CS 310 - PRACTICE QUESTIONS - SET I

Indicate whether the following statements are TRUE or FALSE. **Encircle** the correct choice.

1. $\Theta(\log_a(n)) = \Theta(\log_b(n))$	TRUE / FALSE
2. 2^{2n} **is** $O(2^n)$	TRUE / FALSE
3. $\log_2(n^2)$ **is** $O(\log_2(n))$	TRUE / FALSE
4. $\sqrt{n}$ **is** $O(\log_2(n^2))$	TRUE / FALSE
5. $\sqrt{n}\log_2(n)$ **is** $O(n)$	TRUE / FALSE
6. The time and space complexity of an algorithm does not depend on the choice of the data structure for the problem.	TRUE / FALSE
7. The tree constructed by Breadth-First-Search algorithm is unaffected by the order in which the vertices are explored.	TRUE / FALSE
8. Breadth-First-Search algorithm can be used to find the shortest path from vertex 's' to vertex 't' in an undirected weighted graph G.	TRUE / FALSE
9. Every directed acyclic graph has a topological ordering.	TRUE / FALSE
10. Every execution of the Gale-Shapley algorithm on an instance returns the same matching.	TRUE / FALSE

If f (n) = $O(n^2)$ and g(n) = $O(n^2)$, then f(n) = O(g(n)). Is this statement TRUE or FALSE
__FALSE__
Brief justification of your answer:
Suppose $f(n) = c_1 n^2$, $g(n) = c_2 n$. f(n) is not O(g(n))

If f (n) = O(g(n)) and g(n) = $O(n^3)$, then f(n) = $O(n^3)$. Is this statement TRUE or FALSE
__TRUE__
Brief justification of your answer:
$g(n) = c\ n^3$
f(n) is $O(c\ n^3)$ => f(n) is $O(n^3)$

Consider the following pseudocode. n is the input size.

```
function_A(int n) {
        for i = 1 to n
                Call function_B(n);
         Call function_C(n);
}
```

function_B() takes O(log(n)) time and function_C takes O(n log(n^2)) time.

What is the total running time for function_A in Big-Oh notation? **Justify your answer in terms c and n_0.**

$T(n) = c_1 n \log(n) + c_2 n \log(n^2) = c_1 n \log(n) + 2 c_2 n \log(n)$

$T(n)$ is $O(n \log(n))$

Given an undirected graph **`G(V,E)`** with n vertices and m edges. How long does it take to **delete** an edge (u,w) from G if an adjacency **list** is used to represent the graph?

$O(\text{degree}(u) + \text{degree}(w))$

Given an undirected graph **`G(V,E)`** with n vertices and m edges. How long does it take to **insert** an edge (u,w) from G if an adjacency **matrix** is used to represent the graph?

$O(1)$

Give an example of a sparse graph where number of edges are linear in the number of vertices

Tree

In the following multiple choice questions, **encircle all choices** that are correct.

1. How many simple paths exist in an n-node tree from a vertex x to vertex y.
   a. One
   b. Two
   c. n
   d. Depends on the type of the tree
   e. All of above

f. None of above

2. Which data structure is faster for finding the **degree** of a vertex in a graph?
   a. Adjacency list
   b. Adjacency matrix
   c. Both of them are equally faster

3. Which data structure takes less space for a sparse graph?
   a. Adjacency list
   b. Adjacency matrix
   c. Both of them take the same amount of space

4. Given an **undirected** graph `G(V,E)`. Every edge (u,w) encountered during a breadth-first search is one of the following types. Assume ancestor ≠ parent.
   a. Tree edge (i.e, the edge included in the BFS tree)
   b. Cross edge (i.e., u is neither an ancestor or descendant of w)
   c. Back edge (i.e., an edge to an ancestor vertex in the BFS tree)
   d. All of above
   e. None of above

5. Given **n** elements, you want to construct a heap such that the heap order is preserved. Definition of heap order: For every element **v**, at a node **i**, the element **w** at **i**'s parent satisfies **key(w)≤key(v)**.
   - How long does it take to construct the heap to store n elements?

   a. O(1)
   b. O(n)
   c. O(log(n))
   d. O(nlog(n))
   e. $O(n^2)$
   f. None of above

   - How long does it take to find the minimum element in the heap?

   a. O(1)
   b. O(n)
   c. O(log(n))
   d. O(nlog(n))
   e. $O(n^2)$
   f. None of above

6. Given an undirected weighted graph **G(V,E)** with positive edge weights. G has n vertices and m edges. What is the time complexity of Dijkstra's single source shortest path algorithm if a priority queue is used to store the distances of the vertices from source.
   a. O(nm)
   b. $O(n^2m)$
   c. O(nlog(n))

d. $O(m\log(n))$
e. $O(n\log(m))$
f. **None of above**

7. Given the following recurrence relation:

$T(1) = a,$

$T(n) = qT(n/2) + bn,\ q>2$

a. **T(n) is** $O(qn)$
b. **T(n) is** $O(qn^2)$
c. **T(n) is** $O(q\log_2(n))$
d. **T(n) is** $O(qn\log_2(n))$
e. **T(n) is** $O(n\log(q))$
f. **T(n) is** $O(q^{\log(n)})$
g. **T(n) is** $O(n^{\log(q)})$
h. **None of above**

Given an **undirected** connected graph $G(V,E)$. Is it possible that the depth of any DFS tree rooted at a vertex 'u' of G may be greater than the depth of any BFS tree rooted at 'u'.
TRUE or FALSE ___TRUE_____ If TRUE, illustrate by giving an example of a graph G and the corresponding BFS and DFS trees. If FALSE, prove it.

This is an easy question. Think of an example.

Recall the "Interval Partitioning" problem, where requests are in the form of time intervals and many identical resources are available. We want to schedule all the requests using as few resources as possible. E.g. classrooms are resources and we want to schedule all lectures in as few classrooms as possible. Following is the greedy "Earliest Start-Time First" algorithm for this problem. If the classrooms are stored in a priority queue where the key is the finish time of the last lecture then how much time does the FOR-loop take? Give your answer in Big-Oh notation and **explain** how you estimated the time complexity.

```
EARLIEST-START-TIME-FIRST (n, s1, s2, ..., sn, f1, f2, ..., fn)

SORT lectures by start time so that s1 ≤ s2 ≤ ... ≤ sn.
d ← 0  ⟵ number of allocated classrooms
FOR j = 1 TO n
    IF lecture j is compatible with some classroom
        Schedule lecture j in any such classroom k.
    ELSE
        Allocate a new classroom d + 1.
        Schedule lecture j in classroom d + 1.
        d ← d + 1
RETURN schedule.
```

O(n log(d))

Your friend ran the Prim's algorithm on an undirected, weighted graph G and constructed a minimum spanning tree. Your friend claims that the minimum spanning tree is a bipartite graph. Do you agree?

YES or NO? ___YES_____
Brief explanation of your answer.

Every tree is bipartite.

Given a graph G with positive weights on the edges. S and A are two vertices in G. If all the edge weights are **distinct**, then there is a **unique** shortest path from S to A.

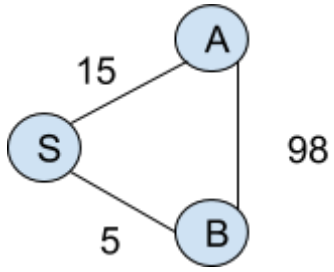
Is the above statement TRUE or FALSE? __FALSE_____
Brief explanation of your answer.

This is an easy question. Think of a counterexample.

Given an undirected weighted graph **`G(V,E)`** with positive edge weights. The graph has n vertices and m edges. You already know Dijkstra's algorithm for finding the single source shortest path to all vertices in **G**. Your friend says that he can slightly modify Dijkstra's algorithm for finding the single source longest path to all vertices in **G**. The change he has made to Dijkstra's algorithm is that instead of checking and updating the minimum distance at each step from source, he now checks and updates the maximum distance from source.

Prove that your friend's algorithm is correct or give a counterexample.

The friend's claim is false.
Counter example:



The friend's algorithm will report 15 as the longest path from S to A, which is not correct.

You have studied that we can find the single source shortest paths in a weighted graph G(V,E) with nonnegative edge weights using Dijkstra's algorithm. The time complexity for Dijkstra's algorithm using heap-based priority queue is O(m log(n)). If you are told that the input graph G will always be a DAG then can you design an algorithm for single source shortest paths that has better time complexity than Dijkstra's algorithm?

If YES, then give a clear description of your algorithm and its running time. If NO, then explain why not.
YES.
This question is similar to the longest path question in the quiz. See quiz solution on LMS.

Let G=(V,E) be an undirected graph with *n* vertices. If G is connected and has *n-1* edges then G does not contain a cycle.

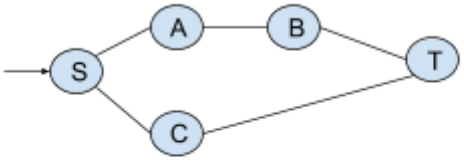
Is the above statement TRUE or FALSE? ____TRUE_____

We know every n-node tree has exactly n-1 edges. The undirected graph G is connected, so it cannot contain a cycle.

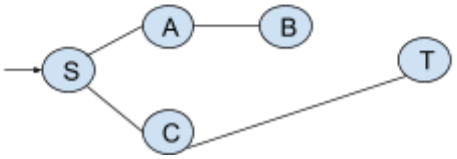
Let G=(V,E) be an undirected graph with n vertices. If there is some path p from vertex s to vertex t in G, then the tree returned by Breadth-First Search on G will contain p.

Is the above statement TRUE or FALSE? __FALSE_____
Brief explanation of your answer.

Graph G contains path p: S,A,B,T



Tree returned by BFS from start node S does not contain p



Suppose there are N companies where each company has a job vacancy for one intern. We are given the preference lists of N companies and N interns. In an instance of the Stable Matching Problem, Company C_1 has ranked intern I_3 as first in its preference list and intern I_3 has also ranked C_1 as first in his preference list. Will the (C1, I3) pair be present in every stable matching for this instance? You have to prove it or give a counterexample.

(C1, I3) pair will be present in every stable matching for this instance TRUE / FALSE
Proof or Counterexample:

Proof by Contradiction.
Assume that a perfect matching 'S' does not contain the (C1, I3) pair, but instead contains pairs (C1, Ix) and (Cy, I3). Since C1 and I3 had ranked each other as first, pair (C1, I3) is an instability with respect to 'S' (meaning pair (C1, I3) does not belong to S, but both C1 and I3 prefer each other to their partners in S). Hence matching S cannot be stable.

While executing the Gale-Shapley algorithm, we want to find if company C_x prefers I_y over I_z in constant time. Describe the data structure you will use to perform this operation in O(1) time.
Suppose the preference list of a company is as follows.

1st	2nd	3rd	4th	5th	6th	7th	8th
8	3	7	1	4	5	6	2

For constant time access, construct an array which stores the ranking of each intern as follows. The indices of this array are interns.

1	2	3	4	5	6	7	8
4th	8th	2nd	5th	6th	7th	3rd	1st

```
int RecFunc(int n) {
  int r;
  if (n==1) return n;
  else {
       RecFunc(n/2);
       r = n/8;
       return subtract(r); }
}
int subtract(int m) {
  int i, diff=578;
  for (i=0; i<m; i++) {
       diff -= i; }
  return diff;
}
```

Write down the recurrence relation for the above code.

$T(1) = b$

$T(n) = T(n/2) + c\ n/8$

Draw the recurrence tree for this code.

Solve the recurrence relation (show all steps for full credit)

$T(n) = b + 2\ (n-1)\ c/8$

T(n) (in Big-Oh notation) is _____O(n)_____

Does the recurrence have a geometric series with decaying exponent? _____YES_____

Explain: The geometric series is $1+½+¼+ \ldots + 1/2^{k-1}$

Consider the following code.

```
unsigned long func(int a, unsigned int b) {
  if (b == 0 )
      return 1;
  else if( b == 1)
      return a;
  else if (b%2 == 0)
      return func(a, b/2)*func(a, b/2);
  else
      return a*func(a, b/2)*func(a, b/2);
}
```

A. What is the above code computing?

a^b

B. Write the recurrence relation for this code.

The input parameters to the function are a and b. c_0, c_1 and c_2 are constants
$T(a, 0) = c_0$, $T(a, 1) = c_1$
$T(a,b) = 2\ T(a, b/2) + c_2$

C. Solve the recurrence relation to determine the time complexity of this code. Show all steps.

$T(a,b) = 2\ T(a, b/2) + c_2$

$T(a,b) = 2\ (2\ T(a, b/4) + c_2) + c_2$

$T(a,b) = 4\ T(a, b/4) + 3c_2$

$T(a,b) = 8\ T(a, b/8) + 7c_2$

Generalizing:
$T(a,b) = k\ T(a, b/k) + (k-1)c_2$

Let $k = b$
$T(a,b) = b\ T(a, 1) + (b-1)c_2$

$T(a,b) = b\ c_1 + (b-1)c_2$

D. What can you do to improve the time complexity of this code?

`func(a, b/2)` is unnecessarily being called twice. Store its value in a variable, e.g. `result = func(a, b/2)` and `return result*result;`

The recurrence now becomes:

$T(a, 0) = c_0$, $T(a, 1) = c_1$
$T(a,b) = T(a, b/2) + c_2$

T(a,b) is O(log(b))

Consider the following code.

```
int compute(int *array, int i, int j) {
  int k, out=1;
  for (k=i; k<=j; k++) {
    out *= array[k];
  }
  return (out);
}

int main() {
 int i, j, m, n;
 ...
 m = n/2;
  ...
  /* Assume that n is an even number. A is a
      single dimensional array size n integers and
      B is a 2-dimensional array of size n x n integers */
  for (i=0; i<n; i++) {      /* outer for-loop */
    for (j=i+1; j<m; j++) {   /* inner for-loop */
      B[i][j] = compute(A, i, j);
    }
  }
  ...
}
```

What is the asymptotic time complexity in Big-Oh notation of the function **compute()? Write your answer in terms of i and j.**
O(j-i)

_____

Fill in the blank for the number of steps executed in the **inner for-loop.**

$\sum_{j=i+1}^{m}$ **(** ___c (j-i)_____ **)** **,** c is a constant

Simply the expression you obtained above (common formulae are at end of this booklet)

c (1+2+3+ … m-i) = c (m-i) (m-i+1)/2

_____

What is the asymptotic time complexity in Big-Oh notation of the **inner for-loop? Write your answer in terms of m and i.** __O($(m-i)^2$)_____

Given an undirected graph **`G(V,E)`** with n vertices and m edges. Suppose graph G is represented as an **adjacency matrix**. What is the time complexity of Depth-First Search in this case?

Time Complexity in Big-Oh notation ___$O(n^2)$_____
Briefly justify your answer.
It will take O(n) time to find neighbors of each vertex.

Given a directed graph **`G(V,E)`** with n vertices and m edges. Suppose graph G is represented as an **adjacency list**. I want to find the number of **in-coming** edges for each of the vertices. What is the time complexity for finding that?

Time Complexity in Big-Oh notation ___O(n+m)_____
Briefly justify your answer.

Traverse the entire adjacency list and keep a count of the incoming edges for each vertex.

Given a **weighted**, undirected graph **`G(V,E)`** with n vertices and m edges with positive weights, you want to find the **shortest paths** from a source vertex 'S' to every other vertex. You know that you can use Dijkstra's shortest path algorithm for this purpose. However, if you are told that G is a **tree**, can you use Breadth-First Search (BFS) to find the shortest paths in O(m+n) time?

Can we use BFS to solve this problem. **YES / NO** ____YES_____
If Yes, **prove** it. If No, give a **counterexample**.

There is only one path between any two vertices in a tree.

Given a weighted, directed graph **`G(V,E)`** with n vertices and m edges with positive weights. Suppose graph G contains **cycles**. Is it possible that Dijkstra's shortest path algorithm will change the distance of a vertex it has already included inside the fringe?

YES / NO ____No_____
If Yes, illustrate by giving an example of a graph. If No, explain why.

A cycle will only increase the distance of a vertex already inside the fringe.

Given a weighted, directed graph **`G(V,E)`** with n vertices and m edges with positive weights. Suppose you find the shortest path 'P' in the graph G from source vertex 'S' to vertex 'A'. Now you modify the graph G by **adding** a constant positive number to all the edge weights. Lets call the modified graph G^+. Is the path 'P' still the shortest path from source vertex 'S' to vertex 'A' in graph G^+?

YES / NO ____**No**_____
Briefly justify your answer.

It's easy to construct a counterexample. Suppose there are two paths P_1 and P_2 from S to A. Suppose P_2 has **more hops** than P_1. If P_2 was originally the shortest path, then after adding a positive number to all edge weights, P_1 may become the new shortest path from S to A in G^+.

Given a weighted, directed graph **`G(V,E)`** with n vertices and m edges with positive weights. Suppose you find the shortest path 'P' in the graph G from source vertex 'S' to vertex 'A'. Now you modify the graph G by **multiplying** a constant positive number to all the edge weights. Lets call the modified graph G^x. Is the path 'P' still the shortest path from source vertex 'S' to vertex 'A' in graph G^x?

YES / NO ___**Yes**_____
Briefly justify your answer.
Scaling by a positive number 's'.
$s\,P_x < s\,P_y$, if $P_x < P_y$

Suppose G is a connected, undirected graph whose edges all have positive weight. Let M be a minimum spanning tree of this graph. Now, we modify the graph by adding 7 to the weight of each edge. Is M guaranteed to be a minimum spanning tree of the modified graph?

YES / NO ___**Yes**_____

In the recurrence tree for Merge-Sort algorithm, the work done at any level 'k' of the tree is equal to half the amount of work at level k-1.

Do you agree with the above statement? **YES / NO** _____**No**_____
If Yes, **prove** it. If No, then explain why.

Same work i.e. O(n) at each level.

Let's fast forward to your big graduation day! All of you have invited guests, who are going to be seated in a big hall. Suppose that guest Gx and Gy know each other and Gx and Gz know each other, then Gx can introduce Gy to Gz (so now Gy and Gz know each other indirectly). LUMS wants your guests to have a great time and has created some seating areas. They want people who know each other (**directly or indirectly**) to be seated together in one area so that they are comfortable talking to each other. You have to find the **minimum** number of seating areas for this purpose.

You have studied a number of algorithms in class. Can you use any of those algorithms to **efficiently** solve this problem? **Note:** You will **not** get marks if you try to design a new algorithm from scratch.

Finding the number of connected components in a graph.

_____

Briefly justify your answer.

The number of connected components in a graph can be found by DFS or BFS.

What is the time complexity of your algorithm in Big-Oh notation ___ O(m+n)_____

Recall the interval scheduling problem discussed in class, where you were given 'n' intervals, each with a start time (s_i) and a finish time (f_i) and you had to find the maximum subset of mutually compatible jobs. You know you can use the Earliest-Finish-Time-First (EFTF) greedy algorithm to solve this problem. Which of the following statements hold true for the EFTF algorithm.

Suppose EFTF says that the following intervals should be scheduled in the order given below.

I_a, I_c, I_d, I_b, I_f, I_e

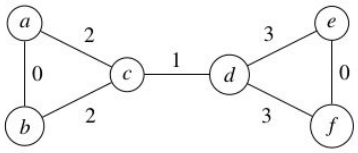
$f(I_a) \leq s(I_c)$ **TRUE / FALSE** _____T_____

$f(I_a) < s(I_d)$ **TRUE / FALSE** _____T_____

$s(I_d) < s(I_b)$ **TRUE / FALSE** _____T_____

$s(I_d) < f(I_b)$ **TRUE / FALSE** _____T_____

Given the following weighted undirected graph G(V, E), what is the maximum number of minimum spanning trees in this graph?



Maximum number of minimum spanning trees in G are: ___4_____

Brief justification of your answer:

a-b, e-f and c-d are in every MST. There is a choice between a-c and b-c and also between d-e and d-f

Recall the document printing problem discussed in class. In this question, we are simplifying its problem statement. We have an ordered list of n words. We cannot reorder the words. The length of the i^{th} word takes up w_i space, (for simplicity, we will assume that there are no spaces between words.) We want to lay out this ordered list of words into lines.
The length of a line is the sum of the lengths of the words on that line. The page width is L and no line can be longer than L. Suppose the length of line is K, then the penalty for that line is L − K. The total penalty is the **maximum** of the line penalties. The problem is to find a layout that minimizes the total penalty. You have to determine if the following pseudocode for a greedy algorithm will correctly solve this problem.

```
for i= 1 to n
      If the ith word fits in the current line, then place it there
      else place the ith word on a new line
```

Will the above greedy approach work? YES / NO ____NO_____
If Yes, prove it. Otherwise provide a counterexample.

Think of a counterexample.

You are given a directed graph G (V, E), where the vertices are numbered v_1, v_2, .. v_n. Each edge is directed from a vertex with lower index to a vertex with higher index (i.e., every directed edge has the form (v_i, v_j) if i<j). **Every vertex, except for v_n has at least one directed edge coming out of it.** The length of the path is the number of edges in it. We want to find the length of the longest path starting from v_1 and ending at v_n. You have to determine if the following pseudocode will find the correct solution.

```
currentNode = v1
L = 0
while there is an edge coming out of currentNode
    Choose the edge (currentnode, vj) such that j is as small as possible
```

```
    currentNode = v_j
    L = L+l
return L as length of longest path
```

Will the above greedy approach work? YES / NO ___NO_____
If Yes, prove it. Otherwise provide a counterexample.

Think of a counterexample.

Solve the following recurrence relation. You have to show all steps to get full marks.

T(1) = b
T(n) = 2T($\frac{n-1}{2}$) + c, where b and c are constants.

What is T(n) in Big-Oh notation? _____O(n)_____