CURTIN UNIVERSITY

School of Electrical Engineering, Computing and Mathematical Sciences
Computing Discipline

Test 2 – S1/ 2018

SUBJECT: Design and Analysis of Algorithms Unit Code COMP3001

TIME ALLOWED:

55 minutes test. The supervisor will indicate when answering may commence.

AIDS ALLOWED:

To be supplied by the Candidate:	Nil
To be supplied by the University:	Nil

Calculators are NOT allowed.

GENERAL INSTRUCTIONS:

This paper consists of Two (2) questions with a total of 50 marks.

ATTEMPT ALL QUESTIONS

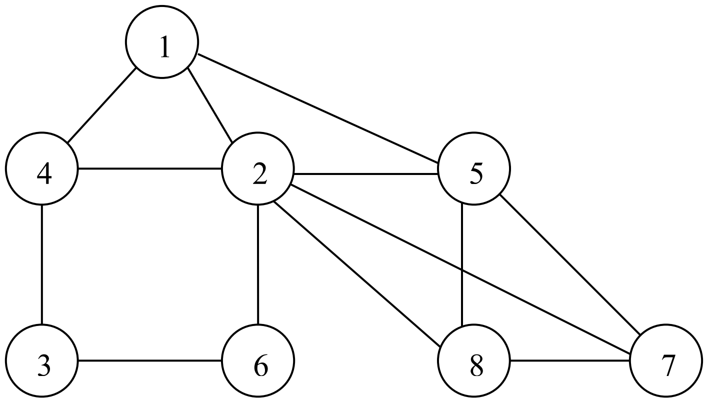
Name: ____

Student No: ____

Tutorial Time/Tutor: ___________________________________

QUESTION ONE (total: 22 marks): Graph and Heap

a) **(Total: 10 marks).** Consider the following undirected graph, and a depth-first-search algorithm to answer the following questions. Note that the graph is represented in an adjacency list.



Procedure DFS_Tree_G(V,E)
Input: G = (V,E) in adjacency list format;. x refers to the value on top of stack; L[x] refers to the adjacency list of x.
Output : The DFS tree T

```
1. Mark all vertices “new” and set T ← {0}
2. Mark any one vertex v ← old
3. push (S, v)
4. while S is nonempty do
5.     while exists a new vertex w in L[x]
6.             T ← T ∪ (x, w)
7.             w ← old
8.             push w onto S
9.     pop S
```

(i) **(4 marks).** Analyse the time complexity of DFS_Tree_G(V, E). State your answer in Big Oh.

(ii) **(4 marks).** Draw the depth-first-search tree of the graph. Assume the root of the tree is node 1, and vertices are always pushed on the stack in increasing order when possible.

(iii) **(2 marks).** Show the contents of stack S to solve part (ii) when it contains the maximum number of nodes.

Answer:

(i)

(ii) Depth First Search tree

(iii)

b) **(Total: 12 marks).** Consider a max-heap A = <16, 12, 14, 11, 9, 6, 8, 5, 4, 7>.

(i) **(2 marks).** Draw the binary tree representation of array A.

(ii) **(2 marks).** Is the binary tree in (i) a leftist tree? Justify your answer.

(iii) **(8 marks).** Consider the following HEAP-XXX-KEY function to answer each of the following four questions.

```
// Assume A[1 … n].
HEAP-XXX-KEY (A, i, key)

1.  if key < A[i]
2.      then error "error message here … "
3.  A[i] ← key
4.  while i >1 and A[PARENT(i)] < A[i]
5.      do exchange A[i] ↔ A[PARENT(i)]
6.         i ← A[PARENT(i)]
```

- Show the contents of array A = <16, 12, 14, 11, 9, 6, 8, 5, 4, 7> after running the function with i = 8 and *key* = 13. **Hint.** You may consider looking at the binary tree representation of the array.

- Describe what the function does.

- What is the worst-case time complexity of the function? What is the worst-case scenario?

- What is the best-case time complexity of the function when the input is valid? What is the best-case scenario?

<u>**Answer:**</u>

(i)

(ii)

(iii)

- Contents of *A* =

- What does the function do?

- Worst case:

- Best case:

END OF QUESTION ONE

QUESTION TWO (total: 28 marks): Greedy Algorithms

a) **(Total: 10 marks).** The owner of a resort is trying to rent out the resort with a goal of **either to rent it out to as many clients as possible or to maximize rental income.** Several rental applications have been received shown as follows. Each application includes the schedule showing when the rental starts and when it ends (based on week number).

(0, 5), (3, 5), (0, 6), (5, 7), (3, 8), (5, 9), (6, 10), (8, 11), (8, 12), (0, 14), (12, 14)

Note that applications (0, 5) and (5, 7) can both be accepted (called **compatible** applications, while (0, 5) and (3, 5) are **not compatible**, i.e., only one of them can be accepted.

(i) **(6 marks).** Suppose the owner wants to rent out the resort to **as many clients as possible**.

- Design a greedy algorithm to select the set of applications that should be accepted. You have to specify your greedy criterion; e.g., for minimum cost spanning tree, its greedy criterion is links with the minimum cost.
- For the applications given above, show in details how your algorithm produces maximum number of applications.
- What is the time complexity of your algorithm? Justify your answer.

Note: you are not asked / required to re-design / re-write any algorithm if you think you can use an existing algorithm that we have discussed in the lecture or available in the attachment provided at the end of this test paper.

(ii) **(4 marks).** Suppose the owner wants to rent out the resort for $500/week, and the goal is to **maximize rental income**.

- Does your solution in part (i) give the maximum rental income? Justify your answer by computing the rental income for your solution in part (i) and compare it with the maximum possible rental income from the applications.

- In general, the solution in part (i) cannot be used to produce compatible applications that give the maximum rental income. Describe one other possible greedy criterion for an algorithm to select compatible applications that give the maximum rental income.

Note: you are not asked to design any algorithm for this question.

Answer:

(i)

(ii)

b) **(Total: 16 marks).** Consider the following Prim's algorithm.

```
1. Insert all v in V into a priority queue Q, each with key of infinity. key[v] is the
   weight of the edge connecting v to the MCST

2.  While Q is not empty
3.      Take out u, the min key vertex, from Q
4.      For each edge (u,v)
5.          If weight of (u, v) < key[v]
6.              Record u as parent of v
7.              Decrease key to weight
8. The final tree is the set of edges { (u, parent[u]) }
```

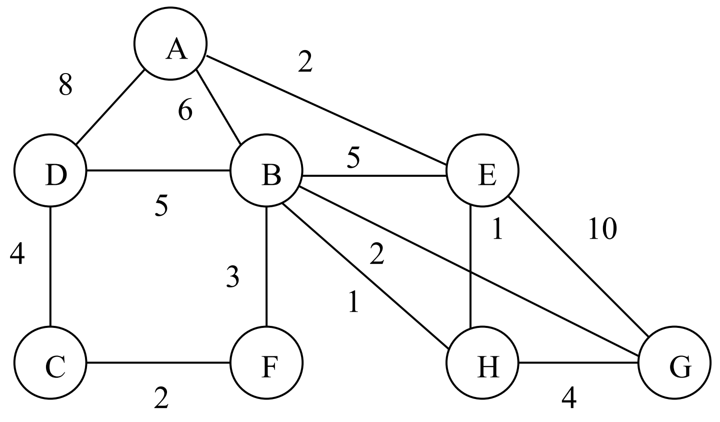
(i) **(2 marks).** Explain the main difference between Prim's algorithm and Kruskal's algorithm.

(ii) **(2 marks).** Does Prim's algorithm need to check for a cycle like in Kruskal's algorithm? Justify your answer by showing which part of the code checks for a cycle or why the algorithm does not need to check for a cycle.

(iii) **(4 marks).** Consider the Prim's algorithm is implemented using a binary heap. What is the time complexity of each of the following lines? **Explain your answers.**

- Line 1?
- Line 3?

(iv) **(8 marks).** Find a minimum cost spanning tree of the following graph using the Prim's algorithm. You have to show the contents of *Q*, where each element of *Q* is a triple (vertex, key[vertex], parent[vertex]). Follow alphabetical order when necessary. Show the spanning tree and its cost.



Answer:

(i)

(ii)

(iii)

- Line 1?

- Line 3?

(iv) MCST using Prim's algorithm

Initial contents of Q

Q: (A, ∞,_) , (B, ∞,_) , (C, ∞,_) , (D, ∞,_) , (E, ∞,_) , (F, ∞,_), (G, ∞,_), (H, ∞,_)

Choose	**The contents of Q**
A	

MCST:

Cost =

END OF QUESTION TWO

Attachment

GREEDY_ACTIVITY_SELECTOR (*s*, *f*)

```
1. $n \leftarrow length[S]$
2. $A \leftarrow \{1\}$
3. $j \leftarrow 1$;
4. for $i \leftarrow 2$ to $n$
5.     do if $s_i \geq f_j$
6.         then $A \leftarrow A \cup \{i\}$
7.             $j \leftarrow i$
8. return $A$
```

BFS_Tree_G(*V*,*E*)
Input: $G = (V, E)$. $L[x]$ refers to the adjacency list of x.
Output: The BFS tree T;

```
1. Mark all vertices new and set $T = \{ \}$
2. Mark the start vertex $v$ =old
3. insert ($Q$, $v$) // $Q$ is a queue
4. while $Q$ is nonempty do
5.     $x$ = dequeue ($Q$)
6.     for each vertex $w$ in $L[x]$ marked new do
7.         $T = T \cup \{x,w\}$
8.         Mark $w$ = old
9.         insert ($Q$,$w$)
```

BUILD-MIN-HEAP (*A*)
Input: An array A of size n= *length*[A]; *heap_size*[A]
Output: A min-heap of size n

```
1.     heap_size[$A$] $\leftarrow$ length[$A$]
2.     for $i \leftarrow \lfloor length[A]/2 \rfloor$ downto 1
3.         do MIN-HEAPIFY($A$, $i$)
```

MIN-HEAPIFY (*A*, *i*)

```
1. $l \leftarrow$ LEFT_CHILD ($i$)
2. $r \leftarrow$ RIGHT_CHILD ($i$)
3. if $l \leq heap\_size[A]$ and $A[l] < A[i]$
4.     then smallest $\leftarrow l$
5.     else smallest $\leftarrow i$
6. if $r \leq heap\_size[A]$ and $A[r] < A[smallest]$
7.     then smallest $\leftarrow r$
8. if smallest $\neq i$
9.     then exchange $A[i] \leftrightarrow A[smallest]$
10.         MIN-HEAPIFY ($A$, smallest)
```

HEAP_EXTRACT_MIN (*A*[1…*n*])

```
1. if heap_size[A] ≥ 1 then
2.     min ← A[1];
3.     A[1] ← A[heap_size[A]];
4.     heap_size[A] ← heap_size[A]-1;
5.     MIN-HEAPIFY(A, 1)
6.     return min
```

HEAP_INSERT (*A*, *key*)

```
1.     heap_size[A] ← heap_size[A]+1;
2.     i ← heap_size[A];
3.     while i > 1 and A[PARENT(i)] > key
4.         A[i] ← A[PARENT(i)];
5.         i ← PARENT(i);
6.     A[i] ← key
```

For a node with *index i*:
PARENT(*i*) is the *index* of the parent of *i*
LEFT_CHILD(*i*) is the *index* of the left child of *i*
RIGHT_CHILD(*i*) is the index of the right child of *i*

END OF TEST PAPER