COP 3502 Exam 1 Review

1. Determine which lines of the code snipped below have a memory violation. Explain.

// assume he’s trying to trick you

// hw: write code where you allocate memory using realloc

int main()

{

int a = 10; // static memory is from the stack while dynamic memory is from the heap

int \* ptr, \*ptr2, \*ptr3, \*ptr4;

ptr = &a; // this is fine because a pointer stores an address

ptr2 = (int \*) malloc(3 \* sizeof(int)); // malloc and calloc return an address

ptr2[3] = 5; // out of bounds because our first index is 0 and the last index is 2

// the square brackets dereference pointers, therefore we dereference a null pointer causing seg fault

// other ways to derefence pointers are (\*) and (->)

ptr3 = realloc(ptr3, sizeof(int)); // You can allocate memory with realloc

\*ptr3 = 10;

\*ptr4 = 15; // this line is bad because we did not allocate memory into pointer 4, so we are dereferencing a null pointer leading to a seg fault

// you did not free any of this which would lead to a memory leak

return 0;

}

// Even at the worst case we still will never look at every value  
0, 1, 2 , 3, 4, target = 2

2. What is the worst case run time of the binary search? (Avg big Oh = O(log(n)) = worst, best = O(1)

a. O(nlog(n))

b. O(n^2)

c. O(n)

d. O(log(n))

// he’s trying to trick you

3. Determine the big oh runtime of the for loop shown below.

You can assume all variables have been properly declared and initialized.

// Since we have 2 nested for loops that each run n times, we say n \* n = n^2, but since we have strcat inside of the second for loop, we have to multiply by n again. So n\*n\*n = O(n^3)

for(int i = 0; i < n; i++)

{

sum++;

for(int j = 0; j < n; j++)

strcat(arr, "b"); // O(n)

}

// if you see strcat, strcmp, strcpy, strlen then those run in O(n)

4. Write a recursive function that will return the sum of all of the odd integers in the passed array.

You may not create any global variables.

Size = 5; what is the last index in the array? Size - 1

int oddSum(int \* arr, int size)

{

// base case

If (size == 0)

Return 0;

// base case 2

// if the number is odd, we return that number + the recursive call to going down the array

If (arr[size - 1] % 2 != 0) // we write – 1 so we don’t go out of bounds

{

Return Arr[size – 1] + oddSum(arr, size - 1);

}

Else

{

Return oddSum(arr, size – 1);

}

// The recursive call

}

5. Write the state of the following array after each pass of the insertion sort:

7 9 3 8 5 11 1

6. Determine the big oh runtime for the following recurrence relation using the iterative method:

T(1) = 1 T(n) = 2T(n/2) + n

7. Determine the worst-case big oh runtime for each of the following operations:

// O(n) because when you add to the front of arraylist, you have to shift all the other n elements down one space

a. Adding an item to the front of an arraylist: \_\_O(n)\_\_\_\_\_

// When we pick the worst pivot each time, we end up with n^2

// every time we pick a pivot it is the smallest or largest number in its subarray

// The best pivot would be the middle number

b. Using quicksort on an array of numbers: \_\_O(n^2)\_\_\_\_\_\_

c. Using a binary search to find a certain number in an array: \_\_O(log(n))\_\_\_\_\_\_

8. What is the best-case runtime for using the selection sort?

O(n^2) is the best case scenario because we always have to find the smallest number each time even if it is already sorted

Dumb sort

9. Find and fix the line that is causing the following code to segfault.

Then, explain why you changed what you did as well as the technical reason it was causing a segmentation fault error specifically.

int function1(int \* arr, int size, int number)

{

if (size == 0)

return 0;

if (arr[size - 1] == number)

return 1;

return function(arr, size + 1, number);

}

// the code is seg faulting because the function never ends and is eating up all the stack memory causing a stack overflow, which leads to a seg fault.

We should change the recursive call to say “size – 1” instead of “size + 1”

10. What benefit does an arraylist present over a regular array?

// an arraylist is resizeable, so we can use dynamic memory and that means we can change the size after runtime  
  
  
  
  
11. Which nlog(n) sort tends to be faster on average? Merge or quicksort?

Quicksort is faster on average.

What is the space complexity of the quicksort?

Quicksort does everything in place to the original array, so the space complexity is O(1)

What is the space complexity of the mergesort?

Merge sort creates new arrays to make the changes to, so the space complexity is O(n)

The quicksort is faster on average because the constant creation of new arrays of the merge sort slows it down