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

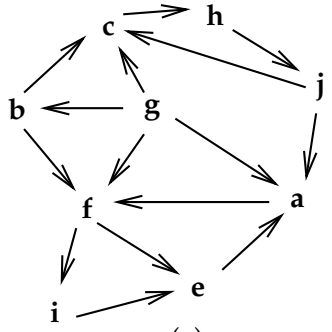
Assignment 5: Due Saturday April 11, 2026 11:59:59 PM

Name:_____

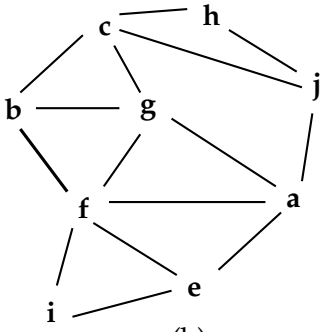
1. Fill in the blanks.
   (a) In _____ hashing, there are no collisions.
   (b) In _____ hashing, each data items has multiple possible hash values.

2. It is advertised that SHA-256 (secure hasing algorithm 256-bit) gives unique hash values to data. Select which of these statements is true.
   (a) Collisions in SHA-256 are impossible.
   (b) SHA-256 is designed so that, in practice, collisions occur only a few times a year, and they are repaired as needed.
   (c) In SHA-256, the probability of a collision is greater than zero, but so small that it is ignored.

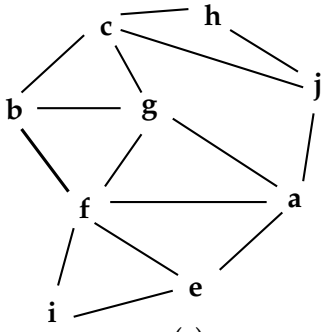
3. Find the strong components of the digraph shown in figure (a) below. Use the other two figures for your work.



(a)



(b)



(c)

4. Here is C++ code for quicksort. Assume that the global array `int A[n]` has been declared. The goal is to sort A.

```
int const n = 20;
int lo;
int hi;

int A[n];

void swap(int&x, int&y)
 {
  int z = x;
  x = y;
  y = z;
 }

void display()
 {
  for(int i = 0; i < n; i++)
  cout << " " << A[i];
  cout << endl;
 }

void quicksort(int first, int last)
 // sorts A[first..last]
 {
  assert(0 <= first and first <= last and last < n);
  if(first < last) // What happens if first = last?
   {
    int pivot;
    int mid = (first+last)/2;
    pivot = A[first];
    swap(A[first],A[mid]);
    pivot = A[first];
    lo = first;
    hi = last;
    while(lo < hi) // This is the main loop
     {
      if(A[hi] < A[lo+1])
       swap(A[hi],A[lo+1]);
      while(A[lo+1] < pivot) lo++;
      while(A[hi] > pivot) hi--;
     }
```

```
    assert(lo == hi);
    swap(A[first],A[lo]);
    quicksort(first,lo);
    quicksort(lo+1,last); // Oops!  There might be a bug here!
   }
 }

int main()
 {
  display();
  quicksort(0,n-1);
  display();
  return 1;
 }
```

The output of my program:

```
70 29 18 45 91 58 64 97 95 67 92 19 63 27 84 73 60 72 42 99
18 19 27 29 42 45 58 60 63 64 67 70 72 73 84 91 92 95 97 99
```

Answer the following questions.

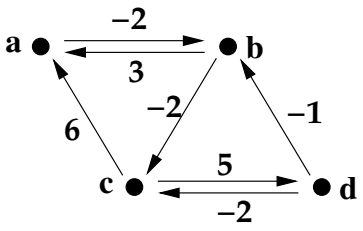
(a) What is the loop invariant of the main loop?

(b) I was careful not to have any duplicate entries in my initial array, the first line of my output. Did I really need that condition for the program to work correctly? Explain.

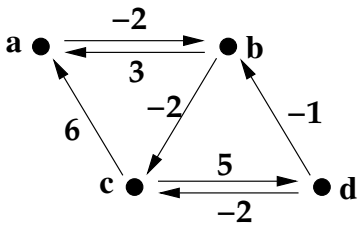
(c) I treated the case where the array has length 2 with special code. Was that necessary?

(d) There might be a bug in the program, as indicated at the second recursive call. Can you find it? Even if there is a bug, the program might run correctly, but it might not.

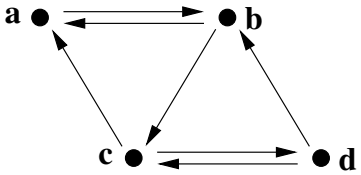
5. Compute the first phase of Johnson's algorithm using the weighted graph below. There are three copies. The first copy shows the original weights. Use the other two copies for your work. The third copy should show the adjusted weights. Do not finish Johnson's algorithm.



(a)



(b)



(c)

6. Write a C++ program to find the Levenstein edit distance between two words. Here is a possible sample run of your program.

```
Enter a word of length no more than 20: quantumchromodynamics
That word is too long.  Try again.
chromodynamics
Enter another word of length no more than 20: crommodynamiks
The Levenstein distance from chromodynamics to crommodynamiks is 3
```

Do not send a photocopy or something that is written by hand. Your C++ program should be submitted in a format that can be executed and run. preferably plain text.

Hint: Get started right away, not Saturday evening.