Addis Ababa University, Amist Kilo July 7, 2011

Algorithms and Programming for High Schoolers

Lecture 4

Sorting: This lecture covers *sorting*. In this problem, we have a list of items (let's say integers) and would like to sort them from smallest to biggest. A natural method for sorting, which is probably what most of us do in real life, is look for the smallest element, put it at the beginning, then sort what's left. This is known as `selectSort`, and here is an implementation in Python:

```
def selectionSort(L):
    if len(L) == 0:
        return []
    smallest = L[0]
    smallestIndex = 0
    for i in xrange(1, len(L)):
        if L[i] < smallest:
            smallest = L[i]
            smallestIndex = i
    # In Python, a,b = b,a swaps the contents of the variables a,b
    L[0],L[smallestIndex] = L[smallestIndex],L[0]
    return [L[0]] + selectionSort(L[1:])
```

Another approach to sorting is to gradually make prefixes of the list sorted. That is, first we'll make sure L[0:1] is sorted, then L[0:2], etc., all the way up to L[0:len(L)]. The following method is known as `insertionSort`.

```
def insertionSort(L):
    for i in xrange(1, len(L)):
        # We assume L[0:i] is already sorted, and now need to put L[i]
        # in its rightful place.
        j = i - 1
        value = L[i]
        while j>=0 and L[j]>value:
            L[j+1] = L[j]
            j -= 1
        L[j+1] = value
    return L
```

Another problem sorting method is `bubbleSort`. This procedure has several iterations. In each iteration you start at the beginning of the list and move to the end, one item at a time, and for each item you encounter you swap it with its adjacent element on the right if the two elements are inverted. This is done repeatedly until there are no more swaps being performed.

```
def bubbleSort(L):
    swapped = True
    while swapped:
        swapped = False
        for i in xrange(len(L)-1):
            if L[i]>L[i+1]:
                L[i],L[i+1] = L[i+1],L[i]
                swapped = True
    return L
```

The last sorting method we'll cover today is `mergeSort`. This is a recursive procedure for sorting a list. We break the list in two equal-sized halves (or as equal-sized as possible if the list size is odd), recursively sort each half, then merge the two lists together.

```
def mergeSort(L):
    if len(L)<=1:
        return L
    # recursively sort the first half of L and put the result in A, and
    # recursively sort the second half of L and put the result in B
    A = mergeSort(L[0:len(L)/2])
    B = mergeSort(L[len(L)/2:])

    # now merge A and B, and put the result in C
    C = []
    aindex = 0
    bindex = 0
    for i in xrange(len(L)):
        if aindex==len(A):
            C += B[bindex:]
            break
        elif bindex==len(B):
            C += A[aindex:]
            break
        else:
            if A[aindex] < B[bindex]:
                C += [A[aindex]]
                aindex += 1
            else:
                C += [B[bindex]]
                bindex += 1
    return C
```

As we will see in the next lecture, `mergeSort` is usually the best choice of these four methods when it comes to algorithm speed.