CS 125 - Lecture 37

Objectives: recursion; Merge sort; Up next: MP7; Android Sunshine app;

1. With your partner, write one or two sentences describing binary search:

2. Given array Contact[], containing names and phone numbers sorted by name, write an efficient search algorithm. Return phone number for name, -1 if not found.

```
public class Contact {
   public int phoneNumber; public String name;
}
```

e.g. `int number = find(friends, “Larry Page”, 0, friends.length-1);`

```
public static int find(contacts[] array, String friend,
                                        int lo, int hi) {















}
```

3. ***Merge Sort:***

12	**14**	**11**	**16**	**18**	**15**	**13**	**17**

```
static void mergeSort(int[] data, int lo, int hi) {
   if (lo >= hi)  return;
   int mid = (lo + hi) / 2;

   mergeSort(data,?                        );
   mergeSort(data,?                        );

   int size = hi - lo + 1;
   int[] temp = new int[size];

   merge(data,temp,lo,mid+1,hi);

   for (int i = 0; i < size; i++) data[i+lo] = temp[i];
}

public static void merge(int[] a, int []tempArray,
                               int lower, int mid, int upper){
    int tempIndex=0;
    int leftLo = lower;
    int leftHi = mid-1;
    int rightLo = mid;
    int rightHi = upper;















}
```

4. Write a recursive song in ABA format:

```
//  Recursive method to create the pitches for son
//  Assume pitches in array have all been initialized to 440.0;

public static void createSong(double[] pitches, int lo, int hi, double
augment) {

   // divide the range of subarray into thirds and work on each third

   int oneThird = (hi - lo + 1) / 3;

}
```

5. You have an array of doubles. You want to search between indices 'lo' and 'hi'. Write a recursive method to find the largest product of two neighboring values. e.g. `findPair({ 1.0 , 1.0 , 7.5 , 4.0, 4.1 , 3.5 },0,5)` returns 30.0 (7.5 * 4.0), which is largest product of two neighboring values.

```
public static double findPair(double[] array, int lo,
                                                int hi) {
```

Write a FORWARD recursive method to find the first index of the largest product of two neighboring values. e.g. `findPair({ 1.0 , 1.0 , 7.5 , 4.0, 4.1 , 3.5 },0,5)` returns 2 because 7.5x4.0=30.0 is largest product of two neighboring values.

```
public static int findPair(double[] array, int lo, int hi)
{
```

CS 125 - Lecture 37

Objectives: insertion sort; recursion review

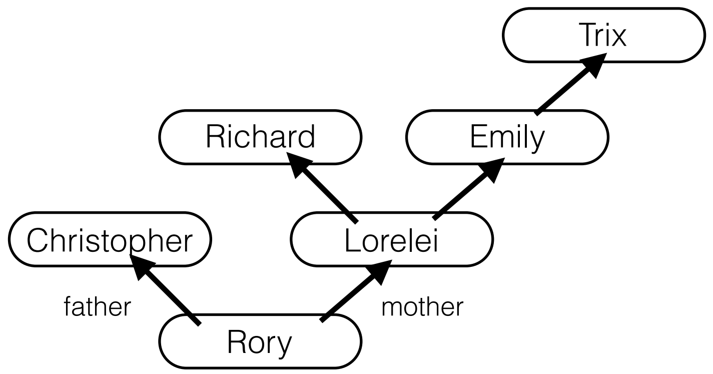
Up next: Midterm 3 tonight;

Midterm #3 (Wed):

Name:	Rm:	Name:	Rm:
A-D	23 PSYCH	Mb-R	103 TL
E-Ke	180 BEVIER	S - V	119 MSEB
Kf-Ma	120 ARCH	W-Z	269 EVRT

Conflict (Thurs):
1MSEB 119

1. Discuss with a neighbor: What is insertion sort? How does the algorithm work?



3. Write a tail recursive with a string accumulator method and a wrapper method to return the father with the longest name. Only consider the male lineage.

2. You have an array of doubles. You want to search between indices 'lo' and 'hi'. Write a recursive method to find the largest product of two neighboring values. e.g. `findPair({ 1.0 , 1.0 , 7.5 , 4.0, 4.1 , 3.5 },0,5)` returns 30.0 (7.5 * 4.0), which is largest product of two neighboring values.

```
public static double findPair(double[] array, int lo,
                                                int hi) {
```

Write a FORWARD recursive method to find the first index of the largest product of two neighboring values. e.g. `findPair({ 1.0 , 1.0 , 7.5 , 4.0, 4.1 , 3.5 },0,5)` returns 2 because 7.5x4.0=30.0 is largest product of two neighboring values.

```
public static int findPair(double[] array, int lo, int hi)
{
```

4. You need to climb a flight of stairs with N steps. You can climb one or jump three steps at a time. How many different ways are there to ascend the stairs?

***Process**: i) Identify the sub-problem; ii) Choose parameters and temp variables; iii) Write the base cases; iv) Write the recursive case;*

Extend your solution above to include:

1) an optional elevator 200 steps from the top.
2) a missing/broken step 15 steps from the top.
3) a non-optional worm-hole exactly 211 steps from the top.

Extend your solution above so that we only count paths that have a maximum of M moves (single steps or jumps).

5. Given a sorted two dimension array with no replicates values, how might one use recursion to efficiently search for the "coordinates" (row, column) of a given value?

```
double[][] mat = {{1.,2.,3.,4.}, {5.,6.,7.,8.}, {9.,10.,11.,12.},
{13.,14.,15.,16.}};
```

Strategy #1:

Strategy #2:

```
public static int findCol(double[] array, double number, int lo, int hi) {
     if (lo == hi) {
        int found = array[lo]==number ? lo : -1;
        return found;
      } else {
        int mid = (lo+hi)/2;

        if (number == array[mid]) return mid;

           if (number < array[mid]) {
              return findCol(array, number, lo, mid-1);
           } else {
              return findCol(array, number, lo+1, hi);
           }
      }
}
public static int findRow(double[][] array, double number, int lo, int hi){

}
public static void printCoords(double[][] array, double number) {

}
```