|  |  |  |
| --- | --- | --- |
| BUE final logo | **Examination Paper Proofing & Printing Confirmation Sheet** | |
| Module Title : **Analysis of algorithms** | | Module Code:  **14CSCI01I** |
| Module Leader : **Abeer Hamdy** | | Semester  **One** |
| Proofed by : **Khaled Nagaty** | | Date of examination |

I hereby confirm:

√

|  |  |
| --- | --- |
| That this examination paper assesses the ILOs defined in the module specification  That appropriate model answers were provided with this examination paper  That this examination paper has been proof-read and is approved for printing  That this examination paper follows the approved University template | √ |
| √ |
| √ |
|  |

**Signed (Proof Reader): Khaled Nagaty**

**Printing instructions & stationery requirements**

|  |  |  |
| --- | --- | --- |
| Number of copies of examination paper to be printed | |  |
| Date of examination | |  |
|  | | Number required per student |
| Stationery Requirement(s) | 8 page answer book |  |
| 12 page answer book |  |
| Graph paper |  |
| Other |  |

**Signed (Module Leader) Abeer Hamdy**

|  |  |  |
| --- | --- | --- |
| BUE final logo | **14CSCI01I**  **Final Examination,**  **2014/ 2015** | |
| Module Title: **Analysis of algorithms** | | |
| Module Leader : **Abeer Hamdy** | | Semester  **One** |
| Equipment NOT allowed |  | |

**Instructions to Students**

* *You should attempt* ***ALL questions****.*
* *The exam paper is 4 pages long.*
* *The total mark of the exam is 100.*
* *The approximate allocation of marks is shown in brackets by the questions.*

This examination is **Two** hours long.

**Attempt all the Questions**

**Q1: [30 Total]**

a) What is the complexity class of each of the following piece of code? [12 Marks, 3 Marks each]

(1) for *i* ← 1 to 2*n*

for *j* ← 1 to *n*

*x* ← *x* + 1

(2) for *i* ← 1 to *n*

for *j* ← 1 to *i*

for *k* ← 1 to *j*

*x* ← *x* + 1

(3) *i* ← *n*

while (*i* ≥ 1)

for *j* ← 1 to *n*

*x* ← *x* + 1

*i* ←*i/* 2

(4) *i* ← *2*

while (*i* < *n*)

*i* ← *i*\**i*

*x* ← *x* + 1

b) Use iterative substitution to solve the following recurrence relation:

 [5 Marks]

c) Use Master theorem to find the Θ notation for the following recurrence relations.

 [4 Marks]

d)Set up a recurrence relation for the following algorithm?

*ALGORITHM* *GraphComplete(A[0..n − 1, 0..n − 1])*

*//Input: Adjacency matrix A[0..n − 1, 0..n − 1]) of an undirected graph G*

*//Output: 1 (true) if G is complete and 0 (false) otherwise*

***if*** *n = 1* ***return*** *1*

***else***

*if* ***not*** *GraphComplete(A[0..n − 2, 0..n − 2])* ***return*** *0*

***else for*** *j ←0* ***to*** *n − 2* ***do***

***if*** *A[n − 1, j]= 0* ***return*** *0*

***return*** *1*

[5 Marks]

e) State true or false for each of the following statements, you must justify your answer

(1) 2n + 3n = O(2n ) [2 Marks]

(2) n(n + 1)/2 = Ω (n) [2 Marks]

**Q2: [50 Total]**

a) You have a set A of n nuts and a set B of n bolts, such that each nut in A has a unique matching bolt in B. Unfortunately, the nuts in A all look the same, and the bolts in B all look the same as well. The only kind of a comparison that you can make is to take a nut-bolt pair (a,b) such that aA, and bB, and test to see if the threads of a are larger, smaller or a perfect match with the treads of b. Describe an algorithm ,based on divide and conquer algorithm, to match up all of the nuts in set A with all of the bolts in set B. *(Hint: Express the essence of the algorithm using simple English words or pseudo code)* [15 Marks]

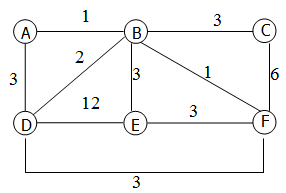
b) Does greedy method produces optimal solution to the fractional knapsack problem? and what is the greedy choice? [5 Marks]

c) Given two strings ‘X’ and ‘Y’ with length “n” and “m” respectively. You are required to design a dynamic programming algorithm to find the length of the longest common substring between the two given strings. For example, if the given strings are “GeeksforGeeks” and “GeeksQuiz”, the output should be “5” as longest common substring is “Geeks”. Write down the recurrence equation (with base condition) , and define its major terms (don’t write the whole algorithm).  [15 Marks]

d) Write pseudo code of floyd's algorithm to find all pairs shortest paths in a graph (The paths themselves and their lengths) [15 Marks]

**Q3: [20 Total]**

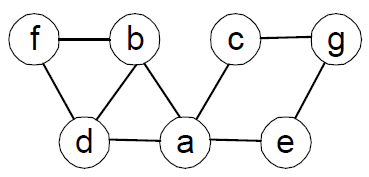
a) Consider the following graph:



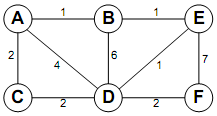
(1) What is the cost of the minimum spanning tree? [ 2 Marks]

(2) How many minimum spanning trees does it have? Give a short argument to justify your answer. [2 Marks]

b) Traverse the following graph by breadth-first search and construct the corresponding breadth-first search tree. Start at node “a”. [7 Marks]



c) Suppose Dijkstra’s algorithm is run on the following graph, starting at vertex "*A"*.



Fill out the following table showing the intermediate distance values of all the vertices after each iteration of the algorithm (Three iterations only are considered here) [9 Marks]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| iteration → | Initial | 1 | 2 | 3 |
| Tree  Q |  |  |  |  |
| A |  |  |  |  |
| B |  |  |  |  |
| C |  |  |  |  |
| D |  |  |  |  |
| E |  |  |  |  |
| F |  |  |  |  |

Where: *Q : Vertex priority queue*

*Tree : Tree of shortest paths*

**Model Answer**

|  |  |  |
| --- | --- | --- |
| BUE final logo | **14CSCI01I**  **Final Examination,**  **2014/ 2015** | |
| Module Title: **Analysis of algorithms** | | |
| Module Leader : **Abeer Hamdy** | | Semester  **One** |
| Equipment NOT allowed |  | |

**Instructions to Students**

* *You should attempt* ***ALL questions****.*
* *The exam paper is 4 pages long.*
* *The total mark of the exam is 100.*
* *The approximate allocation of marks is shown in brackets by the questions.*

This examination is \_\_**TWO**\_\_ hours long.

**Q1: [30 Total]**

a) What is the complexity class of each of the following piece of code? [12 Marks, 3 Marks each]

(1) for *i* ← 1 to 2*n*

for *j* ← 1 to *n*

*x* ← *x* + 1

**O(n2 )**

(2) for *i* ← 1 to *n*

for *j* ← 1 to *i*

for *k* ← 1 to *j*

*x* ← *x* + 1

**O(n3 )**

(3) *i* ← *n*

while (*i* ≥ 1)

for *j* ← 1 to *n*

*x* ← *x* + 1

*i* ←*i/* 2

**O(nlogn)**

(4) *i* ← *2*

while (*i* < *n*)

*i* ← *i*\**i*

*x* ← *x* + 1

**O(loglogn)**

b) Use iterative substitution to solve the following recurrence relations:

 [5 Marks]

**The answer will be O(n)**

c) Use Master theorem to find the Θ notation for the following recurrence relations.

 [4 Marks]

**Ans: O (n)**

d)Set up a recurrence relation for the following algorithm?

*ALGORITHM* *GraphComplete(A[0..n − 1, 0..n − 1])*

*//Input: Adjacency matrix A[0..n − 1, 0..n − 1]) of an undirected graph G*

*//Output: 1 (true) if G is complete and 0 (false) otherwise*

***if*** *n = 1* ***return*** *1*

***else***

*if* ***not*** *GraphComplete(A[0..n − 2, 0..n − 2])* ***return*** *0*

***else for*** *j ←0* ***to*** *n − 2* ***do***

***if*** *A[n − 1, j]= 0* ***return*** *0*

***return*** *1*

[5 Marks]

**The recurrence relation is : T(n)=T(n-1)+2n-1**

e) State true or false for each of the following statements, you must justify your answer

(1) 2n + 3n = O(2n ) [2 Marks]

**False**

(2) n(n + 1)/2 = Ω (n) [2 Marks]

**False**

**Q2: [50 Total]**

1. You have a set A of n nuts and a set B of n bolts, such that each nut in A has a unique matching bolt in B. Unfortunately, the nuts in A all look the same, and the bolts in B all look the same as well. The only kind of a comparison that you can make is to take a nut-bolt pair (a,b) such that aA, and bB, and test to see if the threads of a are larger, smaller or a perfect match with the treads of b. Describe an algorithm ,based on divide and conquer algorithm, to match up all of the nuts in set A with all of the bolts in set B. *(Hint: Express the essence of the algorithm using simple English words or pseudo code)*

[15 Marks distributed over the steps]

Match (A , B) {

1) If A and B contain one element, just match them and finish. Else continue.

2) Pick a nut at random, let this nut be a.

3) Compare each bolt to a.

i) Place each smaller bolt in a set BL.

ii) Place each larger bolt in a set BH.

iii) If it matches, set aside this particular bolt and label it b.

4) Compare every other nut (except a) to b.

i) Place each smaller nut in a set AL.

ii) Place each larger nut in a set AH.

5) recursively call Match(AL, BL)

6) recursively call Match(AH, BH)

}

This work is very similar to QuickSort

b) Does greedy produces optimal solution to the fractional knapsack problem? and what is the greedy choice? [5 Marks]

**Yes , it produces optimal solution. The greedy choice is arranging items in descending order based on their value/weight**

c) Given two strings ‘X’ and ‘Y’ with length “n” and “m” respectively. You are required to design a dynamic programming algorithm to find the length of the longest common substring between the two given strings. For example, if the given strings are “GeeksforGeeks” and “GeeksQuiz”, the output should be “5” as longest common substring is “Geeks”. Write down the recurrence equation (with base condition) , and define its major terms (don’t write the whole algorithm).  [15 Marks]

The idea is to find length of the longest common suffix for all substrings of both strings and store these lengths in a table.

The longest common suffix has following optimal substructure property

LCSuff(X, Y, m, n) = LCSuff(X, Y, m-1, n-1) + 1

( if X[m-1] = Y[n-1])

= 0 Otherwise (if X[m-1] != Y[n-1])

The maximum length Longest Common Suffix is the longest common substring.

LCSubStr(X, Y, m, n) = Max(LCSuff(X, Y, i, j)) where 1 <= i <= m

and 1 <= j <= n

d) Write pseudo code of floyd's algorithm to find all pairs shortest paths in a graph ( The paths themselves and their lengths)

[15 Marks : 5 Marks for path matrix, 4 Marks for distance matrix , 6 marks for three loops ]

Algorithm ModifiedFloyd (W[1..n,1..n])

// input : the weight matrix W of the weighted graph

// output : the distance matrix of the shortest paths' lengths D & the Path matrix

D←W // it is not necessary if W can be overwritten

for each vertex pair (i,j)

path[i,j] ← null

for k ← 1 to n do

for i ←1 to n do

for j ← 1 to n do

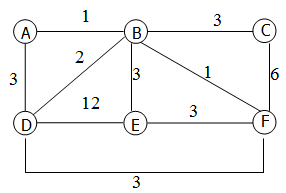
if (d[i,k]+d[k,j] < d[i,j])

path[i,j] ←vertex k

d[i,j] ← d[i,k]+d[k,j]

**Q3: [20 Total]**

a) Consider the following graph:



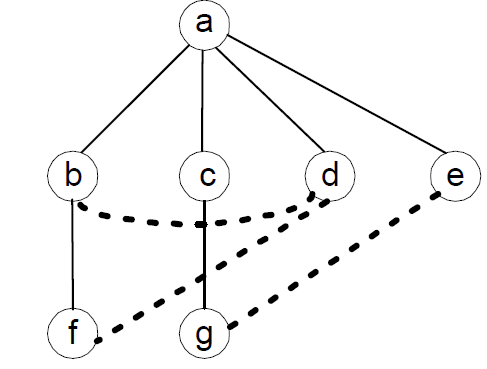
(1) What is the cost of the minimum spanning tree? [2 Marks]

**10**

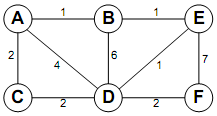
(2) How many minimum spanning trees does it have? Give a short argument to justify your answer. [2 Marks]

**Two MST, The only choice arises when you have the edges BE & EF to choose between.**

b) BFS tree and cross edges [7 Marks]

****

c) Suppose Dijkstra’s algorithm is run on the following graph, starting at vertex "*A"*.



Fill out the following table showing the intermediate distance values of all the vertices after each iteration of the algorithm.

[9 Marks , 3 marks for each of the three iterations]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **iteration →** | **Initial** | **1** | **2** | **3** |
| **Tree**  **Q** | 0 | A | B | C |
| **A** | 0 | 0 | 0 | 0 |
| **B** | ∞ | 1 | 1 | 1 |
| **C** | ∞ | 2 | 2 | 2 |
| **D** | ∞ | 4 | 4 | 4 |
| **E** | ∞ | ∞ | 2 | 2 |
| **F** | ∞ | ∞ | ∞ | ∞ |

|  |  |
| --- | --- |
| **Module Code:** 13CSCI01I | **Title:** Analysis of Algorithms |
| **Modular weight:** 10 | **Examination weighting:** 60% |
| **Prerequisite modules:** MTHC01P | |
| **Reassessment:** No Restrictions | |
| **Module Leader:** Assoc. Prof. Abeer Hamdy | |
| **Semester taught:** 1 | |
| **Key words:** NP problems, Algorithms | |
| **Date of latest revision:** Aug. 2013 | |

**Aims**

The aim of the module is to enable students to analyse the amount of resources needed to solve a given computational problem and to compare the efficiency of using different algorithms in addressing and solving the problem.

**Intended Learning Outcomes**

Upon successful completion of this module students should demonstrate understanding and ability in:

***Knowledge and understanding***

1. Techniques to decide whether a given problem can be solved exactly and in what time-scale, or whether only approximations to their exact solution can be obtained as is the case in many real world problems. A variety of algorithm types and algorithm design techniques and be able to select the appropriate one for a given problem;[2]

***Subject-specific skills***

1. Analyse an algorithm and determine its complexity class;[8]
2. Use a variety of strategies to design algorithms for typical classes of problems;[12]
3. Develop given algorithms in a high level programming language;[14]

***Key/transferable skills***

1. Plan, develop, evaluate and report on individual pieces of work. [20]

**Contents**

* Types of Complexities, Time Complexity Calculations
* Containers, Key Tables & Lists, Binary Search Trees, Disjoint Sets
* Elementary Sorting Algorithms, Recursive Algorithms
* Divide & Conquer Algorithms, Graphs, Greedy Algorithms
* Traversal & Search Algorithms, Dynamic Programming
* Backtracking Algorithms

**Methods of Learning, Teaching and Assessment**

Total student effort for the module is 100 hours on average:

***Learning and Teaching***

1. 12, 2h lectures, informing learning outcomes 1.
2. 12, 2h workshops/labs, informing learning outcomes 2-4.
3. 52 h private study (approx), informing learning outcomes 1-5.

***Assessment:***

1. Two pieces individual assignments includes a set of problems. Student is required to work out solution for each and assess his suggested solution. This method carries 40% of the total mark and aims to develop and assess learning outcomes 2-5.
2. One 2-hour unseen written final exam carrying 60% of the total mark to assess learning outcomes 1-3.

***Feedback given to students in response to assessed work***

Feedback will be provided for each assessed component in written form as appropriate. The coursework will be returned to students with feedback on the accompanying coursework turn in sheet. If students require additional feedback, they are welcome to speak with the TA, and the module-leader.

***Developmental feedback generated through teaching activities***

Dialogue between students and staff in workshops and Labs

**Reading List**

* Anany Levitin, “Introduction to the design and analysis of algorithms”,2nd edition, Pearson,(2007).
* Skiena, S., “*The Algorithm Design Manual*”, Springer-Verlag, (1998).
* Sara Baase & Allen Van Gelder, *Computer Algorithms: Introduction to Design and Analysis*, 3rd edition, Addison-Wesley,(2000).
* Cormen, Leiserson, Rivest & Stein Introduction to Algorithms, 3rd Edition, The MIT Press, (2009).