

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

Answers to Study Guide for Examination October 8, 2026

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) $n^{.01} = \Omega(\log n)$

(b) $n^{\log n} = \Theta(2^{(\log n)^2})$

(c) $\sum_{i=1}^n i^k = \Theta(n^{k+1})$

(d) $\log n^2 = \Theta(\log n^3)$

(e) $\log(n!) = \Theta(n \log n)$

(f) $\log n = \Theta(\ln n)$

(g) $\log n = \Omega(\log \log n)$

2. Fill in the blanks.

- (i) No good programmer would ever use an unsorted list as a search structure. **F** (True or False.)
- (ii) Heapsort is a fast version of **selection sort**
- (iii) Treesort is a fast version of **insertion sort**
- (iv) Shell sort is inspired by **bubblesort** but is faster.
- (v) For any sorting algorithm where all branch points in the code are comparisons, the number of comparisons while sorting n items is $\Omega(n \log n)$. (Use Ω notation.)
- (vi) Name two different divide-and-conquer sorting algorithms. **quicksort mergesort**
- (vii) Name three kinds of priority queues: **stack queue heap**
- (viii) The worst case asymptotic time complexity of sorting an array of length n is:
 - i. $O(n^2)$ using bubblesort.
 - ii. $O(n^2)$ using quicksort.
 - iii. $O(n \log n)$ using heapsort.
- (ix) 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**.
- (x) In a priority queue, each item in the structure represents an **unfulfilled obligation**.
- (xi) The height of a binary tree with 20 leaves must be at least **4**. Exact answer please: no partial credit.

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

Slide.

Wrap. (circular queue)

Use a larger array.

4. The following C++ code is for an array implementation of queue. Fill in the missing code for 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;
};

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. 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 `llocation`. The function `llocation` should be called `find`, but that would cause a redundancy conflict.

```
struct treenode;
typedef treenode*tree;
struct treenode
{
    int item;
    tree lft; // left subtree
    tree rgt; // right subtree
};

void insert(tree&root,int n)
    // inserts n into the binary search tree. Duplicates are not inserted.
{
    if(root)
```

```

        if(n < root.item) insert(tree->lft)
        else if(n > root.item) insert(tree->rght);
        else;
    else
    {
        root = new treenode;
        root->item = n;
    }
}

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

tree llocation(tree root, int n)
    // returns pointer to the node which contains n, if any; nullptr otherwise.
{
    if(root == nullptr) return nullptr;
    else if(root->item == n) return root;
    else if(n < root->item) return llocation(root->lft,n);
    else return llocation(root->rght,n);
}

int higt(tree root)
    // return the height of the binary search tree. If empty returns -1.
{
    if(root)
        return 1 + max(higt(tree->lft),higt(tree->rght));
    else return -1;
}

int kount(tree root)
    // return the number of item in the binary search tree.
{
    if(root)
        return 1 + kount(root->lft)+kount(root->rght);
    else return 0;
}

```

```

void preordertraverse(tree root)
    // writes the items of tree in preorder.
{
    if(root)
    {
        cout << root->item << endl;
        preordertraverse(root->lft);
        preordertraverse(root->right);
    }
}

void inordertraverse(tree root)
    // writes the items of tree in inorder.
{
    if(root)
    {
        inordertraverse(root->lft);
        cout << root->item << endl;
        inordertraverse(root->right);
    }
}

void postordertraverse(tree root)
    // writes the items of tree in postorder.
{
    if(root)
    {
        postordertraverse(root->lft);
        postordertraverse(root->right);
        cout << root->item << endl;
    }
}

```

6. 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

7. Assume that the array `int A[N]` is part of the input. What does `sorted` do?

It returns true if A is sorted, else it false.

What does `gcd` do?

It returns the greatest common divisor of a and b.

What does `ternary` do?

It writes the ternary (base 3) numeral for n.

What does `power` do?

It return 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)
{
    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;
}

```

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

Assume that $F_n = C^n$. Then

$$C^n = C^{n-1} + C^{n-2}$$

Divide both sides by C^{n-2}

$$C^2 = C + 1$$

Subtract $C + 1$ from both sides

$$C^2 - C - 1 = 0$$

By the quadratic formula:

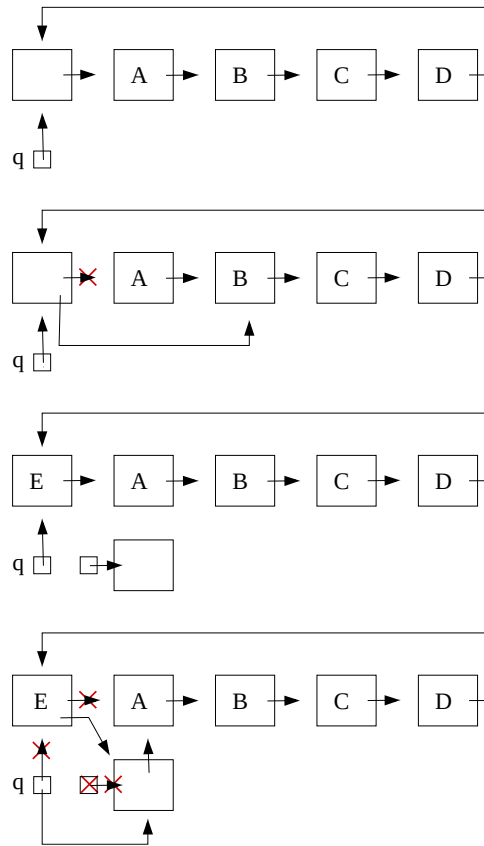
$$C = \frac{1 \pm \sqrt{1+4}}{2}$$

But since C is positive:

$$C = \frac{1 + \sqrt{5}}{2}$$

which is the golden ratio, approximately 1.618

9. 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.



10. The answer to each of these questions is either **bubbleup** or **bubbledown**. If a max-heap is implemented a binary tree implemented as an array, **bubbleup** is needed for insertion, while **bubbledown** is needed for deletemax.
11. For each of the code fragments below, find the asymptotic time complexity in terms of n , using Θ notation.

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

$$\Theta(\sqrt{n})$$

(b) `for(float x = n; x > 2.0; x = sqrt(x))`

$$\Theta(\log \log n)$$

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

$$\Theta(\log n)$$

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

$$\Theta(\log n)$$

- (e) `for(int i = n; i > 1; i--)`
`for(int j = i; j > 1; j = j/2)`
 $\Theta(n \log n)$
- (f) `for(int i = n; i > 1; i--)`
`for(int j = n; j > i; j = j/2)`
 $\Theta(n)$
- (g) `for(int i = 2; i < n; i=i*i)`
 $\Theta(\log \log n)$
- (h) `for(int i = 0; i < n; i++)`
`for(int j = 0; j < i; j++)`
`cout << "Hello world" << end;`
 $\Theta(n^2)$
- (i) `for(int i = 0; i < n; i++)`
`for(int j = 1; j < i; j = 2*j)`
`cout << "Hello world" << end;`
 $\Theta(n \log n)$
- (j) `for(int i = 0; i < n; i++)`
`for(int j = i; j < n; j = 2*j)`
`cout << "Hello world" << end;`
 $\Theta(n)$
- (k) `for(int i = 0; i < n; i++)`
`for(int j = 0; j*j < i; j++)`
`cout << "Hello world" << end;`
 $\sum_{i=0}^{n-1} \sqrt{i}$ is approximated by $\int_0^n \sqrt{x} dx = \Theta(n^{3/2})$

12. Solve each of these recurrences, giving the answer in terms of n using Θ notation.

- (a) $F(n) = 2F(n/2) + n$
 $F(n) = \Theta(n \log n)$
- (b) $F(n) = 2F(n/4) + 1$
 $F(n) = \Theta(\sqrt{n})$
- (c) $F(n) = 2F(n/2) + n^2$
 $F(n) = \Theta(n^2)$
- (d) $F(n) = F(n - \log n) + \log n$
 $F(n) - F(n - \log n) = \log n$
 $\frac{F(n) - F(n - \log n)}{\log n} = 1$
 $F'(n) = \Theta(1)$
 $F(n) = \Theta(n)$

$$\begin{aligned}
\text{(e)} \quad & F(n) = F(n - \sqrt{n}) + n \\
& F(n) - F(n - \sqrt{n}) = n \\
& \frac{F(n) - F(n - \sqrt{n})}{\sqrt{n}} = \sqrt{n} \\
& F'(n) = \Theta(\sqrt{n}) \\
& F(n) = \Theta(n\sqrt{n})
\end{aligned}$$

$$\text{(f)} \quad F(n) = F(\sqrt{n}) + 1$$

Substitute $m = \log n$, hence $m/2 = \log \sqrt{n}$, and let $G(m) = F(n)$. We have:

$$G(m) = G(m/2) + 1$$

$$F(n) = G(m) = \Theta(\log m) = \Theta(\log \log n)$$

13. Simplify each of these expressions. Hint: Answers are integers or simple fractions.

Recall that 2 is the default base of a logarithm. Thus: $2^{\log a} = a$.

$$\text{(a)} \quad \text{Recall that } \log(1/a) = -\log a.$$

$$\log 0.125 = \log(1/8) = \log 2^{-3} = -3$$

$$\text{(b)} \quad \text{Recall that } \log_a b = \frac{\log b}{\log a}.$$

$$\frac{\log_3 5 \log 3}{\log 5} = \frac{\log 5}{\log 3} \frac{\log 3}{\log 5} = 1$$

$$\text{(c)} \quad \log_4 8 = \frac{\log 8}{\log 4} = \frac{3}{2}$$

$$\text{(d)} \quad \text{Recall that } (a^b)^c = a^{bc} = (a^c)^b$$

$$\frac{4^{\log 3}}{9} = \frac{2^{2 \log 3}}{9} = \frac{(2^{\log 3})^2}{9} = \frac{3^2}{9} = 1$$

$$\text{(e)} \quad \log \log 16 = \log 4 = 2$$

$$\text{(f)} \quad \text{Recall that } \log a^b = b \log a.$$

$$\log_4 25 = \frac{\log 25}{\log 4} = \frac{2 \log 5}{2} = \log 5$$

$$\text{Thus } 2^{\log_4 25} = 2^{\log 5} = 5$$

14. The following code computes the product of its two parameters. Write a loop invariant of the loop.

```

int product(float x, int n)
{
    // input condition: n is non-negative
    float y = x;
    float z = 0.0;
    int m = n;
    // Loop invariant holds here.
    while(m > 0) // This was the error!
    {
        // top of an iteration
        if(m%2) z = z+y;
        y = 2*y;
    }
}

```

```

    m = m/2;
    // bottom of the iteration
}
// Loop invariant holds here.
return z;
}

```

The loop invariant is the equation $ym + z = xn$. On the exam, I do not expect you to give the proof of the loop invariant, but here it is. The loop invariant clearly holds initially, since $xn = ym$ and $z = 0$. The inductive property of the loop invariant is that if it holds at the top of any iteration of the loop, it holds at the bottom of that iteration. I do not expect you to write the proof of the inductive property, but here it is.

Let m, y, z be the values of those variables at the top of one iteration, and let m', y', z' be the values at the bottom of that iteration.

If m is even, then $m' = m/2$ and $z' = z$.

If m is odd, then $z' = z + y$, $m' = (m - 1)/2$, and $z' = z + y$.

In either case, $y' = 2y$.

We need to prove that $y'm' + z' = xn$, given that $ym + z = xn$.

Case: m is even. Then $y'm' + z' = (2y)(m/2) + z = ym + z = xn$, and we are done.

Case: m is odd. Then $y'm' + z' = (2y)(m-1)/2 + z + y = y(m-1) + z + y = ym - y + z + y = ym + z = xn$, and we are done.

Correctness of the code follows from the fact that the loop invariant holds. after the loop is done. We have $ym + z = xn$, and $m = 0$ because the loop condition $m > 0$ is false, and thus $z = xn$, which means the function returns the correct value.

15. Walk through the steps of heapsort, sorting an array whose items are X,B,L,F,U,N in that order. For ease of grading, use the matrix below. The number of rows in the matrix might not be exactly what you need. If not, leave rows blank or add extra rows.

X	B	L	F	U	N
X	B	N	F	U	L
X	U	N	F	B	L
L	U	N	F	B	X
U	L	N	F	B	X
B	L	N	F	U	X
N	L	B	F	U	X
F	L	B	N	U	X
L	F	B	N	U	X
B	F	L	N	U	X
F	B	L	N	U	X
B	F	L	N	U	X

16. The following is an unfinished array implementation of stack of integer. Fill in the missing code for empty, push, and pop.

```
struct stak
{
    int item[N];
    int size;
};

void init(stak&S)
{
    S.size = 0;
}

bool empty(stak S)
{
    return S.size == 0;
}

void push(stak&S, int newitem)
{
    S.item[S.size++] = newitem;
}

int pop(stak&S)
{
    assert(not empty(S));
    return S.item[S.--size];
}
```

17. The following is an unfinished linked list implementation of stack of integer. Finish it.

```
struct stacknode;
typedef *stacknode stak;
struct stacknode
{
    int item; Error Fixed Here
    stak link;
};

void init(stak&S)
{
    S = nulptr;
}
```

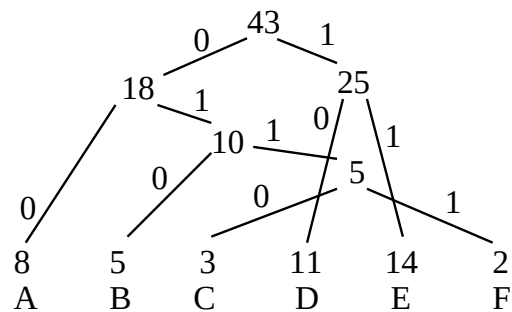
```

bool empty(stak S)
{
    return not(S);
}
void push(stak&S, int newitem)
{
    link top = new stacknode;
    top->link = S;
    top->item = newitem;
    S = top;
}
int pop(stak&S)
{
    tempitem = S->item;
    S = S->link;
    return tempitem;
}

```

18. Find an optimal binary prefix-free code on the alphabet A,B,C,D,E,F where frequencies are given in the following table.

A	8	00
B	5	010
C	3	0110
D	11	10
E	14	11
F	2	0111



19. Write pseudocode for the Bellman Ford algorithm. Be sure to include the shortcut.

```

for all v in V
    V[v] = infinity
V[s] = 0
altered = true
while (altered)
    altered = false
    for all e = (u,v) in E
        if (V[u] + W(e) < V[v])
            V[v] = V[u] + W(e)
            back[v] = u
            altered = true

```

20. Write pseudocode for the Floyd Warshall algorithm.

```
for all 1 <= i <= n
    V[i,i] = 0
    back[i,i] = * \\ undefined
endfor
for all 1 <= i <= n
    for all 1 <= j <= n
        if there is an edge from i to j
            back[i,j] = i
            V[i,j] = W(i,j)
        else
            back[i,j] = * \\ undefined
            V[i,j] = infinity
    for all 1 <= j <= n
        for all 1 <= i <= n
            for all 1 <= k <= n
                temp = V[i,j] + V[j,k]
                if temp < V[i,k]
                    V[i,k] = temp
```

21. Understand **Selection sort**, **Insertion sort**, **Bubble sort**, and **Shell sort**.