Sorting Concepts

- Sorting is considered to be one of the classic areas of study in computer science.
- A few definitions will be helpful:
  - A **field** is a unit of data. (e.g. a name, phone number, ID #, age.)
  - A **record** is a collection of fields.
  - A **file** is a collection of records.
- Sorting is the process of placing the records of a file in some well defined order.
- We sort a file based on one or more fields, which we call **keys**.
- Some order is defined for each key, and the sort is performed on the keys in some order.
- We will focus on sorting with respect to a single key.
- Most sorting algorithms are **comparison sorts**. These use two basic operations: **compare** and **swap**.

Indirect Sorting

- When records are large, swapping is very expensive.
- We can minimize the cost by using an *indirect* sort.
- This can be accomplished in several ways.
- One method is to create a new file whose records contain the key field of the original file, and either a pointer to or index of the original records.
- We sort the new file, whose records are small, and then access the original file using the pointers/indices.

Example

We want to sort the following file by the field **Dept**.

Index	Dept	Last	First	Age	ID number
1	123	Smith	Jon	23	234-45-4586
2	23	Wilson	Pete	54	345-65-0697
3	2	Charles	Philip	19	508-45-6859
4	45	Shilst	Greg	78	234-45-5784

Index	Key
1	123
2	23
3	2
4	45

$\xrightarrow{Sort}$

Index	Key
3	2
2	23
4	45
1	123

- To access the records in sorted order we use the order provided by the *index* column. In this case, $(3, 2, 4, 1)$.

Criteria for Evaluating Sorts

Sorting algorithms can be compared based on several factors.

- **Run-time:** The number of operations performed (usually swap and compare).
- **Memory:** The amount of memory needed beyond what is needed to store the data to be sorted.
  - Some algorithms sort “in place” and use no (or a constant amount of) extra memory.
  - Others use a linear amount, and some an exponential amount.
  - Clearly less memory is preferred, although space/time trade-offs can be beneficial.
- **Stability:** An algorithm is stable if it preserves the relative order of equal keys.

We usually worry mostly about run-time, since the algorithms we discuss all use relatively the same amount of memory.

Example of Stable Sort

We want to sort the following file according to the first letter. Below are the results of a stable and an unstable sort.

File	Stable sorting	Unstable sorting
A Mary	A Mary	A Michel
B Cindy	A Michel	A Peter
A Michel	A Peter	A Mary
B Diana	B Cindy	B Tony
A Peter	B Diana	B Diana
B Tony	B Tony	B Cindy

We will see later why stable sorting is important.

Sorting Algorithms

The following are common sorting algorithms

- Selection Sort
- Insertion Sort
- Bubble Sort
- Quicksort
- Mergesort
- Heapsort
- Radix sort

We will discuss the first three in the remainder of these notes.

Selection Sort

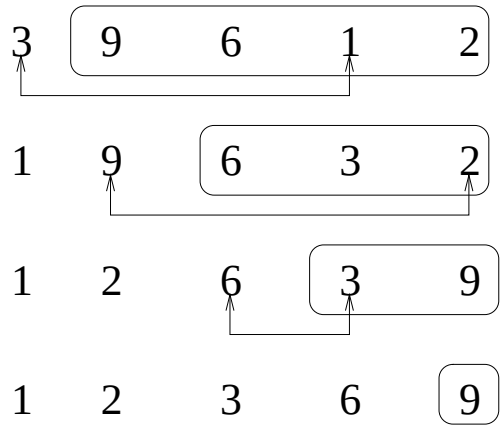
- Find the largest/smallest key, put it in place, and then sort the remainder of the array.
- Sample C++ code for selection sort:

```
void selection_sort(int a[],int n) {
   for (int i=0;i<n-1;i++) {
       int min = i;
       for (int j=i+1;j<n;j++) {
           if(a[j] < a[min]) min = j;
           }
       swap(a[min],a[i]);
       }
    }
```

- This is almost legal Java code, but there is one problem that needs to be fixed.

Selection Sort

Example:



Selection Sort Analysis

```
void selection_sort(int a[],int n) {
   for (int i=0;i<n-1;i++) {
       int min = i;
       for (int j=i+1;j<n;j++) {
           if(a[j] < a[min]) min = j;
           }
       swap(a[min],a[i]);
       }
    }
```

- For each i, we execute $n - i - 1$ compare operations.
- Since i runs from 0 to $n - 2$, we execute

$$(n-1) + (n-2) + \ldots + 1 = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2}$$

  compare operations.
- As we have seen before, $\frac{n(n-1)}{2} = O(n^2)$
- Is this worst-case or average case? How many swaps are executed?

Insertion Sort

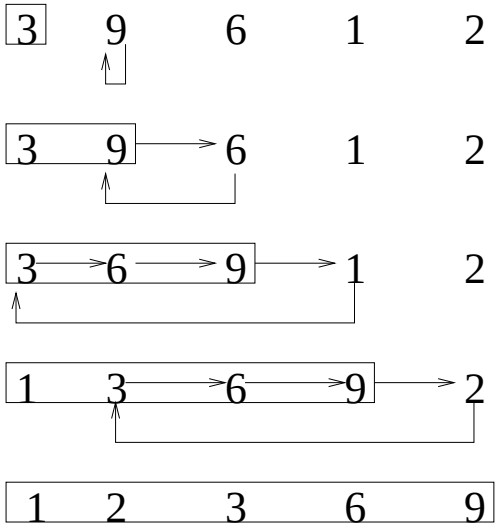
- Insert each key into a sorted sub-list of the keys.
- This is the method often used to sort cards.
- Sample C++ code for insertion sort:

```
void insertion(int a[],int n) {
   for (int i=1;i<n;i++) {
      int j = i;
      int temp = a[i];
      while (j > 0 && temp < A[j-1]) {
         A[j] = A[j-1]; j--;
      }
      A[j] = temp;}
 }
```

- Would this one work as-is in Java?

Insertion Sort

Example:



Insertion Sort Analysis

```
void insertion(int a[],int n) {
   for (int i=1;i<n;i++) {
      int j = i;
      int temp = a[i];
      while (j > 0 && temp < A[j-1]) {
         A[j] = A[j-1]; j--;
      }
      A[j] = temp;}}
```

- For each i, we need to compare the current key with at most i keys.
- Since i runs from 1 to $n-1$, we must do at most

$$1 + 2 + \ldots + n - 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

  comparisons.
- This is the same as selection sort, $O(n^2)$
- What is the average case? How many swaps are performed?

Bubblesort

- Go through the list in order, swapping two elements if there keys are out of order.
- Repeat until no swaps are performed. The list is sorted.
- It is not hard to see that n passes suffice.
- This is similar to selection sort, except with a lot of added swaps.
- Sample C++ code for bubblesort:

```
void bubblesort(int a[],int n)
{
   for (int i=n-1;i>1;i--)
      for (int j=1;j<=i;j++)
         if (a[j-1] > a[j])
            swap(a,j-1,j);
}
```

Comparisons

- Selection sort
  - Average - $\frac{n^2}{2}$ comparisons, n swaps
  - Worst - the same
- Insertion sort
  - Average - $\frac{n^2}{4}$ comparison, $\frac{n^2}{8}$ swaps
  - Worst - double the average
  - linear for *almost sorted* files
- Bubble sort
  - Average - $\frac{n^2}{2}$ comparisons, $\frac{n^2}{2}$ swaps
  - Worst - the same

General Sorting Strategy

- Most well-known (good) sorting algorithms are recursive. They follow the following general strategy:
  Given a list of records L
  - If L has zero or one element, then it is already sorted.
  - Otherwise
    1. Divide in L into two smaller sequences, L_1 and L_2.
    2. Recursively sort L_1 and L_2.
    3. Combine L_1 and L_2 to produce sorted L.
- **Merge Sort** and **Quick Sort** use this sort of technique.
- For more information on these and other more advanced sorting techniques, consult some of my other notes.