Lab 4

Lab Goal

- **Understanding and implementation of searching and sorting algorithms**
- **Searching Algorithms to implement**
  - Linear Search
  - Binary Search
- **Sorting Algorithms to implement**
  - Merge Sort
  - Quick Sort

Code To Write

- **linearSearch(data, target)**
- **binarySearch(data, target)**
  - data: vector<T>
  - target: T
  - T = any given data type
  - Both functions should return an int of the index of the target in the vector
  - If the value is not in the vector, then return -1
  - Difference between these functions are the way they are implemented, and the data given for the binary search is always sorted

Code To Write (cont.)

- **mergeSort(lst)**
- **quickSort(lst)**
  - lst: vector<T>
  - T = any given data type
  - Both functions should sort the vector in a recursive manner and return a sorted version of the vector
  - Difference between these functions are the way they are implemented
  - Feel free to implement helper functions for these functions

Binary Search Example

- {0,2,4,6,8,10,12,14,16,18,20,22,24,26}
- Target = 18
- lowIndex = 0
- highIndex = 13
- midIndex = (13 + 0) / 2 = 6
- array[6] = 12
- 12 is less than 18 so increase lowIndex
- lowIndex = 6 + 1 = 7

Binary Search Example (cont.)

- {0,2,4,6,8,10,12,14,16,18,20,22,24,26}
- Target = 18
- lowIndex = 7
- highIndex = 13
- midIndex = (13 + 7) / 2 = 10
- array[10] = 20
- 20 is greater than 18 so decrease highIndex
- highIndex = 10 - 1 = 9

Binary Search Example (cont.)

- {0,2,4,6,8,10,12,14,16,18,20,22,24,26}
- Target = 18
- lowIndex = 7
- highIndex = 9
- midIndex = (7 + 9) / 2 = 8
- array[8] = 16
- 16 is less than 18 so increase lowIndex
- lowIndex = 8 + 1 = 9

Binary Search Example (cont.)

- {0,2,4,6,8,10,12,14,16,18,20,22,24,26}
- Target = 18
- lowIndex = 9
- highIndex = 9
- midIndex = (9 + 9) / 2 = 9
- array[9] = 18
- return 9 as the answer

Quick Sort Example

- {22,14,8,6,10,2,12,16}
- select a random index and set the value at that index as the pivot
- In this example we will use 6 as the random index
- pivotValue = 12

Quick Sort Example (cont.)

- {12,14,8,6,10,2,22,16}
- Swap that value with the first value
- pivotValue = 12
- pivotIndex = 0

Quick Sort Example (cont.)

- {12,14,8,6,10,2,22,16}
- Loop through the rest of the vector starting with the value after the pivot index
- pivotValue = 12
- pivotIndex = 0
- If a value is less than the pivot value, then increase the pivotIndex and swap the value at the new pivotIndex with the value

Quick Sort Example (cont.)

- {12,14,8,6,10,2,22,16}
- pivotValue = 12
- pivotIndex = 0
- currentValue = 14
- 14 is greater than 12 so 14 stays where it is

Quick Sort Example (cont.)

- {12,14,8,6,10,2,22,16}
- pivotValue = 12
- pivotIndex = 0
- currentValue = 8
- 8 is less than 12 so increase the pivotIndex and swap 8 with the value at pivotIndex
- pivotIndex = 0 + 1 = 1

- {12,14,8,6,10,2,22,16} -> {12,8,14,6,10,2,22,16}

Quick Sort Example (cont.)

- {12,8,14,6,10,2,22,16}
- pivotValue = 12
- pivotIndex = 1
- currentValue = 6
- 6 is less than 12 so increase the pivotIndex and swap 6 with the value at pivotIndex
- pivotIndex = 1 + 1 = 2

- {12,8,14,6,10,2,22,16} -> {12,8,6,14,10,2,22,16}

Quick Sort Example (cont.)

- {12,8,6,14,10,2,22,16}
- pivotValue = 12
- pivotIndex = 2
- currentValue = 10
- 10 is less than 12 so increase the pivotIndex and swap 10 with the value at pivotIndex
- pivotIndex = 2 + 1 = 3

- {12,8,6,14,10,2,22,16} -> {12,8,6,10,14,2,22,16}

Quick Sort Example (cont.)

- {12,8,6,10,14,2,22,16}
- pivotValue = 12
- pivotIndex = 3
- currentValue = 2
- 2 is less than 12 so increase the pivotIndex and swap 2 with the value at pivotIndex
- pivotIndex = 3 + 1 = 4

- {12,8,6,10,14,2,22,16} -> {12,8,6,10,2,14,22,16}

Quick Sort Example (cont.)

- {12,8,6,10,2,14,22,16}
- pivotValue = 12
- pivotIndex = 4
- currentValue = 22
- 22 is greater than 12 so value stays where it is

Quick Sort Example (cont.)

- {12,8,6,10,2,14,22,16}
- pivotValue = 12
- pivotIndex = 4
- currentValue = 16
- 16 is greater than 12 so value stays where it is

Quick Sort Example (cont.)

- {12,8,6,10,2,14,22,16}
- pivotValue = 12
- pivotIndex = 4
- The loop is now done so swap the pivotValue with the number at the pivotIndex

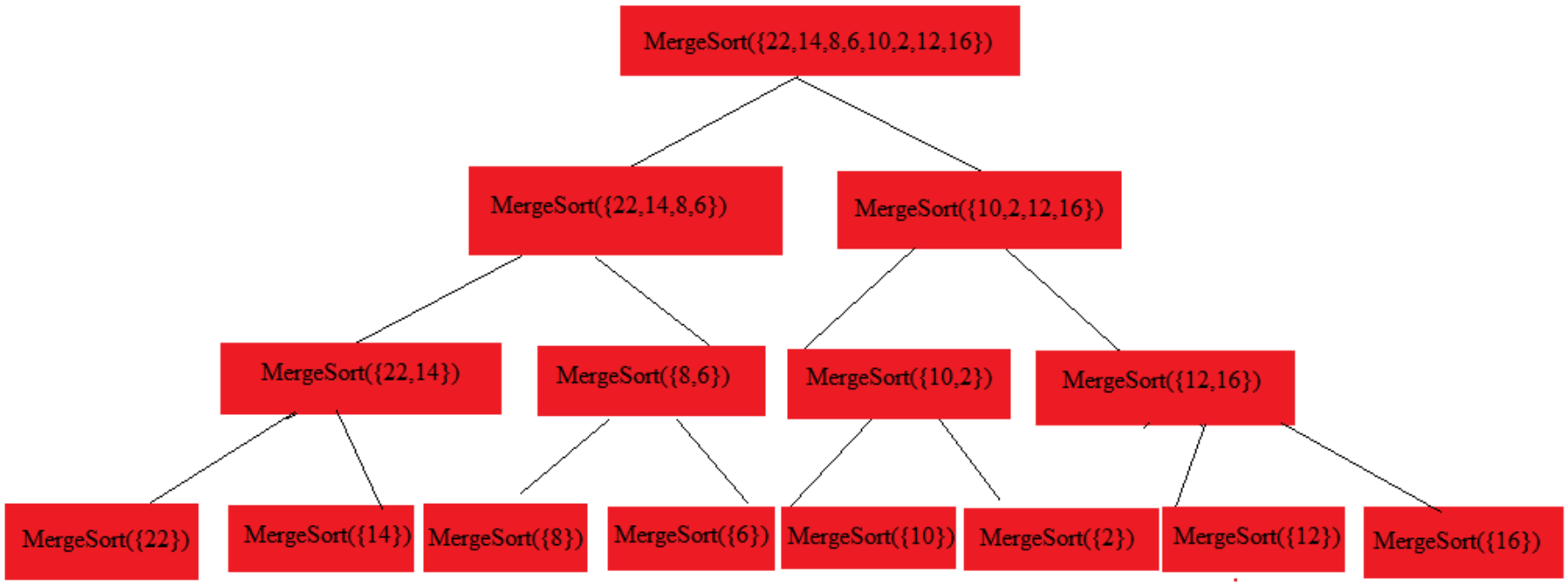
- {2,8,6,10,12,14,22,16}

Quick Sort Example (cont.)

- {2,8,6,10,12,14,22,16}
- pivotValue = 12
- pivotIndex = 4
- Now call quicksort on the two partitions
- quicksort(list, 0, 3)
- quicksort(list, 5, 7)
- Because this is recursive make sure you produce some kind of base case to stop the quick sort

Merge Sort Example

- {22,14,8,6,10,2,12,16}
- Recursively call merge sort on vector until the part of the vector you are merge sorting is a size of 1



Merge Sort Example (cont.)

- original = {22,14,8,6,10,2,12,16}
- After MergeSort is called the two halves you called mergesort on should be sorted
- You then need to merge the two lists together
- So the top call of the merge sort will get back two sorted halves where
  - half1 = {6,8,14,22}
  - half2 = {2,10,12,16}

Merge Sort Example (cont.)

- original = {22,14,8,6,10,2,12,16}
- half1 = {6,8,14,22}
- half2 = {2,10,12,16}
- Start at the first value in each list
- Whichever value is smaller put it in a new vector and move on to the next value in the list

Merge Sort Example (cont.)

- original = {22,14,8,6,10,2,12,16}
- half1 = {6,8,14,22}
- half2 = {2,10,12,16}
- 2 is smaller so place it in the new vector
- sorted = {2}

Merge Sort Example (cont.)

- original = {22,14,8,6,10,2,12,16}
- half1 = {6,8,14,22}
- half2 = {2,10,12,16}
- 8 is smaller so place it in the new vector
- sorted = {2,6,8}

Merge Sort Example (cont.)

- original = {22,14,8,6,10,2,12,16}
- half1 = {6,8,14,22}
- half2 = {2,10,12,16}
- 10 is smaller so place it in the new vector
- sorted = {2,6,8,10}

Merge Sort Example (cont.)

- original = {22,14,8,6,10,2,12,16}
- half1 = {6,8,14,22}
- half2 = {2,10,12,16}
- 12 is smaller so place it in the new vector
- sorted = {2,6,8,10,12}

Merge Sort Example (cont.)

- original = {22,14,8,6,10,2,12,16}
- half1 = {6,8,14,22}
- half2 = {2,10,12,16}
- 14 is smaller so place it in the new vector
- sorted = {2,6,8,10,12,14}

Merge Sort Example (cont.)

- original = {22,14,8,6,10,2,12,16}
- half1 = {6,8,14,22}
- half2 = {2,10,12,16}
- 16 is smaller so place it in the new vector
- sorted = {2,6,8,10,12,14,16}

Merge Sort Example (cont.)

- original = {22,14,8,6,10,2,12,16}
- half1 = {6,8,14,22}
- half2 = {2,10,12,16}
- Once an entire half has been exhausted place the rest of the other half in the sorted vector
- sorted = {2,6,8,10,12,14,16, 22}
- This example only shows the merging at the top level but it should be done like this for every call during the recursion

Some Hints

- **For the merge sort and the quick sort it might help to create a helper function similar to the following**
- **MergeSort/QuickSort(lst, low, high)**
  - lst : reference to list
  - low: int
    - Index of the lower bound to be sorted
  - high: int
    - Index of the upper bound to be sorted
- **Using this function you can sort the values of the list within the given range**
- **This will prevent you from having to make several copies of the list during recursive calls**