I) <https://repl.it/@_Ann_/hw3-N1>

1)

N = 6

000000

000001

000010

000100

000101

001000

001001

001010

010000

010001

010010

010100

010101

100000

100001

100010

100100

100101

101000

101001

101010

2)

Recursive calls 50

3) For n = 5

Strings Count 13

For n = 6

Strings Count 21

For n = 7

Strings Count 34

4)

Fibonacci Sequence

n = 1, count = 2 = fib(3)

n = 2, count = 3 = fib(4)

n = 3, count = 5 = fib(5)

n = 4, count = 8 = fib(6)

n = 5, count = 13 = fib(7)

n = 6, count = 21 = fib(8)

n, count = fib(n+2)

 The ones that end in 0 can all be obtained by adding a 10 on the end of an allowable string of length n−2, and the ones that end in 1 can all be obtained by adding 0 to the end of an allowable string of length n−1. Hence, an=an−1+an−2, (an is number of strings w/o consecutive 1s) like the Fibonacci numbers; only the starting point is different. an=Fn+2an=Fn+2

II) <https://repl.it/@_Ann_/hw3-N2>

III) <https://repl.it/@_Ann_/hw3-N3>

1. N\*(N-1)/2

Because it’s going to first compare from the end couples till it reaches the first two couple

8 7 6 5 4 3 2 1

2 1

3 1

4 1

5 1

6 1

7 1

8 1

Then it is going to check each couple until the second element

3 2

4 2

5 2

6 2

7 2

8 2

Now check until the third element

4 3

5 3

6 3

7 3

8 3

Till fourth element

5 4

6 4

7 4

8 4

Till fifth element

6 5

7 5

8 5

Till seventh element

7 6

8 6

Last couple check

8 7

**7+6+5+4+3+2+1 = 28**

**Largest number of comparisons if the sequence is in decreasing order**

2)

Worst Case running time

O(n^2)

Because it is going to loop through two arrays, doing the comparisons and swapping the elements

3)

Worst

8 7 6 5 4 3 2 1

Best

1 2 3 4 5 6 7 8

IV) <https://repl.it/@_Ann_/hw3-N4>

The program inserts as many numbers into buckets as you provide.

For N numbers there are going to be N insertions into distinct/non distinct (if the characters repeat) buckets.

2)

Worst Case running time: O(n^2)

Because if we have N characters. It is going to loop through all N buckets and then loop through each one of them. Let’s suppose we non distinct characters then first loop will loop n times (let’s suppose we have n length buckets) and n times throught he second loop as we have n characters in one of the buckets.

3)

<https://repl.it/@_Ann_/hw3-N4c> - the implementation with uppercase

Running time will be the same, as only the size of the buckets will change and the decrement/increment implementations. Which is just 2\*m (where m is the previous buckets number). Because I break out of the loop when the index become the length the program won’t loop through the buckets unnecessarily when all the characters where found. O(m+n) without uppercase. O(2m+n) with uppercase.

V) <https://repl.it/@_Ann_/hw3-N5>

1) n\*(n-1)/2 times

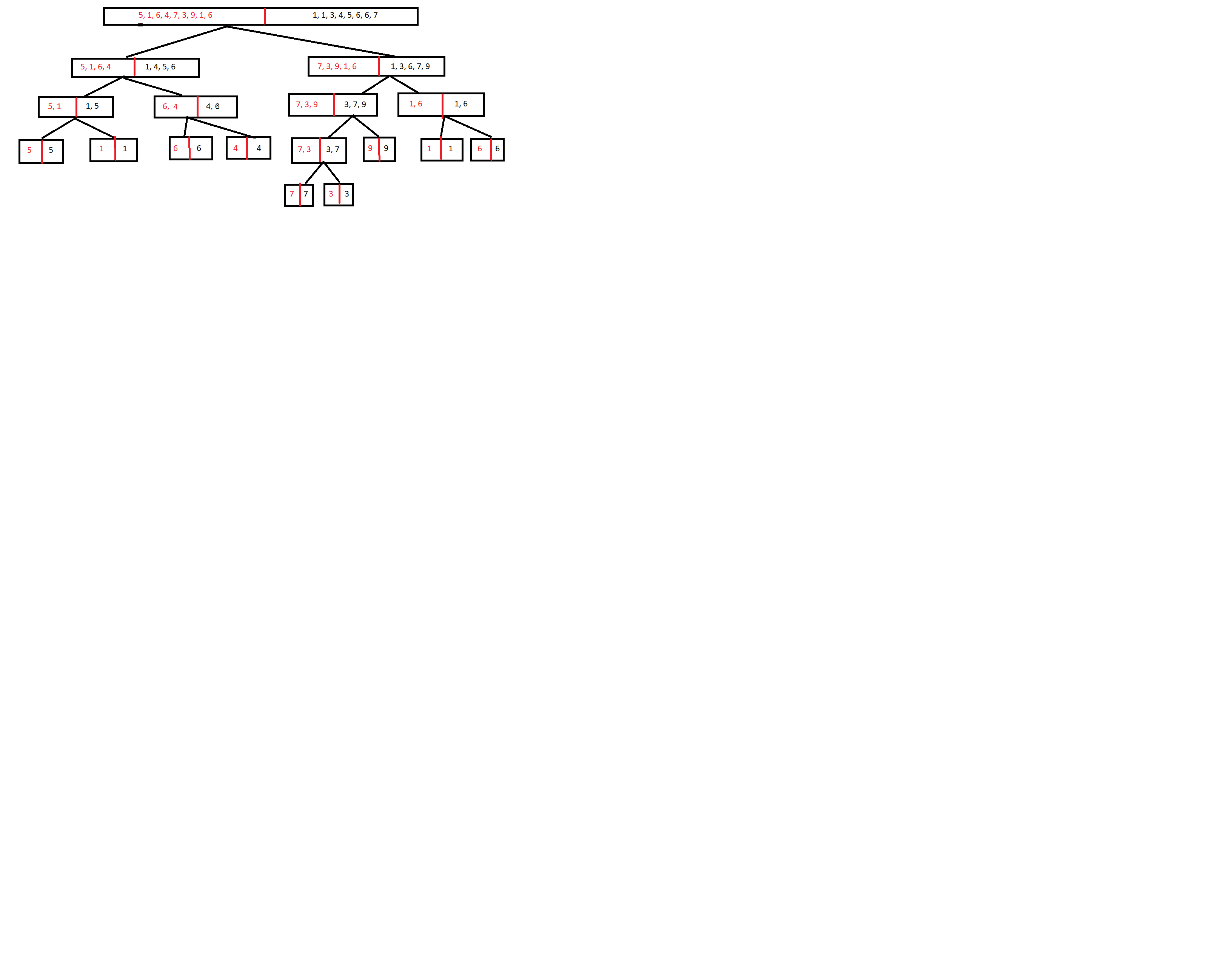
2) worst case O(n^2) because it will loop through for loop for each index and every time it will enter the wile loop and swap n times

3) Best and Worst are the same because it will check and find the minimum for each index anyways.

VI) <https://repl.it/@_Ann_/hw3-N6>

1. N\*(N-1)/2
2. Worst case O(n^2) for each index it will loop again through the array and find the minimum and swap
3. Best 1 2 3 4 5 6 7 8

Worst 8 7 6 5 4 3 2 1

VII)

1. 18
2. O(m+n)
3. O(nlogn) Each level corresponds to one level of recursion. Each of those causes two calls with arrays of size n/4, for a total of 4 such calls at level 2. Thus, the total work at each level is Θ(n). Next, we calculate the number of levels. Since the array size halves with each extra level, and it starts with n and finishes with 1, the number of levels is log2 (n). (Technically, it is ⌈log2 (n)⌉, but such details don’t affect the running time analysis.) So the total work, summing up over all levels, is Θ(n log2 (n)).

In our case there are 3 levels, and each level causes 2^k calls

VIII)

1. I choose always the last one
2. 16 comparisons
3. O(n^2) when the pivot is the smallest or the biggest one. Because in that case it will loop through the array n times (not n/2 or else)

