THE UNIVERSITY OF THE SOUTH PACIFIC

CS214: Design and Analysis of Algorithms
School of Computing, Information and Mathematical Sciences

Final Examination
Semester 2 2017

Mode: Face-to-Face /Blended

Duration of Exam: 3 hours + 10 minutes

Reading Time: 10 minutes

Writing Time: 3 hours

Total Marks: 45 (45% of final grade)

Instructions:

1. This is a closed book exam
2. This exam has three sections – I, II and III
    a. Section I: 12 Marks
    b. Section II: 24 Marks
    c. Section III: 9 Marks
3. All questions are compulsory.
4. Write your answers in this booklet.
5. This exam is worth 45% of your overall mark. The minimum exam mark is 18/45.
6. Number of pages (including this cover page): 22

Name: _____

ID: _____

Section I: Multiple Choice Questions [12 marks]

Circle the letter of the correct answer for Questions 1-12 in the grid provided below. Each question has only one correct answer.

Question	Answer			
1	a	b	c	d
2	a	b	c	d
3	a	b	c	d
4	a	b	c	d
5	a	b	c	d
6	a	b	c	d
7	a	b	c	d
8	a	b	c	d
9	a	b	c	d
10	a	b	c	d
11	a	b	c	d
12	a	b	c	d

1. Which of the following lists (arrays) of integers will produce the worst-case time complexity for *Quicksort* algorithm?
   a. 1, 10, 100, 1000, 1000
   b. 2000, 1000, 200, 100, 20, 10
   c. 1, 1, 1, 1, 1, 1,1
   d. all of the above

2. Which of the following lists complexities in decreasing order?
   a. $\Theta(2^n)$, $\Theta(n^2)$, $\Theta(n\log n)$, $\Theta(1)$, $\Theta(n)$, $\Theta(\log n)$
   b. $\Theta(2^n)$, $\Theta(n^2)$, $\Theta(n)$, $\Theta(n\log n)$, $\Theta(\log n)$, $\Theta(1)$
   c. $\Theta(2^n)$, $\Theta(n^2)$, $\Theta(n\log n)$, $\Theta(n)$, $\Theta(\log n)$, $\Theta(1)$
   d. none of the above

3. Given the three binary encoding below:

CODE1	A:0	B:01	C:010	D:011
CODE2	A:01	B:10	C:1110	D:1111
CODE3	A:01	B:10	C:110	D:111

   a. CODE1 is not a prefix code, as it is not a fixed-length binary code
   b. CODE2 is less efficient than CODE3
   c. Cannot compare the efficiency of CODE2 and CODE3, as the frequencies of the characters are not provided
   d. None of the above

4. Prim's Algorithm is greedy on _____, whereas Kruskal's Algorithm is greedy on _____.
   a. cycles, subsets
   b. subsets, cycles
   c. vertices, edges
   d. edges, vertices

5. Below is an adjacency matrix for a graph, where ∞ means "no edge". Which of the followings is true?

	1	2	3	4
1	0	2	∞	5
2	3	0	∞	∞
3	∞	8	0	4
4	∞	∞	7	0

   a. $D^3[3][2] = D^2[3][2]$
   b. $D^2[3][2] = D^1[3][2]$
   c. All of the above
   d. None of the above

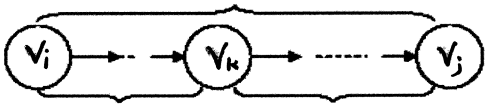
6. If an algorithm has time complexity $T(n)=2n^3+3n^2+4n+5$, it has _____ and _____.
   a. $\Theta(n^3)$ and $O(n^3)$
   b. $\Omega(n^3)$ and $O(n^4)$
   c. all of the above
   d. none of the above

7. Divide-and-Conquer algorithms can improve algorithmic efficiency. However, they should be avoided in the following cases:
   a. An instance of size *n* is divided into two or more instances each almost of size *n*
   b. An instance of size *n* is divided into almost *n* instances of size *n/c*, where c is a constant
   c. When an algorithm has time complexity *T(n)=(n-1)xT(n/2)*
   d. All of the above

8. Shortest paths can be computed by using the equation:

$$D^k[i][j] = minimum(D^{k-1}[i][j], D^{k-1}[i][k]+ D^{k-1}[k][j])$$

   This can be graphically depicted as:



   Therefore,
   a. there is only one intermediate vertex v_k, as showed in the graph above
   b. there is only one intermediate vertex v_{k-1} as showed in the equation above, the problem of *k* is computed from the problem of *k-1*
   c. the vertex v_k is not an intermediate vertex, as D^k is computed from D^{k-1}
   d. none of the above

9. Which of the followings is true for the Travelling Salesperson Problem (TSP)?
    a. TSP is an optimization problem for finding an optimal tour
    b. TSP is a shortest path problem for finding a path of minimum length
    c. TSP is a minimum spanning tree problem for finding a spanning tree with minimum weight
    d. both *a&b* (above)

10. Dynamic programming approach can be applied to which of the followings?
    a. The Travelling Salesperson Problem and Knapsack Problem
    b. The computation of Fibonnacci numbers and Binomial Coefficient
    c. The Backtracking Problem and the Branch-and-Bound Problem
    d. Both *a&b* (above)

11. Compare Greedy approach with Dynamic Programming approach
    a. Both are often used to solve optimization problem
    b. Both apply the Principle of Optimality
    c. Both divide an instance into smaller instances
    d. Greedy approach only determines local optimal values, Dynamic Programming approach determines global optimal values

12. Branch-and-Bound and Backtracking reduce the portion of the state space tree by _____.
    a. iteration and recursion, respectively
    b. bound and backtracking, respectively
    c. traversing
    d. pruning

Section II: Short Answer [24 marks]

1. Give the tree of recursive calls when using the *mergesort* algorithm provided below to sort the following list:

   102 14 196 36 120 10, 50, 60 [3 marks]

Problem: Sort *n* keys in nondecreasing sequence.
Inputs: positive integer *n*, array of keys *S* indexed from 1 to *n*.
Outputs: the array *S* containing the keys in nondecreasing order.

```
void mergesort (int n, keytype S[])
{
    if (n > 1){
        const int h = ⌊n/2⌋, m = n – h;
        keytype U[1...h], V[1..m];
        copy S[1] through S[h] to U[1] through U[h];
        copy S[h+1} through S[n] to V[1] through V[m];
        mergesort (h, U);
        mergesort (m,V);
        merge (h, m, U, V, S);
    }
}
```

Problem: Merge two sorted arrays into one sorted array.
Inputs: positive integers *h* and *m*, array of sorted keys *U* indexed from 1 to *h*, array of sorted keys *V* indexed from 1 to *m*.
Outputs: an array *S* indexed from 1 to *h+m* containing the keys in *U* and *V* in a single sorted array.

```
Void merge (int h, int m, const keytype U[], const keytype V[], keytype S[])
{
    index i, j, k;
    i = 1; j = 1; k = 1;
    while (i <= h && j <= m){
        if (U[i] < V[j]) {
            S[k] = U[i];
            i++;
        }
        else {
            S[k] = V[j];
            j++;
        }
        k++;
    }
    if (i > h)
        copy V[j] through V[m] to S[k] through S[h+m];
    else
        copy U[i] through U[h] to S[k] through S[h+m];
}
```

2. Consider the following two algorithms: [3 marks]

Algorithm A

Problem: Sort n keys in nondecreasing sequence.
Inputs: positive integer n, array of keys S indexed from 1 to n.
Outputs: the array S containing the keys in nondecreasing order.

```
void mergesort (int n, keytype S[])
{
    if (n > 1){
        const int h = ⌊n/2⌋,  m = n – h;
        keytype U[1...h], V[1..m];
        copy S[1] through S[h] to U[1] through U[h];
        copy S[h+1} through S[n] to V[1] through V[m];
        mergesort (h, U);
        mergesort (m,V);
        merge (h, m, U, V, S);
    }
}
```

Problem: Merge two sorted arrays into one sorted array.
Inputs: positive integers h and m, array of sorted keys U indexed from 1 to h, array of sorted keys V indexed from 1 to m.
Outputs: an array S indexed from 1 to $h+m$ containing the keys in U and V in a single sorted array.

```
Void merge (int h, int m, const keytype U[], const keytype V[], keytype S[])
{
    index i, j, k;
    i = 1;  j = 1;  k = 1;
    while (i <= h && j <= m){
        if (U[i] < V[j]) {
            S[k] = U[i];
            i++;
        }
        else {
            S[k] = V[j];
            j++;
        }
        k++;
    }
    if (i > h)
        copy V[j] through V[m] to S[k] through S[h+m];
    else
        copy U[i] through U[h] to S[k] through S[h+m];
}
```

Algorithm B

Problem: Sort *n* keys in nondecreasing order.
Inputs: positive integer *n*, array of keys *S* indexed from 1 to *n*.
Outputs: the array *S* containing the keys in nondecreasing order.

```
void quicksort (index low, index high)
{
    index pivotpoint;
    if (high > low){
        partition(low, high, pivotpoint);
        quicksort(low, pivotpoint-1);
        quicksort(pivotpoint+1, high);
    }
}
```

Problem: Partition the array *S* for Quicksort.
Inputs: two indices, *low* and *high*, and the subarray of *S* indexed from *low* to *high*.
Outputs: *pivotpoint*, the pivot point for the subarray indexed from *low* to *high*.

```
void partition (index low, index high,
                index& pivotpoint)
{
    index i, j;
    keytype pivotitem;

    pivotitem = S[low];
    j = low;
    for (i = low + 1; i <= high; i++)
        if (S[i] < pivotitem){
            j++;
            exchange S[i] and S[j];
        }
    pivotpoint = j;
    exchange S[low] and S[pivotpoint];
}
```

a. Which of the algorithm(s) uses the divide-and-conquer approach? [1 mark]

b. We have learned that, *"In the Divide step, a Divide-and-Conquer algorithm divides an instance of a problem into smaller instance(s). In the Conquer step, each of the smaller instances are solved. We can use recursion until smaller instance is sufficient small."*
In the algorithm(s) you have identified in point *a* above, underline the instruction(s) used to check if the smaller instance is sufficient small. [2 marks]

3. In a competition, Alice was tasked to i) use Huffman's algorithm to encode a text file; and ii) use the resulting binary code to decode some encoded files. Suppose the text file to be encoded has the following character frequencies: [3 marks]

Character	Frequency
a	10
e	12
g	20
r	20

Which of the three encoded files below can be decoded using the Huffman encoding Alice can obtain from the given file above? Justify your answer.

FileA	0010011100
FileB	00110110111
FileC	11011000101

4. Algorithm A and Algorithm B have been designed to solve the same problem. The time complexity for Algorithm A is $T(n)= 200n$ and for Algorithm B is $T(n)=2n^2$.

   a. Which algorithm would be more efficient, eventually? Use the value of n to show when this algorithm can begin to demonstrate that it has better algorithmic efficiency. [1 marks]

   b. To run the algorithm you identified in point *a* (above) on a computer that can perform 10^4 basic operations in one second, what would be the largest problem size that can be solved in two minutes? [2 marks]

5. One way to solve the 0-1 Knapsack Problem is as follows:

For $i > 0$ and $w > 0$, let $P[i][w]$ be the optimal profit obtained when choosing items only from the first i items, such that the total weight less than or equal to w. Then,

$$P[i][w] = \begin{cases} P[i-1][w] & , if\ w_i > w \\ maximum(P[i-1][w], p_i + P[i-1][w-w_i]), & if\ w \geq w_i \end{cases}$$

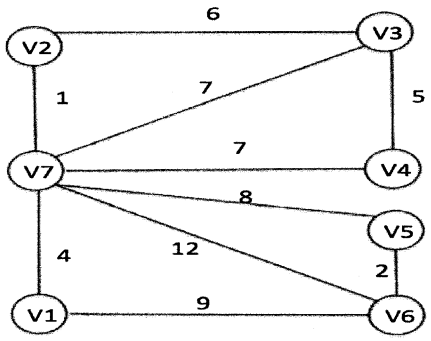
can be used to describe three different cases and compute a solution to the 0-1 Knapsack Problem.

a. In which case(s) the Principle of Optimality applies? Justify your answer. [2 marks]

b. When an item is not considered, which of the three cases given in the equation (above) applies? [1 mark]

6. Give a simple example in which the Greedy approach cannot be applied to find the optimal solution. Justify your answer. [3 marks]

7. Consider the following graph:



a. List the sequence of vertices and edges added using Prim's algorithm to construct a minimum spanning tree. Choose *v1* as the starting vertex. [2 marks]

b. List the sequence of edges added using Kruskal's algorithm to construct a minimum spanning tree. [1 mark]

8. Consider the following 0-1 Knapsack problem: [3 marks]

item 1: $20, 2kg
item 2: $30, 5kg
item 3: $35, 7kg
item 4: $12, 3kg
item 5: $3, 1kg
knapsack maximum capacity: 8kg

Show only the first two levels of the state space tree using a Best-First Search with Branch-and-Bound approach to solve the above 0-1 knapsack problem.

Section III: Problem Solving

1. Suppose there are three services S1, S2 and S3 for event E2. The service time for S1, S2 and S3 are 6, 8 and 5, respectively. Evelyn needs to use these three services for attending E2. Since before E2 there will be an Event E1 that Evelyn needs to attend and there will be limited time for her to travel from the E1 venue to E2 venue. Evelyn will need to get these three services done one by one. To avoid delays, she wants to ask the service providers to arrive at the E2 venue when E1 finishes. The service providers charge the wait time in addition to their actual work done.

   Write an algorithm to help Evelyn to effectively schedule these three services so that she does not need to spend unnecessary money. Your algorithm should also be able to help Evelyn for any other job schedules when jobs need to be done one by one. [5 marks]

2. You are tasked to determine a minimum spanning tree for a given graph by using an algorithm that is at least as good as $O(n^2)$. Which of the following two algorithms would you use? Justify your answer.

[4 marks]

Prim's Algorithm

Problem: Determine a minimum spanning tree

Inputs: integer n>=2, and a connected, weighted, undirected graph containing n vertices. The graph is represented by a two-dimensional array W, which has both its rows and columns indexed from 1 to n, where $W[i][j]$ is the weight on the edge between the ith vertex and the jth vertex.

Outputs: set of edges F in a minimum spanning tree for the graph

```
void prim (int n, const number W[][], set_of_edges& F){
   index i, vnear;
   number min;
   edge e;
   index nearest[2..n];
   number distance[2..n];

   F=∅;
   for (i = 2; i <= n; i++){
      nearest[i]=1;
      distance[i] = W[1][i];
   }

   repeat (n-1 times){
      min = ∝;
      for (i = 2; i <= n; i++)
         if (0 <= distance[i] < min){
            min = distance[i];
            vnear = i;
         }
      e = edge connecting vertices indexed by vnear and nearest[vnear];
      add e to F;
      distance[vnear] = -1;
      for (i = 2; i <= n; i++)
         if (W[i][vnear] < distance[i]){
            distance[i] = W[i][vnear];
            nearest[i] = vnear;
         }
   }
}
```

Kruskal's Algorithm

Problem: Determine a minimum spanning tree

Inputs: integer *n>=2*, positive integer *m*, and a connected, weighted, undirected graph containing *n* vertices and *m* edges. The graph is represented by a set *E* that contains the edges in the graph along with their weights.

Outputs: *F*, a set of edges in a minimum spanning tree

```
void kruskal (int n, int m, set_of_edges E, set_pf_edges& F){
   index i, j;
   set_pointer p,q;
   edge e;

   Sort the m edges in E by weight in nondecreasing order;
   F=∅;
   initial(n); //initializes n  disjoint subsets, each of which contains exactly one of the indices
               //between 1 and n

   while (number of edges in F is less than n-1){
      e = edge with least weight not yet considered;
      i,j = indices of vertices connected by e;
      p = find(i); //makes p point to the set containing index i
      q = find(j);
      if (! equal(p,q)){  // equal(p,q) returns true if p and q both point to the same set
         merge(p,q);  //merges the two sets, to which p and q point, into the set
         add e to F;
      }
   }
}
```

Extra Working Page 1

Extra Working Page 2

Extra Working Page 3

THE END