SEAT NO. 153

NED UNIVERSITY OF ENGINEERING & TECHNOLOGY

SECOND YEAR FALL SEMESTER (ELECTRICAL ENGINEERING)
EXAMINATIONS 2019
BATCH 2018

Dated:03-02-2020

Time: 3 Hours **Max.Marks:60**

Data Structures and Algorithms - EE-264

Instructions:

1. **Attempt all Questions.**
2. **In case, if you find any missing information, assume by yourself and properly mention it.**
3. **All questions carry equal (10) Marks.**

Question 1: **[CLO1]**

Perform a detailed analysis of SELECTION SORT algorithm including general expression of running time, best case and worst case running times. Also, express the running time in θ-notation.

Pseudo Code:

```
SELECTION SORT
1. For i = 1 to n:
2.       min_index = i
3.       For j = i+1 to n:
4.            If A[min_index] > a[j]:
5.                 min_index = j
6.       Swap (A[i],A[min_index])
```

Question 2: **[CLO1]**

Perform the time complexity analysis under worst case for Merge Sort Algorithm (based on Divide and Conquer approach). Also, express your answer in relevant asymptotic notation.

Pseudo Code:

```
MERGESORT(A):
1.  n = length(A)
2.  if n < 2:
3.       return
4.  mid = n/2
5.  left = array of size (mid)
6.  right = array of size (n-mid)
7.  for i = 0 to mid-1:
8.       left[i] = A[i]
9.  for j = mid to n-1:
10.      right[j-mid] = A[j]
11. MERGESORT(left)
12. MERGESORT(right)
13. MERGE(left, right, A)
```

```
MERGE(L,R,A):
1.  nL = length(L)
2.  nR = length(R)
3.  i = j = k = 0
4.  while i < nL and j < nR:
5.       if L[i] < R[j]:
6.            A[k] = L[i]
7.            i = i + 1
8.       else:
9.            A[k] = R[j]
10.           j = j + 1
11.      k = k + 1
12. while i < nL:
13.      A[k] = L[i]
14.      i = i + 1
15.      k = k + 1
16. while j < nR:
17.      A[k] = R[j]
18.      j = j + 1
19.      k = k + 1
```

Question 3: **[CLO1]**

Explain the following Asymptotic Notations by considering an example,

1. O Notation
2. Ω Notation
3. Θ Notation

Question 4: **[CLO2]**

Demonstrate the mechanism of maintaining data in *stacks* and *queues*. What are their typical operations and policy? Write a python class named *Queue* that uses list structure to store the data inside of it. Also define all the functions, which Queue data structure typically offers, in *Queue* class.

Question 5: **[CLO2]**

You have been asked to develop a system which maintains data of employees in an organization. What would be your preference at implementation level (Traditional Procedural-Oriented programming -TPOP or Object-Oriented programming - OOP)? Take the fundamental components of OOP such as Class, use pseudo-code or C++ or Python and implement some of the following basic tasks related to this application.

a) Create an *Employee* class to store basic information of an employee like name, pay and job.
b) Write a method in *Employee* class to increase the salary of a person with desired percentage.
c) Create a sub-class of *Employee* named *Manager* which replaces the inherited method to increase the salary of a person by including an additional bonus of 10% added in the percentage input of a function.

Question 6: **[CLO2]**

Describe the following functions applied on Dynamic Sets:

1)SEARCH (S , k)
2) INSERT (S , x)
3) DELETE (S , x)
5) MAXIMUM (S)
6) PREDECESSOR (S , x)

DS .

unsigned int array **primefactors[]. num** contains the number to be processed by any of the described functions and **ans** contains the result generated by any of the functions except for primeFactors(), for which the result is put into **primefactors[]** array.

The class should be named "NumberTheoretic" and all principles of abstraction and encapsulation must be incorporated in writing it. You are at freedom to define variables and functions other than those described here.

Question No. 4

The following is C++ implementation for the naive prime factorization algorithm:

```
int input,factor,factor_remainder, check=0;
cout<<"\n\nEnter a number to get its prime factors: ";
cin>>input;
cout<<"\n\nThe prime factors of "<<input<<" are:";

factor=2;                    // initialising factor with 2
                             // which could be the first factor

while(factor<=input)         // Termination Condition: factor is
{                            // increasing by 1 and input is reducing
                             // due to division

    if(isprime(factor)==1)   // which means factor is prime
    {
        factor_remainder=input%factor;

        // If factor is prime we have two cases:

        //1) That factor divides the input completely
        if(factor_remainder==0)
        {
            input=input/factor;
            cout<<factor<<",";
        }

        //2) The factor does not divide the input
        if(factor_remainder!=0)
        {
            factor=factor+1;
        }

    }
    else                     // factor is composite
    {
        factor=factor+1;
    }
}
```

(15)

This program has an underlying problem: it is inefficient for some of the factors, as it tests "factor" for primality without consideration. Make this program a bit more considerate by modifying it in such a way that "factor" variable is only tested for primality when a new "factor" is selected. Also discuss how much performance gain would this modification bring to the program.

Question No. 5

Describe the Quick-Sort algorithm using an example and write a C++ function definition to implement it on a character array which starts at index i and ends at index j:

void quickSort(char array[], int i, int j);

(15)

------- X -------

NED UNIVERSITY OF ENGINEERING & TECHNOLOGY

SECOND YEAR SPRING SEMESTER (ELECTRICAL ENGINEERING)
EXAMINATION 2016
BATCH 2014-15 & PREVIOUS BATCHES

Time: 3 Hours

Dated: 28-04-2016
Max. Marks: 60

DATA STRUCTURE & ALGORITHMS- EE-264

Instructions:

1) Attempt four (4) questions in all – all questions carry equal marks.
2) Answer the questions objectively in legible writing. Any missing information may be assumed and clearly mentioned.
3) Use pen to answer the questions and be precise and objective while writing codes. Careless syntax errors will be dealt with serious deduction in marks.
4) Your algorithms/programs should be valid for all possible sets of inputs.

Question No. 1

a) What is "Algorithm Analysis" and why is it an important part of computer programming? Compare "Algorithm Analysis" with other desired characteristics of computer programming (compare with at-least five (05) other characteristics). **(07)**

b) Describe the three cases used to analyze algorithm performance, discussing significance of each method in detail. Demonstrate the application of ~~these~~ any 3 of these techniques on any algorithm of your choice and compute its asymptotic complexity with reference to θ(n) operator. **(08)**

Question No. 2

Using the C++ vector class and algorithm library, write a program that allows user to enter 100 names in a string vector (the user should be allowed to stop entering names intermediately), sort them in ascending order (alphabetical order), and finally display them on screen. **(15)**
After writing the program, discuss the method/algorithm used for sorting the sequence of strings. How different is it from sorting a number sequence?

Question No. 3

Write a C++ class that contains the following number theoretic algorithms as functions: **(15)**

a) bool isPrime(unsigned int N); //checks whether N is prime or not
b) unsigned int gcd(unsigned int A, unsigned int B); //computes greatest //common divisor of A, and B integers
c) void primeFactors(unsigned int N, unsigned int factors[]); //computes prime factors of N and stores //them in factors[] array
d) bool isEven(unsigned int N); //checks if N is even or odd
e) bool isOdd(unsigned int N); //checks if N is odd

Apart from these functions, there may be some general input/output functions in the class. The class needs to have public and private sections, moreover it must contain private data comprising of two unsigned integers **num** and **ans** and an

θ(n) = n

Data Structures and Algorithms (EE-264)

S.E ELECTRICAL

MID-TERM EXAMINATION (Fall 2018)

Note: Attempt Any One Question

Time Allowed: 50-Minutes

Maximum Marks: 20

This question gauges CLO-4 of this course

Q. No. 01

Analyze time complexity (worst & best case) of following *Selection Sort* Algorithm.

```
1: for i ← 1 to n − 1 do
2:     min ← i
3:     for j ← i + 1 to n do
4:         // Find the index of the i^th smallest element
5:         if A[j] < A[min] then
6:             min ← j
7:         end if
8:     end for
9:     Swap A[min] and A[i]
10: end for
```

Q. No. 02

Analyze time complexity (worst case) of following search functions. The worst case in searching problem is that the key entry is not present in the list A. You may work by considering that A = [1, 2, 3, 4, 5, 6, 7, 0] and key to be either 0 or 9. Can you generalize the time complexity in terms of length of A, if A is always sorted and has length equals to 2^n?

```
search1(A, key)
flag = 0
for i=1 to length of A
    if A[i] == key
        break
        flag = 1
return flag
```

```
search2(A, key)
flag = 0
f = 1
l = (length of A)
while f <= l and flag == 0
    m = floor((f + l)/2)  # The usage of floor function is given as: floor((2+3)/2) = 2
    if A[m] == key
        flag = 1
    else if key < A[m]
        l = m - 1
    else
        f = m + 1
return flag
```