1. Simplify each expression.

(a) \( \log_9 3 = \frac{1}{2} \) since \( 9^{\frac{1}{2}} = 3 \)

(b) \( 2^{\log_2 5} = 5 \) because in this course \( \log \) defaults to \( \log_2 \).

2. Fill in the blanks.

(a) Any comparison-based sorting algorithm on a file of size \( n \) must execute \( \Omega(\log n!) = \Omega(n \log n) \) comparisons in the worst case.

(b) Name two divide-and-conquer sorting algorithms.

\textit{quicksort}

\textit{mergesort}

3. The items in a priority queue, such as a stack, represent unfulfilled obligations. In class we discussed a stack algorithm for converting an infix expression to an equivalent postfix expression.

Each operator on the stack represents the unfulfilled obligation to \textit{write that operator}

Each left parenthesis on the stack represents the unfulfilled obligation to \textit{read a right parenthesis}

4. You have an array consisting of thousands of names in alphabetical order. What algorithm would you use to determine whether this array contains the name “Arthur Linkletter”?

\textit{binary search}

5. Find the time complexity of each of these code fragments in terms of \( n \), using \( \Theta \) notation.

(a) 

\begin{verbatim}
for(int = 0; i < n; i++)
    for(int j = i; j > 0; j=j/2);
\end{verbatim}

The inner loop executes \( \Theta(\log i) \) times. Replacing \( i \) by the real variable \( x \) we can approximate the time complexity by the integral

\[
\int_{1}^{n} \ln x \, dx = [x \ln x - x]_1^n = n \ln n - n + 1 = \Theta(n \log n)
\]

(b) 

\begin{verbatim}
for(int i = 0; i < n; i++)
    for(int j = n; j > i; j=j/2);
\end{verbatim}

For each iteration of the outer loop, the inner loop executes approximately \( \log n - \log i \) times. Substituting the real variable \( x \) for \( i \), we approximate the time complexity by the integral

\[
\int_{1}^{n} (\ln n - \ln x) \, dx = [x \ln n - x \ln x + x]_1^n = n \ln n - n \ln n + n - \ln n - 1 = \Theta(n)
\]
(c) for(int i = 1; i < n*n; i = 2*i)
    for(int j = i; j > 0; j=j-1);

Let \( k = \log i \) and \( m = \log n \). Thus \( 2^k = i \) and \( 2^m = n \); hence \( 2^{2m} = n^2 \). Substituting, we have

for(int k = 0; k < 2m; k++)
    for(int j = 2^k; j > 0; j=j-1);

Recall (from calculus) that \( \int 2^x \, dx = \frac{2^x}{\ln 2} + C \)

The inner loop executes \( \Theta(2^k) \) times. Replacing \( k \) by the real variable \( x \), the time complexity is approximated by the integral

\[
\int_0^{2m} 2^x \, dx = \left[ \frac{2^x}{\ln 2} \right]_0^{2m} = \frac{2^{2m} - 1}{\ln 2} = \Theta(2^{2m} = \Theta(n^2))
\]

(d) for(int i = 1; i < n; i++)
    for(int j = 1; j < i*i; j++);

The inner loop executes \( i^2 \) times. Replacing \( i \) by \( x \), we have

\[
\int_1^n x^2 \, dx = \left[ \frac{x^3}{3} \right]_1^n = \frac{n^3 - 1}{3} = \Theta(n^3)
\]

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

Let \( k = \log i \), \( m = \log n \), and \( l = \log j \). We have

for(int k = 0; k < m; k++)
    for(int l = 1; l < k; l=2*l)

The inner loop executes \( \Theta(\log k) \) times. Replacing \( k \) by \( x \) we approximate the time complexity by

\[
\int_1^m \ln x \, dx = \left[ x \ln x - x \right]_1^m = \Theta(m \log m) = \Theta(\log n \log \log n)
\]
6. Find the loop invariant of the following C++ function, which computes \( \lfloor \log n \rfloor \).

```cpp
int flooroflogarithm(int n)
{
    // input condition: n > 0
    int m = n;
    int rslt = 0;
    while(m > 1)
    {
        rslt++;
        m = m/2;
    }
    return rslt;
}
```

The procedure approximates \( \log n \) by repeatedly halving \( n \) until it reaches 1. The variable \( \text{rslt} \) counts the number of times it is halved. At any intermediate point, the value of \( m \) is the original \( n \) halved \( \text{rslt} \) times. Thus the loop invariant is

\[
\lfloor \log n \rfloor = \lfloor \log m \rfloor + \text{rslt}
\]

7. Draw a circular queue with dummy node, holding items X, K, T, F, in that order from front to rear. Draw figures illustrating how the queue changes when you insert L.

![Circular queue diagram]

- Initially, the queue is X → K → T → F with dummy node q.
- After inserting L, the queue becomes L → X → K → T → F.
- If L overflows the queue, it is removed, resulting in a circular queue L → X → K → T → F.
8. A stack of integers is implemented in C++ as a linked list follows.

```cpp
struct stacknode
{
    int item;
    stacknode* link;
};
typedef stacknode* stack;

Write code for the operators push, pop, and empty.

void push(stack&s, int newitem)
{
    stack temp = new stacknode;
    temp->item = newitem;
    temp->link = s;
    s = temp;
}

bool empty(stack s)
{
    return s == NULL;
}

int pop(stack&s)
{
    assert(not empty(s));
    int rslt = s->item;
    s = s->link;
    return rslt;
}
```
9. Walk through the steps of heapsort, sorting an array whose items are R,U,W,F,B,Y in that order. For ease of grading, use the matrix below. (You don't need all the rows.)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>U</td>
<td>W</td>
<td>F</td>
<td>B</td>
<td>Y</td>
</tr>
<tr>
<td>R</td>
<td>U</td>
<td>Y</td>
<td>F</td>
<td>B</td>
<td>W</td>
</tr>
<tr>
<td>Y</td>
<td>U</td>
<td>R</td>
<td>F</td>
<td>B</td>
<td>W</td>
</tr>
<tr>
<td>Y</td>
<td>U</td>
<td>W</td>
<td>F</td>
<td>B</td>
<td>R</td>
</tr>
<tr>
<td>R</td>
<td>U</td>
<td>W</td>
<td>F</td>
<td>B</td>
<td>Y</td>
</tr>
<tr>
<td>R</td>
<td>U</td>
<td>Y</td>
<td>F</td>
<td>B</td>
<td>W</td>
</tr>
<tr>
<td>Y</td>
<td>U</td>
<td>R</td>
<td>F</td>
<td>B</td>
<td>W</td>
</tr>
<tr>
<td>Y</td>
<td>U</td>
<td>W</td>
<td>F</td>
<td>B</td>
<td>R</td>
</tr>
<tr>
<td>R</td>
<td>U</td>
<td>W</td>
<td>F</td>
<td>B</td>
<td>Y</td>
</tr>
<tr>
<td>R</td>
<td>U</td>
<td>Y</td>
<td>F</td>
<td>B</td>
<td>W</td>
</tr>
<tr>
<td>Y</td>
<td>U</td>
<td>R</td>
<td>F</td>
<td>B</td>
<td>W</td>
</tr>
<tr>
<td>Y</td>
<td>U</td>
<td>W</td>
<td>F</td>
<td>B</td>
<td>R</td>
</tr>
<tr>
<td>R</td>
<td>U</td>
<td>W</td>
<td>F</td>
<td>B</td>
<td>Y</td>
</tr>
<tr>
<td>W</td>
<td>U</td>
<td>R</td>
<td>F</td>
<td>B</td>
<td>Y</td>
</tr>
<tr>
<td>B</td>
<td>U</td>
<td>R</td>
<td>F</td>
<td>W</td>
<td>Y</td>
</tr>
<tr>
<td>U</td>
<td>B</td>
<td>R</td>
<td>F</td>
<td>W</td>
<td>Y</td>
</tr>
<tr>
<td>U</td>
<td>F</td>
<td>R</td>
<td>B</td>
<td>W</td>
<td>Y</td>
</tr>
<tr>
<td>B</td>
<td>F</td>
<td>R</td>
<td>B</td>
<td>W</td>
<td>Y</td>
</tr>
<tr>
<td>R</td>
<td>F</td>
<td>B</td>
<td>U</td>
<td>W</td>
<td>Y</td>
</tr>
<tr>
<td>B</td>
<td>F</td>
<td>R</td>
<td>U</td>
<td>W</td>
<td>Y</td>
</tr>
<tr>
<td>F</td>
<td>B</td>
<td>R</td>
<td>U</td>
<td>W</td>
<td>Y</td>
</tr>
<tr>
<td>B</td>
<td>F</td>
<td>R</td>
<td>U</td>
<td>W</td>
<td>Y</td>
</tr>
<tr>
<td>B</td>
<td>F</td>
<td>R</td>
<td>U</td>
<td>W</td>
<td>Y</td>
</tr>
</tbody>
</table>

initial array
bubbledown W
bubbledown R
bubbledown R, heapify completed
swap first and last
bubbledown R, heap order restored
swap first and last
bubbledown B
bubbledown B, heap order restored
swap first and last
bubbledown B, heap order restored
swap first and last
bubbledown B, heap order restored
swap first and last
sorted