Computer Science 477/677 Spring 2019
University of Nevada, Las Vegas
Answers to Final Examination May 14, 2020

Final update: `Fri May 15 17:15:27 PDT 2020`

Print out this test and write your answers on the printout, then scan the pages and email to the GA, Shekhar Singh.with a time stamp of either May 14 PDt or May 15 PDT. If you are in a different time zone, be sure to adjust for that. For example, if you are in New York, your time stamp must be no later than 03:00 May 16 EDT, while if you are in Hawaii, your time stamp must be no later than May 15 21:00 HST.

If you need to attach extra pages, use white paper. If you are unable to print the test, you may write the answers on white paper. Mot lined paper, and not any other color. Please write large and dark enough that your answers are unambigurous Do **not** use lined paper, as it does not scan very well. Use only bright white paper. Make sure your name is on every page, in case pages get separated.

The entire exam is 330 points.

Read this first. Each answer to problems 3, 4, 5, and 13 is O, Ω, or Θ, of one of the following functions: n, n^2, n^3, n^3, $\sqrt{n}$, $\log n$, $\log^2 n$, $\log\log n$, $\log^* n$, $\log n \log\log n$, $n \log n$, $n^2 \log n$, 2^n.

1. True or False. Write "O" if the answer is not known to science at this time. [5 points each]
   (a) **F** All sparse graphs are planar.
   (b) **T** The time complexity of quicksort is $O(n^2)$.
   (c) **T** The time complexity of quicksort is $\Omega(n \log n)$.
   (d) **O** If a problem is $\mathcal{NP}$-complete, there is no polynomial time algorithm which solves it.
2. Fill in the blanks. [5 points each blank.]
   (a) **Dijkstra's** algorithm for solving the single source shortest path problem only works correctly if there are no negative weights.
   (b) In dynamic programming the subproblems must be worked in **topological** order.
   (c) Vertices u and v of a graph G belong to different **components** if there is no path in G from u to v.
   (d) If there is no directed path in a directed graph from vertex u to vertex v, then u and v belong to different **strong components**.
   (e) An $n \times n$ matrix with $n \log n$ non-zero entries would probably be considered **sparse**.
3. Solve the recurrences. Give asymptotic answers in terms of n, using either O, Ω, or Θ, whichever is most appropriate. (5 points each)

(a) $F(n) = 2F\left(\frac{n}{2}\right) + n$

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

(b) $F(n) \geq 4F\left(\frac{n}{2}\right) + n^2$

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

(c) $F(n) = F(n-1) + \frac{n}{4}$

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

(d) $F(n) = F\left(n - \sqrt{n}\right) + \sqrt{n}$

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

(e) $F(n) = F(\log n) + 1$

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

(f) $T(n) < T(n-2) + n^2$

$T(n) = O(n^3)$

(g) $G(n) \geq G(n-1) + n$

$G(n) = \Omega(n^2)$

(h) $H(n) \leq 2H(\sqrt{n}) + O(\log n)$.

$H(n) = O(\log n \log\log n)$

(i) $F(n) = F(n/2) + 1$

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

(j) $F(n) = F(n-1) + O(\log n)$

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

(k) $F(n) = F\left(\frac{n}{2}\right) + 2F\left(\frac{n}{4}\right) + n$

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

(l) $F(n) = F\left(\frac{3n}{5}\right) + F\left(\frac{4n}{5}\right) + n^2$

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

4. Find the asymptotic complexity, in terms of n, for each of these fragments, expressing the answers using O, Θ, or Ω, whichever is most appropriate. (5 points each)

(a) `for(int i = 0; i < n; i = i+1)`

$\Theta(n)$

(b) `for(int i = 1; i < n; i = 2*i)`

$\Theta(\log n)$

(c) `for(int i = 2; i < n; i = i*i)`

$\Theta(\log\log n)$

(d)
```
for(int i = n; i > 0; i--)
  for(int j = i; 2*j <= n; j = 2*j)
```
$\Theta(n)$

(e)
```
for(int i = n; i > 0; i--)
  for(int j = 1; 2*j <= i; j = 2*j)
```
$\Theta(n \log n)$

(f)
```
for(int i = 1; i < n; i++)
  for(int j = n; j > 0; j = j/2)
```
$\Theta(n \log n)$

(g)
```
for(int i = n; i > 1; i = sqrt(i));
```
$\Theta(\log \log n)$

(h)
```
for(int i = 1; i*i < n; i++)
```
$\Theta(\sqrt{n})$

5. [10 points each] Give the asymptotic time complexity, in terms of n, for each of these recursive subprograms.

(a)
```
int f(int n)
 {
  if (n < 2) return 1;
  else return f(n-1)+f(n-1);
 }
```
$\Theta(2^n)$

(b)
```
void hello(int n)
  {
   if(n >= 1)
    {
     for(int i = 1; i < n; i++)
      cout << "Hello!" << endl;
     hello(n/2);
     hello(n/2);
    }
  }
```
$\Theta(n \log n)$

6. [30 points] Which of the following problems are known to be NP-complete? Mark T or F.

(a) **T** Given a weighted graph and a number B, does the graph have a Hamiltonian cycle of weight at most B?

(b) **T** Given a table and a set of tiles of various shapes, can the tiles all be placed on the table so that none overlap and none overhang the edge?

(c) **F** Given a weighted graph, a number D, and two vertices u and v, does there exist a path between u and v of weight at most D?

7. [20 points] What is the name of the algorithm implemented by the following code?

```
int x[n];
read in values of x from some external source;
for(int i = 0; i < n; i++)
 for(j = i+1, j < n; j++)
  if(x[j] < x[i]) swap(x[i],x[j]);
```

Selection sort.

8. [20 points] A directed graph can be reprented in the computer in several ways. Two of them are an array of out-neighbor lists, and an array of in-neighbor lists. Let G be a directed graph whose vertices are the integers $0 \ldots 19$, and whose array of out-neighbor lists is as written below. Construct the array of in-neighbor lists of G.

```
Out-neighbor lists:                          In-neighbor lists:
 0:  1,13                                      0:
 1:  5,19                                      1: 0,13
 2:  5,14                                      2: 12,13
 3:  9                                         3: 19
 4:  7,15                                      4: 5,10
 5:  4,10                                      5: 1,2,19
 6: 11                                         6: 12
 7: 16,18                                      7: 4,11
 8: 17                                         8: 9
 9:  8,15                                      9: 3
10:  4                                        10: 5
11:  7                                        11: 6,14
12:  2, 6                                     12:
13:  1, 2                                     13: 0
14: 11                                        14: 2
15: 17,18                                     15: 4,9
16:                                           16: 7
17:                                           17: 8,15
18:                                           18: 7,15
19: 3, 5                                      19: 1
```

9. [20 points] Give pseudocode for the Floyd-Warshall algorithm. The vertices are the integers $1, 2, \ldots n$. $W[i, j]$ is the given weight of the directed edge from i to j, which is ∞ if there is no edge from i to j. The output of the algorithm is $F[i, j]$ for all i and j, the minimum length of any directed path from i to j, as well as the backpointer $B[i, j]$ for all $i \neq j$. Assume the graph is strongly connected and that there are no negative cycles.

```
for(int i = 1; i <= n; i++)
  for(int j = 1; j <= n; j++)
   {
    F[i,j] = W[i,j];
    B[i,j] = i;
   }
for(int i = 1; i <= n; i++)
  F[i,i] = 0;
for(int j = 1; i <= n; j++)
  for(int i = 1; i <= n; i++)
    for(int k = 1; k <= n; k++)
     {
      temp = F[i,j]+F[j,k];
      if(temp < F[i,k])
       {
        F[i,k] = temp;
        B[i,k] = B[j,k]
       }
     }
```

10. [20 points] Let the function F on positive integers be defined as follows:

$$F(n) = \begin{cases} 1 \text{ if } n <= 2 \\ \left(F(\frac{n}{2}) * (F(\frac{n-2}{2}) + F(\frac{n+2}{2})\right) \; mod \; 23 \text{ if } n > 2 \text{ and } n \text{ is even} \\ \left(F(\frac{n-1}{2})^2 + (F(\frac{n+1}{2})^2\right) \; mod \; 23 \text{ if } n > 2 \text{ and } n \text{ is odd} \end{cases}$$

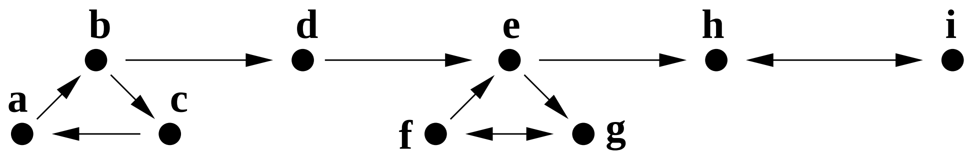
You can compute $F(n)$ for any n in $O(n)$ time using dynamic programming. However, the computation can be done much faster by using memoization.

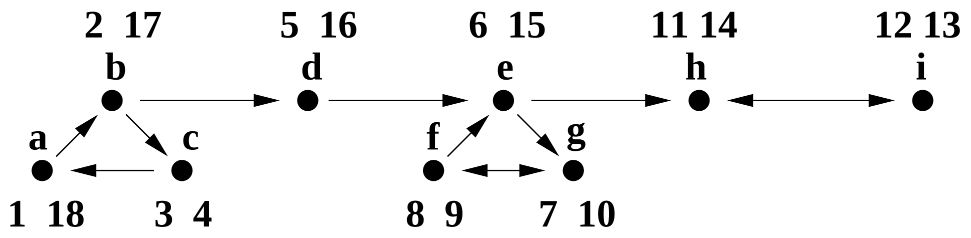
Write pseudo-code for a memoization algorithm which prints F(n) for some given n in less than linear time.

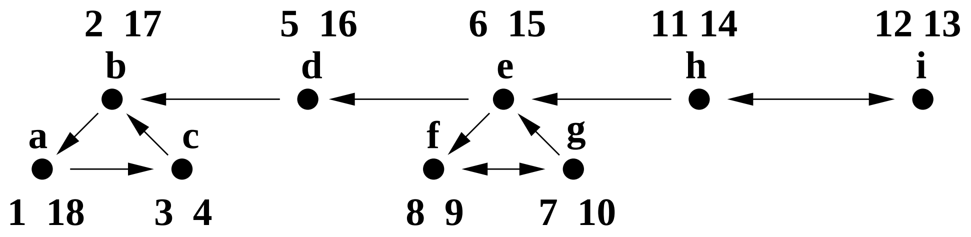
```
int F(int n)
 {
  if (n <= 2) return 1;
  else if (the memo (n,f(n) already exits)
   return f(n);
  else
   {
    if (n is even)
```

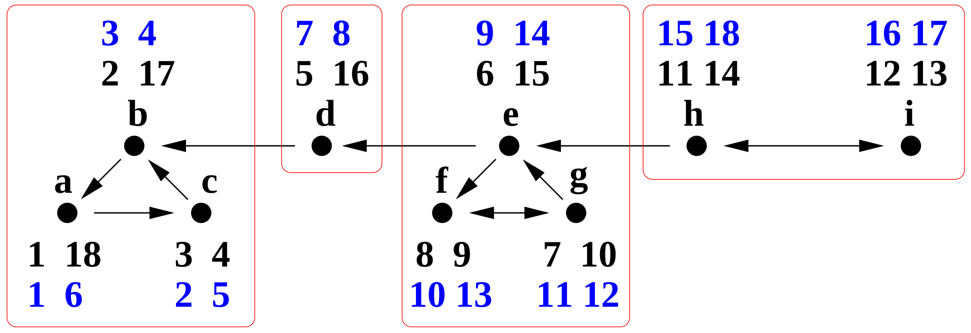
```
      f(n) = F(n/2)*F((n-1)/2)+F((n+2)/2);
     else
      f(n) = (F((n-1)/2))^2+(F((n+1)/2)^2;
     memoize (n,f(n));
     return f(n);
    }
  }
```

11. [20 points] Use the DFS-based algorithm in our textbook to find the strong components of the directed graph G shown below. Of course, you can easily "eyeball" the answer, but I want to see the steps of the algorithm.

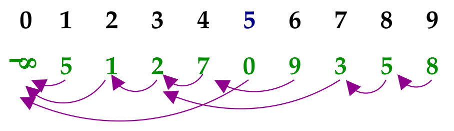








12. [20 points] Find the longest increasing subsequence of the sequence 5, 1, 2, 7, 0, 9, 3, 5, 8, using the dynamic programming algorithm I showed you in the video. Show all work.



back	0	0	2	3	0	4	3	7	8
length	1	1	2	3	1	4	3	4	5

best index of length	~~1~~	3	~~4~~	~~6~~	9
	~~2~~		7	8	
	5				

13. [20 points] **This problem requires serious thought.** Consider the following code:

```
for(int i = 2; i < n*n; i = i*i)
  for(int j = 1; j < i; j++)
```

The asymptotic time complexity of this code is not Θ of any of the functions of n listed at the beginning of the test, but it is O of one of those functions, and is also Ω of another one of those functions. Find the two functions. (Hint: When you are faced with the problem of finding an unknown formula, it frequently helps to experiment by choosing some test numbers. Even bigger hint: try n = 16,17,256, and 257.)

$O(n^2)$ and $\Omega(n)$