Question 1

For each statement below decide whether it is true or false and check the correct answer. No justification is required.

(a) If $f = o(g)$, then $f = O(g)$. ..... ☐ True ☐ False

(b) If $f = \Omega(g)$, then $g = o(f)$. ..... ☐ True ☐ False

(c) $2^{n/10}$ grows asymptotically faster than $8^{2\log_4 n}$. ..... ☐ True ☐ False

(d) $3^{\log_2 n}$ has polynomial asymptotic growth rate. ..... ☐ True ☐ False

(e) $\log(n^n)$ has polylogarithmic asymptotic growth rate. ..... ☐ True ☐ False

(f) A loop variant can be used to prove termination of a program. ... ☐ True ☐ False

(g) A loop variant can be used to obtain an upper bound on the running time of a program. ..... ☐ True ☐ False

(h) An algorithm with worst-case running time $f(n)$ is faster than an algorithm with worst-case running time $g(n)$ on all inputs if $f(n) \in o(g(n))$. ..... ☐ True ☐ False

Question 2

Fill in the sentences below. Each time complexity should be stated as a function of n.

(a) Any algorithm for finding the position of a given element in a sorted array of size n has worst-case time complexity $\Omega($_____$)$.

(b) The *worst-case* time complexity of determining whether a given element exists in the array is $\Theta($_____$)$.

(c) The *best-case* time complexity of determining whether a given element in the array exists is $\Theta($_____$)$.

(d) A program is totally correct,

if it is _____ correct, and if it _____.

Question 3

Answer the questions below by checking the box with the correct answer.
Check only one box for each question.

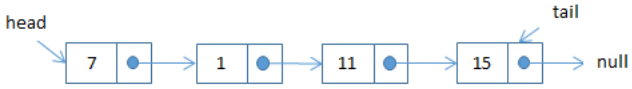
(a) Binary Search, which runs in $\Theta(\log n)$, is more efficient than Linear Search, which runs in $O(n)$. Why do we sometimes prefer to use linear search?

- ☐ If the list is not in sorted order we cannot use binary search.
- ☐ The asymptotic runtime provided is only for a list implemented as an array. A linear Search is more effective when the list is implemented as a linked list.
- ☐ The code for Linear Search is simpler and if the input size is small we may prefer to use linear search.
- ☐ All of the above are reasons to use Linear Search instead of Binary Search.

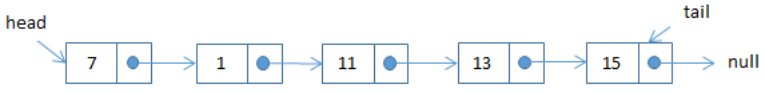
(b) Two main measures for the efficiency of an algorithm are

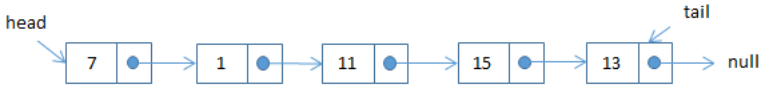
- ☐ Processor and memory;
- ☐ Complexity and capacity;
- ☐ Data and space;
- ☐ Time and space.

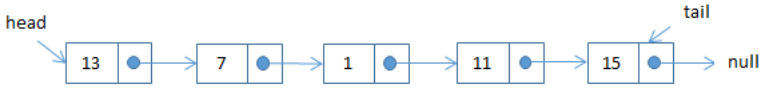
(c) Consider the following singly linked list L:

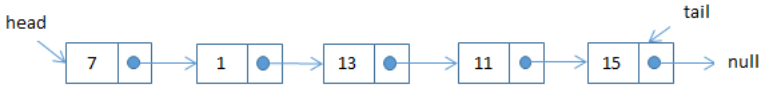


What will L look like after the call $L.\texttt{insert}(13, 3)$ (which inserts 13 at index 3)?

☐ 

☐ 

☐ 

☐ 

(d) For positive constants c_1 and c_2, let $f(n) = c_1 + \sum_{i=1}^{k} c_2$ and let $k = \log(n)$. Which of the following expressions is true?

- ☐ $f(n) \in \Theta(\log(n))$
- ☐ $f(n) \in \Theta(\sqrt{n})$
- ☐ $f(n) \in \Theta(n)$
- ☐ $f(n) \in \Theta(n \log(n))$

(e) The following statement is true for a doubly linked list:

- ☐ It is asymptotically more efficient to remove the head element than the tail element.
- ☐ It is asymptotically more efficient to insert an element at the head than at the tail.
- ☐ If the list is empty, head and tail are both null.
- ☐ Each node has exactly one predecessor and one successor.

Question 4

Consider the program below:

```
Compare(A,B,n) {
/* Precondition:  A and B are integer arrays of length n
   Postcondition: The program returns the sum of all
   values A[i], 0<=i<n, for which A[i]=B[i]. */
    int i = 0;
    int S = 0;
    while(i<n) {
      if A[i] = B[i] {
        S = S + B[i];
      }
      i++;
    }
    return S;
  }
```

(a) Give a loop invariant and argue that it is one. Use your loop invariant to argue that the program is partially correct.

(b) Prove that the program terminates. Use a loop variant for your proof.

Question 5

Consider the following program:

```
Compute(m,d) {
  /* Precondition: m and d are integers and d>1 */
  int k=m;
  while(k>1) {
    if k is a multiple of d {
      k = k/d;
    }
    else {
      return false;
    }
  }
  return k;
}
```

(a) Is it possible that the while-loop does not terminate if the precondition is not satisfied? Why?

(b) Argue that $f(k) = k/d$ is a loop variant for the while-loop, given that the precondition is satisfied.

(c) Now consider the following code:

```
int d=sqrt(n);
Compute(n,d);
```

Give a function $T_{wc}(n)$ such that the worst-case running time of this code is $\Theta(T_{wc}(n))$. Argue why your choice is correct.

(d) Give a function $T_{bc}(n)$ such that the best-case running time of this code is $\Theta(T_{bc}(n))$. Argue why your choice is correct.

Question 6

Consider the following algorithm that copies the entries of an array to a second array in reverse order.

```
/* Precondition: A is a non-null array of integers */
/* Postcondition: B contains the entries of A in reverse order */
  n = A.length
  initialize B to be an integer arragy of length n
  for{i from 0 to n-1}{
    B[i] = A[n-1-i]
  }
  return B
```

(a) State **two** problems with the statement

$$B[j] = A[n-1-j] \text{ for } 0 \leq j \leq i;\ 0 \leq i < n-1$$

that show it is *not* a correct loop invariant for the example. Justify your answer.

(b) Give a correct loop invariant for the example (a proof of correctness is **not** required).

(c) Explain how your loop invariant can be used to prove that the entire algorithm is correct.

Question 7

Consider the following recursive implementation of sum.

```
public int sum(int[] nums, int firstIndex, int length) {
  if (length == 0) return 0;
  if (length == 1) return nums[firstIndex];

  int sublistLength = length/3;
  int sum1 = sum(nums, firstIndex, sublistLength);
  int sum2 = sum(nums, firstIndex + sublistLength, sublistLength);
  int sum3 = sum(nums, firstIndex + 2 * sublistLength, sublistLength);

  return sum1 + sum2 + sum3;
}
```

(a) Give a recurrence that describes the runtime of sum. Make sure your recurrence is defined for values where $n > 1$ and for values $n \leq 1$.

(b) Use your recurrence to find a tight asymptotic bound for sum. Show your work or explain your answer.

Question 8

Let

$$f_1(n) = \frac{2^n}{n^2} \qquad f_2(n) = \frac{n!}{(n/2)!} \qquad f_3(n) = (\log n)^2 \qquad f_4(n) = n^2$$

$$f_5(n) = 1.5^n \qquad f_6(n) = 2^{2\log n} \qquad f_7(n) = (\log_5 n) \cdot (\log_3 n) \qquad f_8(n) = 3^{2\log\log n}.$$

(a) Order the functions by asymptotic growth, i.e., if $f_i = o(f_j)$, then f_i should appear before f_j. (If $f_i = \Theta(f_j)$, then the relative order of f_i and f_j does not matter.)

(b) Indicate all pairs (i, j), $1 \leq i < j \leq 8$, for which $f_i = \Theta(f_j)$.

Question 9

Let

$$f(n) = \frac{2^{2n}}{n^2} \qquad \text{and} \qquad g(n) = \frac{4^n}{n^4} \cdot \log_4 n.$$

Decide whether $f = o(g)$, $f = \Theta(g)$, or $f = \omega(g)$, and prove your statement.

You are **not allowed** to use any statements about the relative asymptotic growth of two functions. (E.g., you cannot simply state that $\log n = o(n)$ without proving it.) You **are allowed** to use the definitions and standard results from page 10.

Definitions. For $f, g : \mathbb{N} \to \mathbb{R}_{>0}$:

$$f = O(g) \Leftrightarrow \exists c > 0\ \forall n \in \mathbb{N} : f(n) \leq cg(n), \qquad f = o(g) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0,$$

$$f = \Omega(g) \Leftrightarrow \exists c > 0\ \forall n \in \mathbb{N} : f(n) \geq cg(n),$$

$$f = \Theta(g) \Leftrightarrow f = O(g) \wedge f = \Omega(g), \qquad f = \omega(g) \Leftrightarrow g = o(f).$$

Limit Test.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & \Rightarrow\ f = o(g) \\ \infty & \Rightarrow\ f = \omega(g) \\ c > 0 & \Rightarrow\ f = \Theta(g). \end{cases}$$

L'Hôpital's Rule.

$$\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = c \in \{0, -\infty, \infty\} \quad \Rightarrow \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}.$$

Derivatives.

$$\frac{d}{dn} n^a = a \cdot n^{a-1}, \qquad \frac{d}{dn} f(n) \cdot g(n) = f'(n)g(n) + f(n)g'(n),$$

$$\frac{d}{dn} e^n = e^n, \qquad \frac{d}{dn} f(g(n)) = f'(g(n))g'(n),$$

$$\frac{d}{dn} \ln n = \frac{1}{n}.$$

Logarithms.

$$\log n = \log_2 n, \quad \ln n = \log_e n, \quad \log_b(a^n) = n \log_b(a), \quad \log_b(a \cdot c) = \log_b a + \log_b c,$$

$$\log_b 1 = 0 \qquad a^{\log_a n} = n, \qquad a^{\log_b c} = c^{\log_b a}, \qquad \log_a n = \frac{\log_b n}{\log_b a}.$$

Exponentials.

$$b^0 = 1, \qquad b^{-x} = \frac{1}{b^x}, \qquad b^m b^n = b^{m+n}, \qquad (b^m)^n = b^{mn}, \qquad a^n b^n = (ab)^n.$$

Common Serieses. For any $k \in \mathbb{N}$ and any $q \in \mathbb{R} - \{1\}$:

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2} \quad \text{and} \quad \sum_{i=0}^{k} q^i = \frac{q^{k+1} - 1}{q - 1}$$