Topics:

1. chapter 1 (preliminary math stuff, preliminary c++ stuff)

2. Chapter 2 (complexity theory)

3. Makefiles and g++

4. Chapter 3 (iterators, standard template library(array, vector , list and

unordered\_maps), smart pointers), review code covered for these topics

Expect true/false, short answers, multiple choice(maybe),

1 coding question- give a problem and you must code up the solution,

1 question will have you analyze the compliexity of some code.

Proofs: identity a proof, and be familiar with the concept, no explicit Proofs

Skiped from chapter 1: Greatest common divisor algorithm

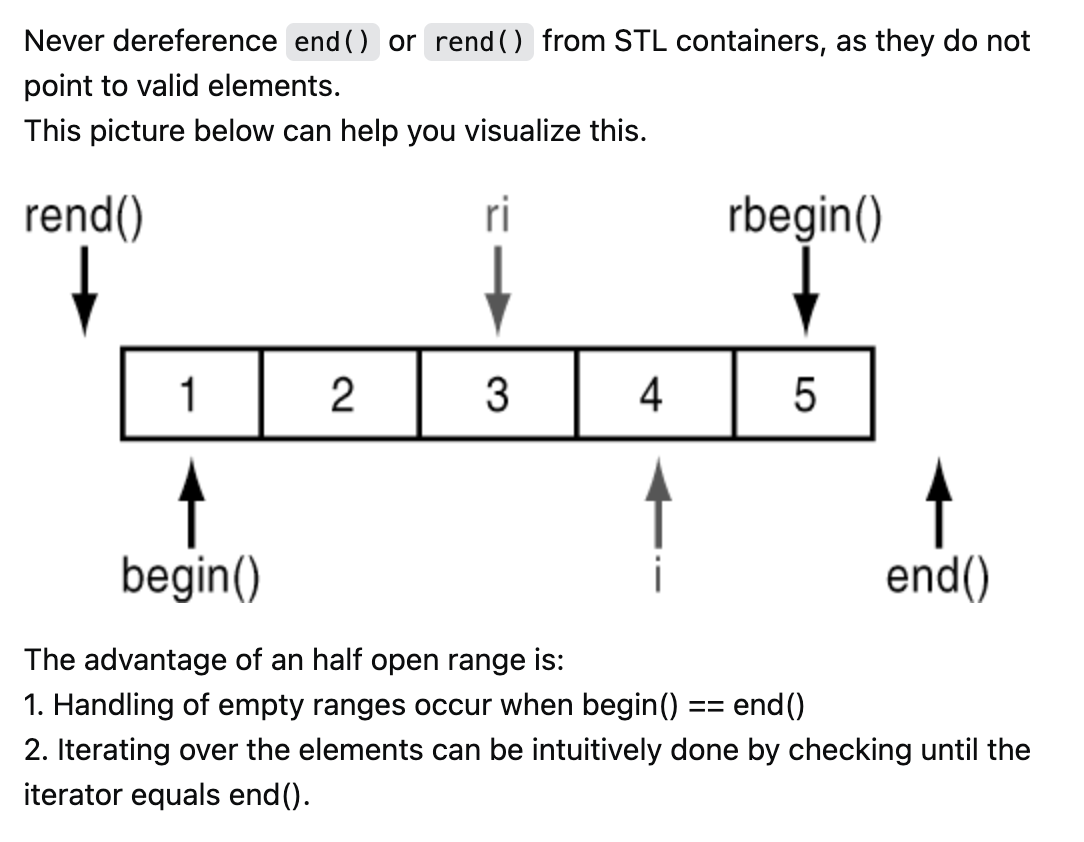
How the g++ compiler works and its architecture(preprocessor, compiler, linker),

and how to use make. In terms of questions I will only ask about things seen in

the Makefile for assignment 1.

https://makefiletutorial.com/  
  
makefile tutorials:  
A black rectangle with white text

Description automatically generated



<https://stackoverflow.com/questions/34830192/what-doesone-past-the-last-element-mean-in-vectors>

Summary of C++ [vector](https://cplusplus.com/reference/vector/vector/at/) and C++ [List](https://cplusplus.com/reference/list/list/):

[iterator](https://cplusplus.com/reference/iterator/), assuming it’s a set, then Random\_Access is the entire set, and   
bidirectional is a subset  
A diagram of a flowchart

Description automatically generated

Vector and list both belong to containers of C++

|  |  |  |
| --- | --- | --- |
| Vector | List |  |
| Has its member functions:  e.g.: .at()  shrintk\_to\_fit()  erase() | Has its member functions  e.g. erase(), begin(), end(), but has less than vector |  |
| Has Random\_access\_iterators:  ++itr; 🡨pre-increment, preferred, return the incremented iterator  Itr++; 🡨 post-increment, less efficient, make a copy, increment the original, and retun the copy.  \*++itr; 🡨increment then dereference  \*it++; 🡨 dereference then increment  Itr == itr;  Itr != itr;  Swap(itr, itr2);  Can do arithmetic operations:  Itr – itr2;  Itr – 5;  Itr + 5;  5 + itr;  Itr[5]; 🡨 this does not move the itr, it only does the offset  Itr->display() | Has Bidirectional\_iterators:  Has mush less options than random\_access\_iterators |  |
| Has access to all member functions of iterator class | Has access to all member functions of iterator class |  |
| List.insert() | does not delete list2 |  |
| List.merge() | DELETES list2 |  |
| List.splice() | DELETES list2 |  |
|  |  |  |

You want to insert n random integers into an initially empty List or Vector, while keeping it sorted after each insertion.

Operations in Vector vs List:

|  |  |  |
| --- | --- | --- |
| Operation | Vector | List |
| Sort() – introSort worse and average | O(n\*logn) | O(n\*logn) |
| Search() on sorted | O(logn) on sorted vector using binary search | Doesn’t have this functionality |
| Search() on unsorted | O(n) have to iterate through | O(n) have to traverse |
| Insert() | O(n), have to move elements down after inserting new | O(1) doubly linked list |
| Overall for each element: | O(logn) + O(n) = O(n) | O(n) + O(1) = O(n) |
| Overall for n elements: | O(n^2) | O(n^2) |

Unordered\_map:

Uses forward iterators:

A diagram of a map

Description automatically generated  
[credit](https://www.alphacodingskills.com/cpp/notes/cpp-unordered-map-end.php)

|  |  |
| --- | --- |
| Operation: | Unorderdmap |
| Find() average | O(1) leverage hash table |
| my\_map1.find(1) | Returns iterator is found, else point to .end() |
| auto it = my\_map1.find(1); it->first; (\*it).first;  Same thing |  |
|  |  |
| auto it = map.find(1); it = map.erase(it); | Now it points to the next element, which is random |

Understanding mod notation, for example: is this true 9 = 7 mod 3 ? (false)

what is a template, how does it compile, explain why .h and .cpp is weird.

n + nlogn + 7= O(nlogn)

know important runtimes like selection sort = O(n^2)

make sure to know l and r values, references, pointers, big 5 constuctors

How to use arithmetic series, be able to explain what the other series are.

These are some examples but not all!