Questions for exam:

1) Pointer

- What is a pointer?
- What are meaning of those operators (*, &, ++) when they are associated with pointers:

int *iptr, i;

*ptr = 1;

&i

iptr++

- What is memory leak? What operations can cause memory leak?

2) OO concepts – inheritance and polymorphism, abstract function, virtual function, static binding, dynamic binding,

3) Algorithms – use your own words to describe linearSearch, binarySearch, selectionSort, BubbleSort, mergeSort and explain their time complexity (you don't have to go through the detailed model to derive the T(n), just some intuition to explain how you get the time complexity).

4) Other concepts:

- How Von neumman architecture works (the fetch-execute cycle)
- Stack overflow
- Constructors/destructors related questions (when do they get called? And the order they got called)

I will ask one question from each category so there will be four questions in total for each person. If your answer to that question is not satisfying, I will give you a second question from the same category to make it up.

Comp Sci Exam is on 4/20/2021.

1) What is a pointer?

A pointer is a data type that points to a memory address.

What does *, & and ++ mean?

***** is the **dereference operator**, which allows you to modify the value that is stored in the address that the pointer points to.

& is the a**ddress-of operator** which allows you to retrieve the address of a variable.

++ will navigate the pointer to the next memory address.

int *iptr, i; = Creates two integer pointer variables, iptr and i.

***ptr = 1; =** Assigns 1 to the address that is stored in ptr.

&i = Retrieves the address of the variable i.

iptr++ = Navigates to the next memory address.

What is a memory leak?

A memory leak is when you allocate memory to something but dont de-allocate it when your done.

ex: When I allocate memory to a pointer using the "new" operator, but dont deallocate it with "delete".

2) Inheritance is when a child class inherits data members of a parent class.

Polymorphism happens when the program determines what function to call during runtime instead of determining what function to call when the program is compiled.

An **abstract class** is a class that has at least one virtual function. Essentially a base class with a virtual function.
Static Binding - The program knows exactly what functions to call at compile time. (Binding is known at compile time)
Dynamic Binding - The program doesnt know what functions to call at compile time and must determine which ones to call during run-time. (Binding is unknown at compile time and is determined during run time.)
Vitual Function - A member funtion that is declared in a class and is overridden by a derived class. its used to achieve run time poly morphism.

3) One for loop = O(n)
Two for loops = O(n*n) = O(n^2)
Recursive functions are generally O(log(n)) if they are computing the N-th term.

Linear Search - O(n) - Uses one for loop - Compares the value your looking for with the value of the array at any index all the way up to n.

BinarySearch - O(log(n)) Recursive function - Requires that the array is sorted. Binary search repeatedly splits the array in half until it finds the value that its looking for.

```
int binarySearch(int arr[], int l, int r, int x)
{
   if (r >= l) {
      int mid = l + (r - l) / 2;

      if (arr[mid] == x)
         return mid;

      // If element is smaller than mid, then
      // it can only be present in left subarray
      if (arr[mid] > x)
         return binarySearch(arr, l, mid - 1, x);

      // Else the element can only be present
      // in right subarray
      return binarySearch(arr, mid + 1, r, x);
   }

   // We reach here when element is not
   // present in array
   return -1;
}
```

SelectionSort - O(n^2) Two nested loops - Searches the array for the smallest number and then moves the smallest number that it can find to the front of the array. Then it repeats this process N times.

```
void selectionSort(int arr[], int n)
{
   int i, j, min_idx;
   // One by one move boundary of unsorted subarray
   for (i = 0; i < n-1; i++)
```

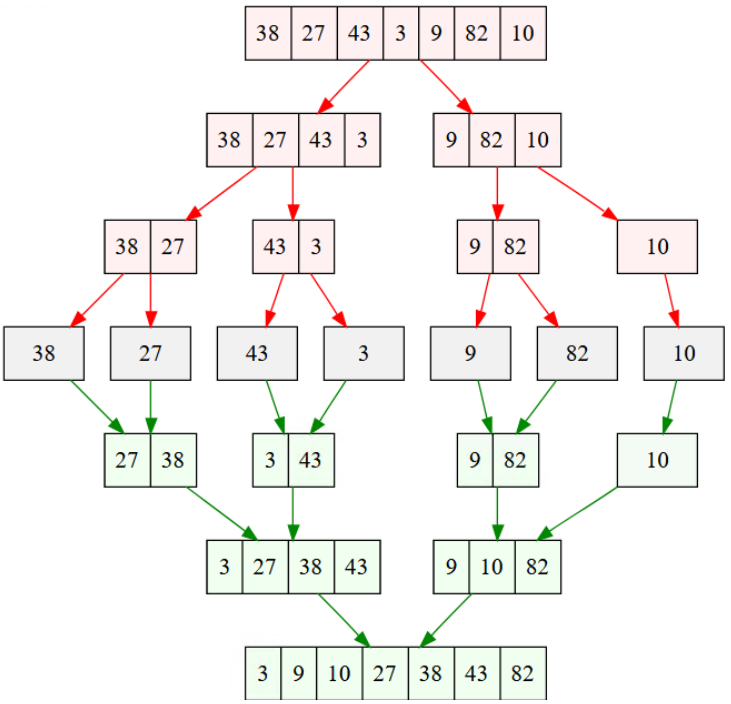
```
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                    min_idx = j;

        // Swap the found minimum element with the first element
        swap(arr[min_idx], arr[i]);
    }
}
```

BubbleSort - O(n^2) Two nested loops - Loops through the whole array of size N and moves the smaller numbers one position forward. It then repeats this N times.

```
void bubbleSort(int arr[], int n)
{
    for (int i = 0; i < n-1; i++)
            for (int j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                            swap(arr[j], arr[j+1]);
}
```

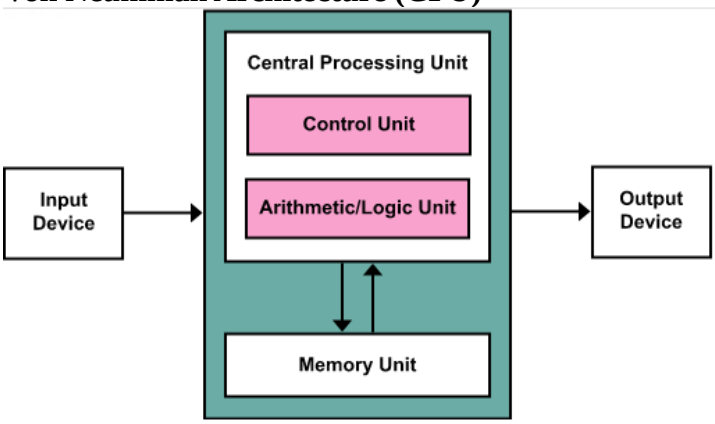
MergeSort - O(n*log(n)) Recursive function, and has to assemble the array of size n. - Repeatedly splits the array in half until the array size is 1, and then it merges the numbers back together in order which results in a sorted array.

```
void merge(int arr[], int l, int m, int r)
{
    int n1 = m - l + 1;
    int n2 = r - m;

    // Create temp arrays
    int L[n1], R[n2];

    // Copy data to temp arrays L[] and R[]
    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    // Merge the temp arrays back into arr[l..r]

    // Initial index of first subarray
    int i = 0;

    // Initial index of second subarray
    int j = 0;

    // Initial index of merged subarray
```



```
    int k = l;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of
    // L[], if there are any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy the remaining elements of
    // R[], if there are any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r){
    if(l>=r){
        return;
    }

    int m =l+ (r-l)/2;
    mergeSort(arr,l,m);
    mergeSort(arr,m+1,r);
    merge(arr,l,m,r);
}
```

Von Neumman Architecture (CPU)



Consists of a control unit, artihmetic and logic unit, memory unit, and registers inputs/outputs

A CPU has the following components:

- **Control Unit** – controls all parts of the computer system. It manages the four basic operations of the Fetch Execute Cycle as follows:
  1. **Fetch** – gets the next program command from the computer's memory
  2. **Decode** – deciphers what the program is telling the computer to do
  3. **Execute** – carries out the requested action
  4. **Store** – saves the results to a Register or Memory
- **Arithmetic Logic Unit (ALU)** – performs arithmetic and logical operations
- **Register** – saves the most frequently used instructions and data

Stack Overflow - A condition where the program tries to use more memroyspace than the call stack has available. This tends to happen when a recursive function infinitely calls itself or calls itself excessively.
Call Stack - A buffer that stores requests that need to be handled.

Constructors and Destructors
Constructors - Called whenever an object of that class is made.
Destructors - Called whenever the object is deleted, or the variable is out of the scope.