Java DSA Notes:

• Trim: Remove the white space from the both ends of the given word. Int ch=in.next().trim()

• SHADOWING :is overriding the variable (global variable is overrides with variable in the local scope).

• VARIABLE LENGTH ARGUMENT:

static void fun(int ...v)// will take the array of

{} integer.

-Length is not constant depends on arguments.

-Must be the last int the arguments List

• MULTIPLE ARGUMENTS:

static void fun(int a,int b,String ...v)

{

}

fun(1,2,”Sanket”,”Kumbhar”);

• FUNCTION OVERLOADING:

-Functions having the same name but different parameters(number of arguments

or different types).

• ARMSTRONG NUMBER: 1 5 3

1\*1\*1+5\*5\*5+3\*3\*3=153 IS AN ARMSTRONG numbers(addition of cubes)

• ARRAYS :-

datatype[] = new datatype[size];

1.Arrays objects are in heaps

2.java language specification says heaps object are not

continuous[continuous blocks] allocated.

3.new is used to create an object .

4.Empty array contains 0 and empty string contains null.

5.Arrays are mutable in Java. While strings are immutable.

6.To find the actual Length of array we can do array.length-1.

7.To Print the array without loops we can simly do Arrays.toString(arrayname);

• Arraylist:-

◦ Size:-

▪ Increases Size automatically as need

▪ Size is fixed internally

▪ if the list fills by some amount

▪ it creates a new list of double size and moves the data to the new list and deletes the old list

▪ Does the same every time when if the list fills by some amount

◦ functions:-

▪ list is an object of arraylist

▪ list.add(value)

▪ list.contains(value)

▪ list.remove(index)

▪ list.set(index,value)

▪ list.get(index)

• Array-list in Array-list (Multidimensional ArrayLIst)

◦ First thr array list is like-

▪ [null,null,null,null]

• So for that we need to initlise the array LIst

◦ ArrayList<ArrayList<Integer>> list = new ArrayList<>();

◦

◦ for (int i = 0; i < 3; i++) {

◦ list.add(new ArrayList<>()); //initilisationing the arraylist

◦ }

• Binary Search

◦ way more efficient that Linear Search.

◦ N=2\*K

◦ applying log on both sides k=log n base 2

◦ Where as in linear search 1m comparisons only 19-20 in Binary Search

◦ simple method is mid = start + end /2

▪ But integer has a fixed Size ,might be possible start + end exceeds the range of integer in java

▪ Better Way is mid = start +(end – start)/2

▪ if(isAsec) {

if (target < arr[mid]) {

end = mid - 1;

} else if (target > arr[mid]) {

start = mid + 1;

} else {

return mid;

} ▪

• For Each Loop

◦ for(int I :array)

◦ {

if(i>1)

count++;

◦ }

◦ the following loop will execute for each element in array till the array is empty

◦ The array value is in the place of I

• Space complexity

◦ O(1)

▪ No extra space is required. Like copying the array and creating a new array

▪ We just perform the operations on the new array

▪ Inplace Sorting Algorithms

• Time Complexity

◦ Best Case :- O(n) – Array is sorted

▪ For the particular pass if the j value is never swapped it says that the array is sorted.

▪ Its actually N-1 but we ignore the constant is TC

◦ Worst Case :- O(n2)-Array is sorted in descending order

* • Bubble sort(Exchange sort/Sinking Sort)
  + two for loops

◦ stable sorting algorithm

▪ When the original order is maintained for the values being the same

• Selection Sort

◦ It performs well in small list/array and two for loops

◦ we are going to select a element and find its right place in array

◦ We will find the **largest element in array and put it in last**

◦ not stable

◦ Because we are finding the maximum element in array

◦ Best Case

▪ O(n2)

◦ Worst Case

▪ O(n2)

* Insertion Sort
  + two for loops
  + we will **start form the starting and will sort the array like at first till index 1 then 2 then 3 then 4**
  + 5,3,4,1,2

o-1

0-2

0-3

* + **at cond j < j-1** we should know that the the left side array is already sorted and no need to check there if j < j-1 then all the elements before j-1 will also be less then j so we should break the loop when this condition gets hitted
  + ith loop will only run till n-2 because if we are going till n-1 we will encounter a condiiton where j=i+1 which will be **out of bound .**
  + **Why we use insertion sort**
    - Steps get reduced if array is sorted
    - number of swaps are reduced as compared to bubble sort
    - Its stable.
    - Used Smaller values of N works => if array is partially sorted
    - It takes parts in hybrid sorting algorithms
  + **Complexity**
    - Worst Case:Descending sorted :-O(n2)
    - N-1 comparisons
    - N(N+1) = N-1(N-1+1) =N(N-1)=O(n2-n) = O(n2)
    - 2 2 2 2
    - Best Case : Ascending sorted:O(n)-**Linear Complexity**
* **Cyclic Sort**
  + **Sorts arrays in single for loop(single pass)**
  + Numbers should be from 1 to N only no number should be greater then 2
  + In Sorted array every elements are in form of index = value – 1;
  + Worst Case O(n-1)(swaps)+n(check present at correct index)
    - O(2n-1)
    - Best Case :O(n)
  + Sorting method
    - Check if the element is present in its correct index using the formula index=value -1;
    - if yes move forward else swap it with correct index
* Tips
  + Everything in in java that start with a capital letter is a Class
* String
  + Every string we create is an object of type string.
  + String name = {“Sanket Kumbhar”};
  + string pool : separate memory structure inside a heap
    - Similar values will point to the same pool
    - Value is not changed if anyone one of the object variable is changed(**Immutability**)
    - Strings objects are immutable in Java(Security Reasons)
  + Why we cant modify string objects ?
    - As Java uses the concept of String literal. Suppose there are 5 reference variables, all refer to one object "Sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why String objects are immutable in Java.
    - String a = “Sanket”;
    - a = “ sanket k”
      * Note that the Object Sanket is not changed but sanket k a new object is created and
  + Because we if we change the
  + Comparison of strings
    - Strings b = “kunal”;
    - a

Kunal

* + - b
    - String a = “Kunal”;
    - Both are equal(==) because both value are same and are pointing to the same location value/
    - Strings a = new String(
  + Different objects of same“Sanket”);
    - Strings b = new String(“Sanket”);
    - a Kunal
    - b Kunal
    - Both are not equal(==) because they are not pointing to the same location.
    - new will force java to create a new object both will be stored in heap but outside the pool.
  + While Comparison if two objects are there(new)
    - .equals() functions for strings its just cares about the value and not the reference variable
  + println
    - first value of => to string => print
  + **Concatenation**
    - ‘a’ + ‘b’ = 195
    - “a” + “b” = ab
    - ‘a ’+2 = 99 same as ‘a’ + ‘2’
    - **+ will work for every type of object if on of its type is Strings**
    - **When an integer is added to string Integer that will call toString**
  + **StringBuilder**
    - **append**
    - **charAt**
    - **indexOf**
    - **contentEquals**
* **PATTERNS**
  + Approach

1. Identity Number of rows
2. Identify for every row how many cols are there/( there are different)types of elements in the columns
3. What do uu need to print

* Note:try to find a formula for row col.

**x**

* **Problem solving**
  + Array is sorted then **apply Binary search**.
  + Given range from 1 to n **apply Cyclic Sort.**
  + **O(n)time and constant space cyclic sort**
  + **% is** placeholder ,f for float,s string
* **Math For DSA**
  + **Bit Manipulations**
    - 1 byte = 8 bits
      * (10)2 = 00001010 Most Significant Bit :- tells +ve(0) or -ve(1)
      * 87654321 Least Significant Bit : tells even(0) or odd(1)
      * LAST bit tells the sign of the number and rest of the tells the value
    - Negative of a number
      * take compliment of a number and add one to it(2’s Compliment method) :-when stored in 1 byte data type
      * WHY ? 2’s compliment gives negative of a number
        + 0 – a number = negative of a number
        + 10000000000 0 1 asach takla ahee (ignored because more
        + -00000001010 -10 equal then 8 bit)
        + 1111111111+1
        + now expression will be 1111111111+1 - 00000001010
        + we can write it as 1111111111 - 00000001010 +1
        + same as taking the compliment
        + So same as taking the compliment and then add one to it
      * Range of a number
        + how many can we store in one byte – 8 bits
        + for 8 bits the total number are 28 =256 total numbers in 8 bits

the first bit tells us the positive and negative as we already know then the total number will be

128 negative numbers and 128 positive numbers

range is -128 to 128

BUT…………0 is also there and it is counted in positive

So now the range is -128 to 127 including 0

* + - * + for n bits

-2(n – 1) to 2(n – 1) - 1

* + **Bit-wise Operators**
    - **Shifting**
      * Left Shift (<<) shall be considered as multiplication by 2^NSimilarly, Right Shift (>>) shall be considered as division by 2^N
    - AND &
      * Both values should be true entire expression is true
      * When you & 1 with any number you will be getting the same number.
      * Use Case
        + Given Number n find if its even or odd
    - OR ||
      * If any value is true entire expression is true.
    - XOR ^(Exclusive OR(if and only if))
      * if you have two number on/
      * XoR any number with 1 getting the compliment(opposite of that number) of that number
        + a ^ 1 = a upper dash(compilment)
        + a ^ 0 = a
        + a ^ a = 0
  + Conversion of Numbers(Number Systems)
    - Decimals number systems (0-9)10
      * Base – 10
    - Binary Numbers (0 and 1(0,1))
      * Base is 2 ()2
    - Octal Numbers(0-7)
      * 0 start
      * No 8
      * Base is 8()8
    - HexaDecimal()
      * 0-9
      * then ABCD
      * Base is ()16
    - CONVERSION
      * Decimal to base b
        + decimal base (17)8 to ()2

Keep diving the number , take remainder,write in oppsite

2 42 0 (101010)2 = (42)10

2 21 1

2 10 0

2 5 1 write in the direction of arrow

2 2 0

1 1

* + - * base b to Decimal
        + Multiply and and the power of base to digits

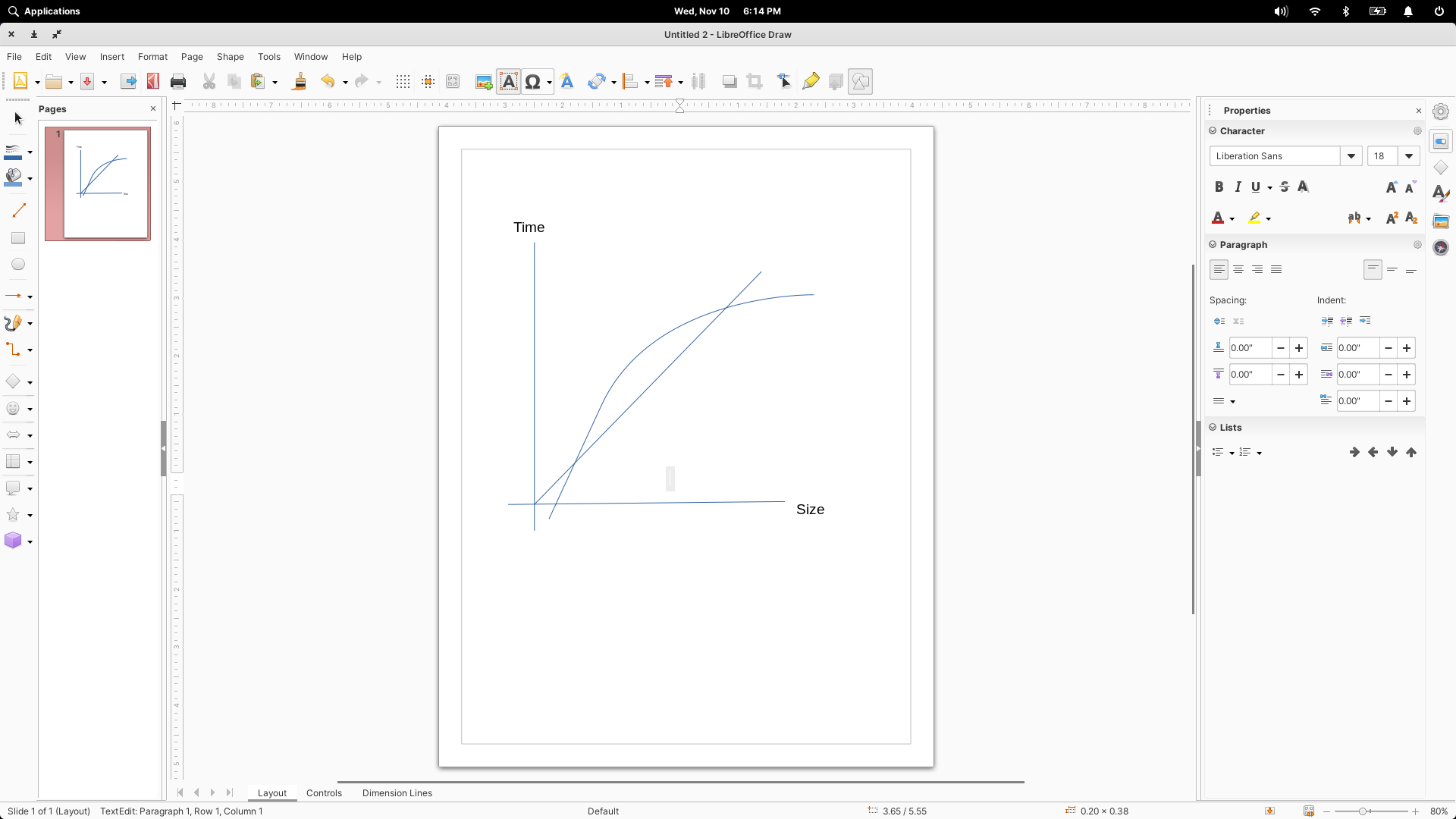
(101010)2 = (?)10

1 \* 2^5 + 0 \* 2^4 + 1 \* 2^3 + 0 \* 2^2 + 1 \* 2^1 + 0 \* 2^0=42

* + - Base Number System(Binary Number System)
    - 1 >> right shift
    - 1 << left shift
* Optimized Ways for sum problems
  + Prime Numbers
    - Can only make checks till the sqrt of that number
    - More optimised way is Sieve of Erathanoneses
      * public class SievePrime {  
         public static void main(String []args)  
         {  
         int n=40;  
         boolean []primes = new boolean[n+1];//n+1 because it also contains 40  
         *sieve*(n,primes);  
         }  
          
         //false in array the number is prime  
         static void sieve(int n,boolean [] primes)  
         {  
         for(int i=2;i\*i <= n;i++)  
         {  
         if(!primes[i])  
         {  
         for(int j=i\*2;j <= n;j+=i) //i\*2 because we start fro the first multiple of it and j+=i because we are fnnding the multipe so +i  
         {  
         primes[j] = true;  
         }  
         }  
          
         }  
          
         for(int k=2;k <= n;k++)  
         {  
         if(!primes[k])  
         {  
         System.*out*.print( k +" ");  
         }  
         }  
         }  
        }
      * Time Complexity
        + 2/n+3/n+5/n+7/2+……………...p/2; ps largest and smallest the n prime number
        + Using the Harmonic Progression for primes

= Log(logN)

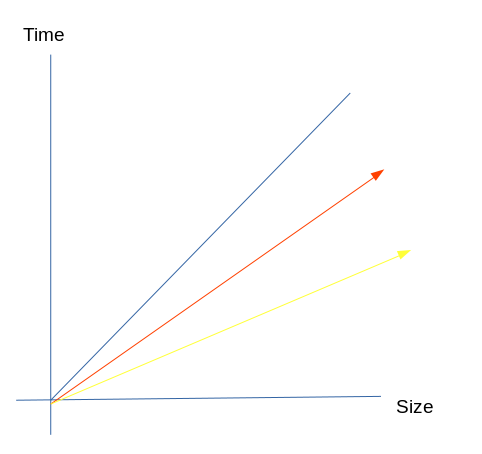
* + - * + O(N \* log(log n))
  + Sqrt of number
    - Binary search for perfect squares
      * package com.Sanket;  
          
        public class BinarySearchSQRT {  
          
         public static void main(String args[])  
         {  
         int n=56;  
         int p=3; for(int i=0;i < p;i++)  
         {  
          
         }  
         System.*out*.printf("%.4f",*sqrt*(n,p));  
         }  
          
         static double sqrt(int n,int p)  
         {  
         int s=0;  
         int e = n;  
          
         double root = 0.0;  
         while(s <= e)  
         {  
         int mid = s + (e - s) / 2;  
          
         if(mid\*mid == n)  
         {  
         return mid;  
         }  
          
         if(mid \* mid < n)  
         {  
          
         s = mid + 1;  
         }  
         else  
         {  
         e = mid - 1;  
         }  
         }  
         double incr=0.1;  
         for(int i=0;i < p;i++)  
         {  
         while(root \* root <= n)  
         {  
         root += incr;  
         }  
         root -= incr;  
         incr/=10;  
         }  
         return root;  
         }  
        }
    1. Newtons raphson sqrt methods
       - package com.Sanket;
       - public class NetwonRaphsonSQRT {  
          public static void main(String []argd)  
          {  
          System.*out*.printf("%.3f",*sqrt*(40));  
          }  
          static double sqrt(int n)  
          {  
          double x=n;  
          double root;  
          while(true)  
          {  
          root = 0.5 \* (x + (n/x));  
          if(Math.*abs*(root-x) < 0.5)  
          {  
          break;  
          }  
           
          x = root;  
          }  
          return root;  
          }  
         }
    - Factors of a number
      * //only check till the sqrt of n because afte that the elennt will get repeated  
         //while checking we should know that if there is a condition that  
         //factors of 20 are n%2==0 then 2 is a factor but the number that is going to be multiplied by two is also an factor i.e n/2 = n/i  
         // so the factors are i and n/2 where n/i != I
    - **Properties of Modulos(%)**
      * (a+b)%m = ((a%m)+(b%m))%m
      * (a-b)%m = ((a%m)-(b%m))%m
      * (a\*b)%m = ((a%m)\*(b%m))%m
      * (a/b)%m = ((a%m)+(b-1 %m))%m
      * b-1 %m Multiplicative modulo inverse
      * (a%m)%m = a % m
      * mx % m = 0 V x C +ve Integers
      * Extra : If p is a prime number which is not a divisor of b then I I can say abp-1 % p = a % p due to farmats little theorem.
* Euclids Algorithm to find GCD
  + gcd(a,b) = gdc(rem(b,a),a) -greatest common factor – when a==0 return b
* LCM
* Recursion
  + Do not overthink before solving
  + A function that calls itself
  + Needs a base condition because for the function to know where to stop
  + WHY Recursion ?
    - It helps us in solving bigger Complex problems in a simple way
    - You can convert recursive solutions into iteration and vice versa
    - Space complexity is not constant because of recursive calls
  + How to know it is an recursive Problem
    - Identify If the problems breaks down into smaller problems
      * Write the recursive solution of problem if needed
      * Draw the recursive tree
      * About the tree
        + See the flow of functions how they are getting in to stack
        + Identify and focus on the left tree and right tree
        + Draw it
      * see how the values are returned at every step and see what type of values we are returning scsc
      * See where the function call will come out of the main function
  + MAKE SURE U RETURN THE RECURSIVE LINE IF THE FUNCTION HAS A RETURN TYPPE
* **TIME AND SPACE COMPLEXITY**
  + Time taken != Time Complexity
  + Function that gives us the relationship about how the time will grow as the input grows .
  + Binary Search : Log (n)
  + Linear Search : O(n)



Linear Search

Binary Search

* + **What do we consider when thinking about complexity** 
    - The Worst Case because its going to crash the code
    - Always look for the worst case
    - Time will from machine from machine

c

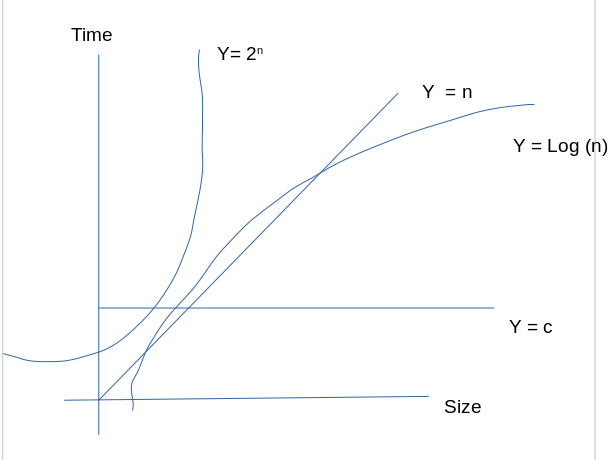
Y= x

Y = 2x

Y = 4x

All have the same time comp : O (n)

* + - Go for the bigger numbers
    - Ignore the constants (numbers)
    - Ignore the less dominating terms

 Very bad exponential complexity

* + - O(1) < O log(n) < O(N) < O (2 n)
* **Big O Notations**
  + Word Definition
    - O (N3)
      * This is the upper bound of the algorithm
      * The algorithm can take any value less then and equal to that I.e O (N3) but will not exceed
      * it can be solved in log n , n2, n2 \*  log n etc.
  + **Big Omega Notations**
    - Opposite of Big O notations
      * Ω (N3)
      * This is the lower bound of the algorithms
      * The algorithm can take any value greater then and equal to that I.e O (N3) but will be less then that
  + **Theta Notation** :
    - θ (N3)
    - Shows that both the upper bound and lower bound is N3.
  + Little O notation :
    - O(N3)
      * This is the strictly upper bound of the algorithm
      * The algorithm can take any value less then that I.e O (N3) but will not exceed ------- Not equal to
  + Little Omega
    - * Ω (N3)
      * This is the Strictly lower bound of the algorithms
      * The algorithm can take any value greater then that I.e O (N3) but will be less then that ------- Not equal to
* **Space Complexity**
  + input space + auxiliary space(extra space or temporary space used by an algorithm)
  + Recursive Algorithm
    - space complexity is the height of the recursion tree
      * calls that are interlinked are in the stack for the same time
      * Longest chain is the space complexity ,m
      * then SC is O(n) : Nth Fibonacci number
* Two types of recursion
  + Linear
    - f(n) = f(n-1)+f(n-2) Fibonacci
  + Divide and Conquer
    - F(n) = f(n/2) + O(1) Binary Search
* Divide and Conquer Recurrence
  + How to actually solve to get complexity
    - Plug and chug
    - Master’s Theorem