Question 1

Original Score: 0 / 2 pts **Regraded Score: 0 / 2 pts**

This question has been regraded.

My friend said she discovered a new efficient algorithm AVERR to calculate the average value of a list of n distinct integers. Which of the following statements about the AVERR **must** be false.

The worst case running time is O(n^2)

Correct Answer

The best case running time is Theta(lgn)

The worst case running time is Omega(logn)

The average case running time is O(n)

None of the above

Question 2

4 / 4 pts

Give a tight bound on the number of times the z = z + 1 statement is executed.

```
i = n
while ( i > 1 ) {
   i = floor(i/2)
   z = z + 1
}
```

Theta(n)

Theta(nlgn)

Theta(sqrt(n))

Theta(lgn)

Theta(1)

Question 3

0 / 4 pts

Is the following a property that holds for all non decreasing positive functions f and g? (True=Yes/ False=No)

If f(n) = O(g(n)) and g(n) = O(h(n)), then f(n)+g(n) = O(g(n))

Correct Answer

True

False

Question 4

0 / 4 pts

Is the following a property that holds for all non decreasing positive functions f and g? (True=Yes/ False=No)

If $f(n) = O(n^2)$ and $g(n) = O(n^2)$, then $f(n) = O(g(n))$.

True

Correct Answer

False

Question 5

4 / 4 pts

Rank the following functions by increasing order of growth:

$$\log(n!),\quad 10000n^2,\quad \log(n^3),\quad 2^n,\quad (0.5)^n,\quad n\log(n)$$

nlog(n), log(n!), 10000n2, log(n3), (0.5)n, 2n

(0.5)n, nlog(n), log(n!), 10000n2, log(n3), 2n

(0.5)n, log(n3), log(n!), nlog(n), 10000n2, 2n

nlog(n), log(n!),log(n3), 10000n2, (0.5)n, 2n

Question 6

10 / 10 pts

```
def goo(n):
  if n == 0:
    return 1
  else:
    x = goo(n-1)
```

y = goo(n-1)

z = x + y

return z

(a) Write a recurrence for the running time T(n) of goo(n)

(b) Solve the recurrence for the asymptotic running time. Assume that addition can be done in constant time. Give the tightest bound possible.

Your Answer:
a) T(0) = 1 where n = 0;

T(n) = 2T(n-1) + c where n >= 1

b) By the Muster Method

a = 2

b = 1

f(n) = c

f(n) = n^d -> d = 0

T(n) = O(n^da^(n/b))

T(n) = O(n^02^(n/1))

Therefore:

T(n) = O(2^n)

Question 7

14 / 14 pts

Consider the 0-1 Knapsack problem where all items have the same benefit of b. In other words, we have n items where w_i is the weight of the i^{th} item, every item has a benefit of b, and the capacity of the knapsack is W. We want to find a subset of items whose total weight is at most W and whose total benefit is as large as possible. (This is knapsack without repetition; each item can be chosen at most once).

(a) What algorithm design paradigm is most appropriate for this problem? Divide and Conquer, Brute Force or Greedy, Dynamic Programming. Why?

(b) Verbally describe an **efficient** algorithm for this problem.

(c) What is the asymptotic running time of your algorithm?

Your Answer:
A)

Greedy - I chose Greedy because this looks a lot like the job scheduling problem. Since the benefit is the same for all the items, one wants to get as many items in the knapsack as possible. To do this, one needs to be greedy to get the ones with the least amount of weight first and then work their way up to the heavier ones.

b)

My efficient algorithm would first Merge Sort the input array in ascending order based on the weight. This means that the items that weight the least would be towards the front of the array(index 0). This would take O(nlgn). The algorithm would then step through the array, placing an item in the knapsack so long as the knapsack has room to accept items. This would take O(n).

c) The asymptotic running time would be O(nlgn).

Question 8

11 / 14 pts

The terms in the Fibonacci sequence are given by :

$F_1 = 1,\ F_2 = 1;\quad F_n = F_{n-1} + F_{n-2}$

(a) Give the pseudocode for a recursive algorithm to calculate the nth term in the Fibonacci sequence.

(b) Describe in words and give the pseudocode for a dynamic programming algorithm to compute the nth term in the Fibonacci sequence.

(c) What is the running time of the DP algorithm.

(d) How does this compare to the running time of the Recursive algorithm?

Your Answer:
a)

```
fib() {
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
```

```
    }
}
```

b)

The goal of the DP is to use the elements that were calculated before. To do this, one could memoize the data. What is means is that, instead of the recursive function having to recalculate the needed value, it would first look back in the already calculated array one and two spots and see if it has it. If it doesnt, then it would calculate what it needs. Below is the code for it.

```
memo = {}
fib (n) {
    if (n in memo) { return memo[0] }
    if (n <= 1) {
        f = n;
    } else {
        f = fib(n-1) + fib(n-2);
    }
    memo[n] = f;
    return f;
}
```

d)

I believe the running time of the Recursive Fibonacci algorithm is O(n!). The running time of the DP version is psuedo polynomial. This means it would be O(nm). In comparison, the DP would run a lot faster than the Recursive.

d,The recursive algorithm is exponential O(2^n) or Theta(phi^n). The DP is much faster since it is linear time Theta(n).

Question 9

17 / 18 pts

OSU student, Benny, is taking his CS 340 algorithms exam which consists on n questions. He notices that the professor has assigned points { p_1, p_2, ..., p_n } to each problem according to the professor's opinion of the difficulty of the problem. Benny wants to maximize the total number of points he earns on the exam, but he is worried about running out of time since there is only T minutes for the exam. He estimates that the amount of time it will take him to solve each of the n questions is { t_1, t_2, ..., t_n }. You can assume that Benny gets full credit for every question he answers completely. Develop an algorithm to help Benny select which questions to answer. **Note: NO** partial credit is assigned to problems that are only partially completed.

(a) What type of algorithm would you use to solve this problem? Divide and Conquer, Greedy or Dynamic Programming. Why?

(b) Describe the algorithm verbally and give pseudo code with enough detail to obtain the running time. If you select a DP algorithm give the formula used to fill the table or array.

(c) What is the running time of your algorithm? Explain.

(d) Now suppose that the professor gave partial credit for problems that are partially completed, would you use the algorithm you described above in this scenario?

(e) What type of algorithm would you use to efficiently solve the partial credit problem? Verbally describe the algorithm.

Your Answer:
a)

Dynamic Programming

This problem is a lot like the "consulting business" problem from the practice test and also the 0-1 Knapsack problem.

b)

Let OPT(i,d) be the maximum point earned for considering problem 1,...., i with t minutes available.

The base case cases are OPT(i,0) = 0 for i = 1,..., n and OPT(0,t) = 0 for t = 1,..., T.

```
for i = 1 to n {
    for t = 1 to T {
        if (ti > t) {
            OPT(i,t) = OPT(i - 1);
        } else {
             OPT(i,t) = max (OPT(i-1,t),OPT(i-1,j-ti) + pi) // pi is the points for the problem
        }
    }
```

}

c)

The running time of my algorithm would be O(nT). This is so because we need to check each problem and see if its point value is more beneficial than the previous. Also, the running time of a dynamic program is the number of subproblems times the time per subproblem.

d)

I would not use the above algorithm.

e)

I would use the Fractional Greedy algorithm. From our lecture, it was shown and proven that when items can be broken up, the Fractional Greedy algorithm is better to use.

for each problem i in P

    xi = 0

    vi = pi/ti (vi is value, pi is point, ti is time)

t = 0 (total current time)

while t < T

    remove item i with the highest vi

     xi = min(ti,T-t)

     w = w + min(wi,T-t)

c) \theta(nT)

Question 10

9.5 / 10 pts

(a) Give the pseudo-code for a recursive algorithm called Find_Smallest(A, n-1) that returns the value of the smallest element in an array of n integers called A. Assume the elements in the array are at locations A[0]..A[n-1].

(b) Give a recurrence T(n) for the running time your algorithm.

(c) Solve the recurrence in part (b)

Your Answer:
a)

base case

Find_Smallest(A,0) = A[0]; where n = 1;

let b = n-1

int min = inf

```
Find_Smallest(A,b)
    if b = 1 return A[0]
    for i from 0 to b
        if min > A[i]
            min  = A[i]
        else
            Find_Smallest(A[i:b],b-i);
      return min;
```

b)

T(n) = T(n-1) + c

c)

Using Muster Method

a = 1

b = 1

f(n) = c

d = 0

T(n) = O(n^d+1)

Therefore,

T(n) = O(n)

c) theta

Question 11

8 / 8 pts

Your friend suggests the following variation of Mergesort: instead of splitting the list into two halves, we split it into three thirds. Then we recursively sort each third and merge them.

Mergesort3 (A[1...n]):

If n <= 1, then return A[1..n]

Let k = n/3 and m = 2n/3

Mergesort3(A[1..k])

Mergesort3(A[k+1..m])

Mergesort3(A[m+1..n)

Merge3(A[1..k], A[k+1,..m], A[m+1..n])

return A[1..m].

Merge3(L0, L1, L2):

return Merge(L0, Merge(L1,L2)).

Assume that you have a function Merge that merges two sorted lists of lengths q and p in time k(q+p) where k is a constant. You may assume that n is a power of 3, if you wish.

(a) What is the asymptotic running time for executing Merge3(L0, L1, L2), if L0, L1, L2 are three sorted lists of length n/3? Give the tightest bound possible. Show your work!

(b) Let T(n) denote the running time of Mergesort3 on an array of size n. Write a recurrence relation for T(n) and solve it using the tightest bounds possible.

(c) Compare the running time of Mergesort3 to the running time of ordinary Mergesort.

Your Answer:
a)

The Merge3 function would be theta(n). This is the same for the normal Merger, it is just done more more time and we know that constants done really matter when it comes to running times.

b)

Using Master Method

a = 3

b = 3

f(n) = cn

nlogba -> nlog33 -> n

This would be case 2

Therefore,

T(n) = theta(nlgn)

c)

They are the exact same.

c) Both are Theta(nlgn)

Question 12

8 / 8 pts

Recall the activity-selection problem from HW 4. You proved that the last-to-finish greedy choice selection criteria yielded an optimal solution. Now consider the following two greedy choice criteria for the activity selection problem:

1. Always select the activity that has earliest starting time.
2. Always select the activity that takes the least amount of time.

Are the above two greedy choice selection methods optimal? If “yes”, give an informal proof. If "no" give counter examples for each of them showing that they are not optimal.

Your Answer:
QUESTION: Do you mean "last to start"? The original way was "first to finish" and then on our hw we did "last to start".

1)

NO. A counter example would be:

A task could start at time 0 but then span the whole allotted time so that other tasks could not be selected.

2)

NO. A counter example would be:

A task could take the least amount of time but then conflict with two other cases that could have been chosen in it place.

Quiz Score: **85.5** out of 100