**! This quiz has been regraded; your score was affected.**

# Midterm

**Due** May 3 at 11:59pm **Points** 100 **Questions** 16
**Available** Apr 30 at 11:59pm - May 3 at 11:59pm 3 days **Time Limit** 75 Minutes

## Instructions

The time limit is 75 minutes and you have one attempt only.

You are allowed to use the following materials:

- Scratch Paper
- One sheet, 8.5x11", double-sided notes (typed or handwritten)
- Students may use any calculator
- Windows Calculator (and MacOS equivalent) "Programmer Mode" is OK!

This quiz was locked May 3 at 11:59pm.

## Attempt History

| | Attempt | Time | Score | Regraded |
|---|---|---|---|---|
| LATEST | Attempt 1 | 75 minutes | 75 out of 100 | 78 out of 100 |

Score for this quiz: **78** out of 100
Submitted May 3 at 3:55pm
This attempt took 75 minutes.

### Question 1 4 / 4 pts

Let $f(n) = n^3$

Let $g(n) = n^2 \log(n^3)$

What is the asymptotic relation between f(n) and g(n)? **Check all that apply.**

☐ f(n) = O(g(n))

Correct! [✓ f(n) = Omega(g(n))]

☐ f(n) = Theta(g(n))

☐ g(n) = Theta(f(n))

Correct! [✓ g(n) = O(f(n))]

☐ g(n) = Omega(f(n))

## 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))

Correct! [✓ Theta( lgn)]

○ Theta(1)

## Question 3

3 / 3 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 $g(n) = \text{Theta}(n^2)$, then $f(n) = O(g(n))$.

Correct!

[✓ True]

False

## Question 4

3 / 3 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!

[✓ False]

Counter example $f(n) = n^2$ and $g(n) = n$

## Question 5

3 / 3 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!

[✓ True]

False

## Question 6

3 / 3 pts

Rank the following functions by increasing order of growth:

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

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

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

Correct!

[✓ $(0.5)^n$, $\log(n^3)$, log( n! ), nlog(n), $10000n^2$, $2^n$]

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

## Question 7

3 / 3 pts

What is the solution of T(n) = T(3n/4) + 1 using the Master theorem?

Theta(log n), Case 1

Theta(n log n), Case 3

Theta(n), Case 1

Correct!

[✓ Theta(log n), Case 2]

## Question 8

Original Score: 0 / 3 pts Regraded Score: 3 / 3 pts

! This question has been regraded.

Which of the following is an example where the greedy method does not achieve the optimal solution to the Coin Change problem? D is the set of denominations and A is the amount to make a change.

- D = { 1, 5, 10, 25} and A = 30
- Correct! [✓ D = { 1, 7, 11} and A = 14]
- D = { 1, 7, 11} and A = 22
- D = { 3, 8, 13, 25} and A = 53
- D = {1, 5, 10, 25, 50 } and A = 52

## Question 9

3 / 3 pts

In dynamic programming, the technique of storing the previously calculated values is called ___________

- Saving value property
- Storing value property
- Correct! [✓ Memoization]
- Mapping

## Question 10

3 / 3 pts

Which of the following is/are property/properties of a dynamic programming problem?

- ○ Optimal Substructure
- ○ Overlapping Subproblems
- ○ Greedy approach
- Correct! [✓ Both optimal substructure and overlapping subproblems]

## Question 11

0 / 2 pts

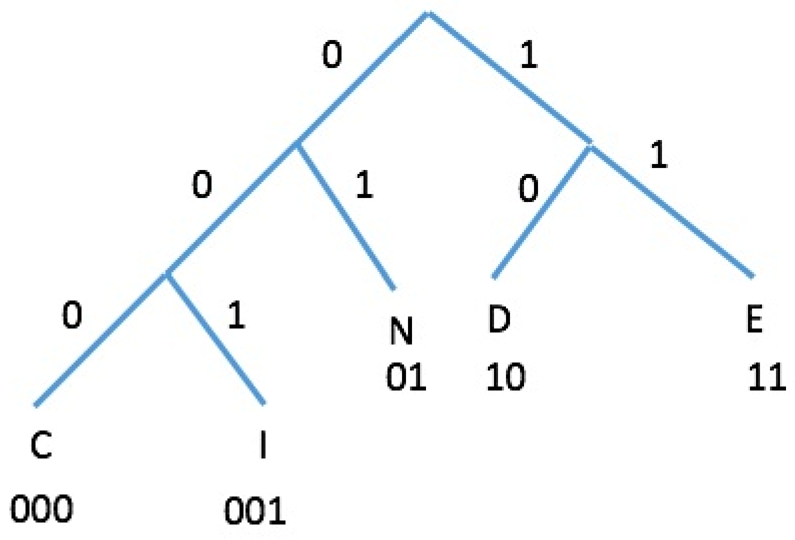
All dynamic programming problems can be solved by using a greedy choice algorithm.

- ou Answered [✓ True]
- orrect Answer ○ False

## Question 12

6 / 6 pts

What would be the result of decoding 0100100011 using the following coding tree?



ICEN

Correct!

[✓ NICE]

NCIE

NEIC

## Question 13

15 / 15 pts

Suppose you are choosing between the following three algorithms:

Algorithm A: solves problems by dividing them into five subproblems of half the size, solve each subproblem recursively and then combines the solutions in linear time.

Algorithm B: solves problems of size n by recursively solving two subproblems of the size n-1, and then combines the solutions in constant time.

Algorithm C: solves problems of size n by dividing them into nine subproblems of size n/3 each, recursively solving each subproblem, and then combines the solutions in $O\left(n^2\right)$ time.

What are the running times of each of the three Algorithms (in big-O notation)?

Your Answer:

**Algorithm A:** T(n) = 5*T(n/2) + n

Using Master's Theorem of Dividing Functions, A = 5, B = 2, k = 1, p = 0

Hence, $\boxed{Log_2^5}$ = 2.3219 > k, which is Case 1: $\boxed{\theta\left(n^{2.3219}\right)}$

**Algorithm B:** T(n) = 2*T(n-1)+ 1

Using Master's Theorem of Decreasing Functions, A = 2 which is > 1

Hence, which is Case 3: $\boxed{\theta\left(2^n\right)}$

**Algorithm C:** T(n) = 9*T(n/3) + n^2

Using Master's Theorem of Dividing Functions, A = 9, B = 3, k = 2, p = 0

Hence, $\boxed{Log_3^9}$ = 2 and that is equal to k, which is Case 2: $\boxed{\theta\left(n^2 Logn\right)}$

## Question 14

**0 / 15 pts**

Prove / Disprove the following 2 properties using Huffman Coding:

a) If some characters occur with a frequency of more than 2/5, then there is a codeword that is guaranteed to be a length of 1.

b) If all characters occur with a frequency of less than 1/3, then there are no codewords guaranteed to be a length of 1.

Your Answer:

A)

B)

no answer

## Question 15

15 / 15 pts

The terms in the Fibonacci sequence are given by :

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

a) Give the pseudo-code for a recursive algorithm to calculate the nth term in the Fibonacci sequence.

b) Give the pseudo-code 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)**

```
int fib (int n)
{
   if (n <= 1) //base case when n = 0 or n = 1
      return n;
   return  fib(n-1) + fib(n-2);     //recursively call 2 functions which is sum of
previous two values
}
```

**B)**

```
int fib (int n)
{
      //create table f[] for bottoms up Dynamic Programming approach
        const int MAX = 999;
        int f[MAX];
   // Base cases
   if (n == 0)
      return 0;
   if (n == 1 || n == 2)
      return (f[n] = 1); // initial values for f array
```

```
    // If fib(n) is already computed use table/Dynamic Programming
approach
    if (f[n])
        return f[n];

     //if code reaches here then value of f[n] does not already exisit on table
and must be computed
   if (n % 2 != 0) //if n is odd
           f[n] = (fib(n)*fib(n) + fib(n-1)*fib(n-1));
    else //n must be even
          f[n] = (2*fib(n -1) + fib(n))*fib(n);

    return f[n]; //return value from table
}
```

**C)**

Run time for this Dynamic Programming algorithm is O(n). This is because we are building up table bottoms up and if value of n was already calculated before we don't need to calculate again and instead it is looked up so all values of n should be calculated O(n) times.

**D)**

The recursive algorithm runs at time of T(n) = 2*T(n-1) + 1 or O(2^n) by using the Master's Theorem of decreasing functions (case 3). Hence, compared to the Recursive algorithm, the Dynamic Programming version is much faster at O(n).

## Question 16

**10 / 15 pts**

Consider the following instance of the knapsack problem with capacity W = 6

| Item | Weight | Value |
|---|---|---|
| 1 | 3 | $25 |
| 2 | 2 | $20 |
| 3 | 1 | $15 |
| 4 | 4 | $40 |
| 5 | 5 | $50 |

a) Apply the bottom-up dynamic programming algorithm to that instance.

b) How many different optimal subsets does the instance of part (a) have?

c) For the bottom-up dynamic programming algorithm, what is the time efficiency, space efficiency and time needed to find the composition of an optimal subset from a filled dynamic programming table?

Your Answer:

A) This is a modified version of the 01 Knapsack problem (since we cannot make fractional components of any of the items). Given max weight of Knapsack, we first take item with max value which is item #5 weight of 5 for value of $50. This leaves weight of 1 remaining which is only item #3 with value of $15. Hence total weight of Knapsack using bottom up dynamic programming algorithm to this instance is $65.

B) there can be two cases for every item: (1) the item is included in the optimal subset, (2) not included in the optimal set
1) Maximum value obtained by n-1 items and W weight (excluding nth item)
2) Value of nth item plus maximum value obtained by n-1 items and W minus weight of the nth item (including nth item)

If weight of nth item is greater than W, then the nth item cannot be included and case 1 is the only possibility.

c) time efficiency = O(nlogn), cspace complexity O(n), and time needed for optimnal subset

Quiz Score: **78** out of 100