University of Nevada, Las Vegas Computer Science 477/677 Fall 2026

Answers to Examination February 11, 2026

Name:_____

No books, notes, scratch paper, or calculators. Use pen or pencil, any color. Use the rest of this page and the backs of the pages for scratch paper. If you need more scratch paper, it will be provided. If you want anything on extra pages to be graded, staple those pages to your test and write, "Please grade this page."

The entire examination is 270 points.

1. Fill in each blank. Write Ω if that is correct; otherwise write O or Ω, whichever is correct. Recall that log means $\log_2$.

   (a) [5 points] $n^{.01} = \Omega\left(\log n\right)$

   (b) [5 points] $n^{\log n} = \Theta\left(2^{(\log n)^2}\right)$

   (c) [5 points] $\sum_{i=1}^{n} i^k = \Theta\left(n^{k+1}\right)$

   (d) [5 points] $\log n^2 = \Theta(\log n^3)$

   (e) [5 points] $\log(n!) = \Theta(n \log n)$

   (f) [5 points] $\log n = \Theta(\ln n)$

   (g) [5 points] $\log n = \Omega(\log \log n)$

2. Fill in the blanks.

   (i) [5 points] Any comparison-based sorting algorithm on a file of size n executes $\Omega(n \log n)$ comparisons. (Use Ω notation.)

   (ii) [15 points] The asymptotic time complexity to find an item in an ordered array of length n is $O(\log n)$ using the divide and conquer algorithm **binary search**.

   (iii) [10 points] In a priority queue, each item in the structure represents an **unfulfilled obligation**.

   (iv) [5 points] The height of a binary tree with 20 leaves must be at least **5**. Exact answer please: no partial credit.

3. [15 points] There are three ways that we've discussed in class to handle the *false overflow* problem for a queue. What are they?

   1. Wrap
   2. Slide
   3. Use a larger array

4. [15 points] The following portion of C++ code contains an array implementation of `queue`. Fill in the missing code for the operators "`enqueue`," "`dequeue`" and "`empty`."

```
struct queue
 {
  int A[N]; // N is a constant large enough to prevent false overflow.
  int rear = 0;
  int front = 0; // initially the queue is empty
 };

void enqueue(queue&q,int newitem) // inserts newitem into q
 {
  q.A[q.rear++] = newitem;
 }

bool empty(queue&q) // returns true if q is empty, false otherwise
 {
  return q.rear == q.front;
 }

int dequeue(queue&q) // returns an item from q and deletes that item
 {
  assert(not empty(q));
  return q.A[q.front++];
 }
```

5. [15 points] What follows is part of a C++ implementation of binary tree of integer. (It runs, I checked.) Fill in the missing code for the recursive functions `insert`, `find`, and `lcation`. The function `lcation` should be called `find`, but that would cause a redundancy conflict.

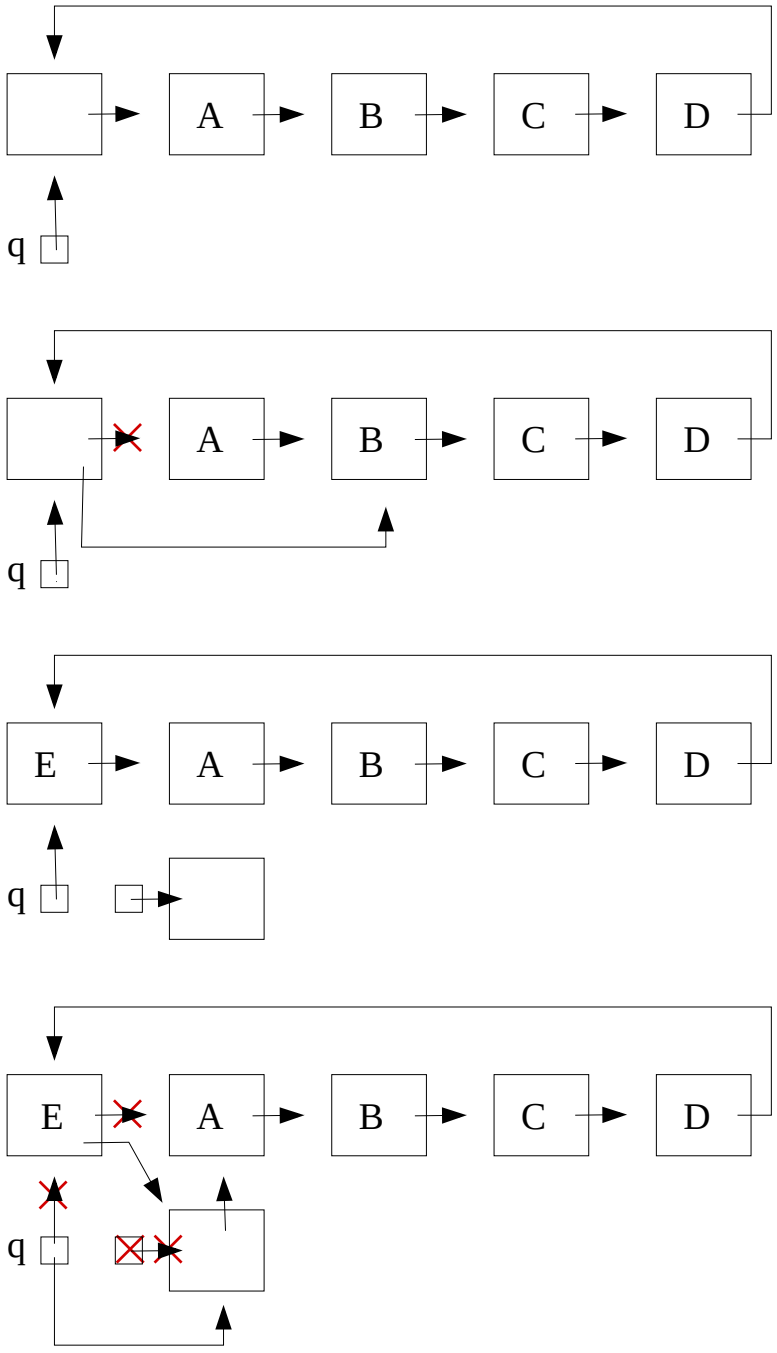
```
struct treenode;
typedef treenode*tree;
struct treenode
 {
  int item;
  tree left;
  tree right;
 };

tree mainroot; // the root of the binary search tree
```

```
void insert(tree&root,int n)
  // inserts n into the binary search tree.  Duplicates are not inserted.
 {
  if(root)
   root->item = n;
  else if(n < root->item) insert(root->left,n);
  else if(n > root->item) insert(root->right,n);
 }

bool find(tree root, int n)
  // returns true if some node contains n, false otherwise.
 {
  if(root)
   if(n == root->item) return true;
   else if (n < root->item) return find(root->left,n);
   else if (n > root->item) return find(root->right,n);
   else;
  else return false;
 }

tree lcation(tree root, int n)
  // returns pointer to the node which contains n, if any; NULL otherwise.
 {
  if(root)
   if(n == root->item) return root;
   else if (n < root->item) return lcation(root->left,n);
   else if (n > root->item) return lcation(root->right,n);
   else;
  else return nilptr;
 }
```

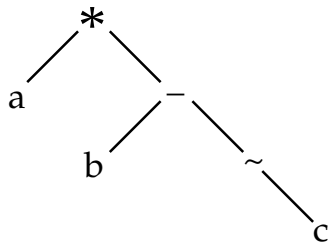
6. [20 points] A max-heap is implemented as a binary tree which is, in turn, is implemented as an array, which is shown in the first row of the table below. We execute two operations. Show the evolution of the heap when "Z" is inserted. Then, show the evolution of the heap when *deletemax* is executed.

8	W	H	R	E	G	F	Q	A	L	B
9	W	H	R	E	G	F	Q	A	Z	B
9	W	H	R	Z	G	F	Q	A	E	B
9	W	Z	R	H	G	F	Q	A	E	B
9	Z	W	R	H	G	F	Q	A	E	B
8	E	W	R	H	G	F	Q	A	E	B
8	W	E	R	H	G	F	Q	A	E	B
8	W	H	R	E	G	F	Q	A	E	B

11. [20 points] Show a circular queue with dummy node, with items A, B, C, D, in that order from front to rear. Then show how the queue changes when you insert E, and then show how the queue changes when you execute dequeue.



7. [10 points] Rewrite the prefix expression $*a - b \sim c$ in infix notation and postfix notation.



Infix: $a * (b - -c)$
Postfix: $abc \sim -*$

8. [10 points] If T is the parse tree for an algebraic expression, the postfix notation for that expression is obtained by writing the nodes of T in **postorder**. (Possible answers: **inorder, preorder, postorder, level order.**)

9. [15 points] Assume that the array int A[N] is part of the input.
What does `sorted` do? It decides whether the array A is sorted.
What does `gcd` do? It computes the greatest common divisor of a and b.
What does `ternary` do? It computes the base 3 numeral for n.
What does `power` do? It computes the value of x^n.

```
bool sorted()
 {
  bool rslt = true;
  for(int i = 1; i < N and rslt; i++)
   if(A[i] < A[i-1]) rslt = false;
  return rslt;
 }

int gcd(int a, int b)
 {
  //cout << "Computing gcd(" << a << "," << b << ")" << endl;
  if(a < 0)return gcd(-a,b);
  else if(b < 0)return gcd(a,-b);
  else if(b < a) return gcd(b,a);
  else if(a == 0) return b;
  else return gcd(b%a, a);
 }

 void mainwork(int n)
  {
   assert(n >= 0);
   if(n > 2) mainwork(n/3);
   cout << n%3;
  }
```

```
void ternary(int n)
 {
  mainwork(n);
  cout << endl;
 }


float power(float x, int n)
 {
  assert(n >= 0);
  float y = x;
  float z = 1.0;
  int m = n;
  while(m > 0)
   {
    if(m%2) z = z*y;
    y = y*y;
    m = m/2;
   }
  return z;
 }
```

10. [10 points] Find the constant C such that the n^{th} Fibonacci number is $\Theta(C^n)$.

    $F_n = F_{n-1} + F_{n-2}$. Assume $F_n = C^n$. We have the quadratic equation $C^2 - C - 1 = 0$. Since $C > 0$, we have $C = \dfrac{-1+\sqrt{5}}{2}$.

11. [10 points] The answer to each of these questions is either **bubbleup** or **bubbledown**. If a max-heap is implemented a binary tree, **bubbleup** is needed for insertion, while **bubbledown** is needed for deletemax.

12. For each of the code fragments below, find the asymptotic time complexity in terms of n.

    (a) [5 points]

    ```
    for(float x = n; x > 1; x = x/2)
    ```

    $\Theta(\log n)$

    (b) [5 points]

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

    $\Theta(\sqrt{n})$

    (c) [10 points] `for(float x = n; x > 2.0; x = sqrt(x))`

    $\Theta(\log \log n)$

    (d) [5 points]

    ```
    for(int i = 1; i < n; i=2*i)
    ```

    $\Theta(\log n)$

    (e) [5 points]

    ```
    for(int i = n; i > 1; i=i/2)
    ```

    $\Theta(\log n)$

    (f) [10 points]

    ```
    for(int i = n; i > 1; i--)
     for(int j = i; j > 1; j = j/2)
    ```

    $\Theta(n \log n)$

    (g) [10 points]

    ```
    for(int i = n; i > 1; i--)
     for(int j = n; j > i; j = j/2)
    ```

    $\Theta(n)$

    (h) [10 points]

    ```
    for(int i = 2; i < n; i=i*i)
    ```

$\Theta(\log \log n)$