Objectives: algorithm analysis; traversing mazes

Up next: MP7: due Monday; all MPs final grading next Wednesday

1. ***Work with a neighbor***: How many seconds will each problem take? It takes 5s to ask each person a question and get a response, 10s for two people to swap positions. All participants are sitting in one row.

b. Find the smallest amount of change carried by one person.
Variables used: ?

Dataset size10 20 40
Best case :
Worst case :

c. Partition (divide) the group of people into two subgroups: "more-obnoxious-than Chapman" group and less-obnoxious-than-Chapman group. Variables used: ?

Dataset size10 20 40
Best case :
Worst case :

2. Write an expression for the ***worst-case*** running time of each algorithm. t(N) = …. Define any constants you need.

```
public static boolean foo1(int N) {     t(N)= ?
   int i = N * 2;
   return (i*i + N*N )>1000;
}

public static void foo2(int N) {
   int i= 2000;

   while( i <= N ){

      i += 2;

   }
}
```

Each take 1 time unit:

arithmetic operations
assignment (=)
boolean comparison
function activation / return
array element assignment/ access
variable declaration

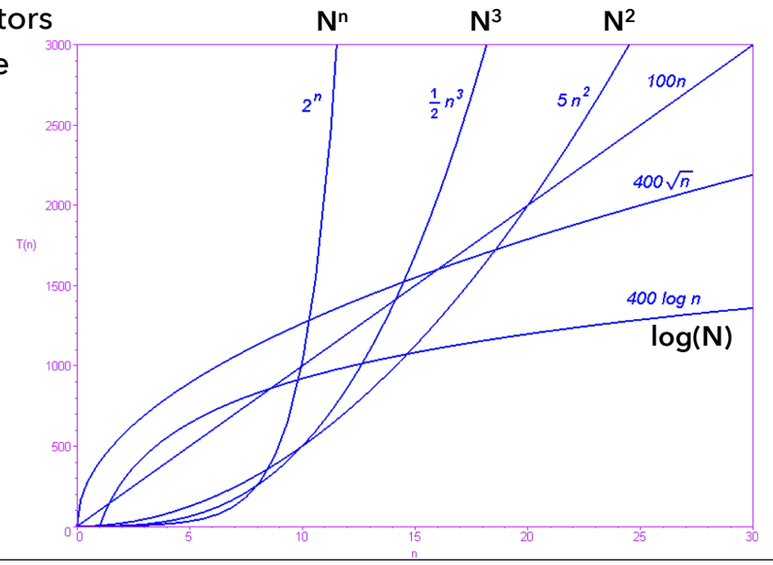
3. **Algorithm analysis - big idea:**

Evaluation and comparison of algorithms.

Method: Characterize the behavior of an algorithm by suppressing constant factors and smaller order terms/constants - we only care when N is large (lower order terms become increasing irrelevant)

Is this all we should ever care about?

4. **What's in a name?**
time complexity / analysis
algorithmic complexity / analysis
asymptotic complexity / analysis



5. **Why is Big-O the way to go?**

1) Abstract enough to ignore language and compiler details
2) Specific enough to allow differentiation and comparison between different algorithms and implementations, especially with large data inputs

Applications: sorting, searching, multiplying two integers, etc.

6. What is the algorithmic complexity for the iterative linear search?

7. What is the algorithmic complexity for the recursive linear search?

8. Write an expression for the ***worst-case*** running time of this algorithm.
t(N) = …. Define any constants you need.

```
public static int maxFunc(int[][] m) {
// assume a matrix with all dimensions N
int max = 0;
for (int i = 0; i < m.length; i++)
  for (int j = 0; j < m[i].length; j++)
     if (max < m[i][j]) max = m[i][j];
}
return max;

What if i++ and j++ above were i+=2 and j+=2?
```

9. Write an expression for the ***worst-case*** running time of each algorithm.
t(N) = …. Define any constants you need.

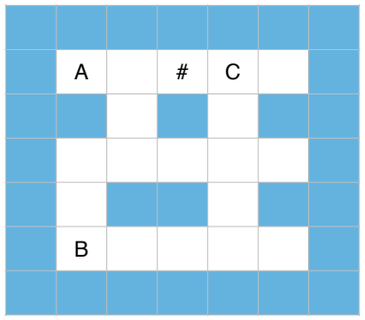
```
public static int foo3(int N, int[] data) {
   // assume N < data.length-1
   return data[N] * 5;
}


public static int foo4(int[] data) {
   int best = 0;

    // N is data.length
   for(int i=1; i<data.length; i++) {


      if(data[best] > data[i])
        best = i;
   }
   return best;
}
```

Each take 1 time unit:

---

arithmetic operations
assignment (=)
boolean comparison
function activation / return
array element assignment/ access
variable declaration

10. Searching Mazes: recursive algorithm
From starting points A,B,C, which (x,y) positions will be checked by the code below?



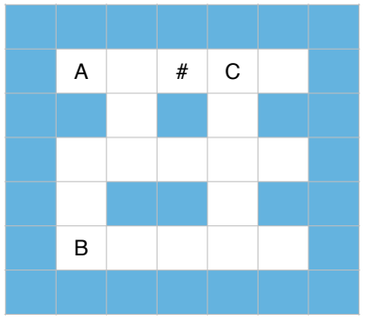
(0,0) is bottom-left of the maze.

Pseudo-code

```
String explore(x,y,flagX,flagY,wall,…         ) {
    if(wall[x][y]) return null; // No path here
    if(x == flagX && y == flagY) return  ""; // Found


     String goNorth = explore(    ,     , …)   ??
     String goEast  = explore(    ,     , …)   ??
     String goSouth = explore(    ,     , …)   ??
     String goWest  = explore(    ,     , …)   ??
// There are more elegant implementations

    if(goNorth != null) goNorth = "N" + goNorth;
    if(goEast != null)  goEast  = "E" + goEast ;
    if(goSouth != null) goSouth = "S" + goSouth;
    if(goWest != null)  goWest  = "W" + goWest ;


if(goNorth == null && goEast  == null
   && goSouth == null && goWest==null ) return null;
// No path 🙁

    String shortestViablePath = … pick shortest non-null path
    return shortestViablePath;
}
```

6. Searching Mazes: recursive algorithm

From starting points A,B,C, which (x,y) positions will be checked by the code below?



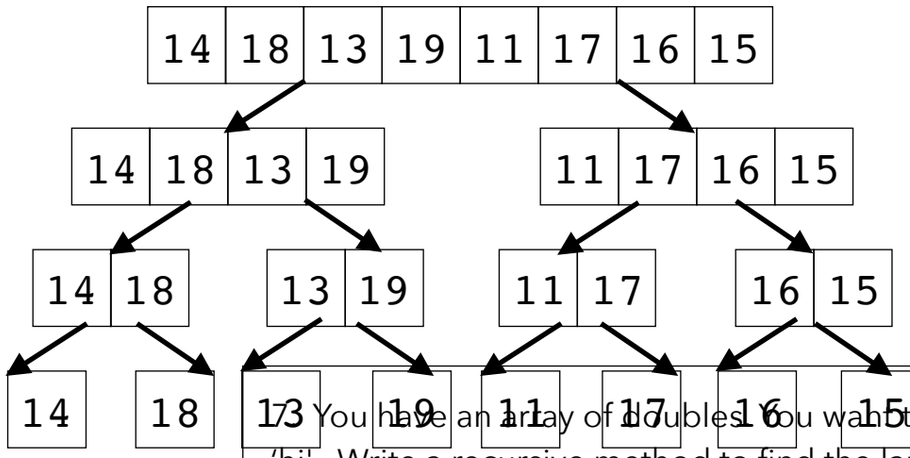
(0,0) is bottom-left of the maze.

Pseudo-code

```
String explore(x,y,flagX,flagY,wall,…          ) {
    if(wall[x][y]) return null; // No path here
    if(x == flagX && y == flagY) return  ""; // Found


     String goNorth = explore(    ,     , …)   ??
     String goEast  = explore(    ,     , …)   ??
     String goSouth = explore(    ,     , …)   ??
     String goWest  = explore(    ,     , …)   ??
// There are more elegant implementations

    if(goNorth != null) goNorth = "N" + goNorth;
    if(goEast != null)  goEast  = "E" + goEast ;
    if(goSouth != null) goSouth = "S" + goSouth;
    if(goWest != null)  goWest  = "W" + goWest ;


if(goNorth == null && goEast  == null
   && goSouth == null && goWest==null ) return null;
// No path ☹

    String shortestViablePath = … pick shortest non-null path
    return shortestViablePath;
}
```

1. Warm up: Think about the iterative algorithm for linear search. Now, write a recursive pseudo-code method for doing linear search:

2. How many levels does the merge sort activation diagram have (roughly)? Why?



3a. At each level of the tree (j =0, 1, 2, …), how many subproblems are there (as a function of N)?

3b. At each level of the tree (j =0, 1, 2, …), how many values are in the array passed into the recursive activation (as a function of N)?

3. You have an array of doubles. You want to search between in 'hi'. Write a recursive method to find the largest product of two values. e.g. `findPair({ 1.0 , 1.0 , 7.5 , 4.0, 4.1 ,` returns 30.0 (7.5 * 4.0), which is the largest product of two neighbo

```
public static double findPair(double[] array, in
                                              in
```

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

4. Write an expression for the ***worst-case*** running time of each algorithm. t(N) = …. Define any constants you need.

```
public static int foo3(int N, int[] data) {
   // assume N < data.length-1

   return data[N] * 5;
}


public static int foo4(int[] data) {
   int best = 0;

   // N is data.length
   for(int i=1; i<data.length; i++) {


      if(data[best] > data[i])
         best = i;
   }
   return best;
}
```

5. **QuickSort introduction**

12	14	11	16	18	17	13	15

```
static void quickSort(int[] data, int lo, int hi) {
      if (hi > lo) {

         int pivot = ?

         int newPivotIndex = ?

         quickSort(data, lo, newPivotIndex - 1);
         quickSort(data, newPivotIndex + 1, hi);
      }
   }
```

6. **QuickSort, partitioning…**

```
static int partition(int[] data,int lo,int hi,int pivotIndex)
{
  // Move the pivot out of the way; for now we'll put
  // it at the start of the list and ignore it until the end.




  // Start working in, from both L and R ends of the list









  // The pivot will need to go to the left of the final
  // boundary if the last value is larger than the
  // pivot value.



}
```

5. **QuickSort summary:**

12	14	11	16	18	17	13	15

How does quick sort differ from merge sort? better? worse?

2. ***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;









}
```