Computer Science 477/677 Fall 2021

Answers to Practice for the Final Examination December 8, 2021

The entire practice test is 675 points.

1. True or False. Write "O" if the answer is not known to science at this time. [5 points each]

   (a) **O** There is a polynomial time algorithm for finding the factors of any integer written as a base ten numeral.

   (b) **T** Planar graphs are sparse.

   (c) **T** Tree graphs are sparse.

   (d) **F** Acyclic directed graphs are sparse.

2. Fill in the blanks.

   (a) [5 points] If a planar graph $\mathcal{G}$ has 5 vertices, then the number of edges of $\mathcal{G}$ cannot be more than **9**.

   (b) [5 points] The height of a binary tree with 10 nodes is at least **3**.

   (c) [5 points] A directed graph has a topological order if and only if it is **acyclic**.

   (d) [15 points] **stack**, **queue**, and **heap** are three examples of priority queues.

   (e) [5 points] The operators of the ADT **stack** are *pop* and *push*.

   (f) [5 points] The operators of the ADT **array** are *fetch* and *store*.

   (g) [10 points] In order to solve a shortest path problem on a weighted directed graph, there must be no **negative cycle**.

   (h) [10 points] The following is pseudo-code for what algorithm?

   selection sort

```
int x[n];
obtain values of x;
for(int i = 0; i < n; i++)
 for(int j = i+1; j < n; j++)
  if(x[i] > x[j])
   swap(x[i],x[j]);
```

   (i) [10 points] The asymptotic time complexity of Johnson's algorithm on a weighted directed graph of n vertices and m arcs is $O(nm \log n)$.

   (j) [5 points] The time complexity of every comparison-based sorting algorithm is $\Omega(n \log n)$.

   (k) [10 points] The postfix expression $xy + xz - *$ is equivalent to the infix expression $(x + y) * (x - z)$.

   (l) [10 points] The items stored in a priority queue (that includes stacks, queues, and heaps) represent **unfufilled obligations**. (2 words)

   (m) [10 points] Name two divide-and-conquer sorting algorithms.

   quicksort and **mergesort**

3. Give the asymptotic complexity, in terms of `n`, of each of the following code fragments. [10 points each]

(a)
```
for(i = 0; i < n; i = i+1);
cout << "Hi!" << endl;
```
$\Theta(n)$

(b)
```
for(int i = n; i > 1; i = i/2)
  cout << "hello world" << endl;
```
$\Theta(\log n)$

(c)
```
for(int i = 1; i < n; i++)
  for(int j = 1; j < i; j = 2*j)
    cout << "hello world" << endl;
```
$\Theta(n \log n)$

(d)
```
for(int i = 1; i < n; i++)
  for(int j = i; j < n; j = 2*j)
    cout << "hello world" << endl;
```
$\Theta(n)$

(e)
```
for(int i = 2; i < n; i = i*i)
  cout << "hello world" << endl;
```
$\Theta(\log \log n)$

(f)
```
for(i = 1; i < n; i = 2*i);
cout << "Hi!" << endl;
```
$\Theta(\log n)$

(g)
```
for(int i = n; i > 1; i = sqrt(i));
 cout << "Hi!" << endl;
```
$\Theta(\log \log n)$

4. For each of these recursive functions, write, and then solve, an appropriate recurrence. [10 points each]

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

(b)
```
void g(int n)
{
 if(n >= 1)
  {
   for(int i = 1; i < n; i++)
    cout << "Hi!" << endl;
   g(n/3);
   g(n/3);
   g(n/3);
  }
}
```
$g(n) = 3g(n/3) + n$
$\Theta(n \log n)$

5. Solve the recurrences. Give the asymptotic value of $F(n)$ using either O, Ω, or Θ, whichever is most appropriate. [10 points each]

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

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

(c) $F(n) = F(n-1) + \frac{n}{4}$ $\Theta(n^2)$

(d) $F(n) \leq F\left(\frac{n}{2}\right) + F\left(\frac{n}{4}\right) + F\left(\frac{n}{5}\right) + n$ $O(n)$

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

(f) $F(n) = F(\log n) + 1$ $\Theta(\log^* n)$

(g) $T(n) < T(n-2) + n^2$ $O(n^3)$

(h) $F(n) \geq F(\sqrt{n}) + \lg n$ $\Omega(\log n)$

(i) $G(n) \geq G(n-1) + n$ $\Omega(n^2)$

(j) $F(n) = 4F(n/2) + n^2$. $\Theta(n^2 \log n)$

(k) $F(n) \leq 2F(\sqrt{n}) + O(\log n)$. $O(\log n \log\log n)$

(l) $K(n) = K(n - \sqrt{n}) + 1$. $\Theta(\sqrt{n})$

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

(n) $F(n) = F(n/2) + 1$ $\Theta(\log n)$

(o) $F(n) = F(n-1) + O(\log n)$ $O(n \log n)$ and $\Omega(n)$

(p) $F(n) = F\left(\frac{n}{2}\right) + 2F\left(\frac{n}{4}\right) + n$ $\Theta(n \log n)$

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

(r) $F(n) = F(n-2) + n$ $\Theta(n^2)$

6. [20 points] Use dynamic programming to compute the length of the longest common subsequence of the strings "011011001" and "1010011001." Show the matrix.

		1	0	1	0	0	1	1	0	0	1
	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1
1	0	1	1	2	2	2	2	2	2	2	2
1	0	1	1	2	2	2	3	3	3	3	3
0	0	1	2	2	3	3	3	3	3	3	3
1	0	1	2	3	3	3	4	4	4	4	4
1	0	1	2	3	3	3	4	5	5	5	5
0	0	1	2	3	4	4	4	5	6	6	6
0	0	1	2	3	4	5	5	5	6	7	7
1	0	1	2	3	4	5	6	6	6	7	8

The longest common subsequences are 01011001 and 11011001 which have length 8.

7. [20 points] Use dynamic programming to compute the Levenshtein distance between the strings "kitchen" and "chicken." Show the matrix.

		c	h	i	c	k	e	n
	0	1	2	3	4	5	6	7
k	1	1	2	3	4	4	5	6
i	2	2	2	2	3	4	5	6
t	3	3	3	3	3	4	5	6
c	4	3	4	4	3	4	5	6
h	5	4	3	4	4	3	4	5
e	6	5	4	4	5	4	3	4
n	7	6	4	5	6	5	4	3

The Levenshtein distance is 3.

8. [20 points] Describe a dynamic programming algorithm to compute the maximum sum of any contiguous subsequence of a given sequence of numbers. For example, if the given sequence is

$$2, 1, -4, 6, -3, 7, -1.2. - 2, 1$$

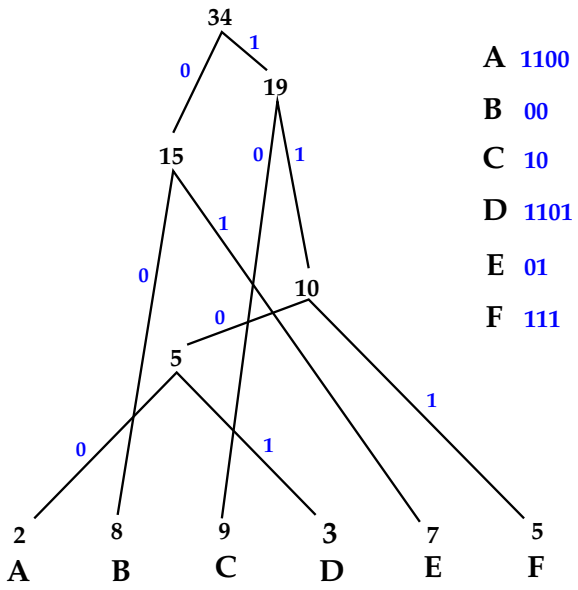
that sum is $6 + (-3) + 7 + (-1) + 2 = 11$. (There is a $\Theta(n)$-time algorithm.)

Let x[1], x[2], ... x[n] be the sequence. For each i, let A[i] be the maximum sum of any contiguous subsequence of the first i terms, and let B[i] be the maximum sum of any contiguous subsequence whose last term is x[i].

```
A[0] = 0;
B[0] = 0;
for(int i = 1; i <= n; i++)
 {
  B[i] = x[i] + max(0,B[i-1]);
  A[i] = max(B[i],A[i-1])l
 }
cout << A[n] << endl;
```

9. Use Huffman's algorithm to construct an optimal prefix code for the alphabet $\{A, B, C, D, E, F\}$ where the frequencies of the symbols are given by the following table.

A	2
B	8
C	9
D	3
E	7
F	5



10. [10 points] Write pseudo-code for binary search.

We have an array A[1] ... A[n] of numbers such that A[i] $\leq$ A[i+1] for all i. We are given a number x, and need to find x in the array.

```
int lo = 1;
int hi = n;
bool found = false;
while(lo <= hi and not found)
 {
  int mid = (lo+hi)/2;
  if(A[mid] == x)
   {
    found = true;
    cout << x << " = A[" << mid << "]" << endl;
   }
   else if (A[mid] < x) lo = mid+1;
   else if (A[mid] > x) hi = mid-1;
 }
if(hi < lo)
 cout << x << " not found" << endl;
```

11. [20 points] What does the following recursive function do? Hint: it is important for cryptography.

```
int f(int m, int n, int e)
 {
  assert(m > 0 and n > 0 and e >= 0);
  if(e == 0) return 1;
  else if(e%2) return f(m,n,e-1)*m%n;
  else
   {
    temp = f(m,n,e/2);
    return temp*temp%n;
   }
 }
```

It computes m^e mod n. It is used as follows. m is the message we need to encode. e and n are public. e is somewhere on the order of 10,000, while n is hundreds of digits long, and is the product of two secret primes. The encrypted message is $y = m^e$ mod n. The original message is $m = y^d$ mod n, where d is the inverse of e in the ring of integers relatively prime to n, modulo n . As yet, no one knows how to find d, in polynomial time without knowing the secret factors of n, therefore, for now, RSA encryption works.

12. [20 points] The usual recurrence for Fibonacci numbers is:
$F[1] = F[2] = 1$
$F[n] = F[n-1] + F[n-1]$ for $n > 2$

However, there is a more efficient recurrence:
$F[1] = F[2] = 1$
$F[n] = F\left[\frac{n-1}{2}\right] * F\left[\frac{n}{2}\right] + F\left[\frac{n+1}{2}\right] * F\left[\frac{n+2}{2}\right]$ for $n > 2$
where integer division is truncated as in C++.

Using that recurrence, Describe a $\Theta(\log n)$-time memoization algorithm which reads a value of n and computes $F[n]$, but computes only $O(\log n)$ intermediate values.

We use a search structure which contains memos. Each memo is an ordered pair $(i, F[i])$ The search structure is initaially empty. We compute the n^{th} Fibonacci number by evaluating $fib(n)$ in the code below.

```
int fib(int i)
 // input condition: i >= 1
 {
  int f;
  if there is a memo (i,f) return f;
  else
   {
    if (i <= 2)
     f = 1;
    else
     f = fib((n-1)/2)*fib(n/2)+fib((n+1)/2)*fib(n+2)/2));
    store the memo (i,f);
    return f;
   }
 }
```

fib(i) is actually stored for no more than $4 \log_2 n$ values of i.

13. [20 points] Describe a randomized algorithm which finds the k^{th} smallest element of an unsorted list of n distinct numbers, for a given $k \leq n$, in $O(n)$ expected time. (By "distinct," I mean that no two numbers in the list are equal.)

Input condition: $1 \leq$ k $\leq$ n. Let x[1] ... x[n] be the unsorted list of numbers. They could be integers, float or whatever, as long as they are of an ordered type, which we call "our-type."

We assume the list is given. We write a recursive function `select(first,last,k)` which returns the k^{th} smallest item of the sublist x[first] ... x[last]. The k^{th} smallest item of the entire list is then simply the value of `select(1,n,k)`. Since the list items are known to be distinct, we don't have to be as careful with our code as we otherwise would.

```
our-type select(int first, int last, int k)
// input condition: first <= k <= last
 {
  if(first == last) return x[first]; // the sublist has length 1
  else
   {
    int r = randomly chosen integer in the range first ... last.
    our-type pivot = x[r];
    swap(x[r],x[first]) // swap the pivot into the first position
    bool done = false;
    int lo = first;
    int hi = last;
    while(lo < hi) // loop invariant: x[i] <= pivot for all i <= lo
                   // and x[i] > pivot for all i > hi
     {
      while(x[lo+1] < pivot) lo++;
      while(x[hi] > pivot) hi--;
      if(lo < hi) swap(x[lo+1],x[hi]);
     }
    // at this point lo = hi
    if(k <= hi-first+1) return select(first,hi,k);
    else return select(hi+1,last,k-hi+first-1);
   }
 }
```

14. [20 points] Give pseudocode for the Bellman-Ford algorithm.

We assume that the vertices are named 0,1 ... n, and that there are m arcs, where the j^{th} arc is the ordered pair (s_j, t_j) and has weight W_j, where both s_j and t_j are integers in the range 1 ... n. We compute V[i], the minimum weight of any directed path from 0 to i, as well as back[i], the next-to-the last vertex on that path. I have included the "shortcut" in the code, as well as a test for negative cycles.

```
V[0] = 0;
for(int i = 1; i <= n; i++)
 V[i] = infinity;
bool updated = true;
for(int k = 1; k <= n and updated; k++) // if not updated, we can stop
 {
  updated = false;
   for(int j = 1; j <= m; j++)
    {
     temp = V[s_j]+w_j;
     if(temp < V[t_j])
      {
       V[t_j] = temp;
       back[t_j] = s_j;
       updated = true;
      }
    }
 }
if(updated) cout << "Negative Cycle" << endl;
```

15. [20 points] Give pseudocode for the Floyd-Warshall algorithm.

Let the vertices be 1 ... n. Let W[i,j] be the weight of the edge from i to j. If there is no edge, W[i,j] defaults to infinity.

For all i,j in 1 ... n, we compute V[i,j], the smallest weight of any path from i to j, as well as back[i,j], the next-to-the last vertex on that path.

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

16. [20 points] Describe, in a very informal way, an $O(\log n)$ time algorithm which computes the median of two sorted lists of length n.

Let A and B be the sorted lists. Let a and b be the media of A and B, respectively. If a = b, that is the answer. If a < b, delete the first half of A and the second half of B, otherwise delete the second half of A and the first half of B. In either case, we must delete the same number of items on each side. The problem is now reduced to the same problem of half the size. Within $O(\log n)$ such steps, each list contains at most one item.

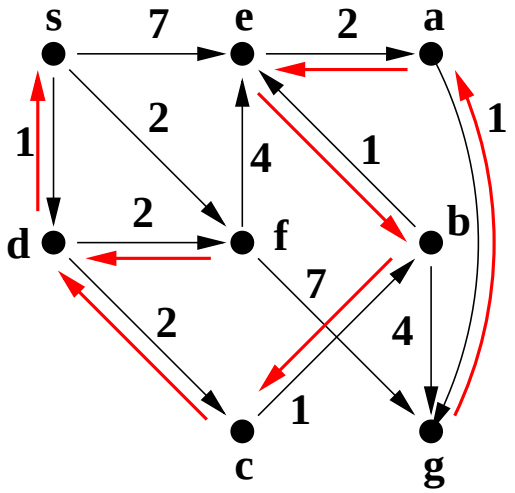
17. [20 points] State Hall's Marriage Theorem.

A *bipartite graph* is a graph (V, E) where V is the union of disjoint sets A and B, and where every edge connects a member of A with a member of B. The question is, does there exists a function $p : A \to B$ such that $\{x, p(x)\} \in E$ for all $x \in A$? Call such a function a *matching*.

If $x \in A$, write Nbrs(x) to mean the set of neighbors of A, which is, of course, a subset of B. For any set $S \subseteq A$, let Nbrs(S) = $\bigcup_{x \in S}$Nbrs(x), consisting of all members of B which are neighbors of some member of A.

Hall's Marriage Theorem states that there is a matching if and only if, for each S⊆A, Nbrs(S) has at least as many elements as S.

18. [20 points] Use Dijkstra's algorithm to solve the single source shortest path problem for the following weighted directed graph, where **s** is the source. Show the steps.



s	a	b	c	d	e	f	g
0	7	4	3	1	~~7~~ 5	2	~~9~~ 8
*	e	c	d	s	~~s~~ b	s	~~f~~ b

Let Q be the minqueue consisting of partially processed vertices. We write Q as a list of vertices, sorted by V. Initially, $Q = (\mathbf{s})$.

In the first step, we dequeue **s** and enqueue **d,e,f**. $Q = (\text{d,f,e})$

In the second step we dequeue **d** and enqueue **c**. $Q = (\text{f,c,e})$

In the third step, we dequeue **f**, enqueue **g**, and change **back(e)** to **f**, decreasing **V(e)** from 7 to 6. $Q = (\mathbf{c,e,g})$

In the fourth step, we dequeue **c** and enqueue **b**. Now, $Q = (\mathbf{b,e,g})$.

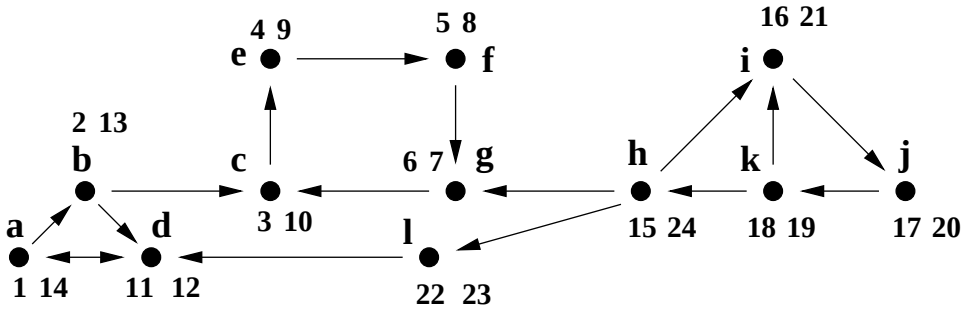
In the fifth step, we dequeue **b**. $Q = (\mathbf{e,g})$, and change **back(g)** to **b**, decreasing **V(g)** from 9 to 8. $Q = (\mathbf{e,g})$.

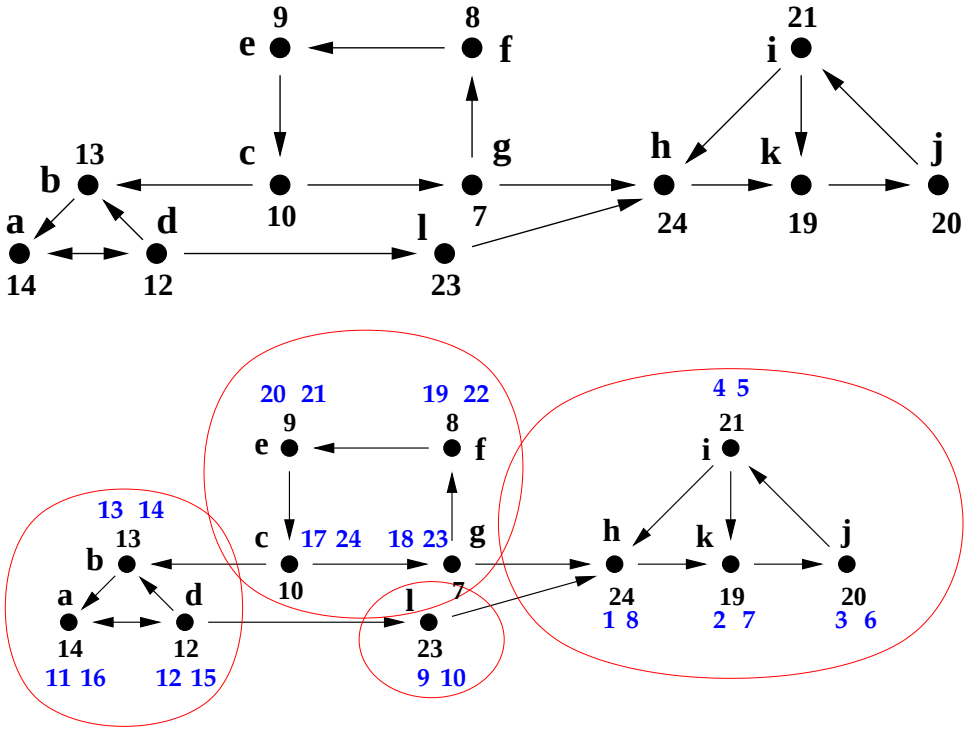
In the sixth step, we dequeue **e** and enqueue **a**. $Q = (\mathbf{a,g})$.

In the seventh step, we dequeue **a**. $Q = (\mathbf{g})$.

In the eighth and last step, we dequeue **g**. Q is now empty and we are done.

19. [20 points] Find the strong components of the following graph, using DFS search. (If you find a better algorithm than the one I showed you in class, you may use that instead.) Circle the strong components.





21. Consider the function george computed by the following recursive function.

```
int george(int n)
 {
  if(n <= 6) return 1;
  else return george(n/2)+george(n/2+1)+george(n/2+2)+george(n/2+3)+n*n;
 }
```

(a) [10 points] What is the asymptotic complexity of george(n), in terms of n? Use Θ notation.

The recurrence is $G(n) = 4G(n/2) + n^2$. The solution is $G(n) = \Theta(n^2 \log n)$.

(b) [10 points] What is the asymptotic time complexity of the recursive code, in terms of n? Use Θ notation.

The recurrence is $T(n) = 4T(n/2) + 1$. The solution is $T(n) = \Theta(n^2)$.

(c) [10 points] What is the asymptotic time complexity, in terms of n, of a dynamic program algorithm which computes george(i) for all i up to n? Use Θ notation.

$\Theta(n)$

(d) [20 points] What is the asymptotic time complexity, in terms of n, of a memoization algorithm which computes george(n)? Use Θ notation.

$\Theta(\log n)$.

20. [20 points] Show the minimum spanning tree of the following weighted graph. Show the evolution of the union-find structure. In case of a tie, always choose the largest letter.