```
void msort(int data[]) {
   msort(data,new int[data.length],0,data.length-1);
}
void msort(int data[],int extra[],int first,int second) {
   if (data.length>1) {
        middle=(first+second)/2;
        msort(data,extra,first,middle-1);
        msort(data,extra,middle,second);
        merge(data,extra,first,middle,second);
   }
}
```

```
void merge(int data[],int extra[],int first,int second,int last) {
   int i,b1,b2;
   b1 = first;  b2 = second;
   for (i=0;i<(last-first+1);i++) {
        if (b1<second && (b2>last || data[b1]<=data[b2])) {
                extra[first+i]=data[b1];
                b1++;
        } else {
                extra[first+i]=data[b2];
                b2++;
        }
   }
   for (i=0;i<(last-first+1);I++) {
        data[first+i]=extra[first+i];
   }
}
```

Sort

Try merge sort on each of the following:

1. sort(1,2,3,4,5,6)
2. sort(3,6,1,2,5,4)

Merge sort

- time: O(n*log(n))
  - Tree of recursion like Tower of Hanoi, but
    - Grows one node per n, not one row
    - Work per node changes
- Space
  - For arrays: O(n)
    - Need a destination list to copy it to
    - Are VERY complicated ways around this
  - For linked lists: O(1)
    - A bit harder to find the mid point, but it's only an extra linear amount of work at each level

```
//QuickSort partition:
static int partition(int[] data, int lo, int hi) {
  int tempPivotindex = lo;
  int pivotVal       = data[tempPivotindex];
  int left           = lo + 1;
  int right          = hi;
  while (left <= right) {
    while (data[left] <= pivotVal && left <= hi) {left++;}
    while (data[right] > pivotVal && right > lo) {right--;}
    if (left < right) {
      int temp    = data[left];
      data[left]  = data[right];
      data[right] = temp;
    }
  }
  data[tempPivotindex] = data[right];
  data[right]          = pivotVal;
  return right;
}
```

```
static void qsort(int[] data, int lo, int hi){
  // quit if empty partition
  if (lo >= hi) {
     return;
  }
  int pivotLocation;
  pivotLocation = partition(data, lo, hi);
  qsort(data, lo, pivotLocation-1);
  qsort(data, pivotLocation+1, hi);
}
```

Sort

Try quick sort on each of the following:

1. sort(1,2,3,4,5,6)
2. sort(3,6,1,2,5,4)

Quick sort

- Time
  - Technically O(n^2) worst case
  - On average, O(nlog n) if pivots aren't bad
    - Good quick sort implementation add some randomization to assure this
- Space
  - For linked lists
    - Fixed storage per procedure
    - Stack space O(log n)

Anagrams

Given a word and a Set of legal English words, provide a function to find all anagrams of the word.

Set anagrams(char word[], Set legalWords);

Example:
ret = anagrams("BEGIN",allEnglishWords)
ret contains two strings: "BINGE" "BEING" "BEGIN"

```
void makeAnagrams(char word[],char newWord[],
                  int pos,Set english,Set results) {
  for (int i=0;i<word.length;i++) {
    if (word[i]!=0) {
      newWord[pos]=word[i];
      if (pos<(word.length-1)) {
        char tmp=word[i];
        word[i]=0;
        makeAnagrams(word,newWord,pos+1,english,results);
        word[i]=tmp;
      } else {
        if (english.contains(new String(newWord))) {
          results.add(new String(newWord));
        }
      }
    }
  }
}
```

Ackermann Function

- A(1,j) = 2^j for j>=1
- A(i,1) = A(i-1,2) for i>=2
- A(i,j) = A(i-1,A(i,j-1)) for i,j>=2

- Try it:
  1. A(2,2)
  2. A(3,2)

Recursion

- Computation
  - Ackermann, Factorial
  - *Frankly it's rarely useful here*
- Divide and conquer
  - Merge sort, Quick sort
- Permutation/Traversal
  - Chess, Anagrams
- Search
  - We'll get there(trees)