CS 202 Recursive Algorithms

Merge Sort

Merge Sort is a classic sorting algorithm which you probably studied in CS 111. We can sort a list by recursively sorting the first half, recursively sorting the second half, and *merging* the resulting sorted list,

```
mergeSort(A[1 ... n]):
   // Input: an array A
   // Ouptut: A in sorted order

   if n == 1:
      return A
   else:
      L = mergeSort(A[1 ... n/2])
      R = mergeSort(A[n/2+1 ... n])
   return merge(L,R)
```

which is done by repeatedly removing the smallest element from the beginning of the two sorted lists.

```
merge(X[1 ... m], Y[1 ... n]):
   // Input: two sorted arrays X and Y
   // Output: a single sorted array of all elements in X and Y

   x = 1 // pointer to current position in X
   y = 1 // pointer to current position in Y
   X[m+1] = infinity // keep from falling off end
   Y[n+1] = infinity // keep from falling off end
   output = [] // initialize output as empty array

   for i = 1 to m + n:
      if X[x] < Y[y]:
         output[i] = X[x]
         x = x + 1
      else:
         output[i] = Y[y]
         y = y + 1

   return output
```

For simplicity, we will always assume that we are sorting a list of 2^k numbers for some $k \in \mathbb{Z}^{\geq 0}$. Thinking only about powers of two make our lives easier without losing any real generality.

Binary Search

Binary Search searches a sorted list for an element by looking at the middle and, if necessary, recursively searching through one of the halves. Since the list is sorted, it knows which half to look.

```
binarySearch(A[1 ... n], x):
   // Input: a sorted array A; an element x
   // Output: is x in the (sorted) array A?

   if n == 0:
      return False
   middle = n/2 rounded down
   if A[middle] == x:
      return True
   else if A[middle] > x:
      return binarySearch(A[1 ... middle-1], x)
   else
      return binarySearch(A[middle+1 ... n], x)
```

Max

Max splits the array in half, finds the max of each half recursively, and then returns the bigger of the two results.

```
max(A[1 ... n]):
   // Input: an array A
   // Output: max element

   if n == 1:
      return A[1]
   middle = n/2 rounded down
   x = max(A[1 ... middle])
   y = max(A[middle+1 ... n])
   if x > y:
      return x
   else:
      return y
```

Bonus: Selection Sort

Selection Sort sorts by repeatedly *selecting* the minimum element in the unsorted segment and swapping it into place.

```
selectionSort(A[1 ... n]):
   // Input: an array A
   // Ouptut: A in sorted order

   for i = 1 to n:
      minIndex = i
      for j = i+1 to n:
         if A[j] < A[minIndex]:
            minIndex = j
      swap A[i] and A[minIndex]
```