

Student Number	
Last Name, First Name	
Section (001 or 002)	

QUEEN'S UNIVERSITY FINAL EXAMINATION
FACULTY OF ENGINEERING AND APPLIED SCIENCE
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ELEC 278 001 – Prof. Zou and ELEC278 002 – Prof. Athersych
December 20, 2017

INSTRUCTIONS TO STUDENTS:

This examination is **3 HOURS** in length. There are **TEN** sections (problems) on this examination. Please answer all questions on the exam paper. No aids are allowed. Put your student number on all pages of all pages, including the front. **GOOD LUCK!**

PLEASE NOTE:

Proctors are unable to respond to queries about the interpretation of exam questions.
Do your best to answer exam questions as written.

This material is copyrighted and is for the sole use of students registered in ELEC278 and writing this exam. This material shall not be distributed or disseminated. Failure to abide by these conditions is a breach of copyright and may also constitute a breach of academic integrity under the University Senate's Academic Integrity Policy Statement.

Question 1 [20 Marks] Algorithm Analysis		Question 6 [15 Marks] Heap	
Question 2 [20 Marks] Linked List		Question 7 [15 Marks] Hash Table	
Question 3 [20 Marks] Stack Processing		Question 8 [10 Marks] Quicksort	
Question 4 [10 Marks] Binary Search Tree		Question 9 [15 Marks] Design of Algorithm	
Question 5 [20 Marks] AVL Trees		Question 10 [20 Marks] Graphs	
TOTAL [165 Marks]			

Question 1. Algorithm Analysis [20 Marks]

Part1. [3 Marks] Define what it means for a function $f(n)$ to be $O(g(n))$

Part2. [3 Marks] The function $f(n) = 3n^2 - 3n + 4$ and $g(n) = n \log_2 n + 10$ is $g(n) = O(n \log_2 n)$. Using only the definition in **Part1**, show that $f(n) + g(n) = O(n^2)$

Part3. [5 Marks] In the following code, estimate the running time of function $sum(n)$ using Big-Oh notation. Please use repetitive substitution to demonstrate how you derive the running time estimation.

```
int sum(int n)
{
    if (n!=0)
        return n + sum(n-1); // sum() function calls itself
    else
        return n;
}
```

Part4. [9 Marks] Consider the following code fragments. Assume that the worst running time for function $F(n)$ is $O(1)$ and it returns a value between 1 and $n + 100$.

CODE 1:

```

1.    sum = 0;
2.    for (i=0; i<= n; ++i)
3.        for (j=0; j< F(n); j++)
4.            sum = sum + 1;

```

Compute the worst case running time for CODE 1 by filling the Table 1 using the Big Oh notation for CODE 1.

Statement	Running Time in Big-Oh Notation
1: <u>line 1</u> , $sum = 0$	
2.a: <u>line 2</u> , the initialization code of the for-statement (i.e., $i=0$)	
2.b: <u>line 2</u> , the termination check for the for-statement (i.e., $i \leq n$)	
2.c: <u>line 2</u> , the increment code for the for-statement (i.e., $i++$)	
3.a: <u>line 3</u> , the initialization code of the for-statement (i.e., $j=0$)	
3.b: <u>line 3</u> , the termination check for the for-statement (i.e., $j < F(n)$)	

3.c: <u>line 3</u> , the increment code for the for-statement (i.e., $j++$)	
4: <u>line 4</u> , $sum = sum + 1$	
Total running time for the entire code	

Question 2. Linked list [20 Marks]

A polynomial function, i.e., $f(x) = c_n x^n + c_{n-1} x^{n-1} + \dots + c_1 x^1 + c_0 x^0$, can be implemented using a linked list data structure as was done in assignment 1. $n \geq 0$ is the *degree* of a polynomial.

You can use the following data structure to represent a term in a polynomial function.

```
typedef struct node
{
    double coef; // a coefficient of a term, e.g., ci is the efficient for cixi
    int exp;      // the exponent for a term, e.g., i is the ex for cixi
    struct node* next;
} Node;
```

You can use the following data structure to represent a polynomial function.

```
typedef struct LinkedList {
    Node *head; // point to the term with highest degree in the polynomial function
} LinkedList;
```

A set of functions can be implemented for calculating the result of a polynomial function for a given x. The following implementation of functions is provided.

```
// Initialize a LinkedList structure
LinkedList* init(LinkedList *l)
{
    l=(LinkedList *)malloc (sizeof(LinkedList));
    l->head = NULL;
    return l;
}

// Check if a LinkedList is empty. It is empty if head pointer is NULL
int isEmpty(LinkedList *l)
{
    return ( l==NULL || l->head == NULL);
}

// Delete the entire Linkedlist by repeatedly deleting first Node
void removeElements(LinkedList* l)
{
    if (!isEmpty (l)) {
        // List is not empty, so there is a first Node to delete.
        Node *ptr = l->head;

        // head of list is now element after first
        l->head = l->head->next;

        // discard node we no longer need
        free(ptr);
    }
}
```

```
    }
    l->head=NULL;
}

//Free linkedlist
void destroy(LinkedList* l)
{
    removeElements(l);
    free(l);
}

// Create a new Node and initialize its value and next pointer
Node *createNode (int _exp, int _coef, Node *nxt)
{
    Node *pNode = (Node *)malloc (sizeof(Node));

    if (pNode != NULL)
    {
        pNode->exp = _exp;
        pNode->coef = _coef;
        pNode->next = nxt;
    }
    return pNode;
}
```

Part1. [10 Marks] Write C code for an **add()** function, which inserts a new term into an existing linked list in the correct position. The terms in the polynomial linked list follow the descending order of the exponents, i.e., the head points to the term with the largest exponent.

```
// l is the pointer to an existing polynomial, the new term will be added to l
// _exp is the exponent value for the new term to be added
// _coef is the coefficient value for the new term to be added

void add (LinkedList* l, int _exp, int _coef)
{

```


Part2. [10 Marks] Implement a *plus* function() which takes two polynomial linked lists as input and updates a result polynomial that represents the sum of the two given polynomial linked lists.

//l1 and l2 are two given polynomials

//l3 keeps the result of the sum of two given polynomials, i.e., l1 and l2

```
void plus(LinkedList* l1, LinkedList* l2, LinkedList* l3)
{
```

```
}
```

Question 3. Stack processing [20 marks]

Suppose you are working with integer stacks. The stack is described by a data structure as follows:

```
struct stack {
    int    tos;    // index of the top-of-stack location
    int    *stk;   // pointer to array holding actual integer values in stack
    int    max;    // actual size of the array pointed to by stk.
};
```

Part 1. [15 marks] Implement an integer stack using the data structure above. The functions you shall write are:

struct stack *createstack (int size) – creates a stack data structure with an array big enough to hold size items. If the stack created successfully, returns a pointer to the stack data structure; otherwise returns NULL.

int isempty (struct stack *s) - returns 1 if stack is empty; 0 if stack not empty.

int isfull (struct stack *s) - returns 1 if stack is full; 0 if stack is not full.

int push (struct stack *s, int x) – pushes the value x onto the stack pointed to by s. Returns 0 if the push was successful; returns -1 if there was a problem (such as full stack).

int pop (struct stack *s, int *px) - pops top value from the stack pointed to by s and stores the value in the integer pointed to by px. Returns 0 if the pop was successful; returns -1 if there was a problem (such as an empty stack).

```
struct stack *createstack (int size)
{
```

```
}
```

```
int isempty (struct stack *s)
{
```

```
}
```

```
int isfull (struct stack *s)
{
```

```
}
```

```
int push (struct stack *s, int x)
{
```

```
}
```

```
int pop (struct stack *s, int *px)
{
```

```
}
```

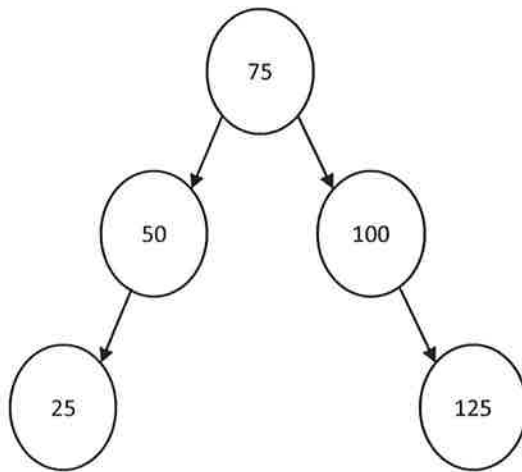
Part 2. [4 Marks] Now suppose you have an existing stack pointed to by a pointer called **pstack**. This stack contains data that includes both positive integers (for our purposes, integers ≥ 0) and negative integers (< 0). You don't know how many integers are actually in the stack, but you do know that the stack is big enough to hold a maximum of 100 integers. You are asked to implement a separating mechanism as follows, using the functions defined for Part 1: (You may assume those functions exist in order for you to write your answer to this part – even if you didn't finish the previous part.)

- Create two new stacks called **positives** and **negatives**.
- Pop values from the original stack one-by-one, and if the number is positive, push it on the **positives** stack. If the number is negative, push it on the **negatives** stack.
- Once all the values are taken from the original, now take all the values on the **negatives** stack and push them on the original stack, then take all the values on the **positives** stack and push them onto the original stack.

Part 3. [1 Mark] What does the process in Part 2 accomplish? (One brief sentence.)

Question 4. Binary Search Tree. [10 Marks]

Suppose you have the following Binary Search Tree:



Part 1. [4 Marks] Draw a diagram showing the final binary search tree after the following values have been inserted into the tree: 21 50 55 60 190 18 17 115

Part 2. [2 Marks] If the tree values were printed using **PostOrder** traversal, what would be printed?

Part 3. [2 Marks] If the tree values were printed using **InOrder** traversal, what would be printed?

Part 4. [2 Marks] If the tree values were printed using **Breadth-First** traversal, what would be printed?

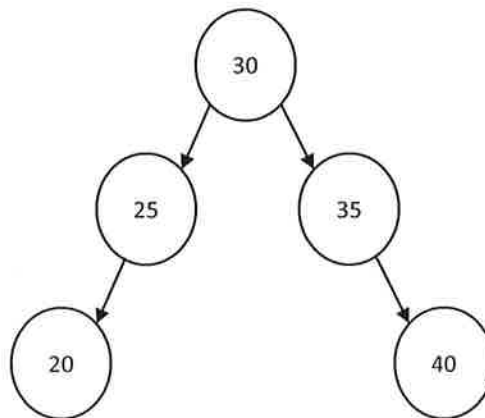
Question 5. AVL Tree [20 Marks]

Part 1. [2 Marks] Both Binary Search Trees and AVL Trees are used to store data that can be searched. What are the advantages of an AVL tree over a BST?

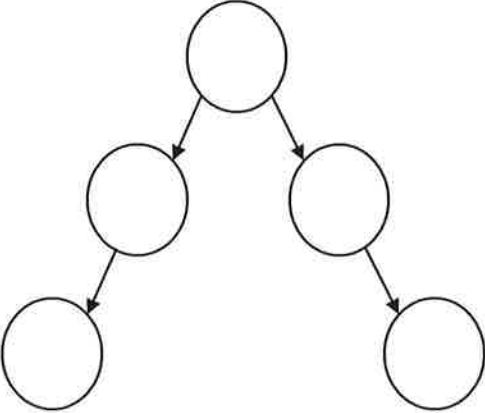
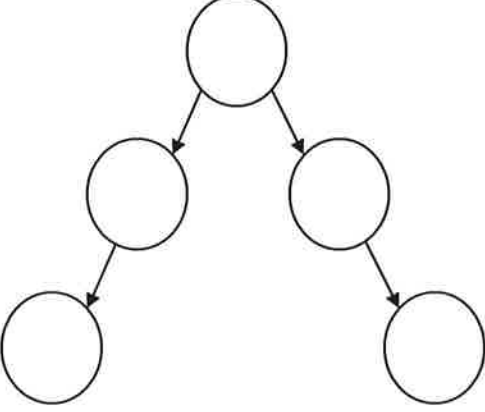
Part 2. [2 Marks] Using a Big-O notation, compare the worst case search times for an AVL tree and a BST, and the worst case insert time for an AVL tree and BST.

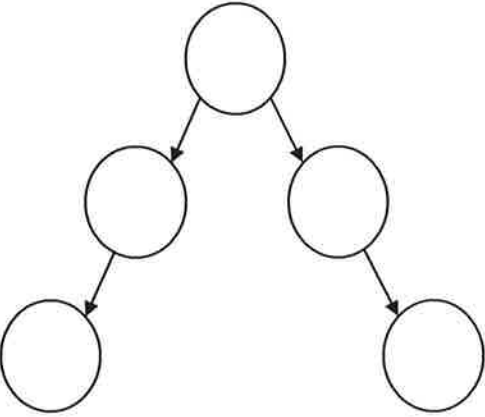
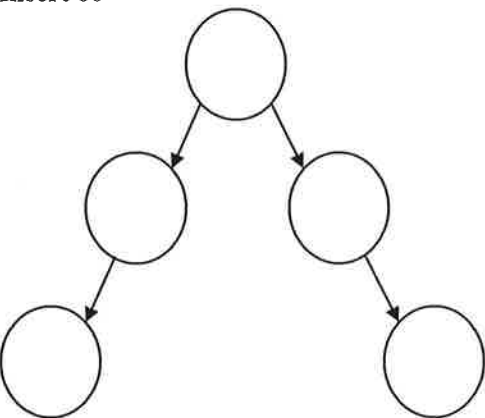
Part 3. [16 Marks] An existing AVL tree is shown below. The charts following the AVL tree are where you will record information about each insert to the tree. An incomplete blank tree is given to make it easier to record your answer – but in all cases you will have to add more nodes to the tree and perhaps move values by rotations. At the top left of each box is the number that will be added to the tree at each step. Insert the integers into the tree, performing any necessary rotation. In the boxes on the right, indicate which node caused the unbalancing and which of the 4 possible rotations was applied.

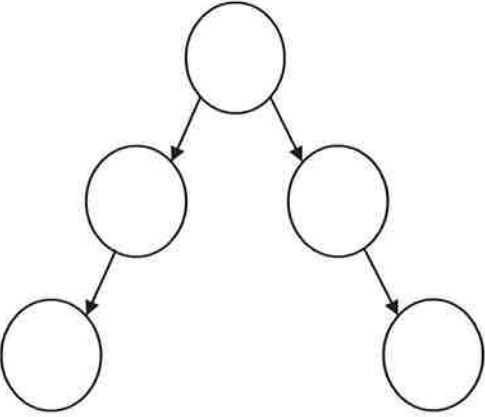
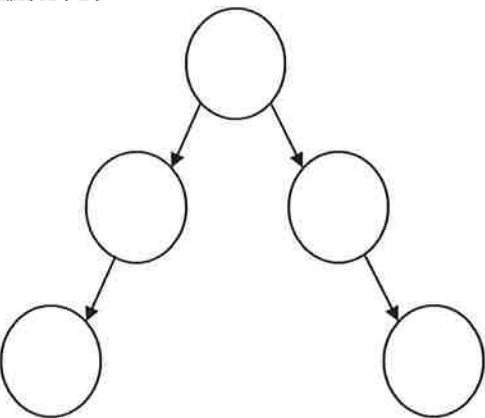
NOTE: An incorrect insertion and rebalancing at any step may result in the loss of marks for subsequent steps.

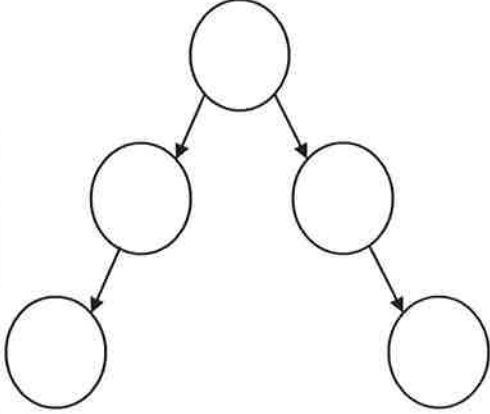
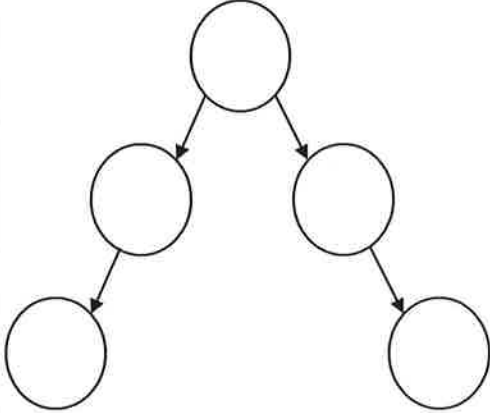


(Chart begins next page.)

Insert 21 	<table border="1"><tr><td data-bbox="1203 245 1378 327">Unbalanced Node</td></tr><tr><td data-bbox="1203 327 1378 409"> </td></tr><tr><td data-bbox="1203 409 1378 491">Rotation Name</td></tr><tr><td data-bbox="1203 491 1378 951"> </td></tr></table>	Unbalanced Node		Rotation Name	
Unbalanced Node					
Rotation Name					
Insert 50 	<table border="1"><tr><td data-bbox="1203 951 1378 1033">Unbalanced Node</td></tr><tr><td data-bbox="1203 1033 1378 1115"> </td></tr><tr><td data-bbox="1203 1115 1378 1197">Rotation Name</td></tr><tr><td data-bbox="1203 1197 1378 1837"> </td></tr></table>	Unbalanced Node		Rotation Name	
Unbalanced Node					
Rotation Name					

Insert 5 	Unbalanced Node
	Rotation Name
Insert 60 	Unbalanced Node
	Rotation Name

Insert 19 	Unbalanced Node
	Rotation Name
Insert 18 	Unbalanced Node
	Rotation Name

Insert 17 	<table border="1"><tr><td data-bbox="1209 254 1377 323">Unbalanced Node</td></tr><tr><td data-bbox="1209 323 1377 392"></td></tr><tr><td data-bbox="1209 392 1377 470">Rotation Name</td></tr><tr><td data-bbox="1209 470 1377 970"></td></tr></table>	Unbalanced Node		Rotation Name	
Unbalanced Node					
Rotation Name					
Insert 15 	<table border="1"><tr><td data-bbox="1209 982 1377 1052">Unbalanced Node</td></tr><tr><td data-bbox="1209 1052 1377 1121"></td></tr><tr><td data-bbox="1209 1121 1377 1661">Rotation Name</td></tr><tr><td data-bbox="1209 1661 1377 1759"></td></tr></table>	Unbalanced Node		Rotation Name	
Unbalanced Node					
Rotation Name					

Question 6. Heap. [15 Marks]

Suppose you have the following integer values stored in an array:

32 4 56 71 23 93 42 67 37 31 8 87.

Part 1. [2 Marks] Draw a diagram to show what this array represents as a **complete binary tree**.

Part 2. [7 Marks] Show the contents of this array after it has been heapified – turned into a maxheap.

--	--	--	--	--	--	--	--	--	--	--	--

Part 3. [1 Mark] The implementation of a heap includes a function to remove a value from the heap. What value would be removed if this function was applied to this maxheap?

Part 4. [5 Marks] The heap needs to be repaired after the removal. Show the contents of this array after the heap has been repaired.

--	--	--	--	--	--	--	--	--	--	--	--

Question 7. Hash Tables [15 Marks]

This question requires that you consider insertions into a hash table. For each part, you will draw a representation of an 11-celled hash table B and its contents after you use the following mapping

$$h(x) = (3x + 5) \bmod 11$$

to insert the elements in the set $\{10, 11, 20, 15, 40, 9, 16, 20\}$ into hash tables in two different collision resolving strategies, as detailed below in Part 1 and Part 2.

Part1. [5 Marks] Construct the hash table by using linear probing

Part2. [5 Marks] Construct the hash table by using separate chaining

Part3. [5 Marks] Rehash the hash table in **Part 2**, so that the load factor, $\lambda = 25\%$. Draw the representation of the rehashed hash table.

Question 8. Quicksort. [10 Marks]

Suppose you have the following values stored in an array:

42	91	50	3	87	22	36	27	9	16	19	82	92	12	32	11	7	94	63	71
----	----	----	---	----	----	----	----	---	----	----	----	----	----	----	----	---	----	----	----

You are going to use Quicksort to sort this data in ascending order (so when complete, the smallest value will be at the left of the array at the smallest index, and the largest value will be at the right). When doing the partition step on a segment of the array, the way to choose the pivot value is to look at the leftmost value, the rightmost value and the middle value and choose the pivot value to be the middle value of those three values. (For example, the left value was 42, the right value was 71 and the middle value is 16, you would choose 42 for your pivot value.) In the table below, show what the array looks like after each of the first 4 partitioning steps. (The original values are repeated at the top of the table for convenience.)

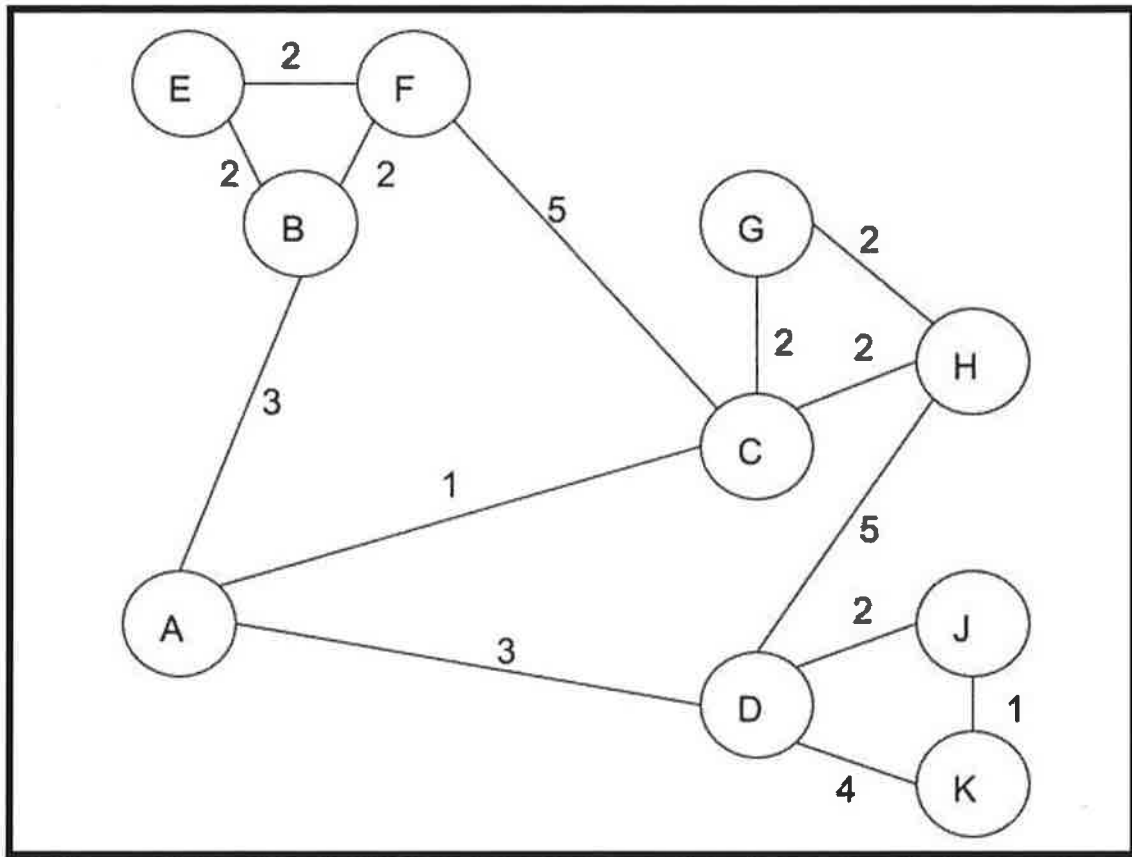
42	91	50	3	87	22	36	27	9	16	19	82	92	12	32	11	7	94	63	71

Question 9: Design of Algorithm [15 Marks]

Suppose you are given an unordered sequence S of n integers in the range $[0, n - 1]$. Write an $O(n)$ time algorithm in C for determining if S contains two elements that are equal to each other.

Question 10. Graphs [20 Marks]

Consider the following graph of a computer network. The vertices are routers, and are labelled {A,B,C,D,E,F,G,H,I,J,K}. The edges have weights that are the time it takes to move a packet from one router to the next (in some tiny time unit). All of the edges are **bi-directional**.



(Questions begin on next page.)

Part 1. [2 Marks] Show an **adjacency matrix** for this graph.

Part 2. [2 Marks] Draw a diagram showing a **spanning tree** starting at vertex A for this graph, showing the shortest paths from A to each of the other vertices in the graph.

Part 3. [16 Marks] Use Dijkstra's algorithm to find the shortest weighted path from starting vertex A to other vertices in the graph. For each pass, complete the following table, showing the information as it is developed about the graph. The table may have too many or too few columns – if too few, stop after you have done pass 8; if too many, stop when you have completed the table. Note that the results of the first pass are already recorded in the table.

In the table, the information in each box is (k,d,p), written as kdp, where:

- k is either + indicating that the vertex has been dealt with, or blank if not;
- d is the weighted distance from the starting vertex to this vertex; an initial value of ∞ is stored in the initial setup;
- p is the predecessor node on which the weighted distance is based. (It will be blank if there is no valid weighted distance yet.)

Vertex	Initial	Pass 1	Pass 2	Pass 3	Pass 4	Pass 5	Pass 6	Pass 7	Pass 8
A	0	+0							
B	∞	3A							
C	∞	1A							
D	∞	3A							
E	∞	∞							
F	∞	∞							
G	∞	∞							
H	∞	∞							
J	∞	∞							
K	∞	∞							