1. True or False. [5 points each]

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

   (b) T 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, if 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 \( \Theta(n \log n) \).

   (c) T 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) Name a well-known divide-and-conquer search algorithm.

       binary search

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

       bubblesort
       selection sort
       insertion sort

   (c) Name two well-known \( \Theta(n \log n) \) time divide-and-conquer sorting algorithms.

       quicksort
       mergesort

3. Solve the recurrences. Give asymptotic answers in terms of \( n \). [10 points each.]

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

       \( F(n) = \Theta(n^2 \log n). \)

   (b) \( G(n) = G(n - 1) + \log n \)

       \( G(n) = \Theta(n \log n) \)

   (c) \( H(n) = 2H(\sqrt{n}) + \log n. \)

       \( H(n) = \Theta(\log n \log \log n) \)
(d) \( K(n) = K(n - \sqrt{n}) + 1. \)
\[
K(n) = \Theta(\sqrt{n})
\]

(e) \( F(n) = 4F\left(\frac{3n}{4}\right) + n^5 \) \quad (No, you don’t need a calculator.)
\[
F(n) = \Theta(n^5)
\]

(f) \( T(n) = T(n/2) + T(n/3) + T(n/6) + n \)
\[
T(n) = \Theta(n \log n)
\]

(g) \( T(n) = T(n/2) + T(n/3) + n \)
\[
T(n) = \Theta(n)
\]

(h) \( T(n) = T(3n/5) + T(4n/5) + n^2 \)
\[
T(n) = \Theta(n^2 \log n)
\]

(i) \( T(n) = T(3n/5) + T(4n/5) + n \)
\[
T(n) = \Theta(n^2)
\]

(j) \( T(n) = T(3n/5) + T(4n/5) + n^3 \)
\[
T(n) = \Theta(n^3)
\]

(k) \( F(n) = F(\log n) + 1 \)
\[
F(n) = \Theta(\log \log n)
\]

(l) \( F(n) = F(\sqrt{n}) + 1 \)
\[
F(n) = \Theta(\log \log n)
\]

(m) \( F(n) = 3F(n/3) + 3F(2n/3) + n \)
\[
F(n) = \Theta(n^3)
\]

(n) \( F(n) = 2F(n/4) + \sqrt{n} \)
\[
F(n) = \Theta(\sqrt{n \log n})
\]

(o) This problem is a challenge. I have not been able to solve it. Can you?
\[
F(n) = F(n/2) + F(n - 1) + 1
\]

I know of no closed form solution to this recurrence. However, I do know that \( F(n) \) is not polynomially bounded, meaning that \( F(n) = \Omega(n^k) \) for any constant \( k \). But is \( F \) at least exponential? That is, can you find a constant \( k > 1 \) such that \( F(n) = \Omega(k^n) \)?
4. [15 points] Consider the following procedure:

```c
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 $O$, $\Omega$, or $\Theta$, whichever is most appropriate.

The time complexity $G$ satisfies the recurrence $G(n) = n + G(n/2)$. Thus $G(n) = \Theta(n)$.

5. [30 points] What follows is a portion of my C++ code of quicksort on an array of integers, with some parts deleted. Fill in the missing parts.

```c
void quicksort(int first, int last) // sort A[first] ... A[last]
{
    if(first < last) // otherwise it is already sorted
    {
        int mid = (first+last+1)/2;
        swap(A[first],A[mid]); // Assume the swap procedure has been written
        int lo = first;
        int hi = last;
        int pivot = A[first] // fill this in
        while(lo < hi) // fill this in
        {
            // fill in lines here
                swap(A[lo+1],A[hi]);
            if(A[lo+1] <= pivot)
                lo++;
            if(A[hi] >= pivot)
                hi--;
        } // lo == hi at this point
        swap(A[first],A[hi]); // place the pivot between the two subarrays
        quicksort(first,hi-1); // recursive call
        quicksort(hi+1,last); // another recursive call. Fill in the parameters
    }
}
```
This is not the only correct solution. But your solution must avoid all the traps that I mentioned in class.

6. [30 points] Walk through polyphase mergesort, where the input file is as given below.

   MDYCOSIZVQWXBANLH
   MCOSVXZN DYZQWANH
   DMYBQWXN CEOSVZALH
   CDEMOSVYZHNAVLQWX
   ABCDELMQSVWXYZHN
   ABCDEHLNMOQSVWXYZ

7. [30 points] Walk through mergesort, where the input file is as given below.

   MDYCOSIZVQWBANLH
   MDYCOSIZ VQWBANLH
   MDYCO SEZ VQWBANLH
   MDYCOSEZ VQWBANLH
   MD YC OS EZ VQ WB ANLH
   MD YC OS EZ VQ WB ANLH
   DM CY OS EZ QV BW AN HL
   CMDY E0SZ BQVW AHLN
   CDEMOSY Z ABH LNQVW
   ABCDEHLNMQASVWXYZ
8. [20 points] The following code implements an algorithm we’ve discussed in class, on an array A. What algorithm does the code implement?

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

void main()
{
    for(int i = 0; i < n; i++)
    for(int j = i+1; j < n; j++)
}
```

Selection sort.

9. [20 points] You are working on computer which lacks multiplication and addition. However, it can add or subtract 1 or 2. What does this function do?

```c
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;
}
```

It returns the number which is two times the parameter.
10. [20 points]

What is the purpose of the following code?

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

Returns x^a
```

11. [20 points]

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. The code below implements binary search, to decide whether there is an entry of the array which is equal to \(b\).

```c
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?

\(A[lo] == b\), alternatively, \(A[hi] == b\), since \(lo == hi\).