CS 170 First Midterm 26 Feb 2010

NAME (1 pt): _____

SID (1 pt): _____

TA (1 pt): _____

Name of Neighbor to your left (1 pt): _____

Name of Neighbor to your right (1 pt): _____

Instructions: This is a closed book, closed calculator, closed computer, closed network, open brain exam, but you are permited a 1 page, double-sided set of notes, large enough to read without a magnifying glass.

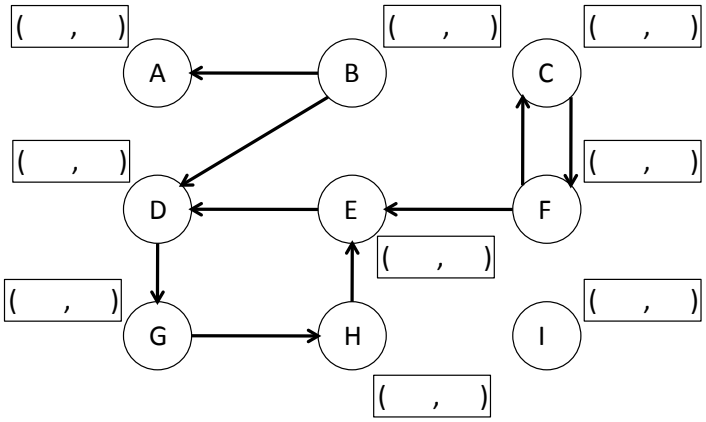
You get one point each for filling in the 5 lines at the top of this page. Each other question is worth 20 points.

Write all your answers on this exam. If you need scratch paper, ask for it, write your name on each sheet, and attach it when you turn it in (we have a stapler).

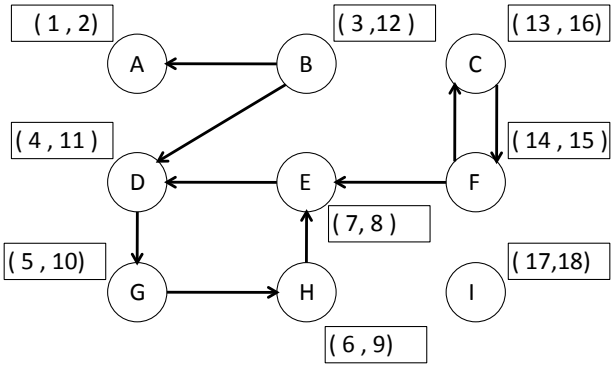
1	
2	
3	
4	
Total	

Question 1 (20 points).

a) Do a depth-first search on the graph below, and show the `pre` and `post` numbers on the figure. Whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.



Answer:



b) What are the cross edges? (write an edge from i to j as (i,j)).

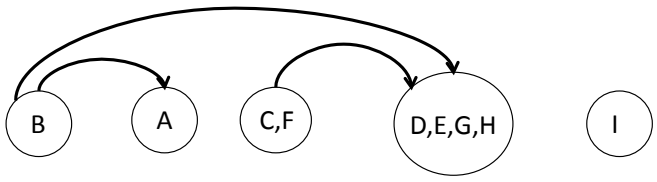
Answer: The cross edges are: (B,A) and (F,E)

c) What are the strongly connected components?

Answer: the SCCs are: $\{D, G, H, E\}$, $\{C, F\}$, $\{A\}$, $\{B\}$ and $\{I\}$.

d) Draw a DAG (consistent with the graph given) from the SCC nodes.

Answer:



e) Can a DAG be built from the SCC nodes of the graph such that I is the first node (the leftmost node if the ordering is from left to right)? Explain briefly.

Answer: YES. There are no edges from or to node I so it can be placed anywhere in the DAG ordering.

f) For a given graph $G = (V, E)$ we define two vertices, i and j, as a "pair" if there is a path from i to j and a path from j to i.

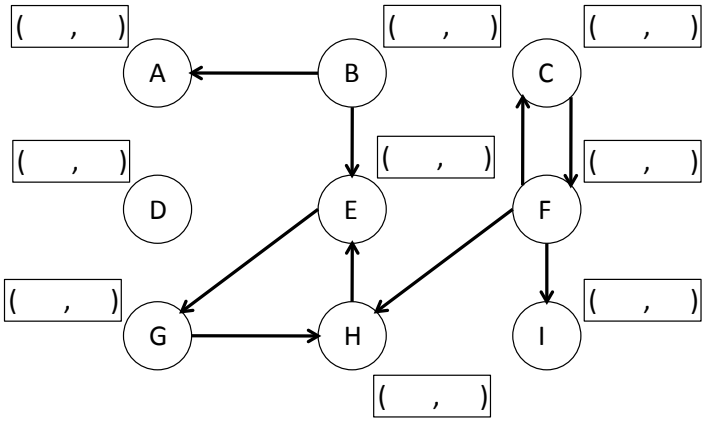
Describe an algorithm that given some graph $G = (V, E)$, returns the number of "pairs" in the graph, and has running time $O(|V| + |E|)$.
(Note: You may use any algorithm learned in class as a black box).

Answer: Compute all of the SCCs with the algorithm learned in class. For all SCCs: $SCC_1, SCC_2, \ldots, SCC_k$, compute there sizes: $s_1, s_2, \ldots, s_k$, (for example by running a depth-first search on each). By definition, every two nodes within a SCC are a "pair". Therefore, the number of "pairs" in the graph is:

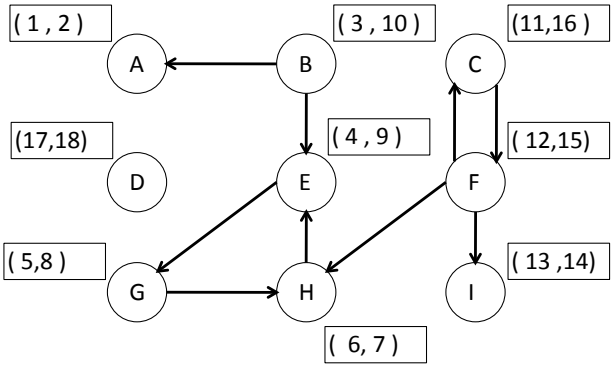
$$\sum_{i=1}^{k} \binom{s_i}{2}$$

Question 1 (20 points).

a) Do a depth-first search on the graph below, and show the `pre` and `post` numbers on the figure. Whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.



Answer:



b) What are the cross edges? (write an edge from i to j as (i,j)).

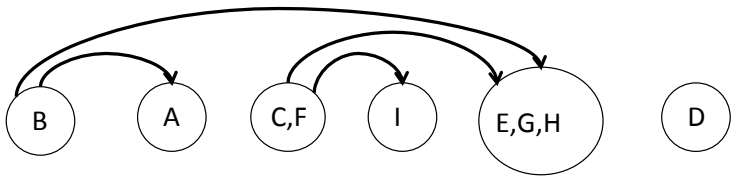
Answer: The cross edges are: (B,A) and (F,H)

c) What are the strongly connected components?

Answer: the SCCs are: $\{G, H, E\}$, $\{C, F\}$, $\{A\}$, $\{B\}$, $\{D\}$ and $\{I\}$.

d) Draw a DAG (consistent with the graph given) from the SCC nodes.

Answer:



e) Can a DAG be built from the SCC nodes of the graph such that D is the first node (the leftmost node if the ordering is from left to right)? Explain briefly.

Answer: YES. There are no edges from or to node D so it can be placed anywhere in the DAG ordering.

f) For a given graph $G = (V, E)$ we define two vertices, i and j, as a "pair" if there is a path from i to j and a path from j to i.

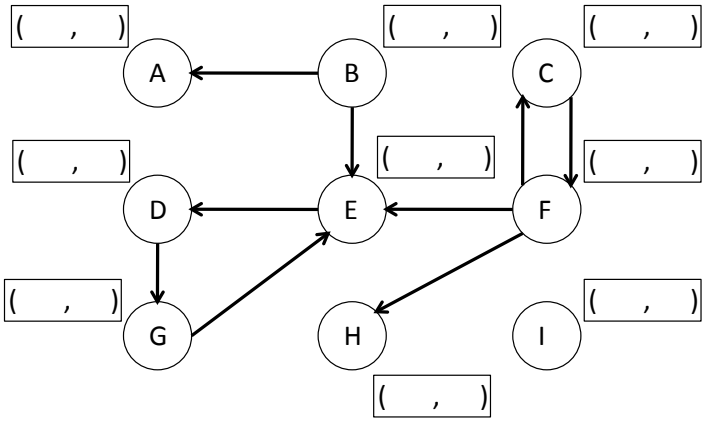
Describe an algorithm that given some graph $G = (V, E)$, returns the number of "pairs" in the graph, and has running time $O(|V| + |E|)$.
(Note: You may use any algorithm learned in class as a black box).

Answer: Compute all of the SCCs with the algorithm learned in class. For all SCCs: $SCC_1, SCC_2, \ldots, SCC_k$, compute there sizes: $s_1, s_2, \ldots, s_k$, (for example by running a depth-first search on each). By definition, every two nodes within a SCC are a "pair". Therefore, the number of "pairs" in the graph is:

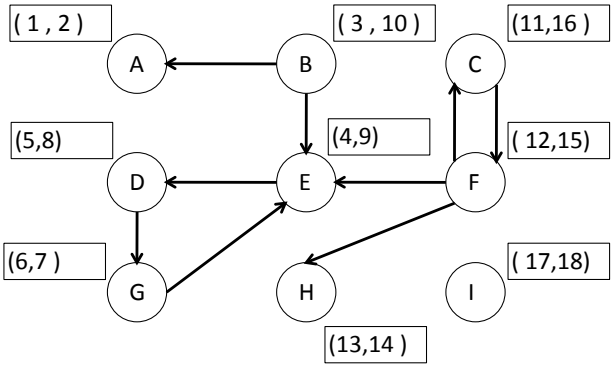
$$\sum_{i=1}^{k} \binom{s_i}{2}$$

Question 1 (20 points).

a) Do a depth-first search on the graph below, and show the `pre` and `post` numbers on the figure. Whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.



Answer:



b) What are the cross edges? (write an edge from i to j as (i,j)).

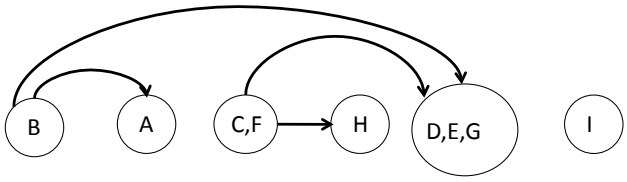
Answer: The cross edges are: (B,A) and (F,E).

c) What are the strongly connected components?

Answer: the SCCs are: $\{A\}$, $\{B\}$, $\{D, E, G\}$, $\{C, F\}$, $\{H\}$ and $\{I\}$.

d) Draw a DAG (consistent with the graph given) from the SCC nodes.

Answer:



e) Can a DAG be built from the SCC nodes of the graph such that I is the first node (the leftmost node if the ordering is from left to right)? Explain briefly.

Answer: YES. There are no edges from or to node I so it can be placed anywhere in the DAG ordering.

f) For a given graph $G = (V, E)$ we define two vertices, i and j, as a "pair" if there is a path from i to j and a path from j to i.

Describe an algorithm that given some graph $G = (V, E)$, returns the number of "pairs" in the graph, and has running time $O(|V| + |E|)$.
(Note: You may use any algorithm learned in class as a black box).

Answer: Compute all of the SCCs with the algorithm learned in class. For all SCCs: $SCC_1, SCC_2, \ldots, SCC_k$, compute there sizes: $s_1, s_2, \ldots, s_k$, (for example by running a depth-first search on each). By definition, every two nodes within a SCC are a "pair". Therefore, the number of "pairs" in the graph is:

$$\sum_{i=1}^{k} \binom{s_i}{2}$$

Question 2 (20 points). You will derive a divide-and-conquer algorithm for finding all the roots of a function $y = f(x)$ in the interval $x \in [0, 1)$ (i.e. $0 \le x < 1$).

The inputs of your recursive algorithm should be (1) the end points of the interval you are searching for roots (initially 0 and 1), (2) a function $\#roots(x_{lower}, x_{upper})$ that can count the number of roots of $f(x)$ that lie in the interval $[x_{lower}, x_{upper})$, and (3) the desired number of bits b of accuracy with which to find the roots (see below for more explanation of b).

The output of your algorithm should be a set of triples $\{(x_{lower,1}, x_{upper,1}, n_1), ..., (x_{lower,k}, x_{upper,k}, n_k)\}$ where (1) the intervals $[x_{lower,1}, x_{upper,1}), ..., [x_{lower,k}, x_{upper,k})$ are disjoint and lie inside $[0, 1)$, (2) each interval $[x_{lower,i}, x_{upper_i})$ has length at most 2^{-b}, and (3) each interval $[x_{lower,i}, x_{upper,i})$ contains exactly $n_i > 0$ roots of $f(x)$. For example the triple $(.5, .75, 6)$ would mean that there are 6 roots in the interval $[.5, .75)$.

For full credit, your algorithms should be as efficient as possible no matter where the roots of $f(x)$ are located.

Part 2.a. (15 points) Give an algorithm to solve this problem, clearly labeling the following parts of your answer: Main Idea (a few sentences), Pseudocode, Proof of Correctness (short!), Running time (worst case) in $O()$ notation (in terms of b), and Running time analysis. You may assume that calling $\#roots(x_{lower}, x_{upper})$ costs $O(1)$.

Answer: The Main Idea is binary search: divide the interval in half at each step, count the number of roots in each half, and keep dividing until either an interval is empty, or has length $\le 2^{-b}$.

The Pseudocode is

```
binary_search(0,1,#roots,b)

proc binary_search(x,y,#roots,b)
    count = #roots(x,y)
    if count > 0
        if y-x ≤ 2^-b
            print [x,y,count]
        else
            mid = (x+y)/2
            binary_search(x,mid,#roots,b)
            binary_search(mid,y,#roots,b)
        endif
    endif
```

The proof of correctness is induction: At each step there are two possibilities: the interval $[x, y)$ contains no roots (in which case the routine returns without doing anything) or it does contain roots. In the latter case there are again two possibilities: the interval is as narrow as desired (in which case the routine prints the desired answer) or it isn't (in which case the routine divides the interval in half and tries again).

The worst case running time is $O(2^b)$.

The worst-case running time analysis assumes that all intervals are nonempty. Then we get a binary tree of intervals, where the width drops in half with each call to binary_search, so there are at most m recursive calls, and a full binary tree of depth b, with at most $1 + 2 + 4 + \cdots + 2^b = 2^{b+1} - 1$ calls to $\#roots$.

Part 2.b. (5 points) Now assume that we know that the function $f(x)$ can have at most n distinct roots. What is the worst case running time now, as a function of b and n? You may assume n is a power of 2.

Answer: Now there can be at most n nonempty intervals at each level of the divide-and-conquer tree. So the maximum number of nonempty intervals (the maximum number of calls to #roots) at each level of the tree can grow like $1, 2, 4, 8, ..., n/2, n, n, ...$ *So there are 2 cases: if* $2^b \leq n$*, the complexity is still* $O(2^b)$*, and if* $2^b > n$*, the complexity is* $O(n + n(b - \log_2 n))$ *which you may simplify to* $O(nb)$*.*

Question 2 (20 points). You will derive a divide-and-conquer algorithm for finding all the roots of a function $z = g(y)$ in the interval $y \in [0, 1)$ (i.e. $0 \le y < 1$).

The inputs of your recursive algorithm should be (1) the end points of the interval you are searching for roots (initially 0 and 1), (2) a function $\#roots(y_{lower}, y_{upper})$ that can count the number of roots of $g(y)$ that lie in the interval $[y_{lower}, y_{upper})$, and (3) the desired number of bits B of accuracy with which to find the roots (see below for more explanation of B).

The output of your algorithm should be a set of triples $\{(y_{lower,1}, y_{upper,1}, n_1), ..., (y_{lower,k}, y_{upper,k}, n_k)\}$ where (1) the intervals $[y_{lower,1}, y_{upper,1}), ..., [y_{lower,k}, y_{upper,k})$ are disjoint and lie inside $[0, 1)$, (2) each interval $[y_{lower,i}, y_{upper_i})$ has length at most 2^{-B}, and (3) each interval $[y_{lower,i}, y_{upper,i})$ contains exactly $n_i > 0$ roots of $g(y)$. For example the triple $(.25, .5, 3)$ would mean that there are 3 roots in the interval $[.25, .5)$.

For full credit, your algorithms should be as efficient as possible no matter where the roots of $g(y)$ are located.

Part 2.a. (15 points) Give an algorithm to solve this problem, clearly labeling the following parts of your answer: Main Idea (a few sentences), Pseudocode, Proof of Correctness (short!), Running time (worst case) in $O()$ notation (in terms of B), and Running time analysis. You may assume that calling $\#roots(y_{lower}, y_{upper})$ costs $O(1)$.

Answer: The Main Idea is binary search: divide the interval in half at each step, count the number of roots in each half, and keep dividing until either an interval is empty, or has length $\le 2^{-B}$.

The Pseudocode is

```
binary_search(0,1,#roots,B)

proc binary_search(y,z,#roots,B)
    count = #roots(y,z)
    if count > 0
        if z-y ≤ 2^-B
            print [y,z,count]
        else
            mid = (y+z)/2
            binary_search(y,mid,#roots,B)
            binary_search(mid,z,#roots,B)
        endif
    endif
```

The proof of correctness is induction: At each step there are two possibilities: the interval $[y, z)$ contains no roots (in which case the routine returns without doing anything) or it does contain roots. In the latter case there are again two possibilities: the interval is as narrow as desired (in which case the routine prints the desired answer) or it isn't (in which case the routine divides the interval in half and tries again).

The worst case running time is $O(2^B)$.

The worst-case running time analysis assumes that all intervals are nonempty. Then we get a binary tree of intervals, where the width drops in half with each call to binary_search, so there are at most m recursive calls, and a full binary tree of depth B, with at most $1 + 2 + 4 + \cdots + 2^B = 2^{B+1} - 1$ calls to #roots.

Part 2.b. (5 points) Now assume that we know that the function $g(y)$ can have at most m distinct roots. What is the worst case running time now, as a function of B and m? You may assume m is a power of 2.

Answer: Now there can be at most m nonempty intervals at each level of the divide-and-conquer tree. So the maximum number of nonempty intervals (the maximum number of calls to #roots) at each level of the tree can grow like $1, 2, 4, 8, ..., m/2, m, m, ...$ *So there are 2 cases: if* $2^B \leq m$*, the complexity is still* $O(2^B)$*, and if* $2^B > m$*, the complexity is* $O(m + m(B - \log_2 m))$ *which you may simplify to* $O(mB)$*.*

Question 2 (20 points). You will derive a divide-and-conquer algorithm for finding all the roots of a function $s = h(t)$ in the interval $t \in [0, 1)$ (i.e. $0 \le t < 1$).

The inputs of your recursive algorithm should be (1) the end points of the interval you are searching for roots (initially 0 and 1), (2) a function $\#roots(t_{lower}, t_{upper})$ that can count the number of roots of $h(t)$ that lie in the interval $[t_{lower}, t_{upper})$, and (3) the desired number of bits B of accuracy with which to find the roots (see below for more explanation of B).

The output of your algorithm should be a set of triples $\{(t_{lower,1}, t_{upper,1}, n_1), ..., (t_{lower,k}, t_{upper,k}, m_k)\}$ where (1) the intervals $[t_{lower,1}, t_{upper,1}), ..., [t_{lower,k}, t_{upper,k})$ are disjoint and lie inside $[0, 1)$, (2) each interval $[t_{lower,i}, t_{upper_i})$ has length at most 2^{-B}, and (3) each interval $[t_{lower,i}, t_{upper,i})$ contains exactly $m_i > 0$ roots of $h(t)$. For example the triple $(.5, .75, 4)$ would mean that there are 4 roots in the interval $[.5, .75)$.

For full credit, your algorithms should be as efficient as possible no matter where the roots of $h(t)$ are located.

Part 2.a. (15 points) Give an algorithm to solve this problem, clearly labeling the following parts of your answer: Main Idea (a few sentences), Pseudocode, Proof of Correctness (short!), Running time (worst case) in $O()$ notation (in terms of B), and Running time analysis. You may assume that calling $\#roots(t_{lower}, t_{upper})$ costs $O(1)$.

Answer: The Main Idea is binary search: divide the interval in half at each step, count the number of roots in each half, and keep dividing until either an interval is empty, or has length $\le 2^{-B}$.

The Pseudocode is

```
binary_search(0,1,#roots,B)

proc binary_search(y,z,#roots,B)
    count = #roots(y,z)
    if count > 0
        if z-y ≤ 2^-B
            print [y,z,count]
        else
            mid = (y+z)/2
            binary_search(y,mid,#roots,B)
            binary_search(mid,z,#roots,B)
        endif
    endif
```

The proof of correctness is induction: At each step there are two possibilities: the interval $[y, z)$ *contains no roots (in which case the routine returns without doing anything) or it does contain roots. In the latter case there are again two possibilities: the interval is as narrow as desired (in which case the routine prints the desired answer) or it isn't (in which case the routine divides the interval in half and tries again).*

The worst case running time is $O(2^B)$.

The worst-case running time analysis assumes that all intervals are nonempty. Then we get a binary tree of intervals, where the width drops in half with each call to binary_search, so there are at most m recursive calls, and a full binary tree of depth B, with at most $1 + 2 + 4 + \cdots + 2^B = 2^{B+1} - 1$ *calls to #roots.*

Part 2.b. (5 points) Now assume that we know that the function $h(t)$ can have at most k distinct roots. What is the worst case running time now, as a function of B and k? You may assume m is a power of 2.

Answer: Now there can be at most k nonempty intervals at each level of the divide-and-conquer tree. So the maximum number of nonempty intervals (the maximum number of calls to #roots) at each level of the tree can grow like $1, 2, 4, 8, ..., k/2, k, k, ...$ *So there are 2 cases: if* $2^B \leq k$*, the complexity is still* $O(2^B)$*, and if* $2^B > k$*, the complexity is* $O(k + k(B - \log_2 k))$ *which you may simplify to* $O(kB)$*.*

Question 3 (20 points).

a) Suppose you have to multiply the polynomials $a(x) = x$ and $b(x) = x^2 + 1$ using the Fast Fourier Transform. Choose an appropriate N and find the FFT of the two polynomials.

Answer: We have N=4 because the product polynomial will have degree 3. So $\omega = e^{2\pi i/4} = i$, and the two FFTs are the polynomials evaluated at $1, i, -1, -i$, so they are $(1, i, -1, -i)$ and $(2, 0, 2, 0)$

b) Once we have the two FFTs, how do we recover the product of the two polynomials? What is the FFT of their product (numerically)?

Answer: First we multiply the two FFTs component-wise, and we get $(2, 0, -2, 0)$ (which is also the FFT of their product). Then we take the inverse FFT, which in this case is equivalent to multiplying by $\frac{1}{4}FFT(-i)$, and we get $(0, 1, 0, 1)$, which represents $x^3 + x$

c) In lectures and problems you saw how to use the Fast Fourier Transform to multiply polynomials. Now you will explore a new application of the FFT: you will show how to use the FFT to find the period of a function. First let's look at the FFT matrix. For a given N, what is the value of ω?

Answer: $\omega = e^{2\pi i/N}$

d) What is the value of the entry at position (j, k) in the FFT matrix (where $0 \leq j < N$ and $0 \leq k < N$)?

Answer: $\omega^{jk} = e^{2\pi ijk/N}$

e) Now we will look at a periodic function $f(z)$, i.e. there exists a number T such that $f(z) = f(z+T)$ for all z (T is called the period of the function). One of the simplest periodic functions is $\cos(z)$. In particular, we will look at a cosine function of period $N/7$, so the function is actually $f(z) = \cos(14\pi/Nz)$. Now we will take the FFT of this function in the following way: multyply the FFT matrix you found above with the column vector that has $f(z)$ at position z. The result will be another column vector V. List all elements of V that are non-zero (i.e. at which positions V has a non-zero entry). Hint: $\cos z = \frac{e^{iz}+e^{-iz}}{2}$, where $i = \sqrt{-1}$.

Answer: $V(x) = \sum_{y=0}^{N-1} FFT(x,y)f(y) = \sum_{y=0}^{N-1} e^{2\pi ixy/N} \frac{e^{14\pi iy/N}+e^{-14\pi iy/N}}{2} = 1/2 \sum_{y=0}^{N-1} e^{2\pi iy(x+7)/N} + 1/2 \sum_{y=0}^{N-1} e^{2\pi iy(x-7)/N}$

Let's look at the first sum: it is geometric with ratio $e^{2\pi i(x+7)/N}$, and evaluating it we get: $S_1 = N/2$ if $e^{2\pi i(x+7)/N} = 1$, and $\frac{e^{2\pi i(x+7)}-1}{e^{2\pi i(x+7)/N}-1} = 0$ otherwise. $e^{2\pi i(x+7)/N} = 1$ is equivalent to saying that $x = N-7$.

Similarly, the second sum is geometric with ratio $e^{2\pi i(x-7)/N}$, and evaluating it we get: $S_2 = N/2$ if $e^{2\pi i(x-7)/N} = 1$, and $\frac{e^{2\pi i(x-7)}-1}{e^{2\pi i(x-7)/N}-1} = 0$ otherwise. $e^{2\pi i(x-7)/N} = 1$ is equivalent to saying that $x = 7$

To conclude, the only non-zero elements of V are at positions 7 and $N-7$

Question 3 (20 points).

a) Suppose you have to multiply the polynomials $a(x) = x + 1$ and $b(x) = x^2$ using the Fast Fourier Transform. Choose an appropriate N and find the FFT of the two polynomials.

Answer: We have N=4 because the product polynomial will have degree 3. So $\omega = e^{2\pi i/4} = i$, and the two FFTs are the polynomials evaluated at $1, i, -1, -i$, so they are $(2, 1+i, 0, 1-i)$ and $(1, -1, 1, -1)$

b) Once we have the two FFTs, how do we recover the product of the two polynomials? What is the FFT of their product (numerically)?

Answer: First we multiply the two FFTs component-wise, and we get $(2, -1-i, 0, -1+i)$ (which is also the FFT of their product). Then we take the inverse FFT, which in this case is equivalent to multiplying by $\frac{1}{4}FFT(-i)$, and we get $(0, 0, 1, 1)$, which represents $x^3 + x^2$

c) In lectures and problems you saw how to use the Fast Fourier Transform to multiply polynomials. Now you will explore a new application of the FFT: you will show how to use the FFT to find the period of a function. First let's look at the FFT matrix. For a given N, what is the value of ω?

Answer: $\omega = e^{2\pi i/N}$

d) What is the value of the entry at position (j, k) in the FFT matrix (where $0 \leq j < N$ and $0 \leq k < N$)?

Answer: $\omega^{jk} = e^{2\pi ijk/N}$

e) Now we will look at a periodic function $f(z)$, i.e. there exists a number T such that $f(z) = f(z+T)$ for all z (T is called the period of the function). One of the simplest periodic functions is $\cos(z)$. In particular, we will look at a cosine function of period $N/11$, so the function is actually $f(z) = \cos(22\pi/Nz)$. Now we will take the FFT of this function in the following way: multyply the FFT matrix you found above with the column vector that has $f(z)$ at position z. The result will be another column vector V. List all elements of V that are non-zero (i.e. at which positions V has a non-zero entry). Hint: $\cos z = \frac{e^{iz}+e^{-iz}}{2}$, where $i = \sqrt{-1}$.

Answer: $V(x) = \sum_{y=0}^{N-1} FFT(x,y)f(y) = \sum_{y=0}^{N-1} e^{2\pi ixy/N} \frac{e^{22\pi iy/N}+e^{-22\pi iy/N}}{2} = 1/2 \sum_{y=0}^{N-1} e^{2\pi iy(x+11)/N} + 1/2 \sum_{y=0}^{N-1} e^{2\pi iy(x-11)/N}$

Let's look at the first sum: it is geometric with ratio $e^{2\pi i(x+11)/N}$, and evaluating it we get: $S_1 = N/2$ if $e^{2\pi i(x+11)/N} = 1$, and $\frac{e^{2\pi i(x+11)}-1}{e^{2\pi i(x+11)/N}-1} = 0$ otherwise. $e^{2\pi i(x+11)/N} = 1$ is equivalent to saying that $x = N - 11$.

Similarly, the second sum is geometric with ratio $e^{2\pi i(x-11)/N}$, and evaluating it we get: $S_2 = N/2$ if $e^{2\pi i(x-11)/N} = 1$, and $\frac{e^{2\pi i(x-11)}-1}{e^{2\pi i(x-11)/N}-1} = 0$ otherwise. $e^{2\pi i(x-11)/N} = 1$ is equivalent to saying that $x = 11$

To conclude, the only non-zero elements of V are at positions 11 and $N - 11$

Question 3 (20 points).

a) Suppose you have to multiply the polynomials $a(x) = x + 1$ and $b(x) = x^2 + 1$ using the Fast Fourier Transform. Choose an appropriate N and find the FFT of the two polynomials.

Answer: We have N=4 because the product polynomial will have degree 3. So $\omega = e^{2\pi i/4} = i$, and the two FFTs are the polynomials evaluated at $1, i, -1, -i$, so they are $(2, 1+i, 0, 1-i)$ and $(2, 0, 2, 0)$

b) Once we have the two FFTs, how do we recover the product of the two polynomials? What is the FFT of their product (numerically)?

Answer: First we multiply the two FFTs component-wise, and we get $(4, 0, 0, 0)$ (which is also the FFT of their product). Then we take the inverse FFT, which in this case is equivalent to multiplying by $\frac{1}{4}FFT(-i)$, and we get $(1, 1, 1, 1)$, which represents $x^3 + x^2 + x + 1$

c) In lectures and problems you saw how to use the Fast Fourier Transform to multiply polynomials. Now you will explore a new application of the FFT: you will show how to use the FFT to find the period of a function. First let's look at the FFT matrix. For a given N, what is the value of ω?

Answer: $\omega = e^{2\pi i/N}$

d) What is the value of the entry at position (j, k) in the FFT matrix (where $0 \leq j < N$ and $0 \leq k < N$)?

Answer: $\omega^{jk} = e^{2\pi ijk/N}$

e) Now we will look at a periodic function $f(z)$, i.e. there exists a number T such that $f(z) = f(z+T)$ for all z (T is called the period of the function). One of the simplest periodic functions is $\cos(z)$. In particular, we will look at a cosine function of period $N/13$, so the function is actually $f(z) = \cos(26\pi/Nz)$. Now we will take the FFT of this function in the following way: multyply the FFT matrix you found above with the column vector that has $f(z)$ at position z. The result will be another column vector V. List all elements of V that are non-zero (i.e. at which positions V has a non-zero entry). Hint: $\cos z = \frac{e^{iz}+e^{-iz}}{2}$, where $i = \sqrt{-1}$.

Answer: $V(x) = \sum_{y=0}^{N-1} FFT(x,y)f(y) = \sum_{y=0}^{N-1} e^{2\pi ixy/N} \frac{e^{26\pi iy/N}+e^{-26\pi iy/N}}{2} = 1/2 \sum_{y=0}^{N-1} e^{2\pi iy(x+13)/N} + 1/2 \sum_{y=0}^{N-1} e^{2\pi iy(x-13)/N}$

Let's look at the first sum: it is geometric with ratio $e^{2\pi i(x+13)/N}$, and evaluating it we get: $S_1 = N/2$ if $e^{2\pi i(x+13)/N} = 1$, and $\frac{e^{2\pi i(x+13)}-1}{e^{2\pi i(x+13)/N}-1} = 0$ otherwise. $e^{2\pi i(x+13)/N} = 1$ is equivalent to saying that $x = N - 13$.

Similarly, the second sum is geometric with ratio $e^{2\pi i(x-13)/N}$, and evaluating it we get: $S_2 = N/2$ if $e^{2\pi i(x-13)/N} = 1$, and $\frac{e^{2\pi i(x-13)}-1}{e^{2\pi i(x-13)/N}-1} = 0$ otherwise. $e^{2\pi i(x-13)/N} = 1$ is equivalent to saying that $x = 13$

To conclude, the only non-zero elements of V are at positions 13 and $N - 13$

Question 4: True/False (20 points). Circle your answer.

T or F: $T(n) = T(n-1) + 200$ has solution $\Theta(n)$.
Answer: True. At each step a constant amount of work is done. Since there are n steps until we reach 1, the running time is $\Theta(n)$.

T or F: $T(n) = 4T(n-1) + 6$ has solution $\Theta(4^n)$
Answer: True. $T(n) = \sum i = 0^n 4^i \cdot \Theta(1) = \Theta(4^n)$

T or F: An algorithm for sorting an array of integers must take $\Omega(n \log n)$ (where n is the size of the array).
Answer: False. For an array of integers, we can use bucket-sort, which has a running time of $O(n + m)$, where m is the difference between the maximal and the minimal elements of the array.

T or F: We want to multiply a polynomial of degree 3 by a polynomial of degree 7. The following statements are *all* true: the final polynomial has degree 10; the minimum number of points we need to sample the final polynomial is 11; if we use the divide-and-conquer FFT algorithm in the book, we need to choose N=16 as the size of the FFT.
Answer: All True.

T or F: In any directed graph, the sum of in-degrees and out-degrees over all vertices is an even number.
Answer: True, because each edge (u, v) adds two to the sum, one to the out-degree of u and one to the in-degree of v.

T or F: In any connected directed graph, by removing one edge we can create at most 2 Strongly Connected Components.
Answer: False: if you have n vertices connected in a ring, and no other edges, removing any one edge makes each vertex a strongly connected component.

T or F: In any connected undirected graph, we have to remove at least k edges to create $k+1$ connected components.
Answer: True, because removing one edge (u, v) from a connected component breaks it into at most 2 connected components (those vertices whose only path to v passes through (u, v), and those with a path to v excluding (u, v)).

T or F: $(\log_2 n)^n = \Omega(n^{\log_2 n})$.
Answer: True: Take logarithms of both sides. On the left you get $n \log_2 \log_2 n$, which is asymptotically larger than $(\log_2 n)^2$.

T or F: Adding an edge to a DAG creates a cycle.
Answer: False: If there are no edges in the DAG at all, adding any one edge (u, v) does not create a cycle.

T or F: If $f = O(g)$ then $2^f = O(2^g)$.
Answer: False: Consider $f(n) = 2n$ and $g(n) = n$.

Question 4: True/False (20 points). Circle your answer.

T or F: Adding an edge to a DAG creates a cycle.
Answer: False: If there are no edges in the DAG at all, adding any one edge (u, v) does not create a cycle.

T or F: $S(n) = 4S(n-1) + 1$ has solution $\Theta(4^n)$.
Answer: True. $S(n) = \sum i = 0^n 4^i \cdot \Theta(1) = \Theta(4^n)$

T or F: An algorithm for sorting an array of integers must take $\Omega(k \log k)$ (where k is the size of the array).
Answer: False. For an array of integers, we can use bucket-sort, which has a running time of $O(k + m)$, where m is the difference between the maximal and the minimal elements of the array.

T or F: We want to multiply a polynomial of degree 4 by a polynomial of degree 8. The following statements are *all* true: the final polynomial has degree 12; the minimum number of points we need to sample the final polynomial is 13; if we use the divide-and-conquer FFT algorithm in the book, we need to choose N=16 as the size of the FFT.
Answer: All True.

T or F: In any directed graph, the sum of in-degrees and out-degrees over all vertices is an odd number.
Answer: False, because each edge (u, v) adds two to the sum, one to the out-degree of u and one to the in-degree of v, so it is even.

T or F: In any connected undirected graph, we have to remove at least s edges to create $s+1$ connected components.
Answer: True, because removing one edge (u, v) from a connected component breaks it into at most 2 connected components (those vertices whose only path to v passes through (u, v), and those with a path to v excluding (u, v)).

T or F: $(\log_3 r)^r = \Omega(r^{\log_3 r})$.
Answer: True: Take logarithms of both sides. On the left you get $r \log_3 \log_3 r$, which is asymptotically larger than $(\log_3 r)^2$.

T or F: If $g = O(h)$ then $2^g = O(2^h)$.
Answer: False: Consider $g(n) = 2n$ and $h(n) = n$.

T or F: $S(n) = S(n-1) + 70$ has solution $\Theta(n)$.
Answer: True. At each step a constant amount of work is done. Since there are n steps until we reach 1, the running time is $\Theta(n)$.

T or F: In any connected directed graph, by removing one edge we can create at most 2 Strongly Connected Components.
Answer: False: if you have n vertices connected in a ring, and no other edges, removing any one edge makes each vertex a strongly connected component.

Question 4: True/False (20 points). Circle your answer.

T or F: We want to multiply a polynomial of degree 5 by a polynomial of degree 8. The following statements are *all* true: the final polynomial has degree 13; the minimum number of points we need to sample the final polynomial is 14; if we use the divide-and-conquer FFT algorithm in the book, we need to choose N=16 as the size of the FFT.
Answer: All True.

T or F: An algorithm for sorting an array of integers must take $\Omega(s \log s)$ (where s is the size of the array).
Answer: False. For an array of integers, we can use bucket-sort, which has a running time of $O(s + m)$, where m is the difference between the maximal and the minimal elements of the array.

T or F: In any directed graph, the sum of in-degrees and out-degrees over all vertices is an odd number.
Answer: False, because each edge (u, v) adds two to the sum, one to the out-degree of u and one to the in-degree of v, so it is even.

T or F: In any connected directed graph, by removing one edge we can create at most 2 Strongly Connected Components.
Answer: False: if you have n vertices connected in a ring, and no other edges, removing any one edge makes each vertex a strongly connected component.

T or F: In any connected undirected graph, we have to remove at least t edges to create $t+1$ connected components.
Answer: True, because removing one edge (u, v) from a connected component breaks it into at most 2 connected components (those vertices whose only path to v passes through (u, v), and those with a path to v excluding (u, v)).

T or F: $R(m) = R(m - 1) + 200$ has solution $\Theta(m)$.
Answer: True. At each step a constant amount of work is done. Since there are m steps until we reach 1, the running time is $\Theta(m)$.

T or F: If $r = O(s)$ then $2^r = O(2^s)$.
Answer: False: Consider $r(n) = 2n$ and $s(n) = n$.

T or F: $(\log_{10} m)^m = \Omega(m^{\log_{10} m})$.
Answer: True: Take logarithms of both sides. On the left you get $m \log_{10} \log_{10} m$, which is asymptotically larger than $(\log_{10} m)^2$.

T or F: Adding an edge to a DAG creates a cycle.
Answer: False: If there are no edges in the DAG at all, adding any one edge (u, v) does not create a cycle.

T or F: $R(m) = 4R(m - 1) + 6$ has solution $\Theta(4^m)$.
Answer: True. $R(m) = \sum i = 0^m 4^i \cdot \Theta(1) = \Theta(4^m)$