Specifications

In C++, create a generic class for implementing an array-based set. Remember, sets are just arrays that do not have duplicate values. The set should be implemented as a template class.

The assignment project uses C++20, which allows you to utilize concepts for constraining the generic type. Specifically, use the `is_arithmetic_v` constraint to ensure that only numeric types can be used with your class.

Project Structure

Your project must consist of three separate files:

1. Header file: Contains class declaration.
2. Source file: Contains the class implementation
3. main.cpp: Contains the `main()` function for testing purposes.

Only include the header file in other files, and ensure you explicitly instantiate all necessary template instances to avoid linking errors during compilation. For more details about handling templates, refer to the following link. Start at the section “Why can’t I separate the definition of my templates class from its declaration and put it inside a .cpp file?”
https://isocpp.org/wiki/faq/templates#templates-defn-vs-decl

Class Requirements

Your set class should include the following:

- ☐ Constructors: Default constructor and capacity constructor to initialize the set's capacity.
- ☐ Destructor: Ensure memory allocated is freed.
- ☐ Methods:
  - ☐ Indexing support
  - ☐ Getters
  - ☐ `find()`: Returns the index of a given element.
  - ☐ `insert()`: Inserts an item (no duplicates allowed). Implement two versions: one that simply appends an item to the set and another that inserts it at a user-specified position.
  - ☐ `remove()`: Removes a specific element.
  - ☐ `removeAt()`: Removes an element at a specified index.
  - ☐ `clear()`: Clears the set.
  - ☐ `toString()`: Returns the contents of the set as a string.

The set should disallow duplicate values. When the array reaches its capacity, it should double in size. If the number of elements drops below a quarter of the capacity, shrink the array by cutting the capacity in half. You must manage memory allocation and deallocation accordingly.

Sorting Algorithms

Your set class should include the following sorting methods:

- ☐ `bubbleSort()`: Standard bubble sort.
- ☐ `InsertionSort()`: Standard insertion sort.
- ☐ `selectionSort()`: Standard selection sort.

Only implement a sorting method after its been introduced in class.

Additional Methods

All methods listed below should be public, with the exception of `swap()`, which should be private.

- ☐ `swap()`
  - Performs swaps for you. Utilize it in your bubble and selection sorts and any variants of them.
- ☐ `bidirectionalBubbleSort()`
  - A modification of the bubble sort method to make it bidirectional. This means the inner for loop will first carry the largest item from left to right as before, but then it will reverse and carry the smallest item from right to left.
- ☐ `median()`
  - This method should return the median value in the set.
- ☐ `oddEven()`
  - This method implements another simple sort called the odd-even sort. The idea is to repeatedly make two passes through the array. On the first pass you look at all the pairs of items, a[j] and a[j+1], where j is odd (j = 1, 3, 5, …). If their values are out of order, you swap them. On the second pass you do the same for all the even values (j = 2, 4, 6, …). You do these two passes repeatedly until the array is sorted.
    - The odd-even sort is actually useful in a multiprocessing environment, where a separate processor can operate on each odd pair simultaneously and then on each even pair. Because the odd pairs are independent of each other, each pair can be checked—and swapped, if necessary—by a different processor. This makes for a very fast sort.
- ☐ `insertionSortVerbose()`
  - A modified insertion sort that counts the number of copies and the number of comparisons it makes during a sort and displays the totals

Testing Requirements

Thoroughly test your set class with a variety of scenarios, including:

- ☐ Empty sets
- ☐ Single-element sets
- ☐ Large sets
- ☐ Sets with different numeric types

Implement test cases that specifically focus on the correctness of the sorting algorithms, the accuracy of median calculation, and the correctness of insertion-related methods. Validate that the set class adheres to the constraint of excluding duplicate values, and conduct tests to confirm the proper handling of various data types dictated by the `is_arithmetic_v` constraint.

Constraints

- All index and size-related variables should use the `size_t` data type.
- Any method that returns an index or size should return `size_t`.
  - Given that `size_t` is an unsigned number, to return -1, utilize the `npos` constant. More on `npos` here:
  - https://cplusplus.com/reference/string/string/npos/
- All dynamically allocated memory must be freed. Use the destructor to handle this.
- All methods defined in your source file must have its signature in the header file (.h).
- C++ uses camelCase. Name variables and methods using camelCase. Classes should be named with PascelCase.
- The default constructor should allocate enough memory for 4 elements.
- If a method will not affect the state of the object, mark the method as constant.

Submission

The submission for this project is the commit ID you want to be graded. You will submit the commit ID via the Canvas assignment. If you need help finding the commit ID, ask for help.

See the grading rubric on Canvas for grading criteria.

Remember to write efficient code. Optimize your code wherever possible. Points will be deducted for unoptimized code.

Tips

- Because it is impossible to see what elements you have in your array, call your toString() method frequently when debugging to help identify errors.

Academic Honesty Policy

All code submissions must be original and authored by the student. Any code sourced from another student, falsely presented as one's own, or derived from third-party websites or generated by AI, will constitute a breach of the school's academic honesty policy.

Daily commits and pushes to your assignment GitHub repository are <u>mandatory</u> while working on your project. Failure to comply will result in deductions from your final grade at minimum, and could lead to academic honesty violations.