Sorting I

Simple Sorts

Project

- Minimum Submission Sunday night
- Final submission –Sunday.
- A couple days of breathing room
  - Before Project 2

Plan

- Things to look forward to
  - Analysis of Algorithms
  - Search Algorithms
  - Sort Algorithms (M, T)
  - Return Exam 1
    - Wasn't here Wednesday? Please pick up after class.
  - Distribute Project 2 (W)

Final Exam – Good news/bad news

- Good news
  - Exam is mid-week and in this building:
    - Wednesday, February 25, 2004
    - 70-3435
- Bad news
  - The time
    - 8:00am – 10:00 am
- Please note all conflicts NOW!

Before we begin

- Any questions on
  - Asymptotic Analysis
  - Searching

Sorting

- Problem:
  - Given:
    - array of objects to be sorted (x)
      - In our examples, this collection will be stored in an array
  - Calculation
    - Sorts the objects in the collection such that
      - x[i-1] <= x[i] for 0 < i < length of x

Sorting

- Problem: Given an array of items, sort the elements in the array
  - Algorithms
    - Selection Sort
    - Insertion Sort
    - Bubble Sort
    - Merge Sort
    - Quicksort

Evaluation

- Time Analysis
  - "Basic Operation"
    - Comparison
    - object swap
  - Best case
  - Worst case
  - Average Case

Swap

- Swaps the item at one index of the array with an object of another

```
public void swap (int x[], int i, int j)
{
      int tmp = x[i];
      x[i] = x[j];
      x[j] = tmp;
}
```

Swap

- Let's generalize to sort for any object
  - By this time, you are probably painfully aware of the Comparable interface:
    - public int **compareTo**(Object o)
      - Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.
      - Assumes that o is of same class of object being compared to.

Swap

- Swaps the item at one index of the array with an object of another

```
public void swap
   (Comparable x[], int i, int j)
   {
         Comparable tmp = x[i];
         x[i] = x[j];
         x[j] = tmp;
   }
```

Sort Algorithms

- Selection Sort
- Insertion Sort
- Bubble Sort
- Merge Sort
- Quicksort

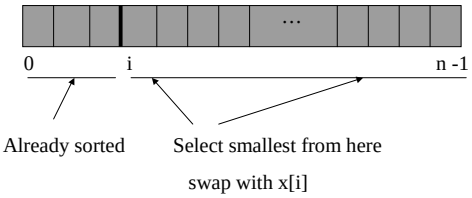
- Let's begin

Selection sort

- Basic idea
  - Iterate through the elements of x
    - For index i, "select" the smallest element of the array from index i to end of the array and swap it with x[i].
    - After i iterations, the smallest i elements will occupy x[0] … x[i-1] in sorted order.

Selection sort

- Basic idea
  - On iteration index i



Selection sort

```
public void selectSort (Comparable x[])
{
    for (int i=0; i < x.length; i++) {
        int small = x[i];
        for (int j=i; j<x.length; j++)
            if (x[j].compareTo(x[small]) < 0)
                small = j;
        swap (x, i, small);
     }
}
```

Selection sort

- Example

Selection sort

- Analysis
  - Comparisons:
    - In the first iteration n comparisons are made
    - In the second iteration n-1 comparisons are made
    - In the third iteration n-2 comparisons are made.
    - And so on…
    - Total for all iterations:
      - $T(n) = 1 + 2 + 3 + \ldots + (n-1) + n = n(n-1) / 2 = \Theta(n^2)$

Selection sort

- Analysis
  - Swaps
    - A swap is made once per iteration (if needed)
      - Best case:
        - » array already sorted (no swaps needed)
        - » $T(n) = c = \Theta(1)$
      - Worst case
        - » Swaps always needed
        - » $T(n) = n = \Theta(n)$
      - Average case
        - » Swaps needed half the time
        - » $T(n) = 0.5n = \Theta(n)$

Sort Algorithms

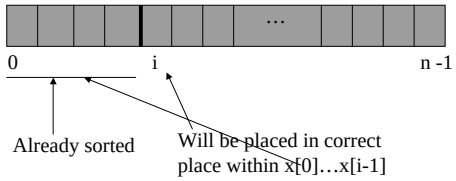
- Selection Sort
- Insertion Sort
- Bubble Sort
- Merge Sort
- Quicksort

Insertion sort

- Basic idea
  - Iterate through the elements of x (from 1 to n-1)
    - On pass i, the element at x[i] will be "inserted" into it's rightful place within the elements of x[0] … x[i-1].
    - Other elements in x[0]…x[i-1] will be shifted accordingly.

Insertion sort

- Basic idea
  - On iteration index i



Insertion sort

```
public void insertSort (Comparable x[])
{
    for (int i=1; i < x.length; i++) {
        Comparable temp = x[i];
        for (int j=i; j >=0; j--) {
          if (j == 0 || (x[j-1].compareTo(temp) < 0)) {
             // insert
             x[j] = temp;
             break;
          }
          else {
             // shift
             swap (x, j, j-1);
        }
     }
}
```

Insertion sort

- Example

Insertion sort

- Analysis
  - Comparisons:
    - In each iteration of the outer loop we will make a comparison as long as x[i] is out of order
    - Best case (array is already sorted):
      - Will only have to make 1 comparison per iteration.
      - This will mean n comparisons = $\Theta(n)$
    - Worst case (array is sorted in reverse)
      - Will have to make i comparisons per iteration
      - This will mean $1 + 2 + 3 + \ldots + (n-1) + n$ comparisons = $\Theta(n^2)$

Insertion sort

- Analysis
  - Comparisons:
    - Average case:
      - Inner for loop will stop, on average, after half of the insertions have been performed
      - Average case = 0.5 worst case
      - $\Theta(n^2)$

Insertion sort

- Analysis
- Swaps
  - In each iteration of the outer loop we will make a swap as long as x[i] is out of order
    - Best case (array is already sorted):
      - Will only have to make 0 swaps per iteration.
      - This will mean 0 swaps = $\Theta(1)$
    - Worst case (array is sorted in reverse)
      - Will have to make i swaps per iteration
      - This will mean $1 + 2 + 3 + \ldots + (n-1) + n$ swaps = $\Theta(n^2)$

Insertion sort

- Analysis
  - Swaps:
    - Average case:
      - Inner while will stop, on average, after half of the swaps have been performed
      - Average case = 0.5 worst case
      - $\Theta(n^2)$

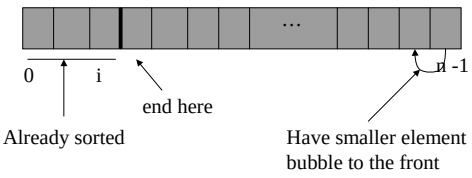
Sort Algorithms

- Selection Sort
- Insertion Sort
- Bubble Sort
- Merge Sort
- Quicksort

Bubble sort

- Basic idea
  - Iterate through the elements of x (from 0 to n-1)
    - Compare adjacent elements and have the larger element "bubble" to the bottom of the array
    - After the i^{th} iteration, the last i elements of the array will be properly sorted

Bubble sort



Bubble sort

```
public void bubbleSort (Comparable x[])
{
    for (int i=0; i < x.length-1; i++) {
        for (int j=x.length-1; j > i; j--)
              if (x[j-1].compareTo(x[j]) < 0)
                  swap (x, j, j+1);
     }
}
```

Bubble sort

- Example

Bubble sort

- Analysis
  - Comparisons
    - First iteration n-1 comparisons are made
    - Second iteration n-2 comparisons are made
    - And so on
    - $T(n) = (n-1) + (n-2) + \ldots + 1 = n(n-1)/2 =$
    - $\Theta(n^2)$

Bubble sort

- Can we do better?
  - If we go through an iteration and none of the elements are swapped during that iteration, then we know we are done.

Better Bubble Sort

```
public void bubbleSort (Comparable x[])
{
    boolean swapMade = true;
    for (int i=0; (i < x.length-1 || !swapMade);
     i++) {
        swapMade = false;
        for (int j=x.length-1; j > i; j--)
              if (x[j-1].compareTo(x[j]) < 0)
                   swap (x, j, j+1);
                   swapmade = true;
        }
     }
}
```

Better Bubble Sort

- With the better bubble sort
  - If the array is initially sorted, only 1 iteration would be performed:
    - Best case:
      - Only n-1 comparisons = $\Theta(n)$
    - Worst case:
      - Array is sorted in reverse…all iterations required
      - $\Theta(n^2)$
    - Average case:
      - Array will be sorted after performing half the iterations
      - Half of the worst case = $\Theta(n^2)$

Better Bubble Sort

- Swaps
  - In each iteration of the outer loop we could make at most, a swap with every comparison.
    - Best case (array is already sorted):
      - Will only have to make 0 swaps.
      - This will mean 0 swaps = $\Theta(1)$
    - Worst case (array is sorted in reverse)
      - Will have to swap every time we compare
      - $\Theta(n^2)$
    - Average case
      - $\Theta(n^2)$

Summary

	Comparisons	Swaps
Selection	Best: $\Theta(n^2)$ Worst: $\Theta(n^2)$ Avg: $\Theta(n^2)$	Best: $\Theta(1)$ Worst: $\Theta(n)$ Avg: $\Theta(n)$
Insertion	Best: $\Theta(n)$ Worst: $\Theta(n^2)$ Avg: $\Theta(n^2)$	Best: $\Theta(1)$ Worst: $\Theta(n^2)$ Avg: $\Theta(n^2)$
Bubble	Best: $\Theta(n)$ Worst: $\Theta(n^2)$ Avg: $\Theta(n^2)$	Best: $\Theta(1)$ Worst: $\Theta(n^2)$ Avg: $\Theta(n^2)$

Sort Algorithms

- Selection Sort
- Insertion Sort
- Bubble Sort
- Merge Sort
- Quicksort

Next time

- Sorting
  - MergeSort
  - Quicksort