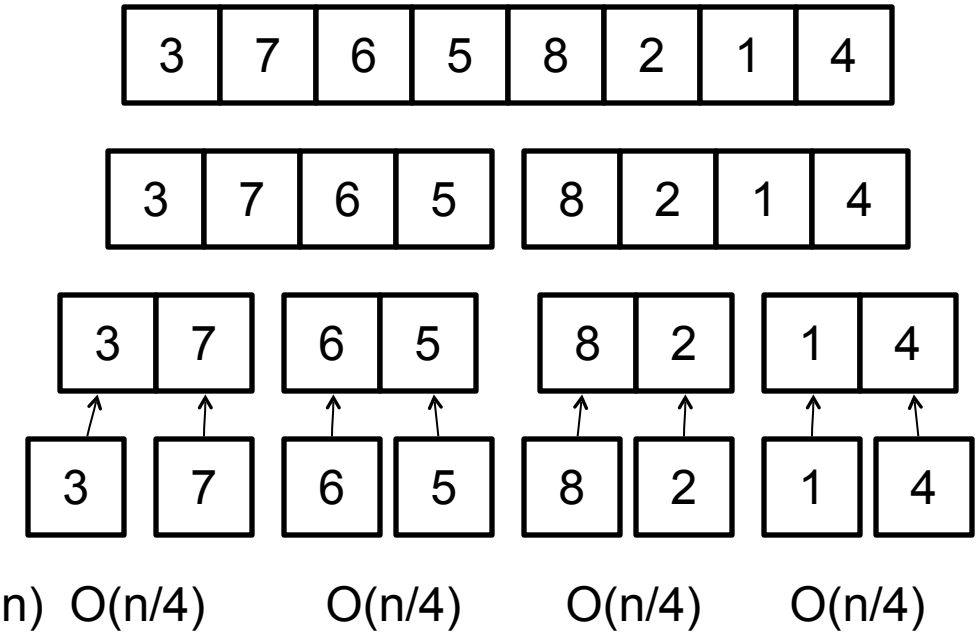
Sorting

- Merge Sort
  - Break the array into half, sort each half, then merge them together.
  - Recursion
    - Base case?
    - How to merge results?
  - Merge complexity?
    - n is array size
    - Each level is O(n)



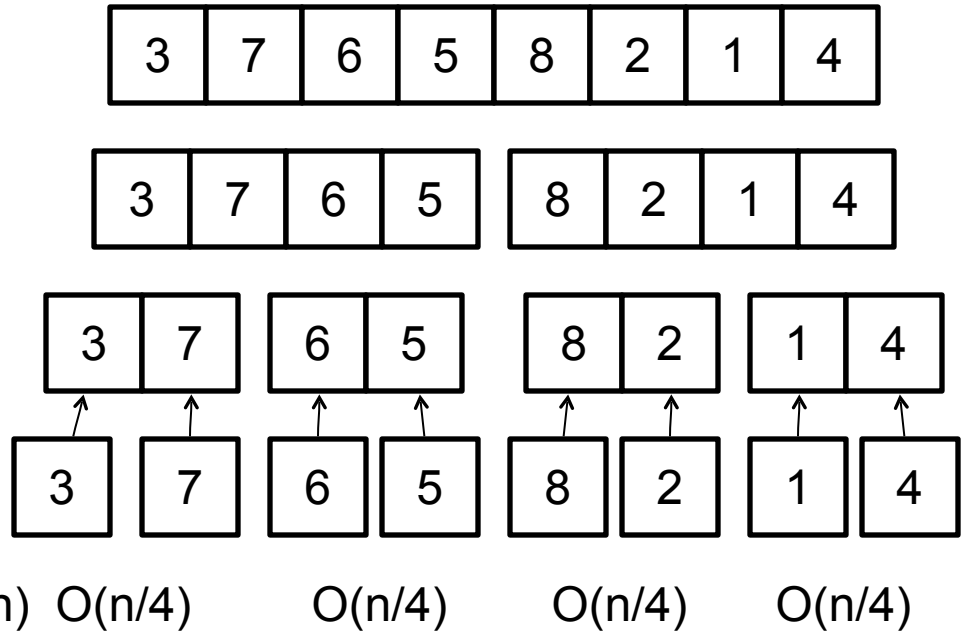
Sorting

- Merge Sort
  - Break the array into half, sort each half, then merge them together.
  - Merge complexity?
    - n is array size
    - Each level is O(n)
  - Number of Level
    - Log_2 n
  - Total complexity
    - O(nlogn)



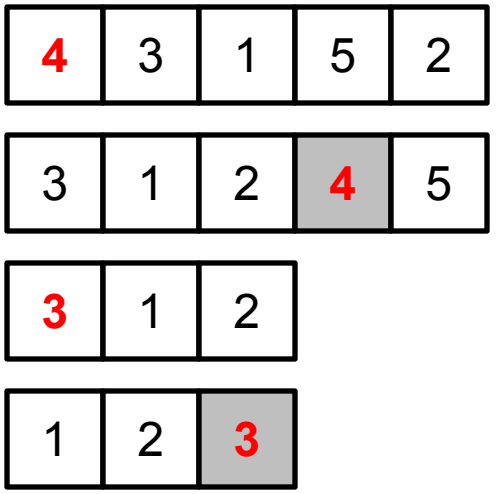
Sorting

- Merge Sort
  - Break the array into half, sort each half, then merge them together.
  - Disadvantage?
    - Need additional memory for merging.



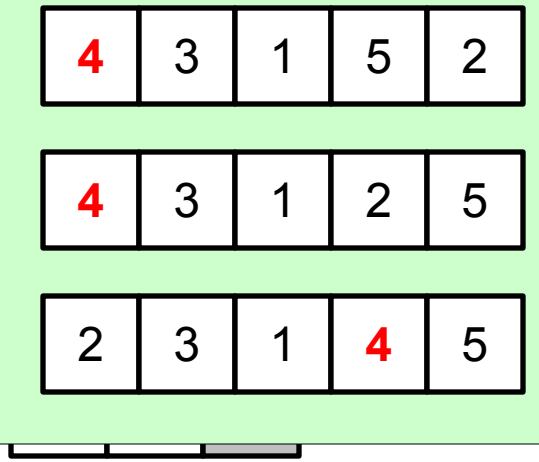
Sorting

- Quick Sort
  - Pick a pivot, move items that are smaller than pivot to the front, move items that are larger than pivot to the end.
  - #Level
    - Expected to be log2n
    - Depending on pivot
  - Complexity:
    - Worst case?
      - O(n^2) for already sorted array
    - Average case?
      - O(nlogn)



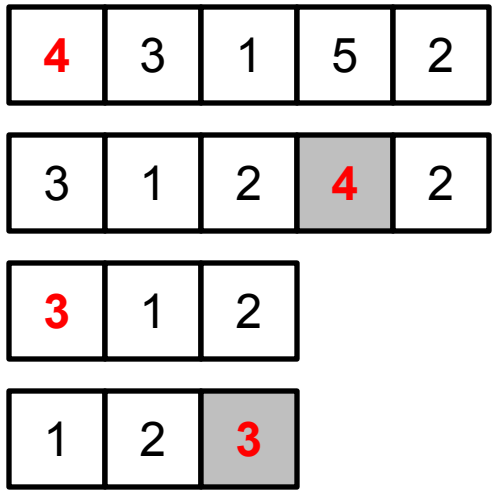
Sorting

```
int Partition(int a[], int low, int high)
{
  int pi = low;
  int pivot = a[low];
  do
  {
     while ( low <= high && a[low] <= pivot )
        low++;
     while ( a[high] > pivot )
        high--;
     if ( low < high )
        swap(a[low], a[high]);
  }
  while ( low < high );
  swap(a[pi], a[high]);
  pi = high;
  return(pi);
}
```



Sorting

- Quick Sort
  - Pick a pivot, move items that are smaller than pivot to the front, move items that are larger than pivot to the end.
  - Partition function
  - How to pick pivot?
    - First? Last? Or random?
    - It's important for the efficiency
  - In-place sorting

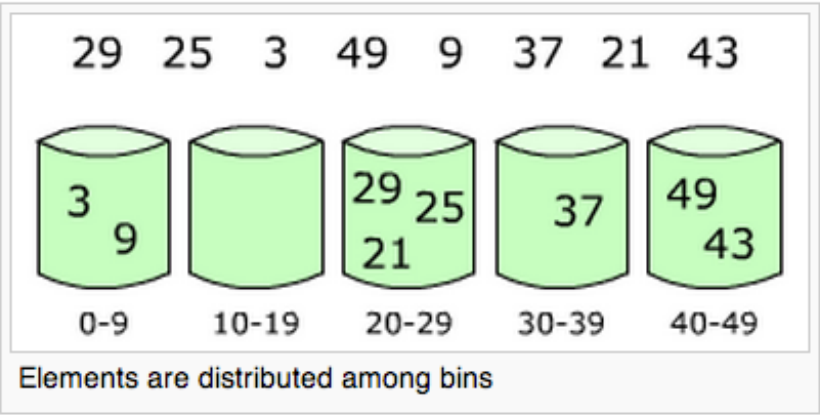


Sorting

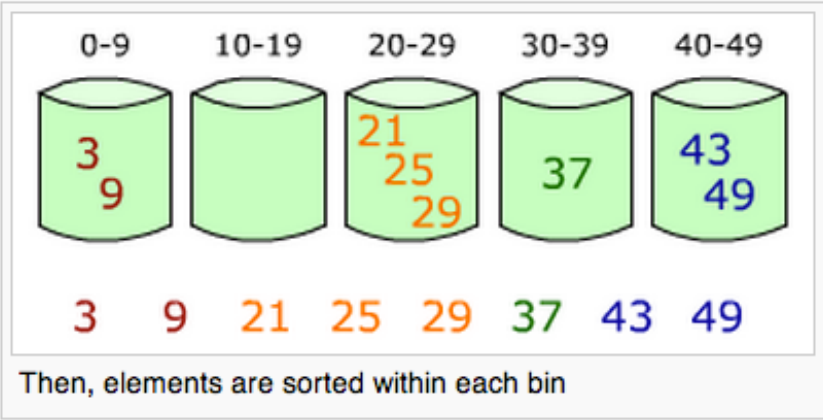
- What's the best algorithm to sort 1,000,000 random numbers that are all between 1 and 5?
- Bucket sort
  - Partition the whole into multiple buckets, sort each buckets individually (could recursively call bucket sort or other sort), and gather together.
  - Bucket: 1, 2, 3, 4, 5

Sorting

- Bucket sort
  - Partition the whole into multiple buckets, sort each buckets individually (could recursively call bucket sort or other sort), and gather together.
  - Average case: O(n+k)



Elements are distributed among bins



Then, elements are sorted within each bin