il(1) Circle whether each of the following statements is true or false. Make sure your circle is clear. No need to explain your answer.

a. T/F Linear search can only be done on unsorted arrays?

F

b. T/F Binary search can only be done on sorted arrays?

T

c. T/F There is only one way to have a good coding style?

F

d. T/F Log2(16) = 4?

T

e. T/F log(xy) = xlog(y)?

F

(2) Suppose I am thinking of a number between m and n-1, inclusive (where m and n are both positive integers and m < n), and as you can guess it. For example, if m = 0 and n = 16, the number could be 0, 15, or anything between 0 and 15. We will always round down. After each guess, I will tell you if your guess is too high, too low, or correct. Note that parts (b) through (f) do not require an explanation.

a. Explain the optimal algorithm for guessing my number in the game described above.

Binary Search

b. How many guesses, in the worst case, will it take to guess a number between 0 and 31, inclusive, in the game described above?

floor(log2(32)) + 1 = 6 Guesses

c. How many guesses, in the worst case, will it take to guess a number between 0 and 50, inclusive, in the game described above?

floor(log2(51)) + 1 = 6 Guesses

d. How many guesses, in the worst case, will it take to guess a number between 16 and 30, inclusive, in the game described above?

floor(log2(15)) + 1 = 4 Guesses

e. How many guesses, in the worst case, will it take to guess a number between 16 and 31, inclusive, in the game described above?

floor(log2(16)) + 1 = 5 Guesses

f. What is the formula for the maximum number of guesses it will take to guess my number, in terms of m and n?

**So we know that its floor(log2(n-m)) + 1 = G**

(3) For each of the following descriptions, write down one sorting algorithm we’ve learned in class. Each algorithm may be used once, more than once, or not at all. You do not need to explain your answers.

a. This algorithm uses the idea of splitting an array by a pivot.

Quick Sort

b. This algorithm has a worst-case running time in O(n^2) no matter how it is implemented.

Selection Sort

c. This algorithm involves inserting an element into an already sorted array.

Insertion Sort

d. This algorithm repeatedly swaps elements in adjacent indices.

Bubble Sort

e. After each pass of this algorithm, the largest element ends up at the end.

Bubble Sort

(4) Simplify these functions using big-o notation

a. 5

O(1)

b. 5n2 + 10n

O(n2)

c. 5n + 5(2n) + 5n2

O(2n)

d. n \* log(n) + log(n)

O(n log(n))

e. n2

O(n2)

(5) Complete the following java method that takes a non-empty array of integers as input and returns its range as output. Your algorithm’s runtime must be in O(n), and should only loop through the array once.

public static int[] range(int[] array) {

int min = array[0];

int max = array[0];

for (int i = 1; i < array.length; i++) {

if (array[i] < min) {

min = array[i];

}

if (array[i] > max) {

max = array[i];

}

}

int[] range = new int[2];

range[0] = min;

range[1] = max;

//int range = max - min;

return range;

}

The solution below does not return but it prints the values the question asked it to return.

public static void range(int[] array) {

int min = array[0];

int max = array[0];

for (int i = 1; i < array.length; i++) {

if (array[i] < min) {

min = array[i];

} else if (array[i] > max) {

max = array[i];

}

}

System.out.println("Range of the array is " + min + " to " + max);

}

public static int[] range(int[] arr) {

int min = arr[0];

int max = arr[0];

for (int i = 1; i < arr.length; i++) {

if (arr[i] < min) {

min = arr[i];

}

if (arr[i] > max) {

max = arr[i];

}

}

int[] range = {min, max};

return range;

}

(6) What is the big-O of space and time of each segment of code below? Please make sure you label which one is for space and which one is for time. Please give brief reasoning for your answers.

a. public int something(int[] numbers){

for(int i = 0; i < numbers.length; i++){ //Time = O(n), Space = O(1)

for(int j = 1; j < numbers.length; j = j \* 2){ //Time= O(log2(n)) Space=O(1)

int k = numbers[i] – numbers[j] //Time = O(1) Space = O(1)

}

}

}

Time Complexity: O(n)\*O(log2(n)) = O(nlog2(n)) - because of nested loops

Space Complexity: O(1)

b. public int something(int[] numbers){

int[] anotherArray = new int[numbers.length \*numbers.length]; //time=O(1), Space= O(n2)

int sum = 0; // Time = O(1) Space = O(1)

for(int i = 0; i < numbers.length; i++){ // Time= O(n) Space = O(1)

for(int j = 0; j < numbers.length; j++){ // Time = O(n) Space = O(1)

sum += numbers[i] + numbers[j] // Time = O(1) Space = O(1)

}

}

}

Time Complexity: O(n)\*O(n) = O(n2 ) - because of nested loops

Space Complexity: O(n^2)

(7) prove 3k-1 can be divided by 2. (k >= 1)

1. Proof

(3k-1)/2 ; k = 1 ; (31-1)/2 = (3-1)/2 = 2/2 = 1

1. Assume

(3k+1-1)/2 ; k = 1 ; (31+1-1)/2 = (32-1)/2 = (9-1)/2 = 8/2 = 4

1. Prove true for the next

(3k+2-1)/2 ; k = 1 ; (31+2-1)/2 = (33-1)/2 = (27-1)/2 = 26/2 = 13

(8) Implement merging two sorted arrays into one sorted array. These arrays are sorted in descending order. For example, input [5, 4, 2], [7, 3, 1], output [7, 5, 4, 3, 2, 1]. Return an array with all the elements of these two arrays sorted, also in descending order.

Public int[] merge(int[] arr1, int[] arr2){

public static int[] merge(int[] array1, int[] array2) {

int[] merged = new int[array1.length + array2.length];

int i = 0;

int j = 0;

int k = 0;

while (i < array1.length && j < array2.length) {

if (array1[i] > array2[j]) {

merged[k] = array1[i];

i++;

} else {

merged[k] = array2[j];

j++;

}

k++;

}

while (i < array1.length) {

merged[k] = array1[i];

i++;

k++;

}

while (j < array2.length) {

merged[k] = array2[j];

j++;

k++;

}

return merged;

}

b.) (6 points) if the two arrays as input are size n each. What is the big-O run time and space complexity of your implementation above?

Time = O(n) + O(n) + O(n) = O(n)

Space = O(n+n) = O(n)

(9) Complete the below java method recursively,



public static int calcFactorial(String str) {

int n = Integer.parseInt(str);

if (n == 0) {

return 1;

}

return n \* calcFactorial(String.valueOf(n - 1));

}

public int CalacFactorial(String word){

int num = Integer.parseInt(word);

if(num >= 1){

int temp = CalacFactorial(String.valueOf(num-1));

return num \* temp;

}

else

return 1;

}

**public int CalacFactorial(String word) {**

**int n = Integer.parseInt(word);**

**if (n == 0) {**

**return 1;**

**}**

**return n \* CalacFactorial(String.valueOf(n - 1));**

**}**

**public static void main(String[] args) {**

**System.out.println(new Factorial().CalacFactorial(“8”));**

**}**

**}**