University of Nevada, Las Vegas Computer Science 477/677 Fall 2026

Answers to Assignment 5: Due Saturday November 7 2026

1. Fill in the blanks.

   (i) **Dijkstra's** algorithm does not permit any arc to have negative weight.

   (ii) The expected height of a treap with n nodes is $\Theta(\log n)$.

   (iii) The maximum possible number of edges of a planar graph with 5 nodes is **9**. (The formula is $3n-6$), provided $n \geq 3$.

   (iv) The worst case time complexity of Johnsonn's algorithm on a weighted directed graph with n vertices and m arcs is $\Theta(m \log n)$.

2. For each of the code fragments below, find the asymptotic time complexity in terms of n.

   (i)
   ```
   for(float x = n; x > 1; x = x/2)
   ```
   $\Theta(\log n)$

   (ii)
   ```
   for(int i = 0; i*i < n; i++)
   ```
   $\Theta(\sqrt{n})$

   (iii)
   ```
   for(float x = n; x > 2.0; x = sqrt(x))
   ```
   $\Theta(\log \log n)$

   (iv)
   ```
   for(int i = 1; i < n; i=2*i)
   ```
   $\Theta(\log n)$

   (v)
   ```
   for(int i = n; i > 1; i--)
     for(int j = i; j > 1; j = j/2)
   ```
   $\Theta(n \log n)$

   (vi)
   ```
   for(int i = n; i > 1; i--)
     for(int j = n; j > i; j = j/2)
   ```
   $\Theta(n)$

   (vii)
   ```
   for(int i = 2; i < n; i=i*i)
   ```
   $\Theta(\log \log n)$

3. Give asymptotic solutions to the following recurrences.

   (i) $F(n) = 2F(n/2) + n$

   $F(n) = \Theta(n \log n)$

(ii) $F(n) = F(n/2) + F(n/3) + n$;

$F(n) = \Theta(n)$

(iii) $F(n) = 4F(2n/5) + F(3n/5) + n^2$

$F(n) = \Theta(n^2 \log n)$

(iv) $F(n) = F(n - \sqrt{n}) + 1$

$F(n) = \Theta(\sqrt{n})$

(v) $F(n) = 3F(n/9) + 1$

$F(n) = \Theta(\sqrt{n})$

(vi) $F(n) = 9F(n/3) + n^2$

$F(n) = \Theta(n^2 \log n)$

(vii) $F(n) = F(\sqrt{n}) + \log n$

$F(n) = \Theta(\log n)$

4. What follows is part of a C++ implementation of a linked list implementation of stack of integer. Fill in the missing code.

```
struct staknode;
typedef staknode*stak;
struct staknode
 {
  int itm;
  stak lnk;
 };

void push(stak&top, int n)
 {
  stak tmp = new staknode;
  tmp->itm = n;
  tmp->lnk = top;
  top = lnk;
 }

bool empty(stak top);
 {
  return top == nullptr;
 }
```

```
int pop(stak&top)
 {
  assert(not empty(top));
  rslt = top->itm;
  top = top->lnk;
  return rslt;
 }
```

5. What follows is part of a C++ implementation of a binary search tree. Fill in the missing code.

```
struct bstnode;
typedef bstnode*tree;
struct bstnode
 {
  int itm;
  tree lft;
  tree rgt;
 };

void insrt(tree&root,int n)
  // inserts n into the binary search tree.  Duplicates are not inserted.
 {
  if(root)
   if(n < root->itm) insrt(root->lft,n);
   else if(n > root->itm) insrt(root->rgt,n);
   else
  else
   {
    root = new bstnode;
    root->itm = n;
   }
 }

bool find(tree root,int n)
  // returns true if n is in the tree rooted at root, otherwise false.
 {
  if(root)
   if(root->itm == n) return true;
   else if(n < root->itm) return find(root->lft,n);
   else return find(root->rgt,n);
  else return false;
 }
```

6. Write pseudocode for the Bellman Ford algorithm. Do not forget the shortcut, and be sure to compute backpointers.

   Assume that the vertices are 0, 1, ... n, where 0 is the source. Assume that W(i,j) is the weight of the arc from i to j, and if there is no such arc, W(i,j) = ∞.

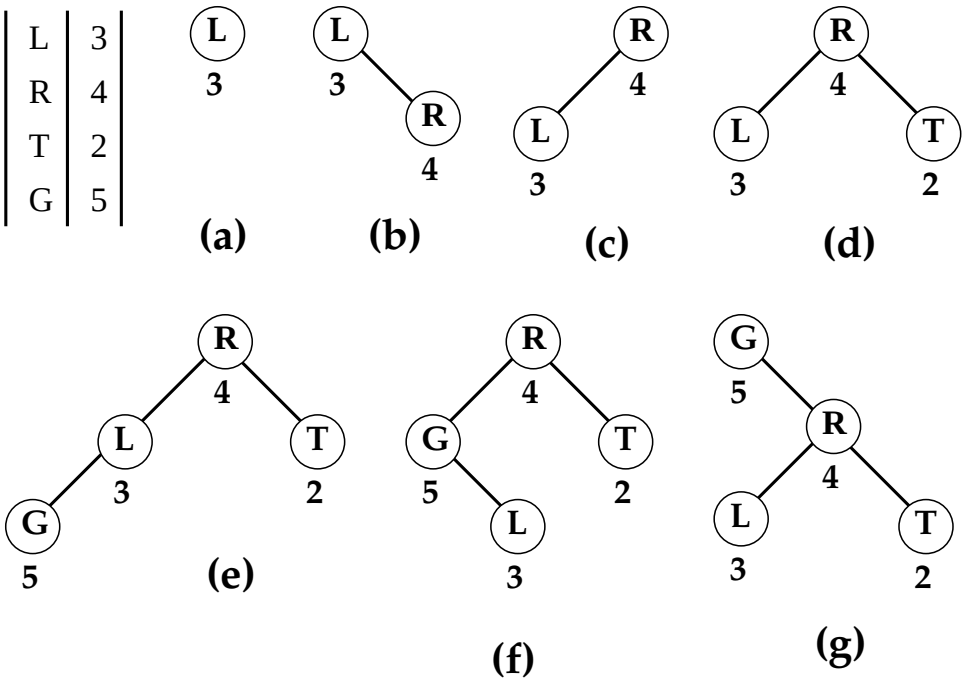
```
V(0) = 0
for all i from 1 to ∞
  V(i) = W(0,i)
  back(i) = 0
done = false
while(not done)
  done = true
  for all i from 0 to n
    for all j from 1 to n
      temp = V(i) + W(i,j)
      if(temp < V(j))
        V(j) = temp
        back(j) = i
        done = false
```

7. Write pseudocode for the Floyd Warshall algorithm. Be sure to compute backpointers. Assume that the vertices are 1, ... n. Assume that W(i,j) is the weight of the arc from i to j, and if there is no such arc, W(i,j) = ∞. We compute V(i,j) to be the least weight of any path from i to j, and back(i,j) is the backpointer of that least weight path.

```
for all i and j
  V(i,j) = W(i,j)
  back(i,j) = i
V(i,i) = 0
for all j from 1 to n
  for all i from 1 to n
    for all k from 1 to n
      temp = V(i,j)+V(j,k)
        if(temp ¡ V(i,k)
          V(i,k) = temp
          back(i,k) = back(j,k)
```

8. Insert the items L, R, T, G into a treap, in that order, using the priorities given in the table below, and using max-heap order. Show each a figure after each insertion or rotation.

L	3
R	4
T	2
G	5



9. The following C++ code computes values of a function.

```
int G(int n)
 {
  if(n <= 2) return n;
  else return G(n/2)+G(n/3)+G(n/6)+n*n;
 }
```

(i) What is the asymptotic value of $G(n)$? $G(n) = G(n/2) + G(n/3) + G(n/6) + n^2$ By Akra Bazzi, $G(n) = \Theta(n^2)$

(ii) What is the time complexity of computation of $G(n)$ using the recursive code above? Assume that each arithmetic operation takes constant time. $T(n) = T(n/2) + T(n/3) + T(n/6) + 1$. By Akra-Bazzi, $T(n) = \Theta(n)$

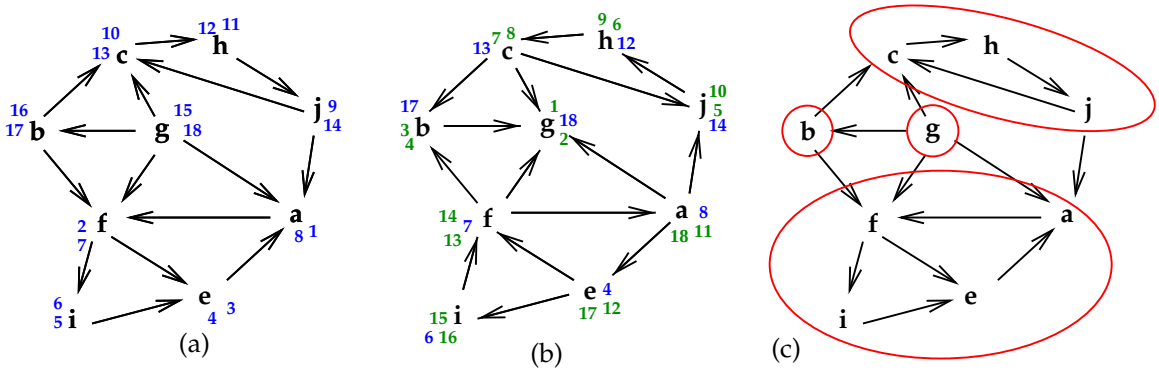
(iii) Write C++ code for a computation of $G(n)$ using dynamic programming, but not memoization. Assume that you are given the value of $n \geq 0$, as well as an array `int G(N)`, where N $> n$.

For i = 1 to n
If(n ¡= 2) G(n) = n
Else G(n) = G(n/2)+G(n/3) + G(n/6) + n*n

(iv) computation of $G(n)$ using memoization? Assume that each arithmetic operation and each fetch or store takes constant time.

The memos are for $G(i)$ for $i = n/(2^i * 3^i)$, provided the denominator is $O(n)$. There are $\Theta(\log^2 n)$ such values.

10. Find the strong components of the digraph shown in figure (a) below. Use the other two figures for your work.



11. Write, in correct C++ code, a dynamic programming algorithm for the coin-row problem. Assume that a sequence of positive numbers $X[N]$ is given, where $X[i]$ is the value of the i^{th} coin in a row. Find the maximum total value of any subset of the coins, with the restriction that no two coins which are adjacent in the original row can be in the subset.

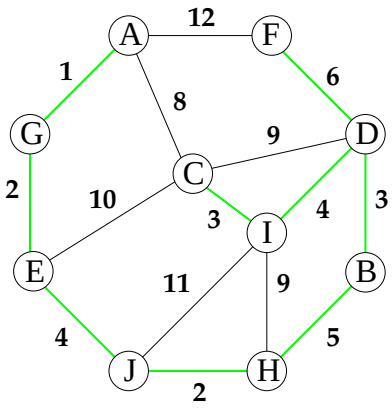
We let A[i] be the least cost of any legal subset of the first i coins. Thus, A[n] is the answer.

```
A[0] = 0; // default
A[1] = X[1];
for(int i = 2; i <= N; i++)
 A[i] = max(A[i-1], X[i]+A[i-2]);
```

12. Use union/find to implement Kruskal's algorithm for the weighted graph below. A step consists of deciding whether a given edge should be part of the minimum spanning tree. The first step considers the edge GH. find(G) = G, find(H) = H, and union(G,H) (arbitrarily) moves G into the set led by H.

    The first array shows the initial values of *parent* and *size* of each vertex The parent of each vertex is itself, and its size is 1, since its only follower is itself. At the first step, the edge $\{A, G\}$ is chosen for the spanning tree. In the second array, G is the leader of a set of size 2, namely $\{A, G\}$.

    Finish the execution of Kruskal's algorithm. At each step, show the matrix of parent and size values. Write "path comression" at each ste where path compression occurs. Attach extra sheets as needed.



		A	B	C	D	E	F	G	H	I	J
	parent	A	B	C	D	E	F	G	H	I	J
	size	1	1	1	1	1	1	1	1	1	1

Edge AG selected

		A	B	C	D	E	F	G	H	I	J
	parent	G	B	C	D	E	F	G	H	I	J
	size	1	1	1	1	1	1	2	1	1	1

Edge EG selected

		A	B	C	D	E	F	G	H	I	J
	parent	G	B	C	D	G	F	G	H	I	J
	size	1	1	1	1	1	1	3	1	1	1

Edge HJ selected

		A	B	C	D	E	F	G	H	I	J
	parent	G	B	C	D	G	F	G	J	I	J
	size	1	1	1	1	1	1	3	1	1	2

Edges BD,CI selected

		A	B	C	D	E	F	G	H	I	J
	parent	G	D	I	D	G	F	G	J	I	J
	size	1	1	1	2	1	1	3	1	2	2

Edges EJ,DI selected

		A	B	C	D	E	F	G	H	I	J
	parent	G	D	I	I	G	F	G	J	I	G
	size	1	1	1	2	1	1	5	1	4	2

path compression

Edge BH selected
leader(B) = I
leader(H) = G

		A	B	C	D	E	F	G	H	I	J
	parent	G	I	I	I	G	F	G	G	G	G
	size	1	1	1	2	1	1	9	1	4	2

path compression

Edge DF selected
leader(D) = G
leader(F) = F

		A	B	C	D	E	F	G	H	I	J
	parent	G	I	I	G	G	G	G	G	G	G
	size	1	1	1	2	1	1	10	1	4	2