CS211 - Data Structures and Algorithms

Lab 05

Instructor: Dr. Sharaf Hussain
E-mail: shhussain@uit.edu

Semester: Fall, 2021

1 Objective

The purpose of this lab session is to practice various searching and sorting techniques as we have discussed in class week.

2 Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

3 How to Submit

- Submit lab work in a single .py file on Google Classroom. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_LW04.py)
- Submit home work in a single .py file on Google Classroom. (No other format will be accepted)
- Lab work file name should be saved with your roll number (e.g. 19a-001-SE_HW04.py)

4 Exercises - Searching

Task 1: Linear search on unsorted sequence

Implement **Linear Search** on an unsorted sequence.

```python
    def linearSearch( theValues, target ) :
      n = len( theValues )
      for i in range( n ) :
        # If the target is in the ith element, return True
        if theValues[i] == target
           return True
      return False # If not found, return False.

```

Task 2: Linear search on a sorted sequence.

Implement **Linear Search** on a sorted sequence.

```python
    def sortedLinearSearch( theValues, item ) :
      n = len( theValues )
      for i in range( n ) :
        # If the target is found in the ith element, return True
        if theValues[i] == item :
          return True
        # If target is larger than the ith element, it's not in the sequence.
        elif theValues[i] > item :
          return False

      return False # The item is not in the sequence.

```

Task 3: The Binary Search

Implement The **Binary serach** algorithm:

```
def binarySearch( theValues, target ) :
# Start with the entire sequence of elements.
low = 0
high = len(theValues) - 1

# Repeatedly subdivide the sequence in half until the target is found.
while low <= high :
  # Find the midpoint of the sequence.
  mid = (high + low) // 2
  # Does the midpoint contain the target?
  if theValues[mid] == target :
    return True
  # Or does the target precede the midpoint?
  elif target < theValues[mid] :
    high = mid - 1
    # Or does it follow the midpoint?
  else :
    low = mid + 1
# If the sequence cannot be subdivided further, we're done.
return False

```

5 Exercises - Sorting

Task 1: Bubble Sort

Implement The **Bubble Sort** algorithm:

```
# Sorts a sequence in ascending order using the bubble sort algorithm.
def bubbleSort( theSeq ):
  n = len( theSeq )
  # Perform n-1 bubble operations on the sequence
  for i in range( n - 1 ) :
    # Bubble the largest item to the end.
    for j in range( i + n - 1 ) :
      if theSeq[j] > theSeq[j + 1] : # swap the j and j+1 items.
        tmp = theSeq[j]
        theSeq[j] = theSeq[j + 1]
        theSeq[j + 1] = tmp

```

Task 2: Selection Sort

Implement The **Selection Sort** algorithm:

```
# Sorts a sequence in ascending order using the selection sort algorithm.
def selectionSort( theSeq ):
  n = len( theSeq )
  for i in range( n - 1 ):
    # Assume the ith element is the smallest.
    smallNdx = i
    # Determine if any other element contains a smaller value.
    for j in range( i + 1, n ):
      if theSeq[j] < theSeq[smallNdx] :
        smallNdx = j
  # Swap the ith value and smallNdx value only if the smallest value is
```

```
    # not already in its proper position. Some implementations omit testing
    # the condition and always swap the two values.
    if smallNdx != i :
      tmp = theSeq[i]
      theSeq[i] = theSeq[smallNdx]
      theSeq[smallNdx] = tmp

```

Task 3: Insertion Sort

Implement The **Insertion Sort** algorithm:

```
# Sorts a sequence in ascending order using the insertion sort algorithm.
def insertionSort( theSeq ):
  n = len( theSeq )
  # Starts with the first item as the only sorted entry.
  for i in range( 1, n ) :
    # Save the value to be positioned.
    value = theSeq[i]
    # Find the position where value fits in the ordered part of the list.
    pos = i
    while pos > 0 and value < theSeq[pos - 1] :
      # Shift the items to the right during the search.
      theSeq[pos] = theSeq[pos - 1]
      pos -= 1

    # Put the saved value into the open slot.
    theSeq[pos] = value

```

6 Lab Tasks

Comlete the Following Lab Taks.

Task 1: Binary Set ADT using Binarry Search

Implement a new **Set** class using a sorted list with the binary search.

Task 2: Modify Binary Search Algorithm

Modify the binary search algorithm to find the position of the first occurrence of a value that can occur multiple times in the ordered list.

Task 3: Return list of negative values

Design and implement a function to find all negative values within a given list. Your function should return a new list containing the negative values.

Task 4: Modify Bag ADT using Binary Search

Implement the Bag ADT from Chapter 1 to use a sorted list and the binary search algorithm. Evaluate the time complexities for each of the operations.

Task 5: Modify MAP ADT using a Sortd list and Binary Search

Implement a new version of the Map ADT from Section 3.2 to use a sorted list and the binary search algorithm.