Discrete Mathematics, Homework 4: Induction and Recursion

1. (1 point) Given a recurrence of the form T(n) = aT(n/b) + cn, what does ‘a’ signify?
a. The fraction of the problem at level n that is passed on to level n + 1.
*b. The number of subproblems into which we divide a problem as we go down the recursion tree.
c. The cost of each leaf node in the recursion tree.
d. The total number of children at every level in the recursion tree.

2. (1 point) Given a recurrence of the form T(n) = aT(n/b) + cn, what does ‘b' signify?
*a. The fraction of the problem at level n that is passed on to level n + 1.
b. The number of subproblems into which we divide a problem as we go down the recursion tree.
c. The cost of each leaf node in the recursion tree.
d. The total number of children at every level in the recursion tree.

3. (1 point) The base case in any recurrence formula ______.
a. Is used to define the memory requirements of the program.
b. Can be safely omitted.
*c. Is the only place where an explicit value is returned, instead of the result of another call.
d. All of the above.

4. (1 point) For the recurrence f(0) = 0, f(1) = 3, f(2) = 5, f(3) = 7, and f(n > 3) = f(n - 2) + f(n -4), then f(7) = ?
a. 12
b. 26
c. 5
*d. -3

5. (1 point) What does this code return for fun1(4, 7)?

```
int fun1(int x, int y)
{
  if(x == 0)
      return y;
  else
      return fun1(x - 1,  x + y);
}
```

*a. 17

b. 32
c. 48
d. 33

6. (1 point) What does this code return for fun1(0)?

```
def fun1(n):
   if(n == 1):
      return 0
   else:
      return 1 + fun1(n/2)
```

a. Not defined
*b. This will recurse forever without returning anything (unless it blows the stack).
c. 0
d. infinity

7. (1 point) Which of the following statements are true for binarysearch1 and binarysearch2?

```
def binarySearch1 (arr, l, r, x):
   # Check base case
   if r >= l:

      mid = l + (r - l)/2

      # If element is present at the middle itself
      if arr[mid] == x:
         return mid

      # If element is smaller than mid, then it
      # can only be present in left subarray
      elif arr[mid] > x:
         return binarySearch(arr, l, mid-1, x)

      # Else the element can only be present
      # in right subarray
      else:
         return binarySearch(arr, mid+1, r, x)

   else:
```

```
        # Element is not present in the array
        return -1

def binarySearch2(arr, l, r, x):

    while l <= r:
        mid = l + (r - l)/2;

        # Check if x is present at mid
        if arr[mid] == x:
            return mid

        # If x is greater, ignore left half
        elif arr[mid] < x:
            l = mid + 1

        # If x is smaller, ignore right half
        else:
            r = mid - 1

    # If we reach here, then the element was not present
    return -1
```

(i) binarysearch1 and binarysearch2 return the output for same set of input.
(ii) binarysearch1 is recursive version of the algorithm and binarysearch2 is iterative version.
(iii) binarysearch2 is recursive version of the algorithm and binarysearch1 is iterative version.
(iv) The recursive version is faster than the iterative version.
(v) Both return correct output only when the array arr is sorted.
*a. (i), (ii), (v)
b. (i), (ii), (iv)
c. (i), (iii), (v), (iv)
d. (i), (iii), (iv)

8. (1 point) Which of the following statements is true for mathematical induction?
a. It can be used to find new theorems.
*b. It can be used to prove a conjecture once it is has been made.

c. Mathematical induction gives insight into the proof and states the reason why it is true.
d. Mathematical induction is prefered over all other proof methods.

9. (1 point) Which of the following cases would tend towards using recursion over iteration?
a. You have lots of disk space
*b. You have plenty of memory
c. You need maximum speed
d. Recursion is always preferred over induction

10. (1 point) Write a recursive definition for f, given that:
f(0) = 3
f(1) = 9
f(2) = 21
f(3) = 45
*a. *f*(n) = f(0) = 3; 2*f*(n) + 3 for n > 0
b. *f*(n) = f(0) = 3; 2*f*(n) + 3 for n >= 0
c. *f*(n) = f(0) = 3; *f*(n) + 6 for n > 0
d. *f*(n) = f(0) = 3; *f*(n) + 6 for n > 1

11. (1 point) Which of the following is a correct, recursive algorithm for n! ?
a. int factorial(int n)
```
    if (n != 0):
        return (n * factorial(n - 1));
```
*b. int factorial(int n)
```
    if (n == 0):
        return 1;
    else:
        return (n * factorial(n - 1));
```
c. int factorial(int n)
```
    if (n == 0):
        return 1;
    else:
        return  factorial(n - 1);
```
d. int factorial(int n)
```
    int fact = 1;
    for (int j = 0; j <= n; j++)
```

```
        fact = fact * j;
    return fact;
```

12. (1 point) How do you prevent a recursive algorithm from becoming an infinite recursion?
a. Define the function arguments carefully
b. Make sure all your variables are initialized
*c. Include a base case that must be reached eventually
d. None of the above

13. (1 point) Which of the following problems can be solved using recursion?
a. calculate the length of a string
b. calculate the factorial of a number
c. calculate the nth fibonacci number
*d. all of the above

14. (1 point) What will happen if the following code is executed?

```
void my_func()
{
  my_func();
}
int main()
{
  my_func();
  return 0;
}
```

a. The program will stop with a syntax error
b. Output will show a sequence of numbers and then the program will exit gracefully.
*c. Execution will stop with a stack overflow, because of the infinite recursion.
d. None of the above.