Sorting Algorithms

- Number of Comparisons
- Number of Data Movements
- Additional Space Requirements

Sorting Algorithms

- Selection Sort
- Insertion Sort
- Bubble Sort
- Shaker Sort

- Merge Sort
- Heap Sort
- Quick Sort

- Bucket & Radix Sort
- Counting Sort

Animation Demos

http://www-cse.uta.edu/~holder/courses/cse2320/lectures/applets/sort1/heapsort.html

http://cg.scs.carleton.ca/~morin/misc/sortalg/

Stable Sort

- A sort is stable if equal elements appear in the same order in both the input and the output.
- Which sorts are stable? Homework!

Radix Sort

3 5 9		3 5 9		3 3 6		3 3 6
3 5 7		3 5 7		3 5 9		3 5 1
3 5 1		3 5 1		3 5 7		3 5 5
7 3 9	→	3 3 6	→	3 5 1	→	3 5 7
3 3 6		3 5 5		3 5 5		3 5 9
7 2 0		7 3 9		7 2 0		7 2 0
3 5 5		7 2 0		7 3 9		8 3 9

Algorithm

for i = 1 **to** d **do**

sort array A on digit i using any sorting algorithm

Time Complexity: $O((N+m) + (N+m^2) + \ldots + (N+m^d))$

Space Complexity: $O(m^d)$

Radix Sort

3 2 9		7 2 0		7 2 0		3 2 9
4 5 7		3 5 5		3 2 9		3 5 5
6 5 7		4 3 6		4 3 6		4 3 6
8 3 9	→	4 5 7	→	8 3 9	→	4 5 7
4 3 6		6 5 7		3 5 5		6 5 7
7 2 0		3 2 9		4 5 7		7 2 0
3 5 5		8 3 9		6 5 7		8 3 9

Algorithm

for i = 1 **to** d **do**

sort array A on digit i using a stable sort algorithm

Time Complexity: $O((n+m)d)$

Counting Sort

Initial Array

1	2	3	4	5	6	7	8
2	5	3	0	2	3	0	3

Counts

0	1	2	3	4	5
2	0	2	3	0	1

Cumulative Counts

0	1	2	3	4	5
2	2	4	7	7	8

External Sorting Methods

- Assumptions:
  - data is too large to be held in main memory;
  - data is read or written in blocks;
  - 1 or more external devices available for sorting
- Sorting in main memory is cheap or free
- Read/write costs are the dominant cost
- Wide variety of storage types and costs
- No single strategy works for all cases

External Merge Sort

- Initial distribution pass
- Several multi-way merging passes

```
ASORTINGANDMERGINGEXAMPLEWITHFORTYFIVERECORDS.$
-----------------------------------------------
AOS.DMN.AEX.FHT.ERV.$
IRT.EGR.LMP.ORT.CEO.$
AGN.GIN.EIW.FIY.DRS.$
-----------------------------------------------
AAGINORST.FFHIORTTY.$
DEGGIMNNR.CDEEORRSV.$
AEEILMPWX.$
-----------------------------------------------
AAADEEEGGGIIILMMNNNOPRRSTWX.$
CDEEFFHIOORRRSTTVY.$
-----------------------------------------------
AAACDDEEEEEEFFGGGHIIIIILMMNNNOOOPRRRRRSSTTTWXY.$
-----------------------------------------------
```

With 2P external devices
Space for M records in main memory
Sorting N records needs
$1 + \log_P(N/M)$ passes

Order Statistics

- Maximum, Minimum   n-1 comparisons

7	3	1	9	4	8	2	5	0	6

- MinMax
  - 2(n-1) comparisons
  - 3n/2 comparisons
- Max and 2ndMax
  - (n-1) + (n-2) comparisons
  - ???

k-Selection; Median

- Select the k-th smallest item in list
- Naïve Solution
  - Sort;
  - pick the k-th smallest item in sorted list.

    O(n log n) time complexity
- Randomized solution: Average case O(n)
- Improved Solution: worst case O(n)


```
QuickSort(A, p, r)
   if (p < r) then
      q = Partition(A, p, r)
      QuickSort(A, p, q)
      QuickSort(A, q+1, r)

Partition(A, p, r)
 x = A[r]
 i = p-1
 for j = p to r-1 do
    if (A[j] <= x) then
       i++
       SWAP(A[i], A[j])
 SWAP(A[i+1], A[r])
 return i+1
```

Partition Procedure Revisited

- The Partition code can be rewritten so that it accepts another parameter, namely, the pivot value. Let's call this new variation as PivotPartition.
- This change does not affect its time complexity.
- RandomizedPartition as used in RandomizedSelect picks the pivot uniformly at random from among the elements in the list to be partitioned.

Randomized Selection

```
RandomizedSelect(A, p, r, i)
   if (p = r) then
      return A[p]
   q = RandomizedPartition(A, p, r)
   k = q – p + 1
   if (i = k)
      return A[i]
  else if (i < k)
      return RandomizedSelect(A, p, q-1, i)
   else
      return RandomizedSelect(A, q+1, r, i-k)
```


Randomized Selection: Rewritten

```
RandomizedSelect(A, p, r, i)
   if (p = r) then
      return A[p]
   Pivot = A[random(p,r)]
   q = PivotPartition(A, p, r, Pivot)
   k = q – p + 1
   if (i = k)
      return A[i]
   else if (i < k)
      return RandomizedSelect(A, p, q-1, i)
   else
      return RandomizedSelect(A, q+1, r, i-k)
```