1. In DSA, the quicksort is _____ .
    a. none of the above
    b. **an algorithm**
    c. a data structure of array
2. Which do you think it is the best case (input array) of QuickSort?
    a. None of the above
    b. **1, 2, 3, 4, 5, 6, 7**
    c. 7, 6, 5, 4, 3, 2, 1
3. Given the Original array: 25, 12, 48, 37, 12, 92, 86, 33.

    What is the output result after 1st pass of bubble sort:
    a. 25, 12, 48, 37, 12, 33, 86, 92.
    b. **12, 25, 37, 12, 48, 86, 33, 92.**
    c. 25, 12, 48, 37, 12, 92, 86, 33.
    d. 25, 33, 48, 37, 12, 12, 86, 92.
4. Given this array: 48, 86, 25, 10, 57, 37, 12, 92, 33.

    Which is the pivote of the 1st partition in QuickSort?
    a. 48
    b. 86
    c. 57
    d. 25
5. Which is the correct formular to evaluate the time complexity of QuickSort?
    a. **T(n) = T(n − i) + T(i) + αn**
    b. T(n) = 2T(n/2) + cn
    c. T(n) = 3T(n-1) + αn
6. Any sorting algorithm that moves elements only one position at a time mút have time complexity at least _____.
    a. nlogn
    b. n^2
    a. c. 1 (x)
    c. n
7. Given: T(n) = 3 T(n/2) + n

    What do you think it is the general form of T(n) after 'i' steps of expansion?
    b. $3T(n/(i) + n(1 + 3 + ... + 3^{i+1})$
    c. $6T(n/(2^{i}) + n(1 + 3/2 + ... + 3/(2^{i}))$
    d. $T(n/(4) + n(1 + 3/2 + ... + 3/(2^{i}))$
    e. $3T(n/(2^{i}) + n(1 + 3/2 + ... + 3/(2^{i+1}))$
8. Given: T(n) = 3 T(n/2) + n

How many steps of expansion needed to express T(n) as a polynomial of T (1) or T (2)?

f. $\log_2 n$

g. $\log_3 n$ (x)

h. $\log_2 n + 1$

i. log(n)

9. Given: T(n) = 2 T(3n/2)

How many steps of expansion needed to express T(n) as a polynomial of T (1) or T (2)?

j. $\log_{3/4} n$

k. None of the above

l. $\log_3 n$

m. $\log_{3/2} n$

10. What is the time complexity of the function T(n) if

T(n) = n + T(n - 1)

a. $O(n^2)$

b. O(n)

c. O(log(n)) (x)

11. Max heap is the array of integer that satisfy:

a. Child >= parent

i. Root is the smallest

b. Child <= parent

i. Root is the largest

c. Child => parent

i. Root is the largest

12. Given that d(n) is O(f(n)), e(n) is O(g(n)).

What is the complexity of d(n) * e(n)?

a. O(f + g)

b. None of the above

c. **O(f * g)**

13. Given that d(n) is O(f(n)), e(n) is O(g(n)).

What is the complexity of d(n) + e(n)?

a. Both of the above (v)

b. min(O(f), O(g)

c. **O(f + g)** (x)

14. What is the complexity O(n) of the following code:

```
for (i = 0; i < N; i++)
for (j = 0; j < N * N; j++)
sum++;
```

---

a. N^2
b. 1
c. N
d. **N^3**

15. What is the complexity Ono of the following code:

```
for (i = 0; i < n; i++)
for (j = 0; j < i * i/2; j++)
for (k = 0; k < j; k++)
sum++;
```

---

a. n^2
b. **n^5**
c. n
d. n^4

16. What is the time complexity of Selection sort?
    a. **O(n^2)**
    b. O(1)
    c. O(n/2 +n)
    d. O(n)
17. What is the time complexity of Straight Insertion Sort in best case?
    a. O(n^2)
    b. **O(n)**
    c. 1
    d. O(n^3)
18. What is the time complexity of Straight Insertion Sort in the worst case?
    a. O(n)
    b. **O(n^2)**
    c. O(1)
    d. O(2^n)
19. What is the time complexity of Heapsort?

a. O(1)
b. O(n)
c. O(n^2)
d. O(n log n)

20. Generally in Bubble sort algorithm, how many time of sorting (pass) do we need to obtain the sorted array?
    a. n/2
    b. n-1
    c. O(n^2)
    d. 2n
21. What is the correct order by growth rate of the functions (min -> max):
    n, log(n), $(1/3)^n$, 30
    a. 30, n, $(1/3)^n$, log(n)
    b. 30, $(1/3)^n$, log(n), n
    c. $(1/3)^n$, 30, log(n), n
22. What is the correct order by growth rate of the functions (min -> max):
    n, $(3/2)^n$, 30, n/log(n)
    a. 30, n, n/log(n), $(3/2)^n$
    b. n, n/log(n), $(3/2)^n$, 30
    c. 30, n/log(n), n, $(3/2)^n$
    d. $(3/2)^n$, n, n/log(n), 30
23. In order to calculate the complexity of an algorithm, there are some steps that you should know. Please select the correct order of the step to calculate the complexity.

    A. Perform the mathematical analysis to find the relationship between T(n) and n

    B. Simplify the result of complexity

    C. Derive the mathematical formula of T(n) from the code (or pseudo-code)

    a. C A B
    b. B A C
    c. A C B
24. To solve a big problem of Mr.T, Dummy try to call some people to help. Each person found that their assigned task is quite similar to others' and so they usually ask other if they have the solution for the similar problem. The final solution will be collected and combined by Dummy. So what type of method is best describe the way Dummy used to solve the problem?
    a. FIT student's method
    b. Divide and conquer method

c. **Dynamic programming method**

d. Balancing subproblems method

25. In terms of complexity analysis, _____ is more important (more informative).

a. **worst case**

b. Normal case

c. best case

26. The _____ can't give us an upper bound on performance.

a. Worst case

b. Best case

27. The best case of an algorithm A is a function

BA: N → N where BAno is the _____ number steps performed by A on an input of size n.

a. Not of all the above

b. **minimum**

c. maximum

28. The number of steps of the recursive (devide and conquer) algorithm for the Tower of Hanoi _____ those of the simple non-recursive algorithm described at the textbook!

a. equals

b. is less than

c. is more than

29. Dijkstra algorithm is to find the _____.

a. shortest parth based on number of edges

b. shortest parth based on cloud computing

c. **shortest parth based on edge weight**

30. The runtime complexity of Dijkstra's Algorithm of a connected graph depends mainly on _____ .

a. the operations of graph

b. None of the above

c. the number of Edges and number of Vertices of the graph

31. WHICH PREQUISITE IS NEEDED BY DIJKSTRA ALGORITHM?

a. Non cycles

b. No vertex with more than 4 edges

c. **Non-negative edge weights**

d. Both of the above

32. What is true about stack?

a. It is Last In First Out

b. **Stack is a variation of List.**

c. All of the above

d. It is First In Last Out

33. What is true about stack?
    a. It is First In First Out List
    b. Stack and List have no relation
    c. All of the above
    d. It is Last In First Out List
34. What is true about Queue?
    a. It is a variation of List
    b. All of the above
    c. It is FIFO
35. In Stack, insertions and deletions can be performed at _____ of it.
    a. one end
    b. both head and tail
    c. two termials
36. What happend if we pop() all the elements from a stack and insert() into a queue, after that, we delete() all the elements from the queue and push() them to the Stack?
    a. we have the stack with revert order of element
    b. we have an empty stack
    c. I don't know
    d. we have done a stupid job
37. What happend if we sequentially POP all the elements from a stack and sequentially INSERT into a queue, after that, we DELETE all the elements from the queue and PUSH them back to the Stack?
    a. we have the stack with revert order of element
    b. b. we have an empty stack (x)
    c. I don't know
    d. we have done a stupid job
38. In ADT definition of FIFO queue, what operation cannot be ignored?
    a. IsFull()
    b. Size()
    c. All of the above
    d. Insert() /// Enqueue
39. What operation(s) can be ignored when defining the ADT of Stack?
    a. IsFull()
    b. Pop()
    c. Push()

40. In a Singly linked list that has only one node, if a Nodes (data, pointer) is a head, the pointer points to _____.
    a. head
    b. somewhere in the memory.
    c. **null**
    d. tail
41. In a Circular linked liste, if a Nodes (data, pointer) is a tail, the pointer points to _____.
    a. tail
    b. somewhere in the memory
    c. **head**
    d. null
42. A linear list, for which all insertions and deletions (and usually all accesses) are made at both ends of the list, is called: _____.
    a. **Dequeue**
    b. all-day queue
    c. Singly Linked Queue
    d. None of the above
43. In the definition of abtract data type of a List, isEmpty() is the method which returns _____ value.
    a. true
    b. **boolean**
    c. wrong
    d. void
44. In the definition of abtract data type of a List, remove (int position) is the method which returns _____ value.
    a. integer
    b. string
    c. long
    d. **void**
45. In the definition of abtract data type of a List, size() is the method which returns _____ value.
    a. no
    b. **integer**
    c. yes
    d. void
46. You write a program to find the LARGEST AND SMALLEST elements in an array of n elements. Generally, this can be done in at least _____ comparisions.
    a. n/2

b. b. 2n -1 (x)

c. n - 1

d. 2n - 3

47. What is true about array-based list and reference-based list?

a. reference-based list cannot perform insertion and deletion

b. **array-based list is not as flexible in size as reference-based list**

c. array-based list is more flexible in size than reference-based list

48. What is true about array-based list and reference-based list?

a. array-based list is lower cost of insertation but not deletion.

b. reference-based list can not perform insertion and deletion

c. **for the same problem, reference-based list has larger size than array-based list**.

49. What is true about array-based list and reference-based list?

a. **reference-based list is lower cost of insertation and deletion.**

b. array-based list is completely bad.

c. array-based list is lower cost of insertation but not deletion.

50. What is true about array-based list and reference-based list?

a. **reference-based list is harder to perform lookup operation compared to array-based list**

b. They can be implemented by Java language only.

c. elements of array can be located dinamically and discontinuous like reference-based list

d. reference-based list is an other name of array-based list

51. What is the result of this code below:

```
---------------------------------------
for (int i=1; i<5; i++)
{
for (int j = 1; j < i-5; j++)
{
System.out.print("*");
}
System.out.println();
}
}
```

a. **nothing**

b. //result:

***

**

*

c. //result:

*

**

***

52. What is the result of this code below:

```
----------------------------------------
for (int i=1; i<5; i++)
{
for (int j = 1; j < 5-i; j++)
{
System.out.print("*"); //Similar to print in C
}
System.out.println(); //similar to print new line in C
}
```

a. //result:

***

**

*

b. //result:

******

c. //result:

*

**

***

53. What is the result of this code below:

```
----------------------------------------
for (int i=1; i<5; i++)
{
for (int k=5-i;k<5;k++)
System.out.print("!");
for (int j = 1; j < 5-i; j++)
{
System.out.print("*");
}
System.out.println();
}
```

a. //result:
   *
   **
   ***

b. nothing

c. //result:
   !*
   !!**
   !!!*
   !!!!

54. What is the result of this code below:

```
----------------------------------------
for (int i=1; i<5; i++)
{
for (int k=i;k<5;k++)
System.out.print("!");
for (int j = 5-i; j < 5; j++)
{
System.out.print("*");
}
System.out.println();
}
```

a. //Result:
   ***
   ***
   ***
   ***

b. //Result:
   *
   **
   ***
   ****

c. //Result:
   !!!!*
   !!!**
   !!**
   !**

d. nothing

55. If we use Adjacency matrix for weighted undirected graph, we will have:
    a. None of the above
    b. **A symmetric matrix over its diagonal**
    c. An asymmetric matrix
56. Which is the appropriate implementation of Graph
    a. Adjacency matrix
    b. Adjacency list
    c. **All of the above**
57. What is maximum number of nodes in a binary tree of depth 5?
    a. 15
    b. **63**
    c. 30
58. What is number of nodes in a full binary tree of depth 4?
    a. 32
    b. 10
    c. 17
    d. 31
59. What is maximum number of nodes in level 4 of a binary tree could have?
    a. **16**
    b. 7
    c. I don't know
    d. 4
60. How many nodes in total of a full binary that has 33 leaves?
    a. 49
    b. 40
    c. 50
    d. 65
61. How many leaves does the complete binary tree that has 19 nodes in total have?
    a. a. 9 (x)
    b. 11
    c. 12
    d. 10
62. How many internal nodes of a full binary tree that has 10001 nodes in total?
    a. 333
    b. 600
    c. 768

d. 500

63. Given 2 result of a binary tree traversal:
    a. *preorder :* YZCDEXBUTA
    b. *inorder :* DCEZYUBTXA
64. What is the root node of the tree?
    a. B
    b. X
    c. Y
    d. A
65. What is the left child of node B?
    a. C
    b. X
    c. U
    d. A
66. What is the right child of node Z?
    a. D
    b. T
    c. Y
    d. null
67. Calculate the result of Prefix expression:

    a. + 4 5 * / 6 2 3

    a. 24
    b. 18
    c. -8
    d. 0
68. Calculate the result of Prefix expression:

    + 10 * 5 + 2 3

    a. 31
    b. 75
    c. 55
    d. 35
69. Calculate the result of Postfix expression:

    6 2 - 3 1 - 4 * 2 + *

    a. 20
    b. 23
    c. 17
    d. 40

70. Calculate the result of Postfix expression:

    6 2 - 3 1 - 4 / 2 + *

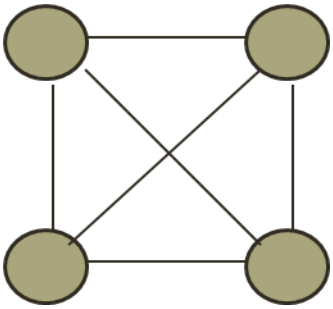
    a. 30
    b. **10**
    c. 7
    d. 5

71. Calculate the result of Postfix expression:

    6 2 + 3 1 - 4 * 2 + *

    a. 120
    b. 80
    c. 72
    d. 36

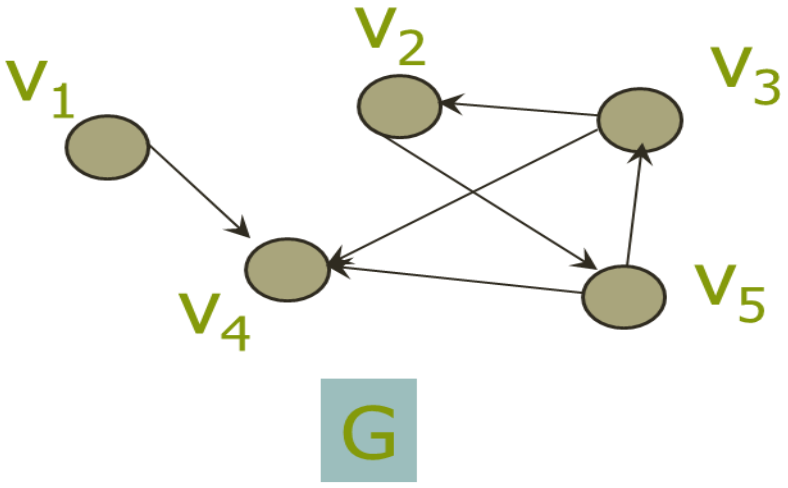
72.



    WHICH IS THE BEST DESCRIBE OF THE GRAPH?

    a. **Unweighted, Undirected, Complete Graph**
    b. Unweighted, undirected, connected graph
    c. Complete Graph
    d. Unweighted, connected graph

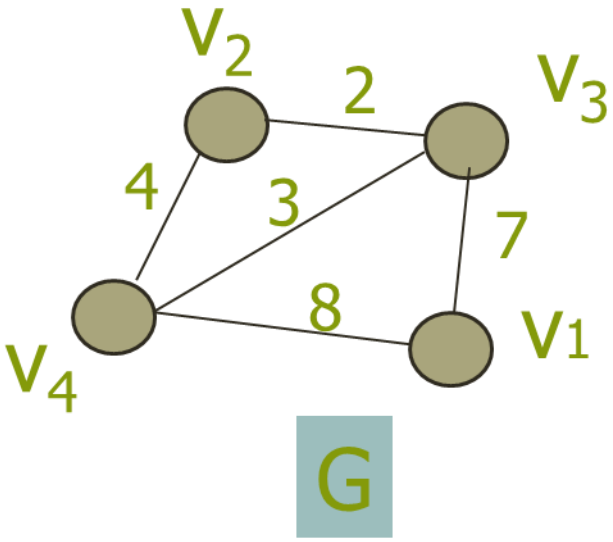
73.



Please fill the table of adjacency matrix for node V1 -> V5 of the graph:

	V1	V2	V3	V4	V5
V1	**0**	**0**	**0**	**1**	**0**
V2	**0**	**0**	**0**	**0**	**1**
V3	**0**	**1**	**0**	**1**	**0**
V4	**0**	**0**	**0**	**0**	**0**
V5	**0**	**0**	**1**	**1**	**0**

74.



Rule to fill the list:

-Increasing index of node

-Node format example v6(10)

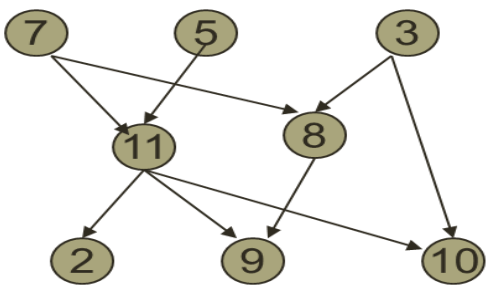
1.[v1] --> [**v3(7)**] --> [**v4(8)**]

2.[v2] --> [**v3(2)**] --> [**v4(4)**]

3.[v3] --> [**v1(7)**] --> [**v2(2)**] --> [**v4(3)**]

4.[v4] --> [**v1(8)**] --> [**v3(7)**] --> [**v4(8)**]

72.



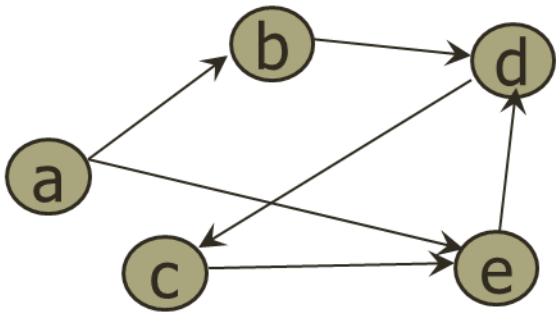
WHAT IS NOT THE RESULT OF TOPOLOGY SORT?

a. 5, 7, 11, 2, 3, 8, 9, 10

b. **5, 7, 8, 11, 3, 2, 9, 10**

c. 5, 7, 3, 8, 11, 2, 9, 10

75.

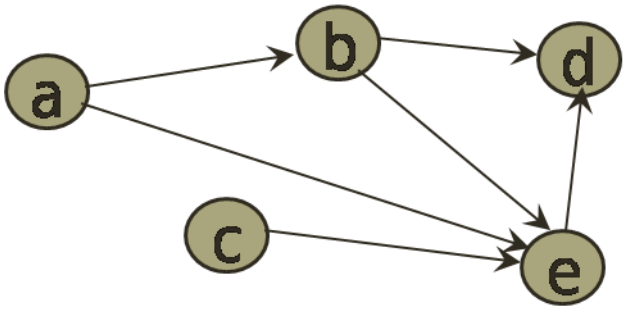


CAN THIS GRAPH PERFORM TOPOLOGICAL SORT?

a. YES

b. NO

76.



//This is topological sort based on given criterium

//CHOOSE THE CORRECT OUTPUT

Algorithm TopoSort

n = |V|;

for i = 1 to n {

_____ select a node v that has the most successors;

_____ print this vertex;

_____ delete node v and its edges from the graph;

}

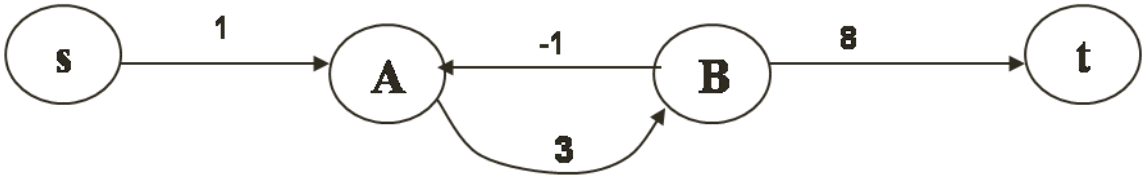
a. C A B D E

b. C A E B D

c. **A B C E D**

d. A C B E D

77.



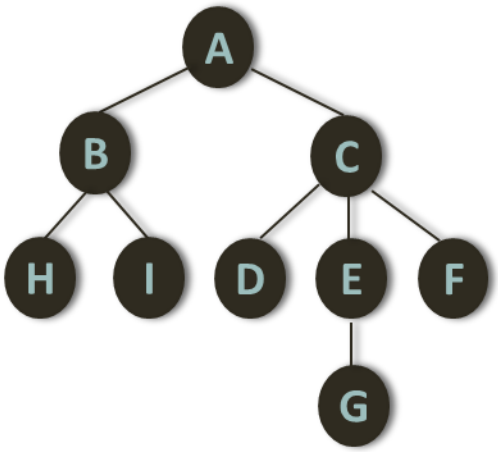
//CAN WE FIND THE SHORTEST PATH FROM S --> T IN THIS SITUTATION?

a. **YES**

b. DON'T KNOW

c. NO

78.



In the picture, the degree of node C is:

a. **3**

b. 4

c. 7

d. 2

79. In the picture, the degree of node E is:

a. **1**

b. 4

c. 5

d. 2

80. In the picture, which statement is correct about (I):

a. (i) has degree of 3

b. **(i) is the subtree of B**

c. (i) is the left child of B

81. In the picture, Depth of this tree is:

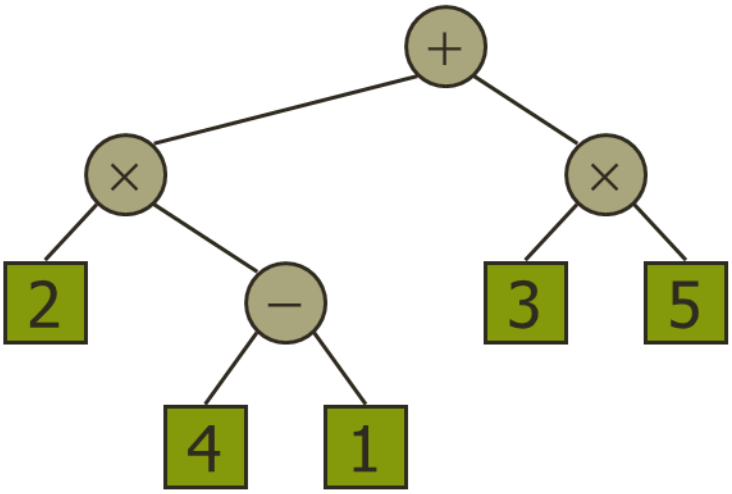
a. **3**

b. 9

c. 7

d. 4

82.



What is the result of this expression using binary tree:

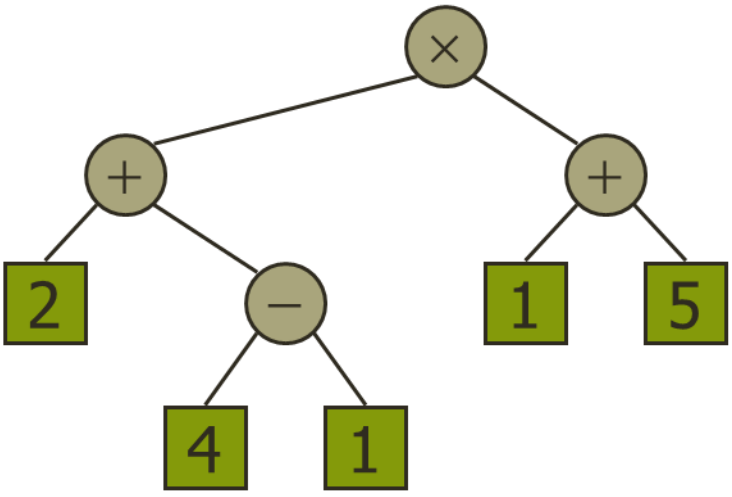
a. 12

b. 40

c. 50

d. **21**

83.



What is the result of this expression using binary tree:

a. 12

b. **30**

c. 21

d. 15

84. This is an Array Representation of a Complete Binary Tree:

A	G	L	D	E	F	Z	H	I	P

What is the left child of G:

a. E

b. Z

c. **D**

d. H

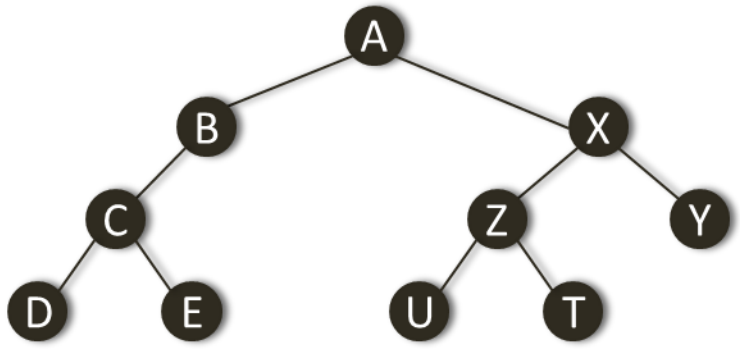
85. What is the right child of D:

a. **I**

b. P

c. L

86.



What is the result of inorder tree traversal? Fill the blanks with correct letter order:

D	C	E	B	A	U	Z	T	X	Y

87. What do you think about the statement below?

In DSA, Tree, Stack, Queue are algorithms to solving common data problem!

a. **False**

b. True

88. A stack is similar to a list, especially they can both perform insertion a new node to the middle of the them.

a. **False**

b. True

89. Heap data structure is a binary search tree?

a. **False**

b. True

90. Complete the code to delete all the nodes from pointer P to the tail of the Singly linked list.

---

1. P->link = _____; //A

---

a. **null**
b. head
c. P->link->link->link
d. P->link->link

91. Complete the code to delete a node: Node(item, link), which P is the pointer pointing to the node right before it in the list.

---

1. P->link = _____; //A

---

a. (A) tail
b. (A) null
c. **(A) P->link->link**

92. Complete the code to delete all the nodes from "current" to the tail of the Singly linked list.

---

1. current.setNext(_____) ; //C code: P->link = _____;

---

a. current.getNext().getNext().getNext()
   //C code: P->link->link->link
b. head
c. current.getNext().getNext()
   //C code: P->link->link
d. null

93. What is the implementation of following code:

---

i. void main()

```
2        {          int i;
3         int A[10];
4
5         for (i=0;i<10;i++)
6         A[i]=i*i;
7        }
```

---

b. reference-based

c. **array-based**

d. nothing

94. ---

2. #define TOTAL_SLOTS 100
3. typedef struct queue Queue;
4. struct queue

{ int front;

5. int rear;
6. int items[TOTAL_SLOTS];
7. };

What implementation type of the queue from the code?

a. J-based implemenation

b. Dont know

c. Linked Implemenation

d. array based implementation

95. What is the maximum nodes in this queue?

a. 99

b. 100

c. n

d. Don’t know

96. What is the maximum nodes in this queue?

---

1. //Initialize MAX_SIZE=1000
2. public class Queue {
3. Queue()
4. { int front = 0;
4. int rear = 0;
5. int data[MAX_SIZE];
6. };

---

a. Don't know

b. 1000

c. n

d. 999

97. void bubblesort_checkpasses(int x[ ], int N)

{

int temp, i,j;

boolean switched = TRUE;

{...}

}

WITH THE INTRODUCTION OF A BOOLEAN VARIABLE switched, WHAT IS THE BEST CASE TIME COMPLEXITY OF THE CODE?

a. O(log n)

b. O(n^2)

c. **O(n)**

d. O(1)

98. //LOOK AT THE GRAPH OPERATION BELOW

//WHAT DOES IT DO?

unmark all vertices in G;

Creat a queue q;

mark s;

insert(s,q)

while (!isempty(q))

__curr = delete(q);

__visit curr; // e.g., print its data

__ for each edge

__ __ if V is unmarked

_____mark V;

____ insert(V,q);

a. **breadth first search**

b. post order graph traversal

c. in order graph traversal

d. depth first search

99. What is the condition of this queue to be empty?

---

1. #define TOTAL_SLOTS 100
2. typedef struct queue Queue;
3. struct queue

4. { int front;

4. int rear;
5. int items[TOTAL_SLOTS];
6. };

---

a. front = 0;
b. **front == rear;**
c. rear = 0;
d. rear = null

100. What does this piece of code do to a List?

---

1. Node current = head;
2. while(current != null)
3. {

4. current=current.getNext(); //C code: current = current->next;

5.}

---

a. insert new node
b. delete the tail
c. **Traversing the list**

101. Pseudo code:

Use a pointer p to traverse the list and search the node and compare the data.

If found: return the pointer to the node.

Otherwise return NULL. Node(data, next)

---

1. NodePtr SearchNode(NodePtr pList, int data)
2. { NodePtr p=pList;
3. while (p!=NULL)
4. { if (p->data== _____) //(A)
5. return p;
6. p = _____; //(B)
7. }
8. return NULL;

9.}

---

a. (A) 0

i. (B) head

b. (A) data

i. (B) p->data

c. (A) data

i. (B) p->next

102. **** FILL THE BLANK ****

**** FILL THE BLANK ****

//THIS CODE IS SELECTION SORT

//PLEASE FILL THE BLANKS WITH THE CORRECT ANSWER

```
void selection_sort(float x[ ], int N)
{
____int target_index, large_index, i;
____float large;

____//Update the data at each target position one by one, from right to left
____for (target_index =N-1; target_index >0; target_index --)
____{
____large = x[ 0 ];//<--TODO
____large_index = 0;

____for (i=1; i <= target_index; i++)
____if (x[i] > large)
____{
_________large = x[ i ]; //<--TODO
_________large_index =  i ; /* and its position number into large_index */
```

____}

____ x[**large_index**] = x[**target_index**]; /* swap */

____ x[target_index] = large;

____}

}

//THIS IS STRAIT INSERTION SORT

//PLEASE FILL IN THE BLANKS WITH THE CORRECT ANSWERS

0.Insertion-Sort(A)

1.____ for j = 1 to n-1 {

2._____ key = A[j];

3._____ i = j-1;

4._____ while i >= 0 and A[i] > key

5._____ { A[i+1] = A[**i**];

6._____ i = i – 1;

7.____ }

8.____ A[**i + 1**] = **key** ;

9.}

//THIS IS THE PARTITION FUNCTION OF QUICKSORT ALGORITHM

//PLEASE FILL THE BLANKS WITH CORRECT ANSWERS

void partition(int x[ ], int idLeftMost, int idRightMost, int *pj)

{

```
_______ int down, up, a, temp;

_______ a = x[idLeftMost];

_______ up = idRightMost;

_______ down = idLeftMost;


_______ while (down < up)

_______ { while ((x[down] <= a) && (down < idRightMost))

_____________ down ++; /* move up the array */

_______ while (x[up] > a)

_____________ up--; /* move down the array */


_______ if (down < up) /* interchange x[down] and x[up] */

_______ {

_____________temp = x[down];

_____________x[ down ] = x[up];

_____________x[up] = temp ;

_______ }

_______ }


_______x[ idLeftMost ] = x[up];

_______x[up] = a;

_______*pj = up;

}
```

//MERGE SORT:

//split the array into two roughly equal subarrays

//sort the subarrays by recursive applications of Mergesort and merge the sorted subarray

```
void merge-sort(int x[ ], int lower_bound, int upper_bound)
{
_____     int pivote;
_____     if (lower_bound != upper_bound)
_____     {
_____________     pivote = (lower_bound + upper_bound) / 2 ;
_____________     merge-sort(x, lower_bound, pivote);
_____________     merge-sort(x, pivote +1, upper_bound);
_____________     merge(x, lower_bound, pivote, upper_bound);
_____     }
}
```

//THIS FUNCTION IS TO MOVE the root value to make the whole tree a max-heap

// Some steps in the functions:

//x is the heap array, no. of elements = N

//Start considering the root node.

//The replacement candidate is Right (or Left) child of id2Down

// If replacement is not necessary then don't do it, stop trickling otherwise replace.

// Prepare for next trickling

```
void trickle_down (int x[ ], int N)

{_____ int id2Down, idReplace; //idReplace is child of id2Down

_____ int temp; //for swapping data

_____ id2Down = 0;

_____ idReplace = 2* id2Down  +2;


_____ while (idReplace <= N-1 )

_____ {

_____ if (idReplace < N-1 && x[idReplace] < x[idReplace -1] )

_____ idReplace -- ;

_____ if ( x[id2Down] >= x[idReplace] )

_____ break;

_____ temp = x[id2Down];

_____ x[id2Down] = x[idReplace];

_____ x[idReplace] =  temp;


_____ id2Down = idReplace;

_____ idReplace = 2* idReplace +2;

_____ }

}


//Program to create Binary tree, with left and right child

//====================================

NodePtr maketree(int value) //create a new.
```

```
{ NodePtr p;
p= (NodePtr)malloc(sizeof(struct node));
p->info = value;
p->right = NULL;
p->left = NULL;
return(p);
}

void setleft(NodePtr p, int value) //create a new left child
{    if (p==NULL)
printf("void insertion\n");
else if (p-> left != NULL)
printf("invalid insertion");
else
p-> left = maketree(value);
}

void setright(NodePtr p, int value) //create a new right child
{    if (p==NULL)
printf("void insertion\n");
else if (p-> right != NULL)
printf("invalid insertion");
else
p-> right = maketree(value);
```

```
}

//To count the number of leaf nodes
//The recursive function to count the number of leaves of a tree
int count_leaf(NodePtr tree)
{_____ if (tree == NULL)
_____ return( 0 );
_____ else if ((tree->left == NULL) && (tree->right == NULL))
_____ return( 1 );
_____ else
_____ return( count_leaf (tree->left) + count_leaf (tree->right));
}

//tree in-order traversal
//definition: intrav is the function to print all tree nodes using in-order, which follows the rules defined in lecture.
struct node {
int info;
struct node* left;
struct node* right;
}

void intrav(NodePtr tree)
{ if (tree != NULL)
```

```
{ intrav(tree-> left );

ntrav(tree-> info );

printf("%d\n", tree-> right);

}

}
```

```
//tree post order traversal

//definition: posttrav is the function to print all tree nodes using post-order, which follows the rules defined in lecture.

struct node {

int info;

struct node* left;

struct node* right;

}
```

```
void posttrav(NodePtr tree)

{ if (tree != NULL)

{ posttrav(tree-> left );

posttrav(tree-> right );

printf("%d\n", tree-> info );

}

}
```

```
//The algorithm of finding duplicate values in array as follow:
```

```
//Create root node to store first no.
//Handle each remaining input no.
//----------Start at the root, traverse the tree
//----------     from top to bottom until we meet
//----------the same value.
//----------------------  At each node,
//-------------------------------- If input no. < current node
//--------------------------------  (we'll look at the left sub-tree)
//-------------------------------- If the left sub-tree is empty,
//---------------------------------           ----------          create a left child to
//---------------------------------           ----------          store input no. and stop traversing
//----------------------------------else go to left subtree.
//-------------------------------- else (input no. > current node)
//-------------------------------- (we'll look at the right sub-tree)
//--------------------------------  …
//---------      If no new node has been created for
//---------      the input number, it is a duplicate
//GIVEN:
//=====================================================
typedef struct node *NodePtr;
struct node
{     int info;
NodePtr left;
NodePtr right;
```

```
};


NodePtr maketree(int value) {..} //create a new

void setleft(NodePtr p, int value) {..} //create a new left child of a given node

void setright(NodePtr p, int value) {..} //create a new right child of a given node


//Fill the blanks below====================================

void main()

{_____     int i, seq[12]={ 3,5,0,2,7,9,5,6,3,7,0,8 };

_____      NodePtr p, T=maketree(seq[0]);

_____      for(i=1;i<12;i++)

_____      {        BOOL bNewNodeCreated=FALSE;

_____________      p=T;

_____________      while (p->info != seq[i])

_____________      {        if (seq[i] < p-> info )

_____________________________      if (p->left==NULL)

_____________________________      { setleft          (p, seq[i]  );

_____________________________       bNewNodeCreated=TRUE;

_____________________________       break;

_____________________      }

_____________________      else

_____________________________      p=p->left;

_____________________      else

_____________________      if (p->right==NULL)
```

_____ { **setright** (p, **seq[i]**);

_____ bNewNodeCreated=TRUE;

_____ break;

_____ }

_____ else

_____ p=p->right;

_____}

_____if (!bNewNodeCreated)

_____printf("%d is a duplicate\n",seq[i]);

_____ }

}

//Complete the code to delete an element from a queue:

* Check if the queue is empty

* get the value at the front of the queue

* Correct the front of the queue

* return the value

//----------

1. //Initialize MAX_SIZE=1000

2. public class Queue {

3. Queue()

4. { int front = 0;

5. int rear = 0;

```
6.  int data[MAX_SIZE];
7.   };


8.  boolean isEmpty()
9.  {
10. return (this.front %   this.rear);
11. }
//-----------------------------------------------


int delete()
{   int rtn_val;
    if (isEmpty())
    {
            printf("underflow error\n");       return null;
    }
    rtn_val = this.data[ MAX_SIZE ];
    front = (  front + 1) %  MAX_SIZE;
    return ( rtn_val );
}
```

Given an array-based Stack.

The pseudo-code of operation Push() is defined as:

* connect new node to stack

* Set new top stack

Complete the code below that use the above pseudo-code:

```
public class SlNode{
    public void setNext();
    public XData getNext();
}
```

```
1. public stack()
2.{
3.   top = new SlNode();
4.};
5.
6. public boolean isEmpty() {..}  //check if stack is empty

7. public void push(SlNode x)
8. {
9.   x. AAA = BBB ;
10. CCC = x;
11. }
```

In link-based Stack, Size() method is descripted as below:

* travel through the stack

* count increases by one if current node is not null

* Return the saved value.

Complete the code below:

```
public class SlNode{
    public void setNext();
    public XData getNext();
}
```

```
1. public stack()
2.{
3.   top = new SlNode();
4.};
5. public boolean isEmpty()  {..}
6. public boolean isFull()  {..}

7. public int size()
8. {          SlNode current = top ;
    int count = 0;
9.   while(current != null )
10. {
11.          count++ ;
12.          current =  current.getNext();
```

13. }

15. return **count** ;

16. }