1. (15 pts)
   (a) Prove the following function is $\Theta(n^4)$.
   \[
   T(n) = 2n^4 - 50n^3 - 100n^2 + 5n + 10.
   \]
   (b) Prove the following tight bound for $\log n!$. (This bound is used to prove a lower bound for sorting. It is also used in analysis of heapsort.)
   \[
   \log(n!) = \sum_{i=1}^{n} \log i = \Theta(n \log n).
   \]
   **Hints:** The upper bound is proved by observing that $\log i \leq \log n$ for all $i$. For the lower bound, you may consider only the larger $n/2$ terms in the summation. That is, $\sum_{i=1}^{n} \log i \geq \sum_{i=\lceil n/2 \rceil}^{n} \log i$.

2. (15 pts)
   (a) Compute and tabulate the following functions for $n = 1, 2, 4, 8, 16, 32, 64$. (The purpose of this exercise is for you to get a feel of how these growth rates compare against each other.)
   Note: All logarithms are assumed to be in base 2, unless stated otherwise. Note that $(\log_2 n)^2$.
   \[
   \log n, \log^2 n, n, n \log n, n^2, n^3, 2^n.
   \]
   (b) Order the following asymptotic complexity functions in increasing order.
   \[
   n^2 \log n, 5, n \log^2 n, 2^n, n^2, n, \sqrt{n}, \log n, \frac{n}{\log n}.
   \]
   The comparison between some of the functions may be obvious (and need not be justified). If you are not sure how a pair of functions compare, you may use the ratio test described below.
   \[
   \lim_{n \to \infty} \frac{f(n)}{g(n)} = \begin{cases} 
   0 & \text{if } f(n) \text{ is asymptotically smaller than } g(n), \\
   \infty & \text{if } f(n) \text{ is asymptotically larger than } g(n), \\
   C & \text{if } f(n) \text{ and } g(n) \text{ have the same growth rate.}
   \end{cases}
   \]
   Note: For any integer constant $k$, $\log^k n$ is a smaller growth rate than $n$. This may be proved using the ratio test.

3. (10 pts) Consider a polynomial of degree $k$, where $k$ is constant. Assume the highest coefficient, $a_k$, is positive. (The remaining coefficients may be positive, negative, or 0.) Prove the polynomial is $\Theta(n^k)$.
   \[
   P_k(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0
   \]
   **Hint:** For the upper bound proof, use the fact that $a_i \leq |a_i|$ so all coefficients become positive. Similarly, for the lower bound proof, use $a_i \geq -|a_i|$ to make all coefficients negative, except the highest term $a_k$ which is positive.

4. (15 pts) Use induction to prove each of the following:
(a) Arithmetic series sum:

\[ S(n) = \sum_{i=1}^{n} (i) = \frac{n(n + 1)}{2} \]

(b) Geometric series sum, \( a \neq 1 \):

\[ S(n) = \sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}. \]

(c)

\[ \sum_{i=1}^{n} \left( \frac{i}{2^i} \right) = 2 - \frac{n + 2}{2^n} \]

(d) For all integers \( n \geq 1 \),

\[ f(n) = 5^n - 1 \]

is divisible by 4.

5. (15 pts) Consider the bubble-sort algorithm below.

```c
void bubblesort (dtype a[], int n)
{int i, j;
 for (i = n - 1; i > 0; i--)
    for (j = 0; j < i; j++) //Bubble the largest of a[0..i] down to position a[i].
        if (a[j] > a[j + 1]) SWAP (a[j], a[j + 1]);
}
```

(a) Analyze the worst-case time complexity, \( T(n) \).

(b) Rewrite the bubble-sort algorithm using recursion.

(c) Let \( f(n) \) be the worst-case number of key-comparisons used by this algorithm to sort \( n \) elements. Write a recurrence for \( f(n) \) and solve it by repeated substitution method.

6. (15 pts) The following algorithm uses a **divide-and-conquer** technique to find the maximum element in an array of size \( n \). The initial call to this recursive function is \( \text{max}(\text{arrayname}, 0, n) \).

```c
dtype FINDMAX(dtype a[], int i, int n)
{ //i is starting index, and n is number of elements.
   //The function uses divide-and-conquer to find max of a[i..i+n-1].
   dtype max1, max2;
   if (n==1) return a[i];
   max1 = FINDMAX(a, i, n/2); //Find max of the first half
   max2 = FINDMAX(a, i+n/2, n-n/2); //Find max of the second half
   if (max1 > max2)
       return max1;
   else return max2;
}
```

Let \( f(n) \) be the worst-case number of **key comparisons** for finding the max of \( n \) elements.

(a) Assuming \( n \) is a power of 2, write a recurrence relation for \( f(n) \).

(b) Prove by induction that the solution is \( f(n) = An + B \) and find the constants \( A \) and \( B \).

(c) Now consider the general case where \( n \) is any integer. Write a recurrence for \( f(n) \). Use induction to prove that the solution is \( f(n) = n - 1 \).
7. (15 pts) Given an array \( A \) of \( n \) elements, we want an efficient algorithm for finding both the minimum and maximum elements in the array. A poor way of doing this is to make a first pass through the array to find the minimum and a second pass to find the maximum. This naive method would take \( 2(n - 1) \) comparisons. We want to design a better algorithm that takes less number of comparisons and is based on a divide-and-conquer method.

(a) Write a recursive function for finding the min and max using divide-and-conquer. The function must work for all integer values of \( n \) and have two parameters for returning the min and max.

(Hint: To make the function efficient, treat both \( n = 1 \) and \( n = 2 \) as base cases.)

\[
\text{MinMax (dtype } A[], \text{ int start, int } n, \text{ dtype& min, dtype& max)}
\]

(b) Let \( f(n) \) be the worst-case number of key-comparisons. Assuming \( n \) is a power of 2, write a recurrence for \( f(n) \). Prove the solution is

\[
f(n) = 3n/2 - 2.
\]

Additional Exercises (Not to be handed in)

8. In analysis of algorithms, we often encounter summations which may be difficult to compute. In this problem, and the next problem, we learn how to find approximate values for certain summations by converting them to integrals.

Let \( f(k) \) be an increasing function of \( k \), and consider the summation

\[
\sum_{k=1}^{n} f(k).
\]

The value of the summation may be viewed as the sum of \( n \) rectangular areas, each with unit width and with height \( f(k), k = 1, 2, \cdots, n \).

- An upper bound for the summation may be derived as follows. Draw an increasing continuous curve to represent \( f(t) \) as a function of \( t \). Label the \( t \) axis at points 1, 2, \cdots, \( n \), \( n + 1 \). Draw the \( n \) unit-width rectangles such that the rectangle with height \( f(k) \) is drawn between points \( k \) and \( k + 1 \). Since \( f(t) \) is an increasing function of \( t \), observe that each rectangle falls entirely under the curve. Therefore, the sum of the areas of the \( n \) rectangles drawn in this way is seen to be smaller than the area under the curve \( f(t) \) between the endpoints 1 and \( n + 1 \). Therefore,

\[
\sum_{k=1}^{n} f(k) < \int_{t=1}^{n+1} f(t)dt.
\]

- A lower bound for the summation is derived in a similar way. Draw each rectangle with height \( f(k) \) to extend to the left of point \( k \), that is, drawn between points \( k - 1 \) and \( k \). Observe that the top of this rectangle extends over the curve \( f(t) \). Thus, the area of this rectangle is greater than the area under the curve \( f(t) \) between points \( k - 1 \) and \( k \). So,

\[
\sum_{k=1}^{n} f(k) > \int_{t=0}^{n} f(t)dt.
\]

- In summary, the summation for an increasing function \( f(k) \) has the following lower-bound and upper bound:

\[
\int_{t=0}^{n} f(t)dt < \sum_{k=1}^{n} f(k) < \int_{t=1}^{n+1} f(t)dt.
\]
(a) Draw diagrams to demonstrate the use of rectangles as explained above to establish the upper-bound and lower-bound.

(b) Use the above method to compute a lower-bound and upper-bound for the following summation.

\[ \sum_{k=1}^{n} (k). \]

How do the bounds compare with the exact sum known from the arithmetic-sum formula?

(c) Derive a lower-bound and upper-bound for the following summation.

\[ \sum_{k=1}^{n} (k^2). \]

9. Let \( f(k) \) be an **decreasing function** of \( k \), and consider the summation

\[ \sum_{k=1}^{n} f(k). \]

As in the last problem, we may establish a lower bound and upper bound for the summation using integrals.

The value of the summation may be viewed as the sum of \( n \) rectangular areas, each with unit width and with height \( f(k) \), \( k = 1, 2, \ldots, n \). These rectangles may be used to establish the following bounds for the summation:

\[ \int_{t=1}^{n+1} f(t) dt < \sum_{k=1}^{n} f(k) < \int_{t=0}^{n} f(t) dt. \]

(a) Draw diagrams to demonstrate the use of the rectangles (as explained in the previous problem) and to explain the above bounds.

(b) Use the above method to compute a lower-bound and upper-bound for the following summation:

\[ \sum_{k=2}^{n} (1/k). \]

(c) Compute a lower-bound and upper-bound for the following summation:

\[ \sum_{k=2}^{n} (1/k^2). \]

10. Consider again the above divide-and-conquer algorithm for finding the max and min of an array of \( n \) elements (Problem 7). Consider the general case where \( n \) is any integer.

(a) Write a recurrence for \( f(n) \). (The recurrence will need two base cases: \( n = 1 \) and \( n = 2 \).) Use induction to prove the solution is

\[ f(n) \leq 5n/3 - 2, \quad n \geq 2. \]

Note the solution is for \( n \geq 2 \). (We avoid \( n = 1 \) in the solution to to achieve the constant 5/3.) Since the solution avoids \( n = 1 \), you will need \( n = 2 \) and \( n = 3 \) for the two bases of induction.
(b) Revise the divide-and-conquer algorithm slightly so that when \( n \) is even, the algorithm never divides it into two subarrays of both odd size. (For example, for \( n = 6 \), the array is not divided into two subarrays of size 3 and 3, but instead 4 and 2.) Show that the worst-case number of key comparisons for this revised algorithm becomes
\[
f(n) = 3n/2 - 2.
\]

11. Use induction to prove each of the following:

(a) For all integer \( n \geq 1 \),
\[
n > \log n.
\]

(b) \[
S(n) = \frac{1}{n + 1} + \frac{1}{n + 2} + \cdots + \frac{1}{2n} \geq \frac{1}{2}, \quad n \geq 1.
\]

12. Consider the recurrence relation: \( S(1) = 1/2 \) and
\[
S(n) = S(n - 1) + \frac{1}{2n - 1} - \frac{1}{2n}, \quad n > 1.
\]
Prove by induction that the solution of the above recurrence satisfies the following relation:
\[
S(n) < 1 - \frac{1}{2n + 1}, \quad n \geq 1
\]

13. Arrays \( A[0..m - 1] \) and \( B[0..n - 1] \) are each sorted in increasing order. Write a procedure to merge them into a sorted array \( C[1..m + n] \). Analyze the running time, \( T(m, n) \).

14. (Towers-of-Hanoi:) This is an interesting puzzle that can be easily solved with recursion. There are three towers: A, B, and C. Initially, \( n \) discs are stacked on tower A, with each disc smaller than the one below it. The object of the game is to move the stack of discs to tower B in the same order. (The third tower is used as intermediate storage.) There are two rules for moving the discs: Only one disc can be moved at a time from the top of one tower to the top of another, and a larger disc can never be placed on top of a smaller one.

(a) Write a recursive function to solve the problem.

(b) To analyze the time complexity, let \( T(n) \) be the total number of moves. Write a recurrence for \( T(n) \) and solve the recurrence. Then, to get a better appreciation for this time complexity, tabulate \( T(n) \) for the values of \( n \) from 1 to 20.

(c) Implement a program for this problem. (Input the number of discs from the keyboard, and printout each move.) Limit \( n \) to at most 6 to avoid excessive output. Hand in a hard copy of the program and the output for two values of \( n \) (e.g., 3 and 5). Below is a sample output:

```
---------------------------------------------------------------
Towers of Hanoi Solver. Please input number of disks (max 6): 2
Initial State: A: 2 1 B: C: 1
1 (Move disc 1 from A to C): A: 2 B: C: 1
2 (Move disc 2 from A to B): A: B: 2 C: 1
3 (Move disc 1 from C to B): A: B: 2 1 C:
```