Radix Sort

We want to sort list `L` based on columns `firstCol` to `lastCol`.

```
radixSort(L, firstCol, lastCol)
{
  for(j = lastCol; j >= firstCol; j--)
  {
    for each element E of L
    {
      d = j'th column of E;
      add E to bucket[d];
    }
    concatenate all buckets;  //can be omitted
  }
}
```

QuickSort

1. Choose a pivot element.
2. Partition based on the pivot:
   - Put everything less than pivot in left sublist
   - Put everything greater than pivot in right sublist
3. Repeat for each sublist until the entire list is sorted.

Partition Algorithm

Partitions an array `A[left..right]` of integers based on `pivot`, which is in `A[right]`.

```
int partition(int left, int right, int pivot)
{
  int i = left-1;
  int j = right;
  while(true)
  {
    while(A[++i] < pivot)
        ;
    while(A[--j] > pivot)
        ;
    if(i >= j)
        break;
    else
        swapReferences(A, i, j)
    }
    swapReferences(A, i, right); // put pivot in correct position
    return i;                    // return pivot location
}
```

Quicksort Algorithm

Suppose we want to sort an array A[left..right].

```
QuickSort(int left, int right)
{
  if(right <= left)
    return;
  else
  {
    pivot = A[right];
    pivotIndex = partition(left, right, pivot);
    QuickSort(left, pivotIndex-1);
    QuickSort(pivotIndex+1, right);
  }
}
```

Note: alternate pivot choices can give better performance.

Merge Algorithm

- Assume `A[leftPos, rightPos-1]` and `A[rightPos,rightEnd]` are both already sorted.
- Merge them and store result in `A[leftPos,rightEnd]`.

```
Merge(A, tmp, leftPos, rightPos, rightEnd)
{
  leftEnd = rightPos-1;
  tmpPos = leftPos;
  numElements = rightEnd-leftPos+1;
  while(leftPos <= leftEnd && rightPos <= rightEnd)
    if(A[leftPos] <= A[rightPos])
       tmp[tmpPos++] = A[leftPos++];
    else
       tmp[tmpPos++] = A[rightPos++];
  while(leftPos <= leftEnd)
    tmp[tmpPos++] = A[leftPos++];
  while(rightPos <= rightEnd)
    tmp[tmpPos++] = A[rightPos++];
  for(i = 0; i < numElements; i++,rightEnd--)
    A[rightEnd] = tmp[rightEnd];
}
```

Mergesort Algorithm

```
Mergesort(A, tmp, lower, upper)
{
  if(lower < upper)
  {
    mid = (lower+upper)/2; //int division
    Mergesort(A, tmp, lower, mid);
    Mergesort(A, tmp, mid+1, upper);
    Merge(A, tmp, lower, mid+1, upper);
  }
}
```

Comparison of Sorting Algorithms

	Best-Case Complexity	**Avg-Case Complexity**	**Worst-Case Complexity**	**Extra Space Reqd?**	**Comments**
Mergesort	O(n log n)	O(n log n)	O(n log n)	Yes	
Quicksort	O(n log n)	O(n log n)	$O(n^2)$	No (except sys stack)	Fastest with good pivot choice
Heapsort	O(n log n)	O(n log n)	O(n log n)	No (using max heap)	
Radix Sort	O(n)	O(n)	O(n)	No (except lists)	Not general purpose