University of Nevada Las Vegas

Computer Science 477/677 Fall 2020

Answers to General Practice Examination

`Sat Oct 10 10:49:05 PDT 2020`

The entire practice examination is 370 points.

1. True or False. [5 points each]

   (a) **F** Computers are so fast today that complexity theory is only of theoretical, but not practical, interest.

   (b) **F** If any problem can be precisely formulated in a mathematical way, there is an algorithm that solves it.

   (c) **T** Heapsort takes $\Theta(n \log n)$ time on an array of size n.

   (d) **T** The inverse Ackermann function, $\alpha(n)$, grows so slowly that, from a practical (as opposed to theoretical) point of view, it might as well be constant.

   (e) **O** If a problem is $\mathcal{NP}$-complete, there is no polynomial time algorithm which solves it.

   (f) **O** Quicksort takes $\Theta(n \log n)$ time on an array of size n.

2. Fill in the blanks. [5 points each blank.]

   (a) What is the **only** difference between the abstract data types *queue* and *stack*?

   Queue is FIFO, stack is LIFO.

   (b) D1 Name a divide-and-conquer searching algorithm.

   Binary Search

   (c) D2 Name two divide-and-conquer sorting algorithms.

   Quicksort

   Mergesort

   (d) The time complexity of every comparison-based sorting algorithm is $\Omega(n \log n)$ (Your answer should use Ω notation.)

   (e) The items stored in a priority queue (that includes stacks, queues, and heaps) represent **unfulfilled obligations**.

3. Write the following asymptotic complexity classes in order, using "=" to mean that two classes are exactly the same, and "$\subset$" to mean that one class is a proper subset of the other; log means $\log_2$.

   $O\left(\frac{1}{n}\right) \subset O(\log 2) \subset O(\log n) \subset O(\log^2 n) \subset O(2^{\log n}) = O(4n+3) \subset O(n \log n) \subset O(n^{1.1}) \subset O(n^3) \subset O(F_n) = O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right) \subset O(2^n) \subset O(3^n)$

4. R1 Solve each of the following recurrences, giving the answer in terms of O, Θ, or Ω, whichever is most appropriate [10 points each].

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5. [15 points] Consider the following procedure:

```
void george(int n)
 {
   int m = n;
   while (m > 1)
    {
      for (int i = 1; i < m; i++)
       cout << "I cannot tell a lie.  I chopped down the cherry tree." << endl;
      m = m/2;
    }
 }
```

Write a recurssence for the time complexity of `george`, then solve that recurrence.

$T(n) = n + T(n/2)$ $\qquad$ $T(n) = \Theta(n)$

6. In each case, assume that X[n] is an array to be sorted. Write correct pseudocode (or C++ code) for:

   (a) Bubblesort.

   This code is already on this practice test.

   (b) Selectionsort.

   This code is already on this practice test.

   (c) The partition loop of quicksort.

   The goal is to sort the subarray A[first] ... A[last]. We assume pivot = A[first]. There are many correct versions, but here is myh favorite.

```
int lo = first+1;
int hi = last;
while(lo < hi)
  {
   if (A[lo] <= pivot) lo++;
   if (A[hi-1] >= pivot) hi--;
   ig (lo < hi and A[hi-1] < pivot and A[lo] > pivot
    swap(A[hi-1],A[lo]);
  }
swap(A[first],A[hi]);
```

7. 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.

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

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

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

   (d) The following code models the first phase of heapsort.

```
for(int i = n; i > 0; i--)
 for(int j = i; 2*j <= n; j = 2*j)
  cout << "swap" << endl;
```

   $\Theta(n)$

(e) The following code models the second phase of heapsort.

```
for(int i = n; i > 0; i--}
 {
  cout << "swap" << endl;
  for(int j = 1; 2*j <= i; j = 2*j)
   cout << "swap" << endl;
 }
```

$\Theta(n \log n)$

(f) The following code models insertion of n items into an AVL tree.

```
for(int i = 1; i < n; i++)
 for(int j = n; j > 0; j = j/2)
   cout << "check AVL property and possibly rotate" << endl;
```

$\Theta(n \log n)$

(g)
```
for(int i = 1; i*i < n; i++)
 cout << "Hi!" << endl;
```

$\Theta(\sqrt{n})$

(h)
```
for(int i = n; i > 1; i = sqrt(i));
 cout << "Hi!" << endl;
```

$\Theta(\log \log n)$

(i)
```
int f(int n)
 {
  if (n < 2) return 1;
  else return f(n-1)+f(n-1);
 }
```

$\Theta(2^n)$

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

$\Theta(n \log n)$

(k)
```
for(int i = n; i > 1; i = i/2)
  cout << "hello world" << endl;
```

$\Theta(\log n)$

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

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

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

8. The following is pseudo-code for which sorting algorithm we've discussed?

   selection sort

```
int x[n];
obtain values of x;
for(int i = n-1; i > 0; i--)
 Find the largest element of x[0], ... x[i] and swap it with x[i]
```

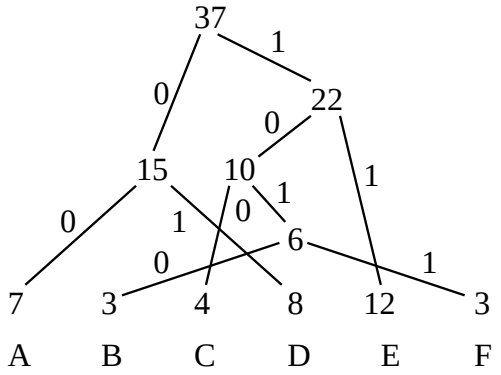
9. The following is pseudo-code for which sorting algorithm we've discussed?

   bubble sort

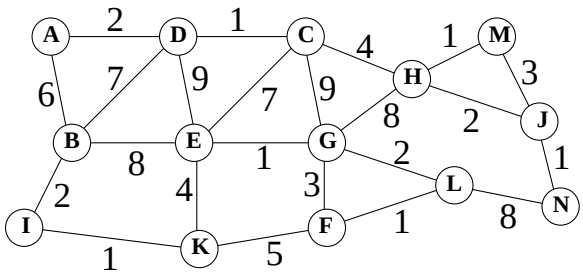
```
int x[n];
obtain values of x;
bool finished = false;
for(int i = n-1; i > 0 and not finished; i--)
 {
  finished = true;
  for(int j = 0; j < i; j++)
    if(x[j] > x[j+1])
     {
      swap(x[j],x[j+1]);
      finished = false;
     }
 }
```

10. [20 points] H1 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	7	00
B	3	1010
C	4	100
D	8	01
E	12	11
F	3	1011

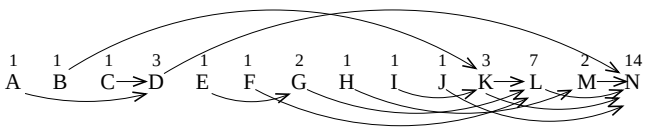


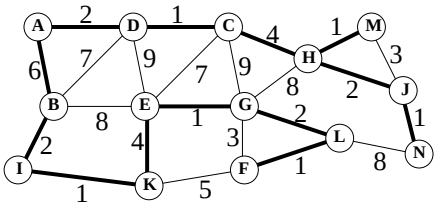
11. Find a minimum spanning tree of the weighted graph shown below.



Use union/find, with path compression. If there is a choice pick the higher letter to be the leader.

If this problem were on a test, I would want to see the following:





However, I will go through the steps.

The edges sorted by weight:

1 CD, 1 EG, 1 FL, 1 HM, 1 IK, 1 JN, 2 AD, 2 BI, 2 GL, 2 HJ, 3 FG, 3 JM, 4 CH, 4 EK, 5 FK, 6 AB, 7 BD, 7 CE, 8 BE, 8 GH, 8 LN, 9 CG, 9 DE

Kruskal's algorithm is defined below, where N is the number of vertices.
Initialize a forest of N singleton trees, each containig one vertex. The function Find(x) returns the leader of the set to which x belongs. If z and w are distinct leaders, Union(z,w) joins the sets together and chooses either z or w to be the leader of the new set. The leader of the larger set becomes the leader of the union. If they are the same size, the larger leader becomes the new leader.

In the example N = 13. We process edges in weight order, selecting N-1 of them for the spanning tree and rejecting the others.
1. Repeat steps 2. through 4. until N-1 edges are selected.

2. Let {x,y} be the next edge.
3. Compute z = Find(x) and w = Find(y). (Path compression may occur here.)
4. If z and w are not equal, execute Union(z,w) and select the edge {x,y}.
5. Let T be the subgraph spanned by all selected edges. T is a minimal spanning tree.

Process(C,D) Find(C) = C. Find(D) = D. Union(C,D). Leader[C] := D. size[D] := 2. {C,D} selected.
Process(E,G) Find(E) = E. Find(G) = G. Union(E,G). Leader[E] := G. size[G] := 2.{E,G} selected.
Process(F,L) Find(F) = F. Find(L) = L. Union(F,L). Leader[F] := L. size[L] := 2. {F,L} selected.
Process(H,M) Find(H) = H. Find(M) = M. Union(H,M). Leader[H] := M. size[M] := 2. {H,M} selected.
Process(I,K) Find(I) = I. Find(K) = K. Union(I,K). Leader[I] := K. size[K] := 2. {I,K} selected.
Process(J,N) Find(J) = J. Find(N) = N. Union(J,N). Leader[J] := N. size[N] := 2. {J,N} selected.
Process(A,D) Find(A) = A. Find(D) = D. Union(A,D). Leader[A] := D. size[D] := 3. {A,D} selected.
Process(B,I) Find(B) = B. Find(I) = K. Union(B,K). Leader[B] := K. size[K] := 3. {B,I} selected.
Process(G,L) Find(G) = G. Find(L) = L. Union(G,L). Leader[G] := L. size[L] := 4. {G,L} selected.
Process(H,J) Find(H) = M. Find(J) = N. Union(M,N). Leader[M] := N. size[N] := 4. {H,J} selected.
Process(F,G) Find(F) = L. Find(G) = L. No union.
Process(J,M) Find(J) = N. Find(M) = N. No union.
Process(C,H) Find(C) = D. Find(H) = N. Leader[H] := N: path compression. Union(D,N). Leader[D] := N. size[N] := 7. {C,H} selected.
Process(E,K) Find(E) = L. Leader[E] := L (path compression) Find(K) = K. Union(L,K). Leader[K] := L: size[L] := 7. {E,K} selected.
Process(F,K) Find(F) = L. Find(K) = L. No union.
Process(A,B) Find(A) = N. Leader[A] := N: path compression. Find(B) = L. Leader[B] = L; path compression. Union(N,L). Leader[L] := N; size[N] := 14. {A,B} selected.
$N - 1 = 13$ edges have been selected. Algorithm ends.

12. Write pseudo-code for binary search.

We assume that A is a sorted array. We are given an item x and we need to determine whether x is an item in the array. If it is an item, we report which its index is in the array. The algorithm recursively searches the subarray A[lo] ... A[hi], where hi and lo are closer together at each recursive call. The items of A could have any ordered type, but for simplicity, I assume that type is float, and that A is a globally declared array.

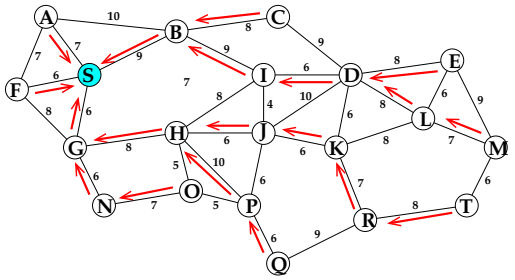
```
bool find(float x, int lo, int hi)
  {
   assert(lo <= hi);
   if(lo < hi)
    {
     int mid = (lo+hi)/2;
     if (A[mid] < x) return find(x,mid+1,hi);
     else return find(x,lo,mid);
    }
   else // lo = hi
    {
```

```
    if(A[lo] == x) cout << "The item is found in position " << lo << endl;
    return A[lo] == x;
   }
 }
```

13. Walk through heapsort for the following array: **A Q R B X S M L N T**

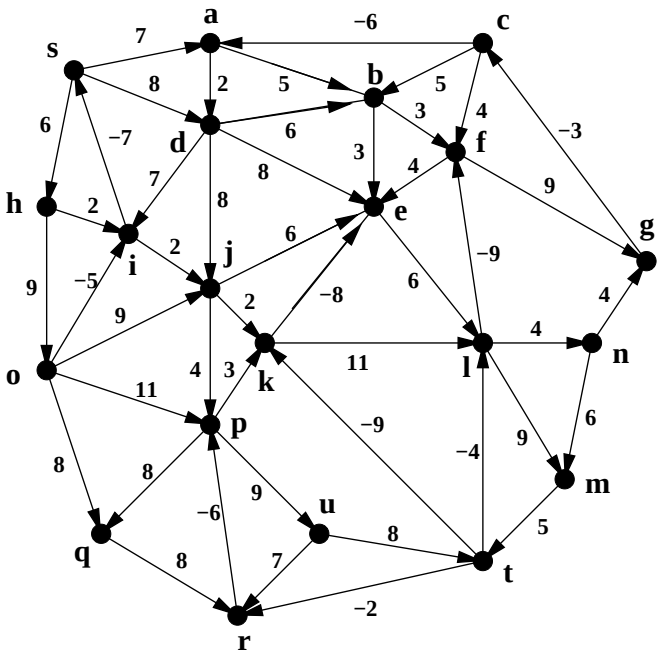
1	2	3	4	5	6	7	8	9	10	
A	Q	R	B	X	S	M	L	N	T	
A	Q	R	N	X	S	M	L	B	T	
A	Q	S	N	X	R	M	L	B	T	
A	X	S	N	Q	R	M	L	B	T	
A	X	S	N	T	R	M	L	B	Q	
X	A	S	N	T	R	M	L	B	Q	
X	T	S	N	A	R	M	L	B	Q	
X	T	S	N	Q	R	M	L	B	A	now in max heap order
A	T	S	N	Q	R	M	L	B	X	
T	A	S	N	Q	R	M	L	B	X	
T	Q	S	N	A	R	M	L	B	X	end of first iteration of second phase
B	Q	S	N	A	R	M	L	T	X	
S	Q	B	N	A	R	M	L	T	X	
S	Q	R	N	A	B	M	L	T	X	end of second iteration
L	Q	R	N	A	B	M	S	T	X	
R	Q	L	N	A	B	M	S	T	X	
R	Q	M	N	A	B	L	S	T	X	end of third iteration
L	Q	M	N	A	B	R	S	T	X	
Q	L	M	N	A	B	R	S	T	X	
Q	N	M	L	A	B	R	S	T	X	end of fourth iteration
B	N	M	L	A	Q	R	S	T	X	
N	B	M	L	A	Q	R	S	T	X	
N	L	M	B	A	Q	R	S	T	X	end of fifth iteration
A	L	M	B	N	Q	R	S	T	X	
M	L	A	B	N	Q	R	S	T	X	end of sixth iteration
B	L	A	M	N	Q	R	S	T	X	
L	B	A	M	N	Q	R	S	T	X	end of seventh iteration
A	B	L	M	N	Q	R	S	T	X	
L	B	A	M	N	Q	R	S	T	X	end of eighth iteration
A	B	L	M	N	Q	R	S	T	X	
B	A	L	M	N	Q	R	S	T	X	end of ninth iteration
A	B	L	M	N	Q	R	S	T	X	done

14. Walk through Dijkstra's algorithm for the following weighted graph to solve the single source shortest pair problem, where S is the source.



S	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	T
0	7	9	17	26 24	34	6	6	14	18	20	26	34	41	12	19	24	32	33	41
	S	S	B	C 1	D	S	S	G	B	H	J	K	L	G	H	H	F	K	R

15. The first step of Johnson's algorithm is to compute the heuristic function On the weighted directed graph (a) below, label each node of (a) with the correct heuristic. (You do not have to show the steps of the algorithm for this. The example is small enough that you can simply compute the values in your head.) The next step is to adjust the arc weights. Label the arcs of (b) with the adjusted weights.



(a)

Oops! In working this problem, I discovered the graph has a negative cycle. There won't be a Johnson's algorithm problem on this test.

16. [20 points] Give pseudocode for a recursive algorithm which computes the median of the union of two sorted lists in logarithmic time.

I have decided not to reveal the solution to this problem.

17. [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.)

    We assume that we are given a set A of n numbers, and '$1 \leq k \leq n$. Here are the steps of the algorithm. If n = 1, simply return the one member of A. Otherwise:
    Pick a random number x from A
    Let B be the set of all members of A which are less than x, and let b be the size of B.
    Let B be the set of all members of A which are greater than x. Note that C has exactly $(n - b - 1)$ elements. if k ¡= b, return the k^{th} smallest number in B, using recursion. else if k = b + 1, return x. else return the $(k - b - 1)^{\text{st}}$ smallest element of C, using recursion.

18. [20 points] Give pseudocode for the Bellman-Ford algorithm.

    We assume that a directed graph G is given, with vertices 0 .. n-1. There is a 2-dimensional array W given, where W[i,j] is the weight of the edge from i to j. If there is no edge from i to j, W[i,j] is given as infinity. The output of the code is two 1-dimensional arrays, dist and back, where dist[i] is the weight of the least cost path from 0 to i, and back[i] is the next-to-the last vertex on that path. Note that back[0] is undefined.

    Let m be the number of edges. The worst case time complexity of the Bellman Ford algorithm is O(nm), but if every optimal path has length at most L, then the time complexity drops to O(n+mL). When we write the code, we can take advantage of the speed up if L is small, but the code does not need to know the value of L. The code here will also detect a negative cycle.

    We assume that the set of all edges of G is stored in such a way that we can visit all the edges in L iterations.

    If the longest optimal path has L edges, the outer loop will iterate L+1 times, after which finished = true, and we exit the outer loop, saving time.

    Any optimal path must be simple (i.e., no repeat vertices) and so can have length at most n-1. If, on the n^{th} iteration of the outer loop, a better path is discovered, that path has length n, and hence there must be a negative cycle.

```
dist[0] = 0;
for(int i = 1; i < n; i++)
  dist[i] = infinity;
bool finished = false; // all optimal paths have been found
int t = 0;  // the number of iterations of the outer loop so far
while (not finished and t < n)
  {
    finished = true // Why?
    for all edges (i,j) of G // there are m iterations of this loop
      {
       t++;
       temp = dist[i] + W[i,j];
       if (temp < dist[j]) // relax
         {
```

```
          dist[j] = temp;
          back[j] = i;
          finished = false;
         }
      }
   }
if(t < n) cout << "Computation completed; no negative cycle" << endl;
else cout "There is a negative cycle: computation faills." << endl;
```

19. [20 points] Give pseudocode for the Floyd-Warshall algorithm.

    We assume that a directed graph G is given, with vertices 0 .. n-1. There is a 2-dimensional array W given, where W[i,j] is the weight of the edge from i to j. If there is no edge from i to j, W[i,j] is given as infinity. The output consists of 2-dimensional arrays dist and back, where dist[i,j] is the weight of the optimal path from i to j, and back[i,j] is the next-to-the last vertex on that path. Note that back[i,i] is undefined.

```
for(int i = 0 to n-1)
  for(int j = 0 to n-1)
    {
     dist[i,j] = W[i,j];
     back[i,j[ = i;
    }
for(int i = 0 to n-1) dist[i,i] = 0;
for(int j = 0 to n-1)
  for(int i = 0 to n-1)
    for(int k = 0 to n-1)
     {
      temp = dist[i,j] + dist[j,k];
      if(temp < dist[i,k]
        {
         dist[i,k] = temp;
         back[i,k] = back[j,k];
         if(i == k) HALT("There is a negative cycle");
        }
     }
```