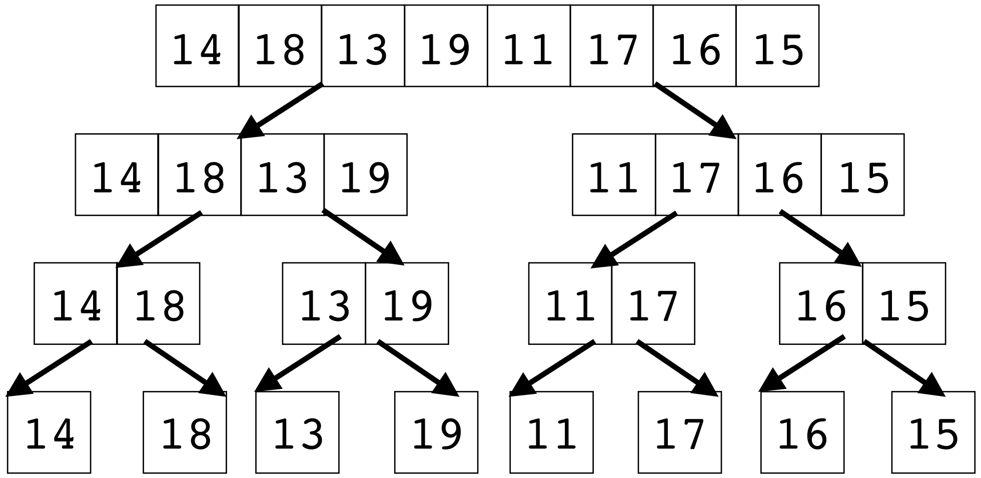
CS 125 - Lecture 42

Objectives: Algorithm analysis; QuickSort

Up next: Xilften E.C. Tu/We this week; MP7 tonight; all MPs regraded last time Wed 8PM

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)?

3c. What is the BigO of Merge Sort?

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

```
public static int foo1(int N, int[] data) {
   // assume N < data.length-1
   return data[N] * 5;
}

public static int foo5(int x) {
   if(x<=0) return 0;
   int half = x/2;
   return 1 + foo5(half);
}

public static int foo2(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

4. What is the Worst case Running Time for the following algorithms - "Big O" notation?

```
public static boolean foo3(int N) {
   int x=0;
   for(int i=0;i<N;i++)
      x+=i*i;
   return x;
}



public static void foo4(int[] data) {
   int index = 0;
   while(index<data.length) {
      index+=Math.max(1,data[index]);
    }
}
```

5. **QuickSort Big Idea:**

6. **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);
      }
   }
```

8. **QuickSort summary:**

12	14	11	16	18	17	13	15

How does quick sort compare to merge sort? better? worse?

7. **Partitioning an array into those elements less than a magic number and those elements greater than a magic number**

```
static int partition(int[] data,int lo,int hi,int magicIndex)
{
  // Move the magic number 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 magic number will need to go to the left of the final
  // boundary if the last value is larger than the
  // magic number.


}
```

9. What is the Worst case running time for the algorithm : "Big O" notation =

```
public static void foo6(int[] array) {
   if(array.length ==0) return;
   int[] space = new int[array.length - 1];
   foo6(space);
   space[0]= 5;
}
```

9. What is the algorithmic complexity, O(?), for linearly searching two arrays of length N for a number p?

10. **Partitioning an array into those elements less than a magic number and those elements greater than a magic number**

```
static int partition(int[] data,int lo,int hi,int magicIndex)
{
  // Move the magic number 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 magic number will need to go to the left of the final
  // boundary if the last value is larger than the
  // magic number.

}
```

11. Write pseudo-code to check if two arrays a[N] and b[N] have a number in common. What is the algorithmic complexity, O(?):

b.) Modify the above to check if array a[N] has duplicate values in the array. What is the algorithmic complexity, O(?):

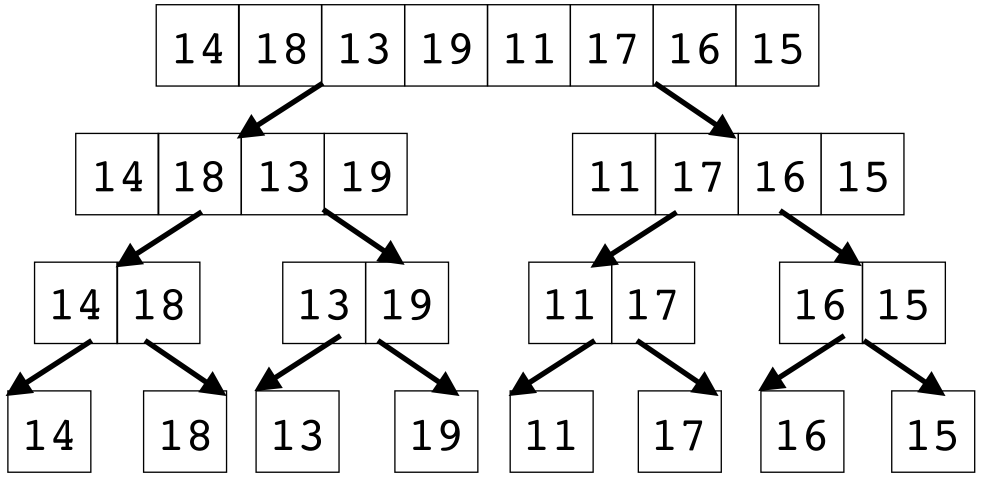
12. Create an activation diagram for f1(128), then write an expression for the running time of f1(128).

```
int f1(int x) {
  if(x == 0) return 1;
  return 2 * f1(x/2);
}
```

13. Create an activation diagram for f2(128), then calculate the running time of f2(128).
Challenge: What does this tell you about MergeSort?

```
void f2(int x) {
  if(x > 1) {;
     f2( x/2 );
     f2( x/2 );
     sleep(x); // sleep for x ms
  } else sleep(1);
}
```

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)?

10. Write pseudo-code to check if two arrays a[N] and b[N] have a number in common. What is the algorithmic complexity, O(?):

b.) Modify the above to check if array a[N] has duplicate values in the array. What is the algorithmic complexity, O(?):

9. What is the algorithmic complexity, O(?), for linearly searching two arrays of length N for a number p?

11. Create an activation diagram for f1(128), then write an expression for the running time of f1(128).

```
int f1(int x) {
  if(x == 0) return 1;
  return 2 * f1(x/2);
}
```

12. Create an activation diagram for f2(128), then calculate the running time of f2(128).

Challenge: What does this tell you about MergeSort?

```
void f2(int x) {
  if(x > 1) {;
     f2( x/2 );
     f2( x/2 );
     sleep(x); // sleep for x ms
  } else sleep(1);
}
```