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

Practice Examination for February 27, 2020

The entire practice examination is 380 points.

1. True or False. [5 points each]

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

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

   (c) _____ If S is a set of distinct items, we say that an $x \in S$ has *rank* k if there are exactly k members of S which are less than or equal to x. If, while implementing quicksort to sort a set of n distinct items, we always pick the pivot (cut) item to be an item whose rank is at least 10% of the size of the subset we are currently sorting, and never more than 90% of the size of that subset, the time complexity of our implementation will be $\Theta(n \log n)$.

   (d) _____ Any comparison-based sorting algorithm must use at least $\log_2(n!)$ comparison to sort n items, in the worst case.

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

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

   _____

   (b) Name a well-known divide-and-conquer searchng algorithms.

   _____

   (c) Name three well-known quadratic time sorting algorithms.

   _____

   _____

   _____

   (d) Name two well-known divide-and-conquer time sorting algorithms.

   _____

   _____

3. Solve the recurrences. Give asymptotic answers in terms of n, using either O, Ω, or Θ, whichever is most appropriate. [10 points each.]

(a) $F(n) = 4F(n/2) + n^2$.

(b) $G(n) \geq G(n-1) + \lg n$

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

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

(e) $F(n) = 4F\left(\frac{3n}{4}\right) + n^5$ (No, you don't need a calculator.)

4. [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;
    }
 }
```

Consider the question of how many lines of output the execution of `george(n)` would produce. Write down an appropriate recurrence for this question, and give an asymptotic solution in terms of `n`, using either O, Ω, or Θ, whichever is most appropriate.

5. [20 points] Walk through radix (or bucket) sort, where you sort the following zip codes:
   `89110 89154 89254 91245 90016 90004 90005 21014`

6. [30 points]

   (a) Illustrate a queue implemented as a circular linked list with a dummy node. The contents of the queue, from front to rear, should be the four items Ann, Ted, Sue, Tom.

   (b) Show the steps of dequeue, using the same example.

   (c) Redraw the original figure, and show the steps of enqueue, were the item Bob is added to the queue.

7. [30 points] Consider the SPLIT procedure used in Quicksort, also known as PARTITION, or (in our textbook) PIVOT. Write pseudo-code for that procedure (not any of the other parts of quicksort) and write a loop invariant for your code.

8. [30 points] Walk through heapsort, where the input file is as given below.

MDYCOSEZVQWBANLH

9. [30 points] Walk through polyphase mergesort, where the input file is as given below.

MDYCOSEZVQWBANLH

10. [20 points] The following code implements selection sort on an array A[n]. There are two loops, each of which has a loop invariant. Find both loop invariants. We let Final[i] be the value of A[i] after A is sorted. Thus, A[i] = Final[i] when the code is finished.

```
void swap(int&x,int&y)
 {
  int temp = x;
  x = y;
  y = temp;
 }

void sort()
 {
  for(int i = 0; i < n; i++)
   for(int j = i+1; i < n; j++)
    if(A[i] > A[j]) swap(A[i],A[j]);
 }
```

For ease of understanding, we rewrite the function using while loops.

```
void sort()
 {
  int i = 0;
  // Outer Loop Invariant holds
  while (i < n-1)
   {
    // Outer Loop Invariant holds
    int j = i+1;
    // Inner Loop Invariant holds
    while(j < n)
     {
      // Inner Loop Invariant holds
      if{A[j] < A[i])
       swap(A[j],A[i])
      j++
      // Inner Loop Invariant holds
     }
    // Inner Loop Invariant holds
    i++;
    // Outer Loop Invariant holds
   }
  // Outer Loop Invariant holds
 }
```

11. [20 points] You are working on computer which lacks multipliation and addition. However, it can add or subtract 1 or 2. What does this function do? What is its loop invariant?

```
int double(int n)
 // input condition: n >= 0
 {
  int p = n;
  int q = 0;
  while(p > 0)
   {
    p = p-1;
    q = q+2;
   }
  return q;
 }
```

12. [20 points]

For any real number x and non-negative integer a, x^a can be computed using recursion. Recall that $x^0 = 1$ if x is real.[1]

```
float power(float x, int a)
 // input condition: a >= 0
 {
  if(a == 0) return 1.0;
  else if(a%2) // a is odd
   return x*power(x,a-1);
  else
   return power(x*x,a/2);
 }
```

We prove correctness of this code by strong induction on a.

Proof: By strong induction. Let x be any real number, and let $a \geq 0$ be an integer.

Inductive Hypothesis: `power(x,i)` returns x^i for all $i < a$.

Case 1. $a = 0$. The return value is $1.0 = x^0$.

Case 2: a is odd. By the inductive hypothesis `power(x,a-1)` returns x^{a-1}. Hence `power(x,a)` returns $x \cdot x^{a-1} = x^a$.

Case 3: a is even and greater than zero. `power(x,a)` returns `power(x*x,a/2)` which is equal to $(x^2)^{a/2} = x^a$ by the inductive hypothesis. ∎

We now give a non-recursive version of the function, using a loop invariant to proove correctness. We write `x^a` to mean x^a.

[1] What about 0.0^0? Isn't any power of zero equal to zero?

```
 1  float power(float x, int a)
 2  // input condition: a >= 0
 3   {
 4    float y = x;
 5    int b = a;
 6    float rslt = 1.0;
 7    // Loop invariant: b >= 0 and rslt*y^b == x^a
 8    while(b > 0)
 9     {
10      // Loop invariant: b >= 0 and rslt*y^b == x^a
11      if(b%2) rslt = rslt*y;
12      y = y*y;
13      b = b/2 // truncated integer division
14      // Loop invariant: b >= 0 and rslt*y^b == x^a
15     }
16     // Loop invariant: b >= 0 and rslt*y^b == x^a
17    return rslt;
18   }
```

This code has only one loop. We need to prove that the loop invariant holds before the first iteration of the loop, and that, if it holds at the beginning of an iteration, it holds at the end of that iteration. It follows that the loop invariant holds after the loop exits.

(a) LI holds at line 7, because `rslt*y^b == 1.0*x^a`

(b) LI holds at line 10 for the first iteration, since it holds at line 7 and none of the variables have been changed.

(c) LI holds at line 10 for any other iteration (after the first, that is) since it holds at line 14 of the previous iteration and none of the variables have been changed.

(d) Assume LI holds at line 10. We must prove that lI holds at line 14 of the same iteration. This statement is non-trivial, since the values of the variables are changed during the loop.

To avoid confusion, we use write `rslt, y, b` for the values of those variables at line 10, and `\rslt', y', b'` for the values of the same variables at line 14. Then $y' = y^2$, while $b' = b/2$ if s is even, $(b-1)/2$ if b is odd.

rslt' = *rslt* if b is even, $rlst \cdot y$ if b is even, rslt if b is odd. We know that $b > 0$, since the loop condition must hold. Thus $b' \geq (b-1)/2 \geq 0$.

If b is even, we have $rslt' \cdot (y')^{b'} = rslt \cdot \left(y^2\right)^{b/2} = rslt \cdot y^b = x^a$ and we are done. If b is odd, we have $rslt' \cdot (y')^{b'} = rslt \cdot y \cdot \left(y^2\right)^{(b-1)/2} = rslt \cdot y \cdot y^{b-1} = rslt \cdot y^b = x^a$ and we are done.

(e) Since LI holds at the end of the last iteration, it must hold at line 16 since no values have been changed.

We now prove correctness of the code. Since the loop condition has failed, $b \leq 0$ at line 17. By the loop invariant, $b \geq 0$. Thus, $b = 0$, hence $y^b = 1$. Thus $rslt = rslt \cdot y^b = x^a$ at line 17, so the correct value is returned.

Give a useful loop invariant of each loop. Indicate the places in the code where the invariant holds.

13. [20 points]

Finding the Minimum of an Array

For this problem, assume that $A[0] \ldots A[n-1]$ is an array of integers, where n is a positive integer.

```
int i = 0;
int j = 0;

while(j < n-1){

  if(A[j] < A[i]) i = j;
  j++;

  }
```

14. [20 points]

Binary Search

For this problem, assume that $A[0] \ldots A[n-1]$ is a sorted array of integers, where n is a positive integer, and that B is an integer.

```
int lo = 0;
int hi = n;

while(lo < hi){

  int mid = (lo+hi)/2; // truncated division, as in C++
  if(A[mid] < B) lo = mid+1;
  else hi = mid;

  }

if (           ) cout << "Yes" << endl;  // I need to insert a condition here!
else cout << "No" << endl;
```

What do you think the condition of the if statement should be?

15. [20 points]

Sum of Positive Items

For this problem, assume that $X[0] \ldots X[n-1]$ is an array of float, where n is a positive integer. Write the loop invariant.

```
float sumPositive = 0.0;
int i = 0;
// Loop invariant:

while (i < n){
   // Loop invariant:

   if (X[i] > 0)
    sumPositive += X[i];
   i++;
   // Loop invariant:

  }
assert(i == n);
// Loop invariant:

// We can conclude that sumPositive is the sum of all positive A[i].
cout << sumPositive << endl;
```