CPE111: Programming with Data Structures

Week5: Sorting Algorithm

Practice I) Sorting with Best, Average, Worst cases

Mission: Choose a sorting algorithm and measure operation time for that sorting algorithm when n = 100, 500, …, 1000,5000,10000 in best, average and worst cases and then plot graph

List1: BubbleSort.py

```
1 # Sorts a sequence in ascending order using the bubble sort algorithm.
2 def bubbleSort( theSeq ):
3       n = len( theSeq ) - 1
4       # Perform n-1 bubble operations on the sequence
5       for i in range( n , 0 , -1) :
6       # Bubble the largest item to the end.
7               for j in range(i) :
8                       if theSeq[j] > theSeq[j + 1] : # swap the j and j+1 items.
9                               tmp = theSeq[j]
10                              theSeq[j] = theSeq[j + 1]
11                              theSeq[j + 1] = tmp
12      return theSeq
```

List2: SelectionSort.py

```
1 # Sorts a sequence in ascending order using the selection sort algorithm.
2 def selectionSort( theSeq ):
3       n = len( theSeq )
4       for i in range( n - 1 ):
5               # Assume the ith element is the smallest.
6               smallNdx = i
7               # Determine if any other element contains a smaller value.
8               for j in range( i + 1, n ):
9                       if theSeq[j] < theSeq[smallNdx] :
10                              smallNdx = j
11
12              # Swap the ith value and smallNdx value only if the smallest value is
13              # not already in its proper position. Some implementations omit testing
14              # the condition and always swap the two values.
15              if smallNdx != i :
16                      tmp = theSeq[i]
17                      theSeq[i] = theSeq[smallNdx]
18                      theSeq[smallNdx] = tmp
19      return theSeq
```

List 3: InsertionSort.py

```
1 # Sorts a sequence in ascending order using the insertion sort algorithm.
2 def insertionSort( theSeq ):
3       n = len( theSeq )
4       # Starts with the first item as the only sorted entry.
5       for i in range( 1, n ) :
6             # Save the value to be positioned.
7             value = theSeq[i]
8             # Find the position where value fits in the ordered part of the list.
9             pos = i
10            while pos > 0 and value < theSeq[pos - 1] :
11                  # Shift the items to the right during the search.
12                  theSeq[pos] = theSeq[pos - 1]
13                  pos -= 1
14
15            # Put the saved value into the open slot.
16            theSeq[pos] = value
17      return theSeq
```

List4: fragment of code for measure time elapsed

```
1 import time
2 start = time.time( )
3 # your algorithm
4 end = time.time( )
5 elapsed = end - start
```

List5: fragment of code for random integer number

```
1 import random
2 seq = []
3 for i in range(0 , n ):
4   seq.append(random.randint( numLow, numHigh ))
```

Practice II) Priority Queue

Mission: 1) Implement priority queue with sorting algorithm

2) Write a task manager program to add tasks with priorities

Priority Queue ADT

A priority queue is a queue in which each item is assigned a priority and items with a higher priority are removed before those with a lower priority, irrespective of when they were added. Integer values are used for the priorities with a smaller integer value having a higher priority. A bounded priority queue restricts the priorities to the integer values between zero and a predefined upper limit whereas an unbounded priority queue places no limits on the range of priorities.

- **PriorityQueue():** Creates a new empty unbounded priority queue.
- **BPriorityQueue(numLevels):** Creates a new empty bounded priority queue with priority levels in the range from 0 to numLevels - 1.
- **isEmpty():** Returns a boolean value indicating whether the queue is empty.
- **length ():** Returns the number of items currently in the queue.
- **enqueue(item, priority):** Adds the given item to the queue by inserting it in the proper position based on the given priority. The priority value must be within the legal range when using a bounded priority queue.
- **dequeue():** Removes and returns the front item from the queue, which is the item with the highest priority. The associated priority value is discarded. If two items have the same priority, then those items are removed in a FIFO order. An item cannot be dequeued from an empty queue.

List 6: Fragment of code for implementing a Priority Queue

```
class PriorityQEntry( object ):
    def __init__( self, item, prioity ):
        self._item = item
        self._priority = priority
```

Reference

Rance D. Necaise. *Data Structures and algorithms using python*. *Chapter4 and 5*. John Wiley&Sons,Inc., 2011

Michael T.Goodrich, Roberto Tamassia, Michael H. Goodwasser. *Data Structures and Algorithms in python*. *Chapter3 and 12*. John Wiley&Sons,Inc. 2013