• Do not open this quiz booklet until you are directed to do so. Read all the instructions first.

• You have 50 minutes to earn 21 points, the test is graded out of 20.

• This booklet contains 5 questions on 7 pages, including this one. You can use the back of the pages for scratch work.

• Write your solutions in the space provided. If you run out of space, continue your answer on the back of the same sheet and make a notation on the front of the sheet.

• You may use a sheet with notes on both sides.

• You may use a calculator, but not any device with transmitting functions, especially ones that can access the wireless or the Internet.

• **Note that** \( \log_a b < c \) **if and only if** \( b < a^c \). You may leave answers in the form \( \log_a b \) though.

• You may use any of the theorems/facts/lemmas/algorithms/home-works that we covered in class without re-proving them unless explicitly stated otherwise.

• If necessary make reasonable assumptions but please be sure to state them clearly.

• When we ask you to give an algorithm in this test, describe your algorithm in English or pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.

• Do not spend too much time on any one problem. Generally, twice a problems point value is an indication of how many minutes to spend on it.

• Good luck!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Title</th>
<th>Points</th>
<th>Parts</th>
<th>Grade</th>
<th>Initials</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Name / student id on top of every page</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Big-O</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Solving Runtime Recurrences</td>
<td>6</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Analyzing Recursion</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Recursive Algorithm</td>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Application of Fast Multiplication</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>20</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
0 (1 point) Write your name and student id on top of every page.

1. (2 × 1 points) Big-O
For each of the following, whether \( f = O(g) \), \( f = \Theta(g) \) or \( f = \Omega(g) \).
Only the final answer will be marked.

(a) \( f(n) = n^3 \log(n) \) and \( g(n) = 17^n \)

\[ \text{SOLUTION } f(n) = O(g(n)) \]

(b) \( f(n) = n^2 \) and \( g(n) = \log^2 n \)

\[ \text{SOLUTION } f(n) = \Omega(g(n)) \]

GRADING (both parts) 1 point for correct answer.
2. (3 \times 2 \text{ points}) Solve the following runtime recurrences using the master theorem. In each of the cases, you may assume \( T(n) = 1 \) for all \( n \leq 100 \) as the base case. \textbf{Your can express your answers using big-O, but they should be tight up to constant factors.}

(a) \( T(n) = 8T(n/2) + n^3 \)

\textbf{SOLUTION} \( a = 8, b = 2, d = 3, \) hence \( \log_b a = d \). Hence \( O(n^3 \log n) \)

(b) \( T(n) = T(n/5) + n^2 \)

\textbf{SOLUTION} \( a = 1, b = 5, d = 2, \) hence \( \log_b a < d \). Hence \( O(n^2) \)

(c) \( T(n) = 10T(n/2) + n^2 \)

\textbf{SOLUTION} \( a = 10, b = 2, d = 2, \) hence \( \log_b a > d \). Hence \( O(n^{\log_2 10}) \)

\textbf{GRADING} \textbf{(for all three parts)}

2 points for correctly applying master theorem.
1 point if any minor error.
3. (4 points) Analyzing Recursion. **Note:** although the question asks you to give the answers in big-O notation, your answers should be tight up to constant factors.

We have the following piece of code, representing a recursive function.

```python
function NoIdea(n)
    if n > 1:
        print 'A'
        NoIdea(n/3)
    for i = 1 \ldots n
        print 'B'
    end for
    NoIdea(n/3)

(a) (2 points) What is the runtime of the above function? Express your answer using the big-O notation.

SOLUTION The runtime recurrence satisfies

\[ T(n) = 2T(n/3) + O(n), \]

since each for loop takes \( O(n) \) time to print all the ‘B’s.

This satisfies the conditions of master theorem with \( a = 2, b = 3, \) and \( d = 1. \)

Since \( \log_3 2 < 1 \), we get \( T(n) = O(n) \)

(b) (2 points) Express the number of times that this algorithm prints ‘A’ in terms of \( n \) using the big-O notation.

The recurrence for the number of times ‘A’ is printed is:

\[ A(n) = 2A(n/3) + 1. \]

This satisfies the conditions of master theorem with \( a = 2, b = 3, \) and \( d = 0. \)

Since \( \log_3 2 > 0 \), we get \( A(n) = O(n^{\log_3 2}) \)

**GRADING (for both parts):**

2 points for the answer. 1 point if answer is incorrect, but one of

- runtime recurrence,
- guess-and-check steps,
- geometric sum from recursion tree

is correct.
4. (6 points) Recursive Algorithm

We have a length $n$ binary (01) array $A(1, 2, \ldots, n)$ such that $A[1] = 0$ and $A[n] = 1$. Call an index $1 \leq i < n$ such that $A(i) = 0$ and $A(i + 1) = 1$ a transition index. The goal of this problem is to give an algorithm that finds a transition index in $O(\log n)$ time.

The question is broken down into several parts to guide the solution. **you may assume each part in later ones.**

(a) (1 point) show that a transition index always exists in an array $A(1, 2, \ldots, n)$ with $A[1] = 0$ and $A[n] = 1$. **You may skip this part if you solved parts (b) and (c)**

**SOLUTION** Consider the first 1 in the array. It must exist because $A[n] = 1$, and it cannot be 1 since $A[1] = 0$. Its predecessor must be 0 by the choice of it being the first 1, so its predecessor must be a transition index.

**GRADING** 1 point for correct proof, or if solutions to parts b) and c) are correct.

(b) (2 points) Suppose $n = 2k + 1$ for some integer $k$. Show that no matter if $A[k + 1]$ is 0 or 1, we can find a sub-array of length $k + 1$ such that it starts with 0 and ends with 1.

**SOLUTION** The value in the middle index: $A[k + 1]$ is either 0 or 1. If it is zero, then the right half is the required subarray because $A[n] = 1$, else the left half is the required subarray because $A[1] = 0$.

**GRADING** 2 points for completely correct. 1 point off for minor errors.
(c) (2 points) Give a recursive algorithm that finds one transition point of a given array. Make sure to include the base case of the recursion.

**SOLUTION**

```python
function findTransition(arr, start, end):
    if start == end - 1
        return start
    m = (start + end) / 2
    if arr[m] != arr[start]:
        return findTransition(arr, start, m)
    else:
        return findTransition(arr, m, end)
```

**GRADING** 2 points for correct pseudocode. 1 point for any error in pseudocode.

(d) (1 point) Show that the algorithm from part (c) runs in $O(\log n)$ time.

**SOLUTION** $T(n) = T(n/2) + O(1)$.

This fits into master theorem with $a = 1$, $b = 2$, and $d = 0$, so we get $T(n) = O(\log n)$

**GRADING** 1 point for correct justification.
5. (2 points)

Application of Fast Multiplication

**NOTE**: this is out of the scope of what we covered in class this time. Homework 1 Problem 4 touches upon this, but this connection to polynomials and such applications will definitely not be on the test this time.

Given 3 subset of numbers $A$, $B$, $C$, each a subset of $\{1 \ldots n\}$, we want to quickly count the number of solutions to

$$5a + 7b = 11c$$

where $a \in A$, $b \in B$, and $c \in C$.

Show that the number of ways of getting each possible value of $5a + 7b$ can be found by building two $O(n)$ sized numbers / polynomials based on $A$ and $B$, and taking their product without carries.

You only need to give the construction, not prove its correctness.

**SOLUTION**

Multiply

$$P(x) = \sum_{a \in A} x^{5a}$$

and

$$Q(x) = \sum_{b \in B} x^{7b}.$$  

The coefficients of $x^k$ in the result gives the number of ways of forming

$$k = 5a + 7b$$

since

$$P(x)Q(x) = \left( \sum_{a \in A} x^{5a} \right) \left( \sum_{b \in B} x^{7b} \right)$$

$$= \sum_{a \in A, b \in B} x^{5a} \times x^{7b}$$

$$= \sum_{a \in A, b \in B} x^{5a+7b}.$$  

**GRADING** 2 points for any valid polynomials.