THE UNIVERSITY OF MANITOBA

DATE_____ Algorithms Candidacy Examination

Enter NAME and STUDENT NUMBER:

_____
STUDENT NUMBER

_____
WRITE NAME IN FULL ON THIS LINE

_____
PRINT NAME IN FULL ON THIS LINE

Instructions: Attempt an answer each question as best you can. There are many questions so allocate your time accordingly. It is generally impossible to answer all in completion so pace yourself and answer as many as can.

Also this test requires to modular arithmetic calculators such as through Wolfram Alfa or a number of other on-line resources.

1) Questions of run time complexity for recursive algorithms. Where T(n) is run time for problems of size n.

a) Argue that the solution to the recurrence $T(n) = T(n/3) + T(2n/3) + cn$, where c is a constant, is $\Omega(n \lg n)$ by appealing to a recursion tree.

b) Determine the Big-Oh complexity using visualization the following recurrence relations:
$T(n) = T(n/2) + cn$
$T(n) = 2T(n/2) + cn$
$T(n) = T(n/2) + c$
$T(n) = 2T(n-1) + c$
$T(n) = T(n-1) + c$

c) Use a recursion tree to give an asymptotically tight solution to the recurrence $T(n) = T(n - a) + T(a) + cn$, where $a \geq 1$ and $c > 0$ are constants.

d) Determine the Big-Oh complexity using the Master Method the following recurrence relations:

$T(n) = T(n/2) + cn$
$T(n) = 2T(n/2) + cn$
$T(n) = T(n/2) + c$
$T(n) = 2T(n-1) + c$
$T(n) = T(n-1) + c$

Appendix 1): Master Method.
Bound a recurrence of the form:
$T(n) = aT(n/b) + f(n) \quad a \geq 1 , \quad b > 1$

1. if $f(n) = O(n^{\log_b a - \varepsilon})$, $\varepsilon > 0$ then $T(n) = \Theta(n^{\log_b a})$
2. if $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$
3. if $f(n) = \Omega(n^{\log_b a + \varepsilon})$, $\varepsilon > 0$ and $af(n/b) \leq cf(n)$ for $c < 1$ then $T(n) = \Theta(f(n))$

2) General Algorithm Knowledge

a) Draw a map of "algorithm and problem world". Include complexity classes and example of problems as well as algorithms in those sets or classes. If possible provide the complexity of the examples you use. Use the correct complexity notation. i.e. At the least used problem complexity classes of P, NP, NPC and exponential.

b) What does the term NP mean in terms of a short certificate or being able to verify the solution? (Not just its definition)

c) What does the term NPC mean in terms of polynomial time mapping or transformations?

d) You are working on a graph problem of unknown complexity. How would you establish it is in NPC in terms of mappings?

e) Once established to be in the set of NPC problems, how would you go about trying to "solve" the problem?

f) Using visualization solve the recurrence $T(N)=4T(N/2) +N$

3) Randomization and probability.

a) Given a sporting event play-off with 16 teams. Teams are paired and play a best of seven series (the first team to win 4 games, wins the series and no further games are played). The losing team is eliminated and the winning team goes on to the next round. After 4 rounds there is only one team remains as the overall winner.

What is the expected number of games that will be played in a play-off series? That is, how many games does the tournament winner expect to play? The tournament winner is the team that wins all 4 of its series.

Provide a Monte Carlo solution to the problem. That is, provide a pseudo-code algorithm and any other justification, comments or arguments that you would like to add.

b)Consider Quicksort

```
QUICKSORT(A, p, r)
1 if p < r
2    then q ← PARTITION(A, p, r)
3         QUICKSORT(A, p, q - 1)
4         QUICKSORT(A, q + 1, r)
```

To sort an entire array A, the initial call is QUICKSORT(A, 1, length[A]).

The key to the algorithm is the PARTITION procedure, which rearranges the subarray A[p…r] in place.

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p - 1
3  for j ← p to r - 1
4       do if A[j] ≤ x
5             then i ← i + 1
6                  exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

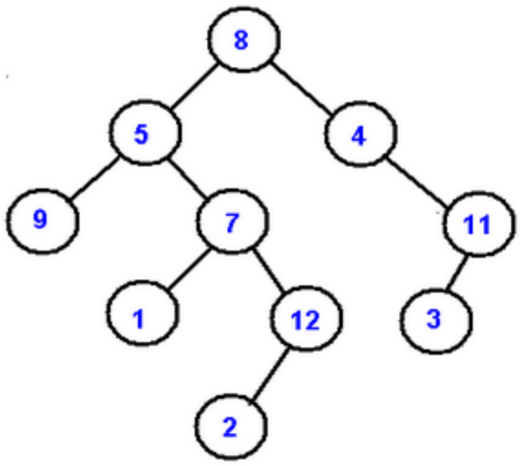
Argue why the complexity of Quicksort is typically considered O(nlgn), while the worst case is $O(n^2)$. Determine the complexity (solve the T(n) recurrance) of the algorithm provided, for corner cases of best and worst case partitions.

How can randomization prevent the worst case from ever occurring with vanishingly small probability? Hint: The answer has to do with selection of the partition element.

4) Data Structures

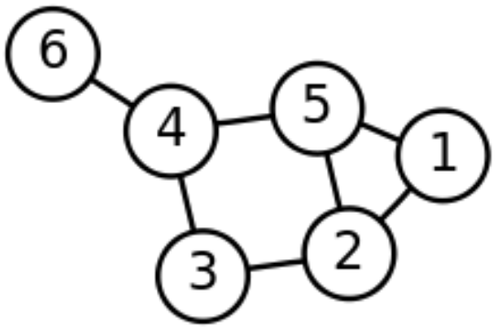
a) Discuss advantages and disadvantages of using arrays or linked lists in term of operations of insertion, deletion and searching. Consider the case of sorted data.

For the binary tree below, provide an example of output from a depth first traversal and breath first traversal.



b) Given that it is better to use a hash function depends on all of the bits for the key, which hash function is better and why?
h(k) = k mod 8 or h(k) = k mod 7. .

c) Given a graph:



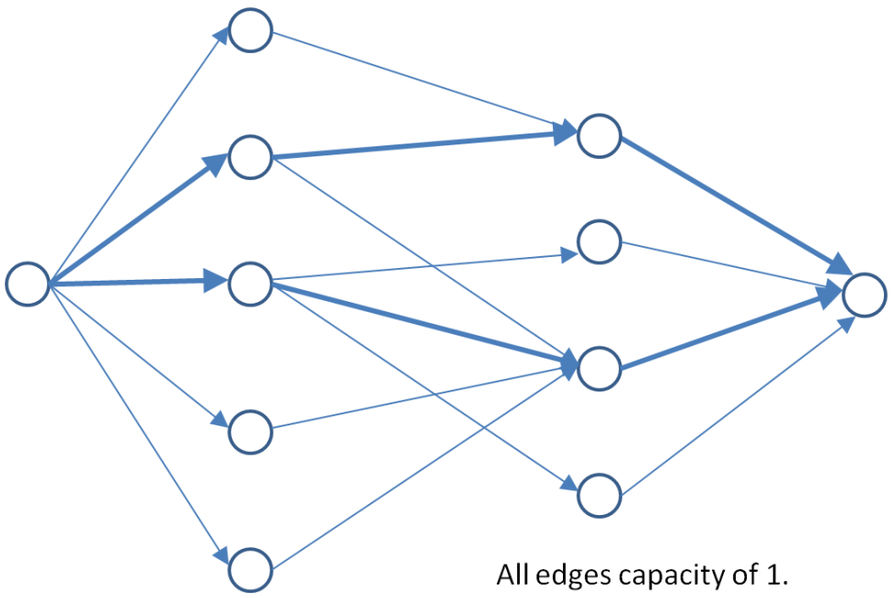
Provide an adjacency list representation of the graph as well as adjacency matrix representation of the graph.

Demonstrate a Breath First Search for the graph above for finding the shortest path from node 3 to node 1.

5) Max-Flow: The Ford-Fulkerson method for finding maximum flow is iterative. We start with f(u, v) = 0 for all u, v ε V, giving an initial flow of value 0. At each iteration, we increase the flow value by finding an "augmenting path," which we can think of simply as a path from the source s to the sink t along which we can send more flow, and then augmenting the flow along this path. We repeat this process until no augmenting path can be found.

FORD-FULKERSON-METHOD(G, s, t)
1 initialize flow f to 0
2 while there exists an augmenting path p
3 do augment flow f along p
4 return f

For the graph below a flow is shown along the bold (dark) lines. Form this point forward complete the method to obtain maximal flow. Hint: Create a residual network and then BFS search can be employed to find an augmenting path. Capacity along an edge on a residual network is estimated as follows: $c_f(u,v)=c(u,v)-f(u,v)$.



6) Number theory question:

Generating a large number that is probably prime.

Given that you want to generate a probably prime 200 digit number N, with an uncertainty of 1 part in 2^{50}.

Assume that composites numbers are detected using base-a pseudo-prime tests (Miller-Rabin tests). We want to know how many modular exponentiation computations (base-a pseudo-prime tests) of the type $a^{n-1}=1$ modn will be performed on average before a prime with the required certainty is found. Composites are eliminated if $a^{n-1}!=1$.

Hints:

i) The number of prime numbers up to N is approximately N/lnN.

ii) If the probability of an event (success) is p, the expected number of trials until a success is 1/p.

a) The probability of a 200 digit number (picked at random) being prime is ____, (calculated from the density of prime numbers). What is the Expected number of trials before successfully selecting a prime at random?

b) If the pseudo-primality test has a probability of 3/4 of eliminating composites from the search for each iteration within the Miller-Rabin test, what is the number of Expected number of iterations (of the type $a^{n-1}=1$ modn) required to eliminate a composite from contention as a prime?

c) Once a number is generated that is very likely prime, how many iterations of the Miller-Rabin pseudo primality test are required to be almost certain ($1-1/2^{50}$) that the number selected is almost certainly prime?

d) Put that all together. What is the Expected number of computations of the type $a^{n-1}=1$ modn before a 200 digit very likely prime number is generated?

e) Assume someone discovered a polynomial time deterministic algorithm $O(\log^{11}n)$, which they have. How does it compare to the above result of the time it took to generate a big prime using a pseudorandom algorithm that is probabilistic?

7) RSA type question:
Your friend selects 11 and 17 as p and q for their RSA encryption implementation.

a) In general p and q (two large primes are selected), maybe of 100 and 120 decimals.
n = p.q (this is the unique prime factors for n). Generate n.

b) Select a small odd integer e relatively prime to ø(n), i.e. to (p-1)*(q-1). (use gcd)

```
Euclid(a,b)
if b = 0 then return a
else return Euclid(b, a mod b)
```

Alternatively, between e= 7 or 11, which is a better choice for e?

Compute d as the multiplicative increase of e, modulo ø(n), d exists, easy to compute using the extended_euclid and modular equation solver.

```
Modular_linear_equation_solver (a, b, n) (Trying to solve ax = b (mod n))
(d, x', y') <= Extended_Euclid (a, n)
if d | b then x0  <=  x'(b/d) mod n
for i = 0 to d-1 print (x0 + i(n/d)) mod n
else
print ( no solutions) because d did not divide b
```

```
Extended-Euclid(a, b)
if b = 0 then return (a, 1, 0)
(d', x', y') ← Extended-Euclid(b, a mod b)
(d, x, y) ← (d', y', x' - ⌊a / b⌋ y')
return ( d, x, y)
```

Alternatively, verify which of the following would be d provided e=7.

c) P = (e, n) as the friends public key. S = (d, n) as the friend's secret key.

The message you want to send is "Hello" encoded as 48 65 6C 6C 6F in hex or 72 101 108 108 111 in decimal.
Symbolically, **demonstrate** how you would encrypt such a big number and send it to your friend. **Encode** the H in Hello.

d) Similarly, symbolically, **demonstrate** how the recipient of such an important message would decrypt the message. **Decode** the first symbol.

8) Diffie Hellman:

You want to get the code 101 to your friend so that they can decode a message using a secret key algorithm. You decide to use Diffie Hellman. You and you friend agree to use 17 as the big prime p, and 3 as g, a generator for Z^*_{17}.

You **pick a** as your secret and **calculate** $g^a \bmod p = A$

Your friend **picks b** as their secret and **calculates** $g^b \bmod p = B$

You send your friend A, and they send you B.

You **calculate** $B^a \bmod p$

Your friend **calculates** $A^b \bmod p$

a) Are these two equal? If so they represent the shared secret K.

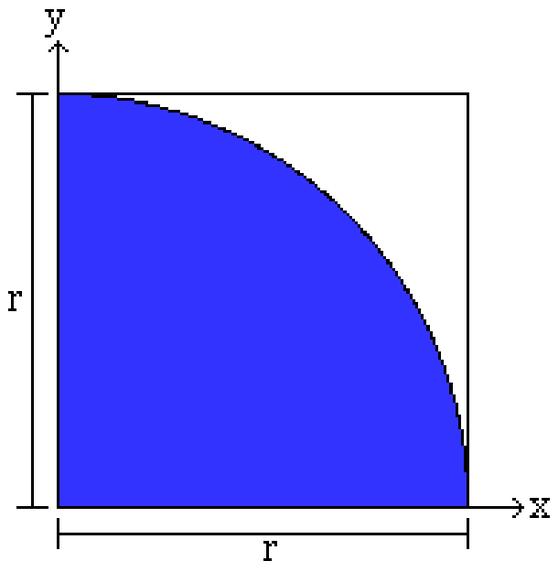
b) How do you use K to get the code 101 to your friend?

9) Monte Carlo Question:

Using the following 15 pairs of random number between 0 and 99999 and the image below calculate π. (assume r=1) (from "A million Random Digits")(Normalize as required)

Describe your method in a readable pseudo-code or clear paragraph.

x	y
73735	45963
02965	58303
98859	23851
33666	62570
81666	26440
15838	47174
89793	34378
78155	22466
16381	66207
75002	80827
99982	27601
84543	87442
77757	54043
80871	32792
30500	28220



What is the estimate of π, when you **also** use these 15 random number pairs? (from random.org)

x	y
38484	79938
40719	73052
20779	8438
82213	18481
43151	48517
39667	28343
42187	6989
46297	34392
81222	92435
7661	16363
92054	97838
65137	17749
18243	8098
51066	12035
97255	67682

Hint: Pythagoras Theorem $a^2+b^2=c^2$ and area of a circle is πr^2

10) Complexity:

a) If a problem is $\Omega(n)$, does it imply that it **can be** solved with an algorithm that is O(n)? Yes or No.
Provide an example

b) If a problem is $\Omega(n^2)$, does it imply that it **can not be** solved with an algorithm that is O(n)? Yes or No.

c) If a problem is in an element of the set NP, does it imply that it **can be** solved with a polynomial time algorithm? Yes or No.

d) If a problem is in an element of the set NP, does it imply that a solution **can be** checked or verified with a polynomial time algorithm? Yes or No.

e) If an algorithmic problem is an element of the set NPC, and you can transform your algorithmic problem **to it** with a polynomial time mapping or reduction, have you now proven that your algorithmic problem is an element of NPC? Yes or No.
(Assume your algorithmic problem can be checked in polynomial time.) **Draw** a picture.

f) How would you establish that an algorithmic problem is an element of the set of problems that are NPC?

11) An Alternative

In the event these questions were not specifically studied for. Present a problem and your solution below.