Sorting

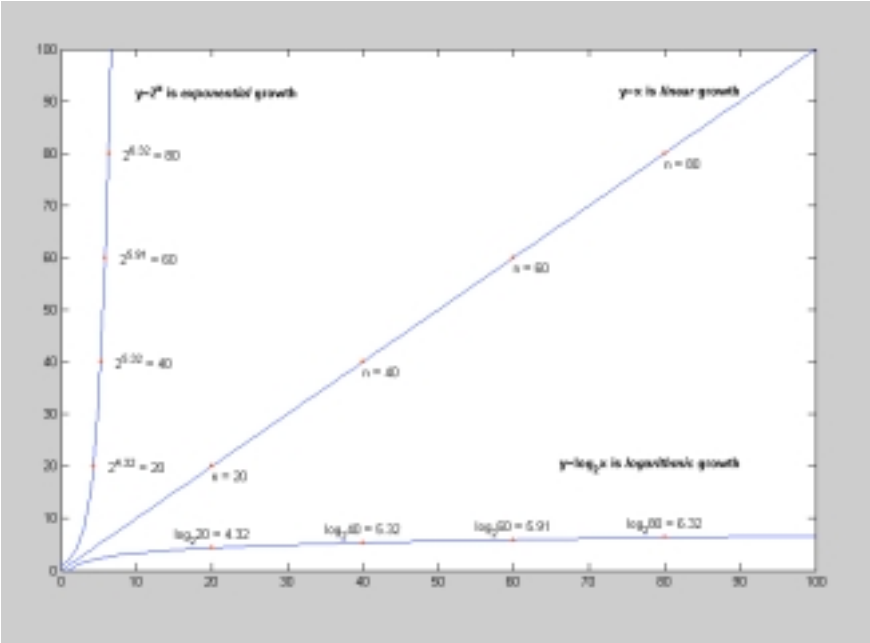
CSE 142, Summer 2003

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/03su/

Linear vs Binary Search

- Recall work needed to search a list of *n* items
  - » Linear search ~ *n*
  - » Binary search ~ log *n*
- For all but small lists, binary search is much, much, much faster
  - » For *n* = 1,000, log *n* ~ 10
  - » For *n* = 1,000,000, log *n* ~ 20



matlab generated using ex20\showlog.m

Simple sort is easy, fast sort is tricky

- There are lots of possible sorting algorithms
- The speed of the sort may depend on the data
  - » completely random
  - » already sorted, sorted but in reverse order
  - » odd patterns of partially sorted data
- Today's goal
  - » understand the concept of sorting
  - » understand that there are good sorters available

Recall addItem from ListManager

- The existing list is already in order
- Find the correct spot for the new item
- Insert the new item in the correct spot

```
public void addItem(Comparable obj) {
   int j = theList.size();
   while(j > 0 && ((Comparable)theList.get(j-1)).compareTo(obj)>0 ) {
       j--;
   }
   theList.add(j,obj);
}
```

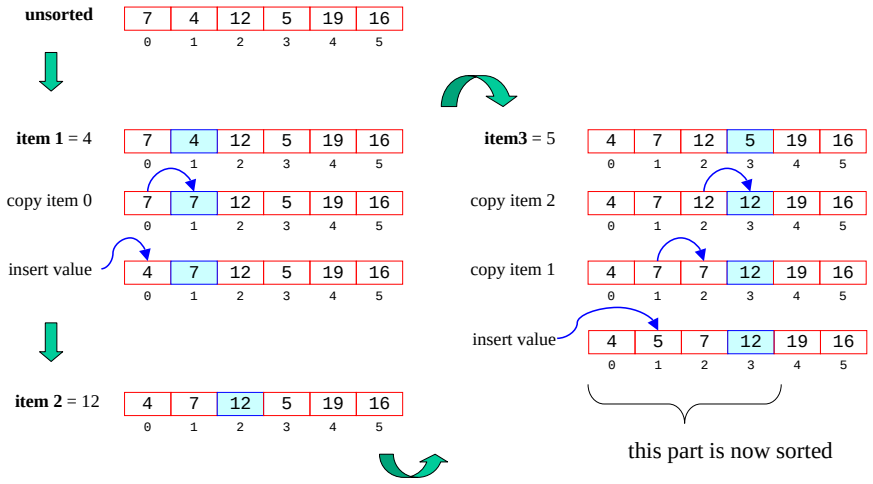

Insertion Sort

This technique can be the basis of a simple sort algorithm

```
initialize the sorted part to be empty
for each item in the list
    copy the tail of the list down until we find
      the correct spot for the item in the sorted part
    insert the item in the correct position

the list is now sorted
```


Insertion sort example



Insertion Sort

```
public void InsertionSort(int x[]) {
     for(int p = 1; p < x.length; p++ ) {
         int j;
         int tmp = x[p];
         for(j=p; j > 0 && x[j-1] > tmp;j-- )
             x[j] = x[j-1];
         x[j] = tmp;
     }
}
```

- Insertion sort is used as the bottom level of many other sorting routines, so you will see this implemented in a variety of forms if you study sorting algorithms

Insertion Sort Characteristics

- Running time
  - » Worst case is proportional to N^2
    - reverse order input
    - must copy every element every time
  - » Best case is proportional to N
    - in-order input
    - copy down stops with first comparison every time

N^2 is too slow for big arrays

- There are better sort algorithms for larger lists
  - » slightly more complex, so there is more overhead
  - » improvement is often sensitive to input data
  - » typically divide the list to be sorted into sub-arrays and sort those
    - the dividing process does some sorting
    - insertion sort sub-arrays with less than ~7 elements
- Use the sort methods in Arrays and Collections to sort large lists

Conclusion

- Performance Tradeoffs
  - » Sorting is relatively expensive
  - » Pays off if searches are frequent and clustered together compared to additions to the list
- Can either
  - » maintain lists in sorted order at all times
    - add operation is more expensive
  - » sort when needed for a find operation
    - find operation is expensive when it includes a sort