Student Name: _____ Student ID # _____

OU Academic Integrity Pledge

On my honor I affirm that I have neither given nor received inappropriate aid in the completion of this exercise.

Signature: _____ Date: _____

Notes Regarding this Examination

Open Book(s) You may consult any printed textbooks in your immediate possession during the course of this examination.

Open Notes You may consult any printed notes in your immediate possession during the course of this examination.

No Electronic Devices Permitted You may not use any electronic devices during the course of this examination, including but not limited to calculators, computers, and cellular phones. All electronic devices in the student's possession must be turned off and placed out of sight (for example, in the student's own pocket or backpack) for the duration of the examination.

Violations Copying another's work, or possession of electronic computing or communication devices in the testing area, is cheating and grounds for penalties in accordance with school policies.

Part I. Definitions of Time and Space Complexity

Definition of Big O: Let $f(n)$ and $g(n)$ be functions mapping non-negative integers to real numbers. We say that $f(n) \in O(g(n))$ if there is a real number $c > 0$ and a fixed integer $n_0 \geq 1$ such that $f(n) \leq cg(n)$ for every integer $n \geq n_0$.

Definition of Big Ω: Let $f(n)$ and $g(n)$ be functions mapping non-negative integers to real numbers. We say that $f(n) \in \Omega(g(n))$ if there is a real number $c > 0$ and a fixed integer $n_0 \geq 1$ such that $f(n) \geq cg(n)$ for every integer $n \geq n_0$.

Definition of Big Θ: Let $f(n)$ and $g(n)$ be functions mapping non-negative integers to real numbers. We say that $f(n) \in \Theta(g(n))$ if there are real numbers $c, d > 0$ and a fixed integer $n_0 \geq 1$ such that $dg(n) \leq f(n) \leq cg(n)$ for every integer $n \geq n_0$.

1. (2 points) Which is typically used to provide an upper bound on the time complexity of an algorithm?
   A. Big O
   B. Big Ω
   C. Big Θ
   D. A and B
   E. A, B, and C

2. (2 points) Which is typically used to provide a lower bound on the time complexity of an algorithm?
   A. Big O
   B. Big Ω
   C. Big Θ
   D. A and B
   E. A, B, and C

3. (2 points) Which is typically used to provide both an upper and a lower bound on the time complexity of an algorithm?
   A. Big O
   B. Big Ω
   C. Big Θ
   D. A and B
   E. A, B, and C

4. (2 points) Which is generally considered to be a better time complexity?
   A. $\Theta(n \log_2 n)$
   B. $\Theta(n^2)$
   C. $\Theta(n)$
   D. $\Theta(\log_2 n)$
   E. $\Theta(n^2 \log_2 n)$

5. (2 points) Which is generally considered to be a better time complexity?
   A. $\Theta(n + \log_2 n)$
   B. $\Theta(n^2)$
   C. $\Theta(n + n \log_2 n)$
   D. $\Theta(n^2 + \log_2 n)$
   E. $\Theta(n^2 + n)$

6. (2 points) A time complexity of $\Theta(n \log_2 n + \log_2 n + n^2)$ can be simplified to which?
   A. $\Theta(n \log_2 n)$
   B. $\Theta(n \log_2 + \log_2 n)$
   C. $\Theta(\log_2 n)$
   D. $\Theta(n^2 \log_2 n)$
   E. $\Theta(n^2)$

Part II. Applications of Time and Space Complexity

Consider the following pseudocode:

```
Algorithm RecursiveFunction (a, b)  // a and b are integers
  if (a ≤ 0)
    return b;
  else
   return RecursiveFunction (a/2, a + b);
  endif
```

7. (2 points) What is the time complexity of the `RecursiveFunction` pseudocode shown above?
   A. $\Theta(a)$
   B. $\Theta(\log_2 a)$
   C. $\Theta(1)$
   D. $\Theta(a/2)$
   E. $\Theta(a+b)$

8. (2 points) What is the space complexity of the `RecursiveFunction` pseudocode shown above?
   A. $\Theta(a)$
   B. $\Theta(\log_2 a)$
   C. $\Theta(1)$
   D. $\Theta(a/2)$
   E. $\Theta(a+b)$

Consider the following pseudocode:

```
Algorithm IterativeFunction (a, b)  // a and b are integers
  while (a > 0)
    b ← a + b
    a ← a / 2
  endwhile
  return b;
```

9. (2 points) What is the time complexity of the `IterativeFunction` pseudocode shown above?
   A. $\Theta(a)$
   B. $\Theta(\log_2 a)$
   C. $\Theta(1)$
   D. $\Theta(a/2)$
   E. $\Theta(a+b)$

10. (2 points) What is the space complexity of the `IterativeFunction` pseudocode shown above?
    A. $\Theta(a)$
    B. $\Theta(\log_2 a)$
    C. $\Theta(1)$
    D. $\Theta(a/2)$
    E. $\Theta(a+b)$

Consider the following pseudocode:

```
Algorithm DoubleIterativeFunction (n)  // n is an integer
  i ← 1
  while (i < n) do
    j ← 1
    while (j < n) do
      print (i + j) / 2
      j ← j + 1
    endwhile
    i ← i + 1
  endwhile
```

11. (2 points) What is the time complexity of the `DoubleIterativeFunction` pseudocode shown above?
    A. $\Theta(n)$
    B. $\Theta(\log_2 n)$
    C. $\Theta(1)$
    D. $\Theta(n^2)$
    E. $\Theta(\log_2(i+j))$

12. (2 points) What is the space complexity of the `DoubleIterativeFunction` pseudocode shown above?
    A. $\Theta(n)$
    B. $\Theta(\log_2 n)$
    C. $\Theta(1)$
    D. $\Theta(n^2)$
    E. $\Theta(\log_2(i+j))$

Consider the following pseudocode:

```
Algorithm DoubleIterativeFunction2 (n)  // n is an integer
  i ← 1
  while (i < n) do
    j ← 1
    while (j < i) do   // note the change in this line
      print (i + j) / 2
      j ← j + 1
    endwhile
    i ← i + 1
  endwhile
```

13. (2 points) What is the time complexity of the `DoubleIterativeFunction2` pseudocode shown above?
    A. $\Theta(n)$
    B. $\Theta(\log_2 n)$
    C. $\Theta(1)$
    D. $\Theta(n^2)$
    E. $\Theta(\log_2(i+j))$

14. (2 points) What is the space complexity of the `DoubleIterativeFunction2` pseudocode shown above?
    A. $\Theta(n)$
    B. $\Theta(\log_2 n)$
    C. $\Theta(1)$
    D. $\Theta(n^2)$
    E. $\Theta(\log_2(i+j))$

Part III. Time and Space Complexity of Searching and Sorting

15. (2 points) What is the time complexity of the most efficient algorithm for **searching** for an arbitrary item in an **unsorted** array of n items?

A. $\Theta(n)$
B. $\Theta(\log_2 n)$
C. $\Theta(n \log_2 n)$
D. $\Theta(n/2)$
E. $\Theta(n^2)$

16. (2 points) What is the time complexity of the most efficient algorithm for **searching** for an arbitrary item in a **sorted** array of n items?

A. $\Theta(n)$
B. $\Theta(\log_2 n)$
C. $\Theta(n \log_2 n)$
D. $\Theta(n/2)$
E. $\Theta(n^2)$

17. (2 points) What is the time complexity of the most efficient algorithm for **sorting** an array of n arbitrary items?

A. $\Theta(n)$
B. $\Theta(\log_2 n)$
C. $\Theta(n \log_2 n)$
D. $\Theta(n/2)$
E. $\Theta(n^2)$

18. (2 points) What is the time complexity of the most efficient algorithm for performing $\boldsymbol{n}$ **searches** for arbitrary items in an **unsorted** array of n items?

A. $\Theta(n)$
B. $\Theta(\log_2 n)$
C. $\Theta(n \log_2 n)$
D. $\Theta(n/2)$
E. $\Theta(n^2)$

19. (2 points) What is the time complexity of the most efficient algorithm for performing $\boldsymbol{n}$ **searches** for arbitrary items in a **sorted** array of n items?

A. $\Theta(n)$
B. $\Theta(\log_2 n)$
C. $\Theta(n \log_2 n)$
D. $\Theta(n/2)$
E. $\Theta(n^2)$

20. (2 points) What is the time complexity of the most efficient algorithm for **sorting** an array of n arbitrary items and then performing $\boldsymbol{n}$ **searches** for arbitrary items in the **sorted** array?

A. $\Theta(n)$
B. $\Theta(\log_2 n)$
C. $\Theta(n \log_2 n)$
D. $\Theta(n/2)$
E. $\Theta(n^2)$

Exam continues with short answer questions.

Short Answer Question 1: Bubble Sort (10 points)

A. What is the time complexity of Bubble Sort?

B. Explain why Bubble Sort has the time complexity you listed above.

Short Answer Question 2: Merge Sort (10 points)

A. What is the time complexity of Merge Sort?

B. Explain why Merge Sort has the time complexity you listed above.

Short Answer Question 3: Binary Selection Sort (10 points)

A. What is the time complexity of Binary Selection Sort?

B. Explain why Binary Selection Sort has the time complexity you listed above.

Short Answer Question 4: Quicksort (30 points)

```
Algorithm Quicksort (A, left, right)
  if (left < right)
    pivotPoint ← ⌊(left+right)/2⌋  // note central pivot
    i ← left - 1
    j ← right + 1
    do
      do
        i ← i + 1
      while (i < A.size) and (A[i] ≤ A[pivotPoint])
      do
        j ← j - 1
      while (j > 0) and (A[j] > A[pivotPoint])
      if (i < j) then swap (A[i], A[j])
    while (i < j)
    swap A[pivotPoint], A[j])
    Quicksort (A, left, j-1)
    Quicksort (A, j+1, right)
  endif
```

Show the steps followed by the Quicksort algorithm given above in pseudocode when sorting the following array. Draw one figure for each call to Quicksort. You may omit calls where `left` $\not<$ `right`.

value	88	6	63	36	43	16	11	32	13	25
index	0	1	2	3	4	5	6	7	8	9

Additional space to answer Short Answer Question 4.