National University of Sciences & Technology (NUST)
School of Natural Sciences (SNS)
Department of Mathematics

CS250: Data Structures and Algorithm

Class: BS2k21_Mathematics – Fall 2023

Lab 10:

Implementation of Sorting Algorithms and Complexity Analysis

Date: 22 November, 2023

Time: 10:00 am - 01:00 pm

Instructor: Fauzia Ehsan

Submission Deadline: 25th November, 2023 (11:59 P.M)

National University of Sciences & Technology (NUST)
School of Natural Sciences (SNS)
Department of Mathematics

Lab 10: Implementation of Sorting Algorithms and Asymptotic Complexity Analysis

Introduction
This lab is based on the analysis of different algorithms.

Objectives
Objectives of this lab are:

- To make students analyze different algorithms and their asymptotic complexities.
- Implement basic sorting algorithms and compare the running times of sorting algorithms.

Tools/Software Requirement
Python 3/ PyCharm IDE / MS Visual Studio

Contents

Sorting Algorithms

Selection Sort:

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two sub arrays in a given array.

1) The sub array which is already sorted.

2) Remaining sub array which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted sub array is picked and moved to the sorted sub array.

```
SELECTION-SORT(A,N)
    for current=0 to N-2
        min_index=current
        for j=current+1 to N-1
            if (A[j]<A[min_index])
                min_index=j
        swap(A[current], A[min_index])
```

Insertion Sort

Insertion sort is a popular sorting algorithm, which is quite simple to implement. The pseudo code is as follows:

```
INSERTION-SORT(A,N)
   for i=1 to N-1
      key = A[i]
      j = i - 1
      while j >= 0 and A[j]>key
         A[j+1] = A[j]
         j = j - 1
      A[j+1] = key
```

Bubble Sort:

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. The pseudo code is as follows:

```
BUBBLE-SORT(A,N)
    for i=0 to N-2
        for j=0 to n-i-1
            if (A[j]>A[j+1])
                swap(A[j], A[j+1])
```

Task 1 a: Implement Selection sort, Insertion sort and Bubble sort algorithms in Python.

Create a menu that displays all the algorithms and sort the values according to the user's choice. (Use match-case statements)

Task 1b (average case complexity):

The next step is to compare the running time of algorithms. Generate arrays of random numbers in the range 1 to 1000 with sizes 100, 1000, and 5000. Compare the running times of the three algorithms on each array. How do they compare? Are the results what you expected, and why? Answer the questions in at the end of the word file.

Task 1c (best- and worst-case complexity):

Now sort the arrays using built-in sort function, once in ascending order and then in descending order. Given both sorted arrays as inputs to all the three algorithms and compute their running time. The running time of which algorithm shows most variations based on the structure of the input and why? Answer the questions in at the end of the word file.

Task 02.

a. You have already implemented a function that prints all elements of a list of size n, where n>=0. What is the Big-O complexity of that operation?
b. Suppose you have an **array-based list** of size **n**. Implement a function takes a position number **pos** as input from the user, and returns the value stored at that position. What is the Big-O time complexity of this function?
c. Suppose you have a **singly linked list** of size **n**. Implement a function takes a position number **pos** as input from the user, and returns the value stored at that position. What is the Big-O time complexity of this function? What is its best-case time complexity?

Task 3. Big-O Complexity Analysis

In the below given table, determine the asymptotic complexity of different operations:

Operation	**Big-O Complexity**
Insert an element at the front of a singly linked list of size n	
Insert an element at the tail end of a singly linked list of size n.	
Delete the last node of a singly linked list of size n.	
Insertion at the front of an array list of size n	
Insertion at the tail end of an array list of size n	
Enqueue in a queue of length n.	
Dequeue in a queue of length n.	
Converting an expression of length n from infix to postfix form using stack	
Finding an element via Binary Search algorithm in a sorted array-list of size n.	
Finding an element via Binary Search algorithm in an **unsorted** array-list of size n. Think about it!	

Happy Coding!

Deliverables

Comment your program heavily. Intelligent comments and a clean, readable formatting of your code account for 20% of your grade.

You should submit your codes and report as a compressed zip file. It should contain all files used in the exercises for this lab.

The submitted file should be names `cs250_firstname_lastname_lab10.zip`