NAME: Student ID:

KEY

QUESTION 1. Short Answer.

Briefly justify your answers.

(a) *(4 points)* **Give the big-O solution to the recurrence relation $T(n) = 3T(\frac{n}{2}) + \mathcal{O}(n)$.**

$a = 3, \; b = 2, \; d = 1$

$a > b^d \quad (3 > 2^1)$

By M.T., $\boxed{T(n) = O(n^{\log_2 3})}$

(b) *(3 points)* **Match the following for an edge (u, v) in a directed graph G, when depth-first search is executed. (Match the column entries on the left to the ones on the right.)**

i. Tree Edge or Forward Edge	a. $\text{pre}(u) < \text{pre}(v) < \text{post}(v) < \text{post}(u)$
ii. Back edge	b. $\text{pre}(v) < \text{post}(v) < \text{pre}(u) < \text{post}(u)$
iii. Cross edge	c. $\text{pre}(v) < \text{pre}(u) < \text{post}(u) < \text{post}(v)$

(Note: There was another variant with minor reordering.)

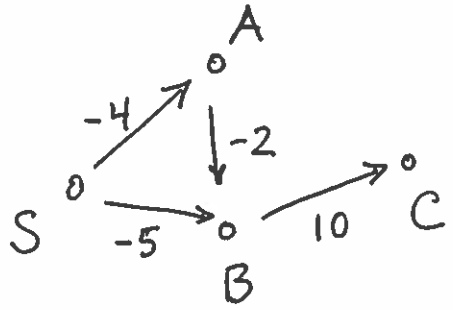
KEY

(c) *(4 points)* **Dijkstra's algorithm does not necessarily give correct results when the input graph has negative edge weights. Bob proposes that we add a large constant to each edge weight so that all edge weights are made positive, and then use Dijkstra's algorithm to find shortest paths from a given source to all reachable vertices.**

Bob's idea unfortunately is flawed. Explain why, and show a small counterexample with at most five vertices.

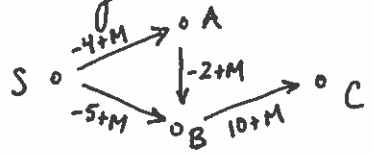
Flaw: A shortest path in Bob's transformed (re-weighted) graph is not necessarily a shortest path in the input graph.

Counterexample:



True SP from S to C is S − A − B − C.

Adding a large positive constant M (>5) to each edge weight doesn't prevent Dijkstra from returning S − B − C in the shortest-paths tree.



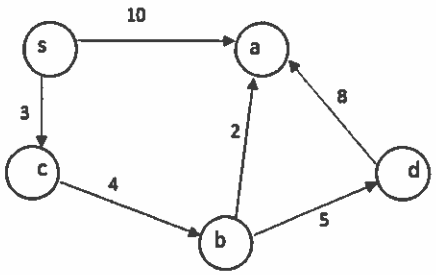
(d) *(4 points)* **Prove or disprove : If $f(n)$ is $O(g(n))$ then $2^{f(n)}$ is $O(2^{g(n)})$.**

- Consider $f = \log n^2$, $g = \log n$

  $f = 2 \cdot g = O(g)$.

- But $2^{\log n^2} = n^2 \notin O\left(2^{\log n}\right) = O(n)$

(e) *(5 points)* Consider the following directed graph G, with source vertex s. The labels on the edges represent their weights.

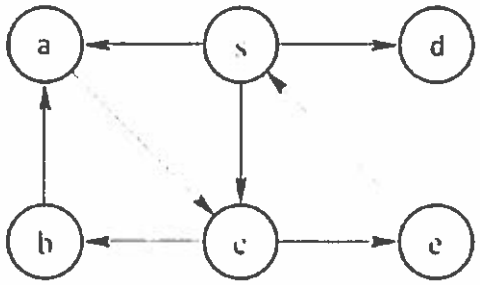


Execute the Bellman-Ford algorithm on this graph, with s as the source vertex. (Recall that $l_j^{(k)}$ denotes the shortest $s \rightsquigarrow v_j$ pathlength using $\leq k$ edges.) Fill in the table below with the distance values $l_j^{(k)}$ of all the vertices v_j for the first three iterations of the algorithm ($k = 1, 2, 3$).

Iteration	s	a	b	c	d
$k = 0$	0	∞	∞	∞	∞
$k = 1$	0	10	∞	3	∞
$k = 2$	0	10	7	3	∞
$k = 3$	0	9	7	3	12

KEY

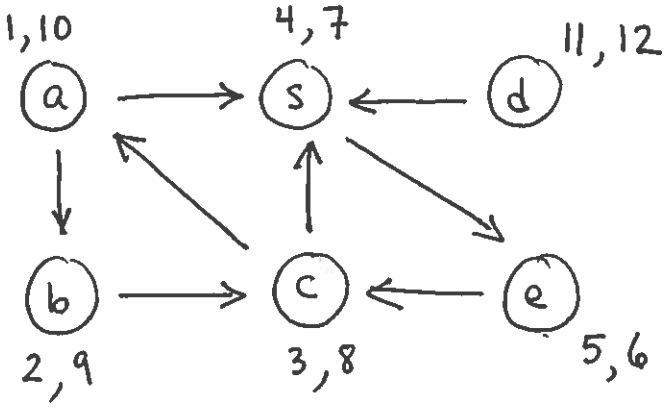
QUESTION 2. DFS, BFS, SCC-Finding.



For this question, refer to the graph above.

(a) ***(4 points)*** **List the strongly connected components (SCCs) in the given graph G, in the order that they are found using the algorithm given in class. Show your work, including G^R and pre/post numbers.**

G^R with pre, post numbers



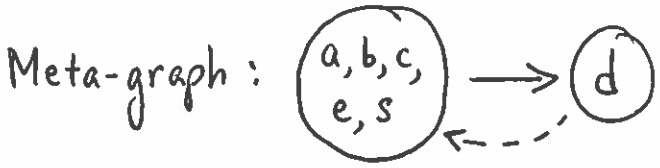
- d has highest post number in DFS of G^R.
- explore(d) in G finds the SCC {d}.
- a has next highest post number.
- explore(a) in G finds the SCC {a, b, c, e, s}.

List of SCCs in order found: {d}, {a, b, c, e, s}

KEY

(b) *(2 points)* **What is the minimum number of edges that, when added to the given graph G, will make it strongly connected?**

1



Adding any (one) of the edges (d,a), (d,b), (d,c), (d,e), (d,s) suffices (to make the graph strongly connected).

(c) *(4 points)* **Execute breadth-first search on the given graph starting at vertex s. List the vertices along with their distances from s in the order that the vertices are enqueued during the BFS.**

Vertex	Distance from s
s	0
a	1
c	1
d	1
b	2
e	2

NAME: Student ID:

KEY

QUESTION 3. *Peak* **Finding**

Let $A[1...n]$ be an array of n distinct numbers. The sequence of numbers in A has a *peak* at index i if the sub-array $A[1...i]$ is increasing (i.e., $A[j] < A[j+1]$ for $1 \le j \le i-1$) and the sub-array $A[i...n]$ is decreasing (i.e, $A[j] > A[j+1]$ for $i \le j < n$). The index i is called the *peak* of A.

For example, the array $[4, 6, 9, 5, 1]$ has a *peak* 3 since the array elements from index 1 to index 3 are in increasing order, and the array elements from index 3 to index 5 are in decreasing order.

Design an efficient algorithm that, when given an array A of n distinct numbers *that has a peak*, returns the index i of the peak. (NOTE: A linear-time algorithm will not receive points.)

(a) *(3 points)* **Give an English description of your algorithm.**

If an element is peak, it will be greater than its previous and next element

| 0 → i ← n
| increasing decreasing.

If A[mid-1] < A[mid] > A[mid+1] then mid is peak of A.

If A[mid-1] < A[mid] < A[mid+1] then peak will lie on right subarray (between mid+1 and high) since mid element is in increasing part of the array.

If A[mid-1] > A[mid] > A[mid+1] then peak will lie on left subarray.

If subarray size is 1 then that index will be peak since A is guaranteed to have a peak.

(b) *(4 points)* **Give pseudocode of your algorithm.**

PeakFind (A, 1, n).

```
Procedure : PeakFind (A, low, high) {
        if (low >= high)
            return low;
        mid = (low + high) / 2 ;
        if (A[mid] > A[mid-1] && A[mid] > A[mid+1])
            return mid;
        Else if (A[mid] > A[mid-1] && A[mid+1] > A[mid])
            return PeakFind (A, mid+1, high);
        Else return PeakFind (A, low, mid-1);
}
```

KEY

(c) *(3 points)* **Analyze the runtime of your algorithm.**

I divide array into two equal-sized subarray.
Decide in constant time which subarray to solve &
solve only one subarray.

Hence, $T(n) = T(n/2) + O(1)$.

Using master's theorem ($a=1, b=2, d=0$;
$d = \log_b(a)$)

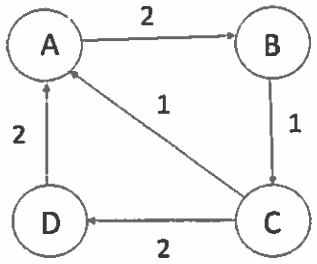
$T(n) = O(\log n)$.

KEY

QUESTION 4. Minimum-Weight Cycle Finding

Let $G = (V, E)$ be a positively-weighted, directed graph. Design an efficient algorithm to find the weight of the minimum-weight simple cycle in the graph. To be considered "efficient", your algorithm's big-O runtime complexity must be no more than a factor of $|V|$ larger than some runtime complexity given on the cover sheet of the exam.

A *simple cycle* is a cycle in which no vertex or edge is repeated, except the starting or the ending vertex. The weight of a cycle is the sum of the weights of the edges in the cycle.
[Hint: A cycle may be viewed as being composed of an edge $u \to v$ and a path $v \rightsquigarrow u$.]



For example, in the graph shown above, $A \to B \to C \to A$ is the minimum-weight simple cycle, with weight = 4. For the above graph, your algorithm should return the weight value 4.

(a) *(3 points)* Give an English description of your algorithm.

A simple cycle is composed of an edge $u \to v$ and a path $v \rightsquigarrow u$

We find shortest paths for all pairs (v, u) by running Dijkstra's algorithm starting from each vertex $v \in V$. We will label the length of shortest path $dist(v, u)$

Next, over all edges $(u', v') \in E$, we find the cycle which has $\min\left(\ell(u', v') + dist(v', u')\right)$

(b) *(4 points)* **Give pseudocode of your algorithm.**

$\ell \rightarrow$ weights of edges

```
Min_simple_cycle (G, ℓ)

for every v ∈ V :
    dist(v, u) = dijkstra (G, ℓ, v)

min = ∞
for every (u', v') ∈ E :
    if ℓ(u', v') + dist(v', u') < min :
        min = ℓ(u', v') + dist(v', u')
return min
```

(c) *(3 points)* **Analyze the runtime of your algorithm.**

Dijkstra's algorithm takes $O((|V|+|E|)\log|V|)$ time

Running time on $|V|$ vertices $= |V| \times O((|V|+|E|)\log|V|)$

Finding min $= O(|E|)$ time

Total running time $= O((|V|+|E|)|V|\log|V|) + O(|E|)$

$= O((|V|+|E|)|V|\log|V|)$