THE UNIVERSITY OF THE SOUTH PACIFIC

CS214: Design and Analysis of Algorithms

Faculty of Science, Technology and Environment / School of Computing, Information and Mathematical Sciences

Final Examination

Semester I 2015

Flexi Mode

Special Question-Answer Booklet

Duration of Exam: 3 hours + 10 minutes
Reading Time: 10 minutes
Writing Time: 3 hours

Instructions:

1. 9 questions in this special question-answer booklet. Paper is out of 95 points.
2. This exam contributes to 40 % of course.
3. Open Book Exam: Printed copy of lecture notes, codes and tutorials are allowed. Calculator is allowed.
4. **Write your answers in the space given in Question paper.**
5. Number of pages: 24

NAME: …..

ID: …..

SEAT NO: …..

Problem 1 (14 points): Algorithms and Time Complexity

Give the time complexity for each of the following algorithms. If you know of more than one algorithm to solve a given problem, then you must give the time complexity of the most efficient algorithm. You do not need to give any justification for your answers.

a) Time complexity for checking whether two indices *i* and *j* exist in an **unsorted array** of integers such that *(a[i] * a[j]) /2= x* for some value *x*.

b) Time complexity of Prim's algorithm for *n* vertices.

c) Time complexity for the multiplication of two *n x n* matrices using matrix multiplication code as given below.

```
for (c = 0; c < m; c++) {
      for (d = 0 ; d < n; d++) {
         sum[c][d] = first[c][d] + second[c][d];
         printf("%d\t", sum[c][d]);
      }
      printf("\n");
   }
```

d) Time complexity for computing the factorial as given in code below:

```
long factorial(int n)
{
  int c;
  long result = 1;

  for (c = 1; c <= n; c++)
    result = result * c;

  return result;
}
```

e) Is the time complexity same for the recursive implementation?

```
long factorial(int n)
{
  if (n == 0)
    return 1;
  else
    return(n * factorial(n-1));
}
```

f) Explain major difference between the above implementations of Fibonacci and which one is better and why?

g) Time complexity for solving the 0-1 Knapsack Problem using Greedy Method.

h) Time complexity for solving the Traveling Salesman Problem using a brute-force approach.

Question 2: Comparison of Algorithms (10 Points)

a) What is the difference between **Branch and Bound** and **Backtracking.** Which approach is better for optimization problems ? Explain with examples. (5 points)

b) What is the major difference between **insertion sort** and **merge sort** and what are their similarities ?

Which one is the faster and why? (5 points)

Problem 3 (10 points): Backtracking Algorithms

In the sum-of-subsets problem you are given as input:

# a set of n positive integers (weights) {w1, w2, w3, ... , wn}, and
# a target sum, W

with the task of finding all subsets that sum to the target sum, W.
Consider an instance of the sum-of-subsets problem: For the weights of { 4, 6, 10, 11, 14 }, find all the subsets adding to 25.

To solve a problem using backtracking, you need to answer the following questions:

a) Give the state-space tree (7 points)

b) What state information is needed at each node? (2 point)

c) Time complexity (2 point)

d) How can this algorithm be implemented and in future be improved further? (3 Point)

Problem 4. Heuristic Algorithms (12 Points). You can use code and diagrams to explain your answers.

a) What are the advantages of simulated annealing and why does it perform better than Greedy Algorithm for the TSP problem? (3 Points)

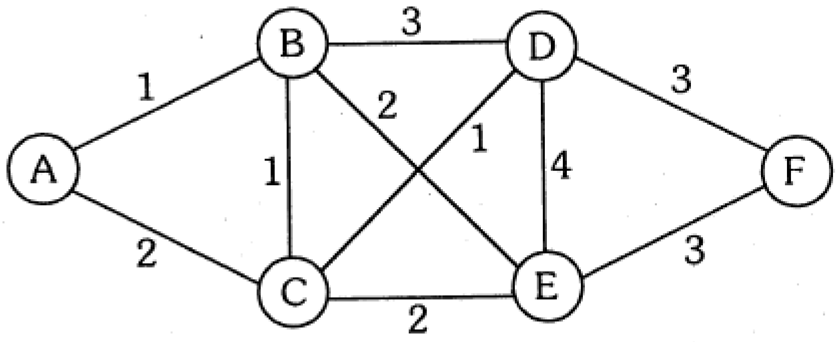
b) Explain how Global search is enforced in the beginning and local search at the end. What will you do to improve simulated annealing algorithm? (4 points)

c) Give the code for the Fitness/Energy Function for a 0-1 Knapsack problem. (3 Marks)

d) List some limitations of Simulated Annealing (2 Marks).

Problem 5: MST (10Points)

a) Using Kruskel's algorithm find the minimum spanning tree for the following graph. Give detailed comments and intermediate steps with reference to the Algorithm below. (7 Points)



Kruskal's Algorithm
MST-KRUSKAL(G, w)

```
1.  A ← Ø
2.  for each vertex v # V[G]
3.      do MAKE-SET(v)
4.  sort the edges of E into nondecreasing order by weight w
5.  for each edge (u, v) # E, taken in nondecreasing order by weight
6.      do if FIND-SET(u) ≠ FIND-SET(v)
7.          then A ← A # {(u, v)}
8.              UNION(u, v)
9.  return A
```

b) What are the major differences and similarities between Floyd's and Kruskel's Algorithm? (3Points)

Problem 6: Sorting Algorithms (7 Points)

Pseudocode for top down merge sort algorithm (using Lisp) which recursively divides the input list into smaller sublists until the sublists are trivially sorted, and then merges the sublists while returning up the call chain.

```
function merge_sort(list m)
      // Base case. A list of zero or one elements is sorted, by
definition.
    if length(m) <= 1
        return m

      // Recursive case. First, *divide* the list into equal-sized
sublists.
    var list left, right
    var integer middle = length(m) / 2
    for each x in m before middle
         add x to left
    for each x in m after or equal middle
         add x to right

    // Recursively sort both sublists
    left = merge_sort(left)
    right = merge_sort(right)

    // Then merge the now-sorted sublists.
    return merge(left, right)


In this example, the merge function merges the left and right sublists.

function merge(left, right)
    var list result
    while notempty(left) and notempty(right)
        if first(left) <= first(right)
            append first(left) to result
            left = rest(left)
        else
            append first(right) to result
            right = rest(right)
    // either left or right may have elements left
    while notempty(left)
        append first(left) to result
        left = rest(left)
    while notempty(right)
        append first(right) to result
        right = rest(right)
    return result
```

Refer to the MergeSort Algorithm Above and Sort: **5, 26, 7, 14, 3, 7, 2**
Give detailed diagram of the sorting procedure with comments.

Problem 7: Knapsack Problem (12 Points)

Consider the 0-1 knapsack problem with n= 4 and W = 14

Item	Price	Weight	Price/Weight
1	$20	4	10
2	$50	4	6
3	$10	8	15
4	$15	5	2

(a) Use Greedy Method to compute the optimal profit in filling the knapsack (3 Points)

(b) Use Branch and Bound to solve the knapsack problem for the same items. Show the state space tree. (9 Points).

Problem 8: Huffman Coding

Use Huffman Coding Algorithm to Construct the tree for the Following String (10 Marks). Give the bit representation that will be used for encoding. Note that string is case sensitive and you must cater for the space.

“NAMASTE FIJIANS”

Problem 9: Design of Algorithms (10 Points)

Design an Algorithm for calculating Total Price for selected items in Vending machine.

```
class Item{
        double Price ;
        int NumberItems;
         string ItemName;
         }
```

Assume that you have an Array of selected items:

```
int SelectedItems [ ] = {1, 2, 2, 1, 7, 7, 3}
```

(if 1 is water and 2 is orange, then this means that you selected 2 bottles of water and 2 oranges etc – order does not matter)

Construct the required data structure to store 10 items with any values (no need to show item price and values – we assume that they are already initialized) . (2 Points)

Using OOP, develop a method that calculates total price for selected items while checking if you have the item (i.e. NumberItems > 0). (8 Points)

EXTRA Page

EXTRA Page

EXTRA Page