Elementary Data Structures and Algorithms

Order Notation and the Mathematics for Analysis of Algorithms

Concept: *mathematics notation*

1. log 2 n is:

   A. Θ (log 10 n)

   B. ω (log 10 n)

   C. o (log 10 n)

2. log 2 n is equal to:

   A. log 10 n log 10 2

   B. log 10 n log 2 10

   C. log 2 n log 2 10

   D. log 2 n log 10 2

3. log (n m) is equal to:

   A. m log n

   B. n log m

   C. (log n) m

   D. log n + log m

4. log (n m) is equal to:

   A. log n + log m

   B. (log n) m

   C. n log m

   D. m log n

5. log 2 2 can be simplified to:

   A. 1

   B. 4

   C. 2

   D. log 2 2 cannot be simplified any further

6. 2 log 2 n is equal to:

   A. *n*

   B. log 2 n

   C. n 2

   D. 2 n

7. n 2 is o (n 3) . Therefore, log n 2 is ? (log n 3) . Choose the tightest bound.

A. theta

B. big omega

C. little omicron

D. little omega

E. big omicron

8. log n n is Θ (?) .

A. log *n*

B. *n* log *n*

C. log n log n

D. *n*

9. log 2 n is Θ (?) .

A. n log n

B. *n*

C. 2 n

D. log n

10. The number of permutations of a list of *n* items is:

A. *n*!

B. log n

C. 2 n

D. n log n

E. *n*

Concept: relative growth rates

11. What is the correct ordering of growth rates for the following functions:

- f (n) = n 0.9 log n
- g (n) = 1. 1 n
- h (n) = 9.9 n

A. $h < g < f$

B. $g < f < h$

C. $h < f < g$

D. $g < h < f$

E. $f < h < g$

F. $f < g < h$

12. What is the correct ordering of growth rates for the following functions:

- f (n) = n (log n) 2
- g (n) = n log 2 n
- h (n) = n log (log n)

A. $h > g > f$

B. $h > f > g$

C. $f > g > h$

D. $g > f > h$

E. $f > h > g$

F. $g > h > f$

Concept: *order notation*

13. What does big Omicron roughly mean?

    A. worse than or the same as

    B. the same as

    C. worse than

    D. better than or the same as

    E. better than

14. What does ω roughly mean?

    A. the same as

    B. worse than

    C. better than or the same as

    D. worse than or the same as

    E. better than

15. What does θ roughly mean?

    A. the same as

    B. better than or the same as

    C. better than

    D. worse than

    E. worse than or the same as

16. **T** or **F**: All algorithms are ω (1) .

17. **T** or **F**: All algorithms are θ (1) .

18. **T** or **F**: All algorithms are Ω (1) .

19. **T** or **F**: There exist algorithms that are ω (1) .

20. **T** or **F**: There exist algorithms that are O (1) .

21. **T** or **F**: All algorithms are O (n n) .

22. Consider sorting 1,000,000 numbers with mergesort. What is the time complexity of this operation? [THINK!]

    A. constant, because *n* is fixed

    B. n 2 , because mergesort takes quadratic time

    C. *n* log *n*, because mergesort takes *n* log *n* time

Concept: *comparing algorithms using order notation*

The phrase *by a stopwatch* means the actual amount of time needed for the algorithm to run to completion, as measured by a stopwatch.

23. **T** or **F**: If f = ω (g) , then algorithm *f* always runs slower than *g* (by a stopwatch), in all cases.

24. **T** or **F**: If f = ω (g) and the input causes worst-case behaviors, then algorithm *f* always runs slower than *g* (by a stopwatch), regardless of input size.

25. **T** or **F**: If f = ω (g) and the input causes worst-case behaviors, then algorithm *f* always runs slower than *g* (by a stopwatch), above a certain input size.

26. **T** or **F**: If f = ω (g) , then algorithm *f* always runs slower than *g* (by a stopwatch), above a certain input size.

27. **T** or **F**: If f = θ (g) , then algorithm *f* always takes the same time as *g* (by a stopwatch), in all cases.

28. **T** or **F**: If f = θ (g) and the input causes worst-case behaviors, then algorithm *f* always takes the same time as *g* (by a stopwatch), regardless of input size.

29. **T** or **F**: If f = θ (g) and the input causes worst-case behaviors, then algorithm *f* always takes the same time as *g* (by a stopwatch), above a certain input size.

30. **T** or **F**: If f = θ (g) , then algorithm *f* always takes the same time as *g* (by a stopwatch), above a certain input size.

31. **T** or **F**: If f = θ (g) , then algorithm *f* always takes the same time as *g* (within a constant factor), in all cases.

32. **T** or **F**: If f = θ (g) and the input causes worst-case behaviors, then algorithm *f* always takes the same time as *g* (within a constant factor), regardless of input size.

33. **T** or **F**: If f = θ (g) and the input causes worst-case behaviors, then algorithm *f* always takes the same time as *g* (within a constant factor), above a certain input size.

34. **T** or **F**: If f = θ (g) , then algorithm *f* always takes the same time as *g* (within a constant factor), above a certain input size.

35. **T** or **F**: If f = ω (g) , then *f* and *g* can be the same algorithm.

36. **T** or **F**: If f = Ω (g) , then *f* and *g* can be the same algorithm.

37. **T** or **F**: If f = o (g) , then *f* and *g* can be the same algorithm.

38. **T** or **F**: If f = O (g) , then *f* and *g* can be the same algorithm.

39. **T** or **F**: Suppose algorithm f = θ (g) . *f* and *g* can be the same algorithm.

40. **T** or **F**: If f = Ω (g) and f = O (g) , then f = Θ (g) .

41. **T** or **F**: If f = Ω (g) and f = O (g) , then *f* and *g* must be the same algorithm.

Concept: *analyzing code*

In the pseudocode, the lower limit of a `for` loop is inclusive, while the upper limit is exclusive. The additive step, if not specified, is one.

Tracing recursive functions

When asked about the number of recursive calls, do not include the original call.

42. How many recursive calls are made if n = 5? Assume the initial value of *i* is zero.

```
function f(i,n)
    {
    if (i < n)
        {
        println(i);
        f(i+1,n);
        }
    return 0;
    }
```

A. the function recurs infinitely

B. 3

C. 6

D. 4

E. 5

F. none of the other answers are correct

43. How many recursive calls are made if n = 81. Assume the initial value of *i* is one.

```
function f(i,n)
    {
    if (i < n)
        {
        println(i);
        f(i*3,n);
        }
    return 0;
    }
```

A. 3

B. 5

C. the function recurs infinitely

D. 6

E. 4

F. none of the other answers are correct

44. How many recursive calls are made if n = 64. Assume the initial value of *i* is one.

```
function f(i,n)
    {
    if (i < n)
        {
        println(i);
        f(i*sqrt(n),n);
        }
    return 0;
    }
```

A. 7

B. 9

C. the function recurs infinitely

D. none of the other answers are correct

E. 1

F. 4

G. 8

H. 2

45. How many recursive calls are made if n = 64. Assume the initial value of *i* is zero.

```
function f(i,n)
    {
    if (i < n)
        {
        println(i);
        f(i+sqrt(n),n);
        }
    return 0;
    }
```

A. 1

B. the function recurs infinitely

C. 7

D. 2

E. 8

F. 4

G. 9

H. none of the other answers are correct

Time complexity, recursive functions, single recursion

46. What is the time complexity of this function? Assume the initial value of *i* is zero.

```
function f(i,n)
    {
    if (i < n)
        {
        println(i);
        f(i+1,n);
        }
    return 0;
    }
```

A. θ (log n)

B. θ (n)

C. θ (n 2)

D. θ (1)

E. θ (log n)

F. θ (n)

47. What is the time complexity of this function? Assume the initial value of *i* is one.

```
function f(i,n)
    {
    if (i < n)
        {
        println(i);
        f(i*3,n);
        }
```

```
return 0;
}
```

A. θ (log n)

B. θ ((log n) 3)

C. θ (n log n)

D. θ (n 2)

E. θ (n)

F. θ (n n)

48. What is the time complexity of this function? Assume the initial value of *i* is one.

```
function f(i,n)
    {
    if (i < n)
        {
        f(i*sqrt(n),n);
        println(i);
        }
    }
```

A. θ (log n)

B. θ (n)

C. θ (1)

D. θ (n)

E. θ (n 2)

F. θ (n n)

49. What is the time complexity of this function? Assume positive, integral input and integer division.

```
function f(n)
    {
    if (n > 0)
        {
        for (var i from 0 until n)
            println(n);
        f(n/2);
        }
    return 0;
    }
```

A. θ (1)

B. θ (n 2)

C. θ (n log n)

D. θ (log n)

E. θ (n)

F. θ (n n)

Time complexity, recursive functions, double recursion

Space complexity, recursive functions, single recursion

50. What is the space complexity of this function? Assume positive, integral input and integer division.

```
function f(x,n)
    {
    if (x > 0)
        {
        f(x/2,n);
        for (var i from 0 until n)
            println(n);
        }
    }
```

A. θ (1)

B. θ (n log x)

C. θ (x)

D. θ (log n)

E. θ (log x)

F. θ (x / 2)

G. θ (n x)

H. θ (n)

51. What is the space complexity of this function? Assume the initial value of *i* is zero.

```
function f(i,n)
    {
    if (i < n)
        {
        f(i+2,n);
        println(i);
        }
    }
```

A. θ (n)

B. θ (log n)

C. θ (1)

D. θ (n 2)

E. θ (n)

F. θ (n log n)

52. What is the space complexity of this function? Assume the initial value of *i* is one.

```
function f(i,n)
    {
    if (i < n)
        {
        f(i*3,n);
        println(i);
        }
    }
```

A. θ (log n)

B. θ (n log n)

C. θ (n 2)

D. θ (n)

E. θ (n)

F. θ (1)

53. What is the space complexity of this function? Assume the initial value of *i* is one.

```
function f(i,n)
    {
    if (i < n)
        {
        f(i*sqrt(n),n);
        println(i);
        }
    }
```

A. θ (1)

B. θ (n n n)

C. θ (n)

D. θ (n - n)

E. θ (n)

F. θ (n n)

54. What is the space complexity of this function? Assume the initial value of *i* is one.

```
function f(i,n)
    {
    if (i < n)
        {
        f(i+sqrt(n),n);
        println(i);
        }
    }
```

A. θ (n n)

B. θ (n n n)

C. θ (n)

D. θ (n - n)

E. θ (1)

F. θ (n)

Space complexity, recursive functions, double recursion

Time complexity, iterative loops, single loops

55. What is the time complexity of this code fragment?

```
for (i from 0 until n by 1)
    println(i);
```

A. θ (n)

B. θ (n n)

C. θ (n 2)

D. θ (n n)

E. θ (log 2 n)

F. θ (n log n)

56. What is the time complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    println(i);
    i = i * 2;
    }
```

A. θ ((log n) 2)

B. θ (n n)

C. θ (n)

D. θ (n log n)

E. θ (log n)

F. θ (n 2)

57. What is the time complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    println(i);
    i = i * sqrt(n);
    }
```

A. θ (n n)

B. θ (n - n)

C. θ (n)

D. θ (n n n)

E. θ (n)

F. θ (1)

Time complexity, iterative loops, double loops

58. What is the time complexity of this code fragment?

```
for (i from 0 until n by 1)
    for (j from 0 until i by 1)
        println(i,j);
```

A. θ (n n)

B. θ (n log n)

C. θ (n 2)

D. θ (log 2 n)

E. θ (n)

F. θ (1)

59. What is the time complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    for (j from 0 until n by 1)
        println(i,j);
    i = i * 2;
    }
```

A. θ (n n)

B. θ (log 2 n)

C. θ (1)

D. θ (n)

E. θ (n log n)

F. θ (n 2)

60. What is the time complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    for (j from 0 until i by 1)
        println(i,j);
    i = i * 2;
    }
```

A. θ (n n)

B. θ (n 2)

C. θ (n log n)

D. θ (1)

E. θ (log 2 n)

F. θ (n)

61. What is the time complexity of this code fragment?

```
for (i from 0 until n)
    {
    j = 1;
    while (j < n)
        {
        println(i,j);
        j = j * 2;
        }
    }
```

A. θ (n n)

B. θ (n)

C. θ (n 2)

D. θ (n log n)

E. θ (1)

F. θ (log 2 n)

62. What is the time complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    println(i);
    i = i * 2;
    }
for (j from 0 until n by 2)
    println(j);
```

A. θ (n n)

B. θ (n)

C. θ (n log n)

D. θ (log 2 n)

E. θ (1)

F. θ (n 2)

63. What is the time complexity of this code fragment?

```
for (i from 0 until n by 2)
    println(i);
for (j from 0 until n by 1)
    println(j);
```

A. θ (log 2 n)

B. θ (n)

C. θ (n n)

D. θ (n 2)

E. θ (1)

F. θ (n log n)

64. What is the time complexity of this code fragment?

```
for (i from 0 until n by 1)
    println(i);
j = 1;
while (j < n)
    {
    println(j);
    j = j * 2;
    }
```

A. θ (n 2)

B. θ (n)

C. θ (log 2 n)

D. θ (n log n)

E. θ (n n)

F. θ (1)

Space complexity, iterative loops, single loops

65. What is the space complexity of this code fragment?

```
for (i from 0 until n by 1)
    println(i);
```

A. θ (n 2)

B. θ (n)

C. θ (n)

D. θ (log n)

E. θ (n log n)

F. θ (1)

66. What is the space complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    println(i);
    i = i * sqrt(n);
    }
```

A. θ (1)

B. θ (n)

C. θ (n log n)

D. θ (n)

E. θ (log n)

F. θ (n 2)

67. What is the space complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    println(i);
    i = i * 2;
    }
```

A. θ (n)

B. θ (1)

C. θ (log n)

D. θ (n log n)

E. θ (n)

F. θ (n 2)

Space complexity, iterative loops, double loops

68. What is the space complexity of this code fragment?

```
for (i from 0 until n by 1)
    println(i);
for (j from 0 until n by 1)
    println(j);
```

A. θ (n 2)

B. θ (log n)

C. θ (n)

D. θ (n log n)

E. θ (n)

F. θ (1)

69. What is the space complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    println(i);
    i = i * 2;
    }
for (j from 0 until n by 1)
    println(j);
```

A. θ (n)

B. θ (n log n)

C. θ (log n)

D. θ (n 2)

E. θ (n)

F. θ (1)

70. What is the space complexity of this code fragment?

```
for (i from 0 until n by 1)
    println(i);
j = 1;
while (j < n)
    {
    println(j);
    j = j * 2;
    }
```

A. θ (1)

B. θ (log n)

C. θ (n log n)

D. θ (n)

E. θ (n 2)

F. θ (n)

71. What is the space complexity of this code fragment?

```
for (i from 0 until n by 1)
    for (j from 0 until i by 1)
        println(i,j);
```

A. θ (n log n)

B. θ (log n)

C. θ (n)

D. θ (n 2)

E. θ (n)

F. θ (1)

72. What is the space complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    for (j from 0 until i by 1)
        println(i,j);
    i = i * 2;
    }
```

A. θ (n)

B. θ (1)

C. θ (n)

D. θ (log n)

E. θ (n 2)

F. θ (n log n)

73. What is the space complexity of this code fragment?

```
for (i from 0 until n)
    {
    j = 1;
    while (j < n)
        {
        println(i,j);
```

```
        j = j * 2;
        }
    }
```

A. θ (n)

B. θ (n 2)

C. θ (n)

D. θ (log n)

E. θ (1)

F. θ (n log n)

74. What is the space complexity of this code fragment?

```
i = 1;
while (i < n)
    {
    for (j from 0 until n by 1)
        println(i,j);
    i = i * 2;
    }
```

A. θ (1)

B. θ (n)

C. θ (n log n)

D. θ (n 2)

E. θ (n)

F. θ (log n)

Concept: *analysis of classic, simple algorithms*

75. Which of the following describes the classic recursive fibonacci's time complexity?

A. θ (n)

B. θ (Φ)

C. θ (Φ n)

D. θ (n - n)

E. θ (n n)

F. θ (1)

G. θ (Φ n)

76. Which of the following describes the classic recursive fibonacci's space complexity?

A. θ (n n)

B. θ (n)

C. θ (1)

D. θ (Φ n)

E. θ (n)

F. θ (n - n)

77. Which of the following describes iterative factorial's time complexity?

A. θ (Φ n)

B. θ (n n)

C. θ (n)

D. θ (n)

E. θ (n - n)

F. θ (1)

78. Which of the following describes iterative fibonacci's space complexity?

A. θ (n - n)

B. θ (1)

C. θ (n)

D. θ (n n)

E. θ (n)

F. θ (Φ n)