Lab 7 Notes

CS 61BL Summer 2019

Lab 7

July 5, 2019

Name: Jon

SID:

Please complete this worksheet during your lab, and turn it in to your TA by the end of your section. You are encouraged to work with your neighbors collaboratively.

Section Number: 01 ~~02~~ 03 04 05 06 07 08 09 10 11 12

1 Timing

1.1 Regarding the results of `Timer.java`, why are there differences between some students numbers and other students numbers? Why is it that the amount of time it takes to sort the same number of elements isnt always the same? What might contribute to these differences?

Processor speed, other processes running,

(+ the 61C answer: better caching, better compilation/assembling)

2 Counting

2.1 Consider an `if ... else` of the form,

```java
if (A) {      — a
    B;        — b
} else {
    C;        — c
}
```

true:

```
if (A) {
    B
}
```

= a+b

false:

```
if (A) {
} else
    C
}
```

= a + c

where `A`, `B`, and `C` are program segments. (`A` might be a method call, for instance.)

Let a be defined as the number of steps it takes to evaluate `A`, b be that for `B`, and c be that for `C`. How many steps does it take to evaluate the entire `if ... else` block in terms of a, b, c? *Assume that* $a + b < c$.

Best case: a+b

Worst case: a+c

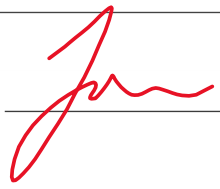
do NOT ever do this in the future, there is much more complexity than just # steps.

3 Using the Right Bounds

3.1 Copy the statement below, replacing "(your name here)" with your name, and add your signature acknowledging your understanding of the "Best Case and Worst Case" section of the spec.

I, (your name here), agree that Big Omega is not the same as Best Case and Big O is not the same as Worst Case.

No thanks.

Your Signature: 

4 Analyzing Functions

Provide the tightest bound possible for each of these functions, in terms of the function parameter `n` or in terms of the length of the `array` (which you may also call `n`).

4.1 What is the runtime? Provide the appropriate bound notation and order of growth.

```
1 public void f1(int[] array) {
2     for (int i = 0; i < array.length; i++) {
3         for (int j = 0; j < Integer.MAX_VALUE; j++) {
4             System.out.println("Hi!");
5         }
6     }
7 }
```

n

constant

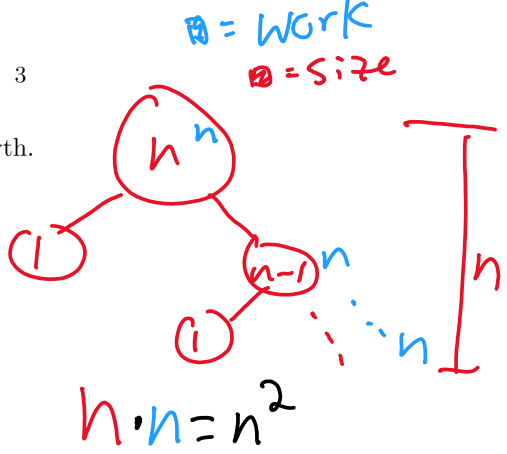
$\Rightarrow \Theta(const \cdot n)$

Answer: $\Theta(n)$

4.2 What is the runtime? Provide the appropriate bound notation and order of growth.

```
1  public int f2(int n) {
2      if (n <= 1) return n;
3      f1(new int[n]);
4      return n + n * f2(n - 1) + n * n * f2(1);
5  }
```

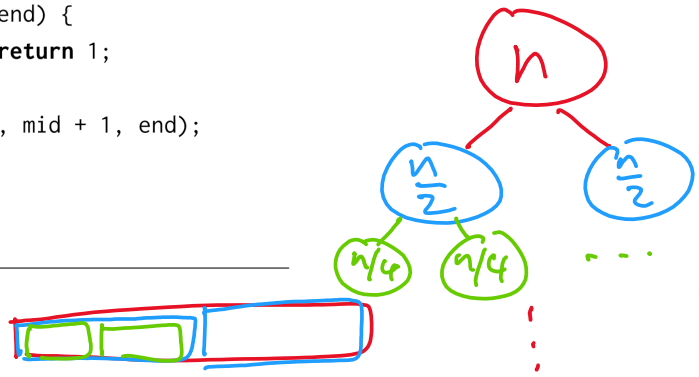
Answer: $\Theta(n^2)$



4.3 What is the runtime? Provide the appropriate bound notation and order of growth.

```
1  /* When f3 is first called, start will be 0 and end will be the length of the array - 1 */
2  public int f3(char[] array, int start, int end) {
3      if (array.length <= 1 || end <= start) return 1;
4      int mid = start + ((end - start) / 2);
5      return f3(array, start, mid) + f3(array, mid + 1, end);
6  }
```

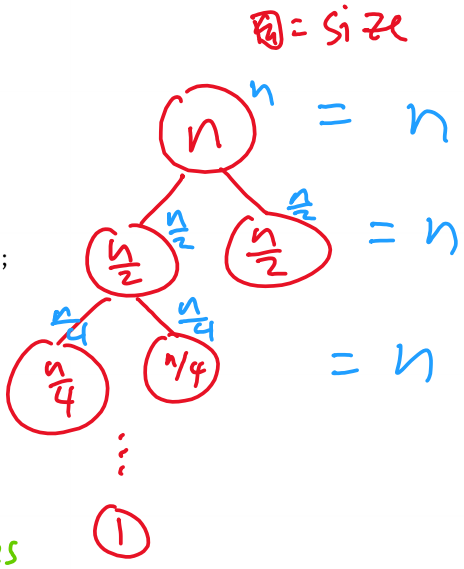
Answer: $\Theta(n)$



4.4 What is the runtime? Provide the appropriate bound notation and order of growth.

```
1  /* When f4 is first called, start will be 0 and end will be the length of the array - 1 */
2  public int f4(char[] array, int start, int end) {
3      if (array.length <= 1 || end <= start) return 1;
4      int counter = 0;
5      for (int i = start; i < end; i++) {
6          if (array[i] == 'a') counter++;
7      }
8      int mid = start + ((end - start) / 2);
9      return counter + f4(array, start, mid) + f4(array, mid + 1, end);
10 }
```

Answer: $\Theta(n \log n)$



how many layers?
divide by 2 every time, until 1.
$\Rightarrow \log_2(n) = \Theta(\log n)$
math: $2^{\log_2 n} = n$, aka $2 \cdot 2 \cdots 2$ ($\log(n)$ times) is n.

n work per layer,
height $= \Theta(\log n)$ → $\log n$ layers $\Rightarrow \Theta(n \log n)$

4.5 What is the runtime? Provide the appropriate bound notation and order of growth.

```
1  public void f5(int n) {
2      int[] array = {1, 2, 3};
3      while (n > 0) {
4          f1(array);
5          n = n / 2;
6      }
7  }
```

3 3 3 . . .

n n/2 n/4 ... 1

= 3 log n

logn

same as above,

Answer: $O(\log n)$

4.6 What is the runtime? Provide the appropriate bound notation and order of growth.

```
1  public void f6(int[] array) {
2      for (int i = 1; i < array.length; i++) {
3          if (array[i] == array[i-1]) {
4              System.out.println("Sarah is the potatoest");
5              return;
6          }
7      }
8  }
```

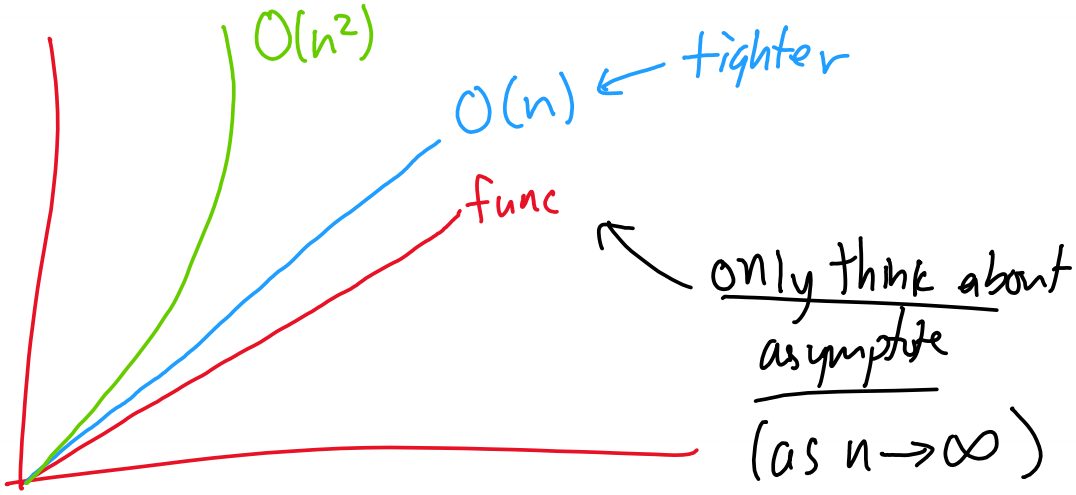
can return early!

$\Rightarrow O(n)$ upper bound

Answer: $O(n)$

- want to make as tight as possible

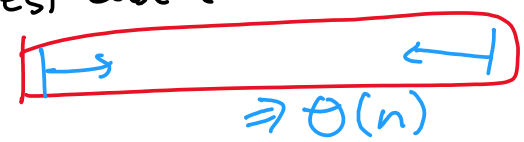
also $O(n^2)$, but not as tight



4.7 What is the runtime? Provide the appropriate bound notation and order of growth.

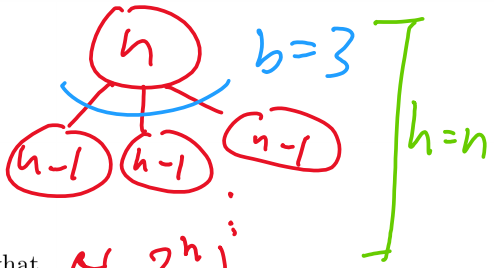
```
1  /* When f7 is first called, start will be 0 and end will be the length of the array - 1 */
2  public int f7(int[] array, int start, int end) {
3      if (array.length <= 1 || end <= start) {
4          return 1;
5      } else if (array[start] <= array[end]) {
6          return f7(array, start + 1, end - 1);
7      } else {
8          int tmp = array[start];
9          array[start] = array[end];
10         array[end] = tmp;
11         return f7(array, start + 1, end) + f7(array, start, end - 1)
12                 + f7(array, start + 1, end - 1);
13     }
14 }
```

best case (sorted)



$\Rightarrow \Theta(n)$

← confusing, just say 3 calls, all don't work out (worst case)



$\Theta(3^n)$

Answer(s): $\Omega(n), O(3^n)$

4.8 For `f7` in the previous subquestion, instead of asking "What is the runtime", what if we asked you "What is the best and worst case runtime?" instead?

Best Case: $\Theta(n)$ Worst Case: $\Theta(3^n)$