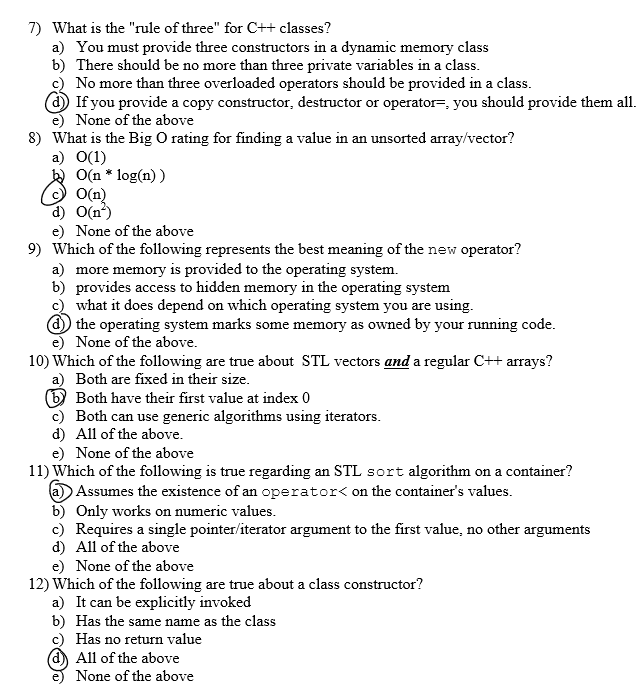
**CLASSES: Struct VS Class:** A struct, by default, members are public while private in class. Inheritance in case of a struct public by default, while class is private. Classes can take part in templates. Structs and classes can have associated methods. Classes and structs can have private data members. There can be constructors and destructors in both.Class’s **copy constructor** can be explicitly invoked; when a class instance is passed to a function, when a class instance is returned from a function. **Accumulate:** computes the sum of the given value in it and the elements in the range (first,last). **Transform:** applies the given f(x) to a range and stores the result in another range std::transform(set.begin(),set.end(),set.begin(), ::change(2upper))

**VARIABLES/TYPES: C++ map** requires 2 types to declare the container. Cannot respond to [] or .at() (neither does STL list). **Basic array data structure:** It is of fixed size. Can use generic algorithims on an array via use of pointers. First value is at index 0. A 4 byte **unsigned int** can hold 8\*4 = 32 bits so from 0 -> 2^32. **This variable**: a pointer; you can’t change what it points to; it is bound to calling instance of a method. Implementing a **Stack** can be done by an array, linked list or a vector.



**Pointers/Iterators: C++ iterators:** act as pointers to a container value, are required to use generic algorithms such as sort or accumulate and are typed specifically to the container they are use with. C++ pointer byte size depends on the OS. An iterator/pointer to the first element is required to utilized the STL generic sort algorithm on an entire container’s value. **Single linked data structures** can change its size during execution, is efficient at inserting an element at a particular position uses pointers to track the next element in the list. **head\_ptr -> node1.** Head is the first pointer to the start of the list. **Node 1[val|next\_ptr]** val is the val stored at node 1, next\_ptr points to node 2. **=> head\_ptr.next = pointer to node 2.**

**Rules:Big-Oh Notation:** Fastest: )(n•log(n)) Slowest: O(n^n). **O(1)** constant time **O(log n)**  time grows as the log of the size **O(n)** time grow linearly with size **O(n2)** time squares with growth **O(2n)** time grows as the power of size. **Rule of 3:** For any object that dynamically allocates memory. In this case you probably define a copy constructor; define an assign & destructor. **‘Strongly typed’** C++: every variable must have type. Class constructor doesn’t have to return type; it initializes an object. Requirement for a properly written recursive function is to provide a base case.

**Memory: new:** gets/allocates memory from the OS and returns a pointer to it (single object or array of those object) new type; new type [size] **delete:** de-allocates the memory, delete ptr gives ptr back to OS; delete [] ptr deletes all of array elements, back to OS. **Stack** Used implicitlu; think push& pop; grows towards Heap; may not have ‘holes’; may overflow. **Heap:** used explicitly; think malloc & new; grows towards stack; may have ‘holes’;may OF(run2Stack). **Leacking Mem: new** some memory; use that memory; reassign the pointer. Since it was never deleted it isn’t owned by the OS nor the user.

