Algorithms: Searching and Sorting

Instructor: Dmitri A. Gusev

Spring 2007

CSC 120.02: Introduction to Computer Science

Lecture 12, March 8, 2007

Sequential Search

```
public boolean IsThere(int item)
    {
        for (int i=0;i<listLength;i++)
        {
            if (list[i]==item)
                return true; // item found
        }
        return false; // item not found
    }
```

Selection Sort

```
public void SelectionSort()
   {
      int currentMin; // declare a variable for the current minimum value
      int currentMinIndex; // declare an index

      for (int i=0;i<listLength;i++)
      {
         // Find the maximum in the sublist [i..listLength-1]:
         currentMin = list[i];
         currentMinIndex = i;
         for (int j=i+1;j<listLength;j++)
         {
            if (list[j] < currentMin)
            {
               currentMin = list[j];
               currentMinIndex = j;
            }
         }

         // Swap list[i] with list[currentMinIndex] if necessary:
         if (currentMinIndex != i)
         {
            list[currentMinIndex] = list[i];
            list[i] = currentMin;
         }
      }
   }
```

Bubble Sort

```
Set current to index of first item in the list;
do
{
   Set the value of  a boolean variable named swap to false;
   for (index going from (listLength-1) to (current+1))
   {
      if (list[index]<list[index-1])
      {
         Swap list[index] and list[index-1];
         Set the value of swap to true;
      }
   }
   Increment current to shrink the unsorted portion of the list;
}
while ((current<(listLength-1))&&swap);
```

Quicksort

if (*there is more than one item in the list*)
{
  Select splitVal;
  Split the list so that
    (list[0]..list[splitPoint-1]<=splitVal)&&
    (list[splitPoint]=splitVal)&&
    (list[splitPoint+1]..list[listLength-1]>splitVal);
  Quicksort list[0]..list[splitPoint-1];
  Quicksort list[splitPoint+1]..list[listLength-1];
}

Binary Search

We search in a list that's **already sorted**. We compare the value of the middle element with the one that we're searching for. If they are equal, return true. If the value of the middle element is larger, we continue our search by examining the middle element of the half where the smaller values are located. Otherwise, check the middle element of the other half of the list. If the half to examine is an empty list (no elements left to check), return false.