University of Nevada, Las Vegas Computer Science 477/677 Spring 2020

Practice Examination for April 30, 2020

Updated `Sat Apr 25 12:12:28 PDT 2020`

The entire practice examination is 340 points.

The current closure order extends to April 30.

1. True or False. [5 points each] T = true, F = false, and O = open, meaning that the answer is not known to science at this time.

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

   (b) **O** If a problem can be worked in $O(n)$ time by a single processor, then it can be worked in polylogarithmic time, that is, $O(\log^k n)$ time for some constant k, if polynomially many processors are used.

   (c) **T** The asymptotic space complexity of a program cannot excced its asymptotic time complexity.

2. (10 points each) Find the asymptotic final value of `kount` for each of these code fragments in terms of n. $\Theta(n)$, $\Theta(n^2)$, $\Theta(n \log n)$, $\Theta(\log^* n)$, $\Theta(\log\log n)$, $\Theta(\sqrt{n})$, In each case, I will give an equivalent integral.

   (a)
```
int kount = 0;
for(int i = 0; i < n; i++)
 for(int j = i; j > 0; j--)
  kount++;
```
$$\int_{x=0}^{n}\int_{y=0}^{x} dydx = \int_{x=0}^{n} y]_{y=0}^{x}\, dx = \int_{x=0}^{n} xdx = \left.\frac{x^2}{2}\right]_0^n = \frac{n^2}{2} = \Theta(n^2)$$

   (b)
```
int kount = 0;
for(int i = 1; i < n*n; i = 2*i)
 kount++;
```
Substituting $\ell = \log i$ we can write:
```
int kount = 0;
for(int j = 0; j < 2*log n; \ell++)
 kount++;
```
$$\int_{y=0}^{2\log n} dy = y]_0^{2\log n} = 2\log n = \Theta(\log n)$$

   (c)
```
int kount = 0;
for(int i = 0; i*i < n; i++)
 kount++;
```
Substituting $n = m^2$ we obtain:

```
int kount = 0;
for(int i = 0; i*i < m*m; i++)
 kount++;
```

Note that $i^2 < m^2$ if and only if $i < m$, since i, m are positive integers:

```
int kount = 0;
for(int i = 0; i < m; i++)
 kount++;
```

$\int_{x=0}^{m} dx = x]_{x=0}^{m} = m = \sqrt{n} = \Theta(\sqrt{n})$

(d)
```
int kount = 0;
for(int i = 1; i < n; i = 2*i)
 for(int j = 0; j < i; j++)
  kount++;
```

Let $k = \log_2 i$. We obtain

```
int kount = 0;
for(int k = 0; k < log2(n); i = 2*i)
 for(int j = 0; j < 2^k; j++)
  kount++;
```

Recall that the the antiderivative of e^x is e^x, and the ant-derivative of K^x is $K^x \ln K$ for any positive constant K. Recall also that $\ln 2$ is a constant. We have: $\int_{z=0}^{\log_2 n} \int_{y=0}^{2^z} dydz = \int_{z=0}^{\log_2 n} y]_{y=0}^{2^z} dz = \int_{z=0}^{\log_2 n} 2^z dz = 2^z \ln 2]_{z=0}^{\log_2 n} = n \ln 2 - \ln 2 = \Theta(n)$

(e)
```
int kount = 0;
for(int i = 1; i < n; i = 2*i)
 for(int j = i; j < n; j++)
  kount++;
```

Let $k = \log_2 i$. We obtain

```
int kount = 0;
for(int k = 0; k < log2(i) ; k++
 for(int j = 2^k; j < n; j++)
  kount++;
```

We have:
$\int_{z=0}^{\log_2 n} \int_{y=2^z}^{n} dydz = \int_{z=0}^{\log_2 n} y]_{y=2^z}^{y=n} dz = \int_{z=0}^{\log_2 n} (n - 2^z)dz$
$= nz - 2^z \ln 2]_{z=0}^{z=\log_2 n} = n \log_2 n - (n-1)\ln 2 = \Theta(n \log n)$

(f)
```
int kount = 0;
for(int i = n; i > 0; i = log(i))
 kount++;
```

(g)
```
int kount = 0;
for(int i = 0; i < n*n; i = i+2*sqrt(i)+1)
 kount++;
```

Hint: Use the substitution $\texttt{i} = \texttt{j}^2$. Note that $i + 2\sqrt{i} + 1 = (\sqrt{i} + 1)^2$.

```
int kount = 0;
for(int j = 0; j*j < n*n; j++)
 kount++;
```

Of course, $j^2 < n^2$ if and only if $j < n$. Thus we have:

```
int kount = 0;
for(int j = 0; j < n; j++)
 kount++;
```

Hence $\texttt{kount} = \Theta(n)$.

(h) Deleted.

(i) Deleted.

(j)
```
int kount = 0;
for(int i = 2; i < n; i = i*i)
 kount++;
```

Let $j = \log_2 i$

```
int kount = 0;
for(int j = 1; j < log2(n); j = 2*j)
 kount++;
```

Let $k = \log_2 j$

```
int kount = 0;
for(int k = 0; k < log2(log2(n)); k++)
 kount++;
```

Hence $\texttt{kount} = \Theta(\log \log n)$.

(k)
```
int kount = 0;
for(int i = 1; i < n; i++)
 for(int j = i; j < n; j=2*j)
  kount++;
```

Let $k = j/i$, which is always a power of 2. Initially, $k = 1$, and it doubles at each iteration of the inner loop, and it's bounded above by n/i. Let $u = \log_2 k$, which is always an integer. Initially, $u = 0$, it increments by 1 at each iteration of the inner loop, and it's bounded above by $\log_2(n/i) = \log_2 n - \log_2 i$.

```
int kount = 0;
for(int i = 1; i < n; i++)
 for(int k = 1; j < n/i; k=2*k)
  kount++;
```

```
int kount = 0;
for(int i = 1; i < n; i++)
 for(int u = 0; u < log2(n)-log2(i);u++)
  kount++;
```

$\log_2 x = \ln x/\ln 2$, hence the antiderivative of $\log_2 x$ is $\dfrac{\ln x - x}{\ln 2} = x\log_2 x - \dfrac{x}{\ln 2}$

The final value of `kount` is approximately

$$\int_{x=1}^{n}\int_{w=0}^{\log_2 n - \log_2 x} dw dx = \int_{x=1}^{n}(\log_2 n - \log_2 x)dx = \left(x\log_2 n - x\log_2 x + \frac{x}{\ln 2}\right)\Big]_{x=1}^{n} = \frac{n-1}{\ln 2} - \log_2 n = \Theta(n)$$

3. (10 points each) Find the asymptotic complexity of $F(n)$ for each recurrence, expressed using Θ if possible, Ω or O otherwise.

   For these problems use the master theorem.

   (a) $F(n) \le F(n/2) + n$ $\qquad O(n)$

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

   (c) $F(n) = 4F(n/2) + n$ $\qquad \Theta(n^2)$

   (d) $F(n) \ge F(n/2) + 1$ $\qquad \Omega(\log n)$

   (e) $F(n) = 2F(n/4) + \sqrt{n}$ $\qquad \Theta(\sqrt{n}\log n)$

   For these problems, use the anti-derivative method.

   (f) $F(n) = F(n-1) + n$ $\qquad \Theta(n^2)$

   (g) $F(n) = F(n-2) + n^2$ $\qquad \Theta(n^3)$

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

   Move the first term on the left to the right, then divide both sides by $\sqrt{n}$

   $$\frac{F(n) - F(n-\sqrt{n})}{\sqrt{n}} = \sqrt{n}$$

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

   $$F(n) = \Theta\left(n^{\frac{3}{2}}\right)$$

   For these problems, use the generalized master theorem.

   (i) $F(n) = F(n/3) + F(n/4) + F(n/5) + n$ $\qquad \Theta(n)$ because $\frac{1}{3} + \frac{1}{4} + \frac{1}{5} < 1$

   (j) $F(n) = 2F(n/4) + F(n/2) + n$ $\qquad \Theta(n\log n)$ because $2(\frac{1}{4}) + \frac{1}{2} = 1$

   (k) $F(n) = F(3n/5) + F(4n/5) + n$ $\qquad \Theta(n^2)$ because $\left(\frac{3}{5}\right)^2 + \left(\frac{4}{5}\right)^2 = 1$

   (l) $F(n) = F(3n/5) + F(4n/5) + n^2$ $\qquad \Theta(n^2\log n)$ because $\left(\frac{3}{5}\right)^2 + \left(\frac{4}{5}\right)^2 = 1$

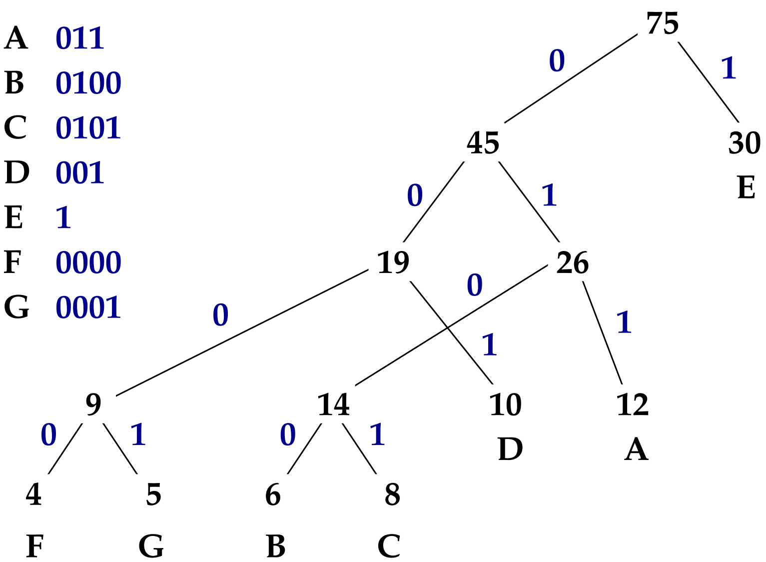
(m) $F(n) = 2F(2n/3) + F(n/3) + 1$ $\qquad$ $\Theta(n^2)$ because $2\left(\frac{2}{3}\right)^2 + \left(\frac{1}{3}\right)^2 = 1$

(n) $F(n) \le F(n/5) + F(7n/10) + n$ $\qquad$ $O(n)$ because $\frac{1}{5} + \frac{7}{10} < 1$

4. [20 points] Find an optimal prefix-free code for the alphabet $\{A, B, C, D, E, F, G\}$ with the following frequency distribution.

A	12
B	6
C	8
D	10
E	30
F	4
G	5

A 011
B 0100
C 0101
D 001
E 1
F 0000
G 0001



This answer is not unique, but the code strings for any optimal code have the same lengths.

5. [20 points] Consider a array of n numbers. The sum of those numbers can be computed in logarithmic time by using n processors working in parallel.

Suppose that the numbers in the array are:

1, 2, 9, 0, 5, 7, 2, 8, 6, 3, 4, 1, 5, 9, 5, 6.

Walk through the parallel algorithm which finds the sum using n processors. At each level, show the intermediate results. Your diagram should clearly indicate each time two numbers are combined into one number.

1 2 9 0 5 7 2 8 6 3 4 1 5 9 5 6

3 9 12 10 9 5 14 11

12 22 14 25

34 39

73

With n processors, the problem can be worked in $O(\log n)$ time. Initially, we have 16 numbers in the example. In the first phase, 8 processors combine them in pairs, obtaining 8 numbers. In the next phase, 4 processors combine those in pairs, obtaining 4 numbers. In the next phase, 2 processors combine those in pairs, obtaining 2 numbers. In the final phase, one processor combines those to find the overall sum. There are four phases, and each phase can be worked in $O(1)$ time. Note that $\log_2(16) = 4$.

6. [20 points] Let A be an array of n numbers. Consider the problem of finding the maximum sum of any contiguous subarray. For example, if the items of A are -3, 2, 4, -5, 3, 2, -1, 4, the contiguous array with

the maximum sum is 2, 4 -5, 3, 2, -1, 4; If the items of A are -5, 3, -2, 4, 6, -8, 1, -3, 5 then the answer is 3, -2, 4, 6. There are at four known algorithms for this problem:

We write $S[i,j] = \sum_{k=i}^{j} A[k]$. The problem is to find $\max_{1 \leq i \leq j \leq n} S[i,j]$.

(a) An exhaustive algorithm which takes $O(n^3)$ time.

Compute $S[i,j]$ for all i,j and select the largest. There are $\Theta(n^2)$ choices of i,j and it takes $O(n)$ time to compute each one. Thus, this method takes $O(n^3)$ time.

(b) A slightly more intelligent algorithm which takes $O(n^2)$ time. Note that $S[i,j]+A[j+1] = S[i,j+1]$. Using this equation, for each i, we can compute $S[i,j]$ for all $j \geq i$ in $O(n)$ time. Thus, we compute all $S[i,j]$ in $O(n^2)$ time.

Since the number of values of S is $\Theta(n^2)$, any algorithm faster than that will have to avoid computing all $S[i,j]$.

(c) A rather clever divide and conquer algorithm, which takes $O(n \log n)$ time.

I will skip the explanation of this one.

(d) A sophisticated dynamic programming algorithm which takes $O(n)$ time.

Define $M[k] = \max\{S[i,j] : i \leq j \leq k\}$, Define $N[k] = \max\{S[i,k] : i \leq k\}$. Our dynamic algorithm has the following structure:

```
for(k = 1; kn; k++)
 {
  Compute N[k];
  Compute M[k];
 }
return M[n};
```

It is possible to compute both $N[k]$, and then $M[k]$ in $O(1)$ time. Here is the pseudo-code.

```
N[1] = A[1];
M[1] = A[1];
for(int k = 2; k <= n; k++)
 {
  N[k] = max{A[k],N[k-1]+A[k]};
  M[k] = max{N[k],M[k-1]};
 }
return M[n];
```

Do you see how it works?

7. [20 points] The *distance* between two vertices x, y of a connected unweighted graph is defined to be the minimum number of edges of a path from x to y. The *diameter* of such a graph is defined to be the maximum distance between any two vertices.

Suppose you are given a connected undirected graph G with n vertices and m edges, where n is one billion and m is approximately $10n$, and no vertex has degree more than 100. (Think of the internet.) Your job is to find the diameter of G.

(a) How long would that take if you use the Floyd-Warshall Algorithm?

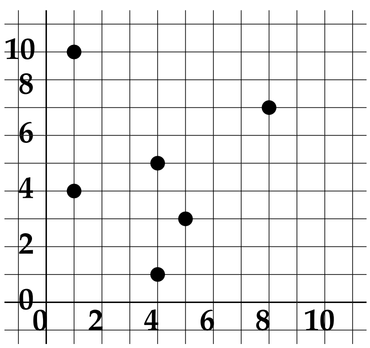
$O(n^3)$. That would be approximately 10^{27} steps.

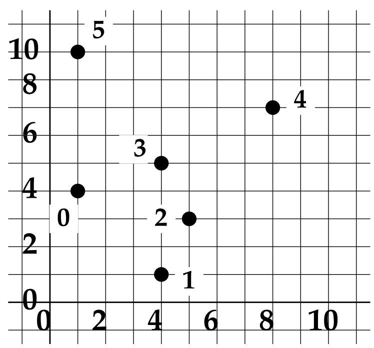
(b) Describe the algorithm you would recommend.

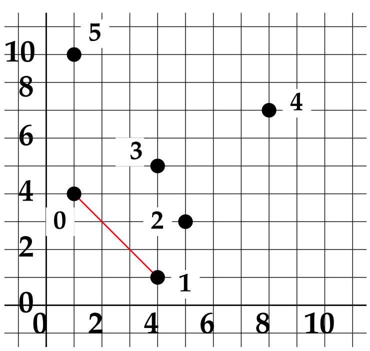
One idea is, for each vertex v, to use breadth first search to find the vertex farthest from v. This would take $O(nm)$ time, about 10^{19} steps.

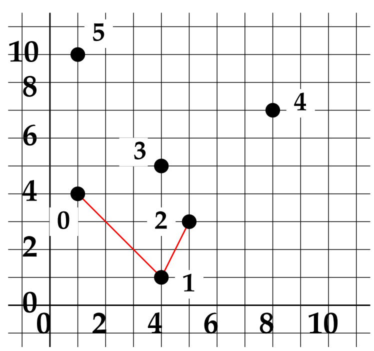
(c) Can your computation be efficiently parallelized if you hava a parallel machine with a billion processors? If you have p processors, for $p \leq n$. assign each processor a set of n/p vertices, and have it use breadth first search to find the farthest vertex from each of those. Thus the problem can be solved in $O(m)$ time using n processors, or $O(mn/p)$ time using $p \leq n$ processors.

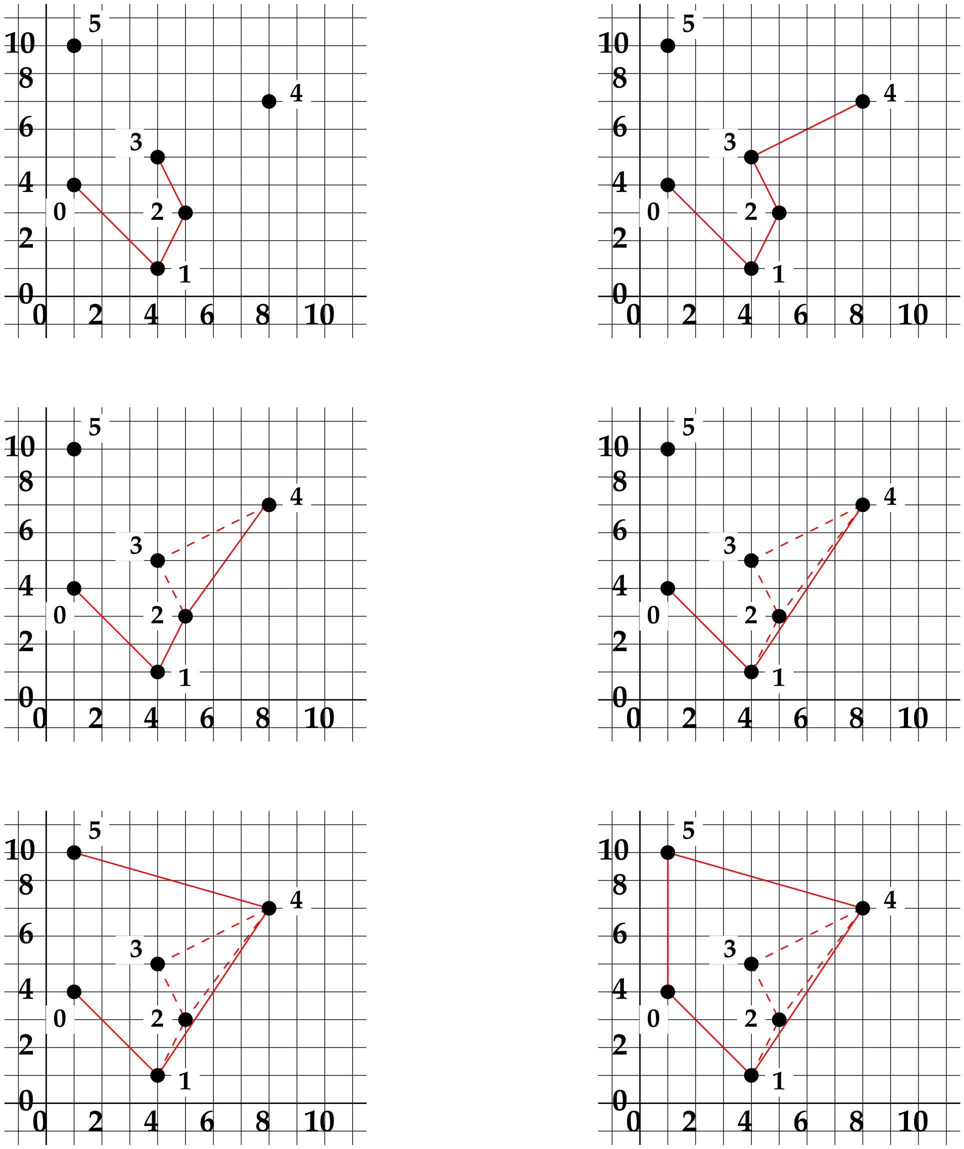
8. Use Graham scan to find the convex hull of the set of dots in the figure below. Use the point (1,4) as the pivot.











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

   (a) The items in a priority queue represent **unfulfilled obligations**.

   (b) If a hash table has n places and there are n data items, What is the approximate percentange of places that will hold more than one item? **26** or **27** (Within 1 percentage point.)

   If n is large, the probability that there are exactly k items in a given place is approximately $\dfrac{1}{k!\,e}$ which is $\dfrac{1}{e}$ if $k = 0$ or $k = 1$. Thus the probability that there are 2 or more items is $1 - \dfrac{2}{e} \approx 0.264$

   More generally, if there are n items placed randomly in a hash table of size m, the average number of items in a place is n/m, and the probability that a given place will have exactly k items is approximately $\dfrac{(n/m)^k}{k!\,e^{n/m}}$