Due to the print book page limit, we cannot inlcude all good CheckPoint questions in the physical book. The CheckPoint on this Website may contain extra questions not printed in the book. The questions in some sections may have been reordered as a result. Nevertheless, it is easy to find the CheckPoint questions in the book on this Website. Please send suggestions and errata to Dr. Liang at y.daniel.liang@gmail.com. Indicate the book, edition, and question number in your email. Thanks!

Chapter 22 Check Point Questions

Section 22.2

▼22.2.1

Why is a constant factor ignored in the Big O notation? Why is a nondominating term ignored in the Big O notation?

The constant factor is ignored in big O notation, because it has no impact on the growth rate of the time complexity function. A nondominating term is ignored in Big O notation, because as the input size grows, the dominating term grows much faster than the nondominating term.

Hide Answer

▼22.2.2

What is the order of each of the following functions?

(a) $(n^2 + 1)^2/n$
(b) $(n^2 + \log^2 n)^2 / n$
(c) $n^3 + 100n^2 + n$
(d) $2^n + 100n^2 + 45n$
(e) $n2^n + n^2 2^n$

(a) $(n^2 + 1)^2/n = O(n^3)$
(b) $(n^2 + \log^2 n)^2 / n = O(n^3)$
(c) $n^3 + 100n^2 + n = O(n^3)$
(d) $2^n + 100n^2 + 45n = O(2^n)$
(e) $n2^n + n^2 2^n = O(n^2 2^n)$

Hide Answer

Section 22.3

▼22.3.1

Count the number of iterations in the following loops.

```
(a)
int count = 1;
while (count < 30) {
  count = count * 2;
}

(b)
int count = 15;
while (count < 30) {
  count = count * 3;
}

(c)
```

```
int count = 1;
while (count < n) {
  count = count * 2;
}
```

```
(d)
int count = 15;
while (count < n) {
  count = count * 3;
}
```

(A) 5
(B) 1
(C) The ceiling of log2n times
(D) The ceiling of log3(n/15) times

Hide Answer

▼22.3.2

How many stars are displayed in the following code if n is 10? How many if n is 20? Use the Big O notation to estimate the time complexity.

```
(a)
for (int i = 0; i < n; i++) {
  System.out.print('*');
}
```

```
(b)
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    System.out.print('*');
  }
}
```

```
(c)
for (int k = 0; k < n; k++) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      System.out.print('*');
    }
  }
}
```

```
(d)
for (int k = 0; k < 10; k++) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      System.out.print('*');
    }
  }
}
```

if n is 10: (a) 10 (b) 10^2 (c) 10^3 (d) 10*10^2
if n is 20: (a) 20 (b) 20^2 (c) 20^3 (d) 20*20^2
Using Big-O notation: O(n), O(n^2), O(n^3), O(n^2)

Hide Answer

▼22.3.3

Use the Big O notation to estimate the time complexity of the following methods:

```
(a)
public static void mA(int n) {
  for (int i = 0; i < n; i++) {
    System.out.print(Math.random());
  }
}

(b)
public static void mB(int n) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++)
      System.out.print(Math.random());
  }
}

(c)
public static void mC(int[] m) {
  for (int i = 0; i < m.length; i++) {
    System.out.print(m[i]);
  }

  for (int i = m.length - 1; i >= 0; )
  {
    System.out.print(m[i]);
    i--;
  }
}

(d)
public static void mD(int[] m) {
  for (int i = 0; i < m.length; i++) {
    for (int j = 0; j < i; j++)
      System.out.print(m[i] * m[j]);
  }
}
```

(a): O(n)
(b): O(n^2)
(c): O(n)
(d): O(n^2)

Hide Answer

▼22.3.4

Design an O(n) time algorithm for computing the sum of numbers from n1 to n2 for (n1 < n2). Can you design an O(1) for performing the same task?

An O(n) time algorithm for this is

```
int sum = 0;
for (int i = n1; i <=  n2; i++)
  sum += i;
```

An O(1) time algorithm for this is

```
int sum = n2 * (n2 + 1) / 2 - n1 * (n1 - 1) / 2;
```

Hide Answer

▼22.3.5

Example 7 in Section 22.3 assumes $n = 2^k$ Revise the algorithm for an arbitrary n and prove that the complexity is still O(logn).

```
result = a;
i = 2;

while (i <=  n) {
  result = result * result;
  i *= 2;
}

for (int j = i / 2 + 1; j <=  n; j++)
  result = result * a;
```

Assume that $2^{k-1} <= n < 2^k$. The while loop is executed k-1 times. The for loop is executed at most $2^k-2^{k-1}=2^{k-1}$ times. So, the total complexity is O(n). Consider another implementation:

```
public static int f(int a, int n) {
  if (n == 1) {
    return a;
  }
  else {
    int temp = f(a, n / 2);
    if (n % 2 == 0) {
      return temp * temp;
    }
    else {
      return a * temp * temp;
    }
  }
}
```

This implementation results in O(logn) complexity.

Hide Answer

Section 22.4

▼22.4.1

Put the following growth functions in order: $5n^3/4032$, 44logn, 10nlogn, 500, $2n^2$, $2^n/45$, 3n

500,44logn,3n,10nlogn $2n^2$,$5n^3/4032$,$2^n/45$

Hide Answer

▼22.4.2

Estimate the time complexity for adding two n by m matrices, and for multiplying an n by m matrix by an m by k matrix.

Adding two matrices: O(nm). Multiplying two matrices: O(nmk)

Hide Answer

▼22.4.3

Describe an algorithm for finding the occurrence of the max element in an array. Analyze the complexity of the algorithm.

The algorithm can be designed as follows: Maintain two variables, max and count. max stores the current max number, and count stores its occurrences. Initially, assign the first number to max and 1 to count. Compare each subsequent number with max. If the number is greater than max, assign it to max and reset count to 1. If the number is equal to max, increment count by 1. Since each element in the array is examined only once, the complexity of the algorithm is O(n).

Hide Answer

▼22.4.4

Describe an algorithm for removing duplicates from an array. Analyze the complexity of the algorithm.

The algorithm can be designed as follows: For each element in the input array, store it to a new array if it is new. If the number is already in the array, ignore it. The time for checking whether an element is already in the new array is O(n), so the complexity of the algorithm is O(n^2).

Hide Answer

▼22.4.5

Analyze the following sorting algorithm:

```
for (int i = 0; i < list.length - 1; i++) {
  if (list[i] > list[i + 1]) {
    swap list[i] with list[i + 1];
    i = -1;
  }
}
```

This is similar to bubble sort. Whenever a swap is made, it goes back to the beginning of the loop. In the worst case, there will be O(n^2) of swaps. For each swap, O(n) number of comparisons may be made in the worst case. So, the total is O(n^3) in the worst case.

Hide Answer

▼22.4.6

Analyze the complexity for computing a polynomial f(x) of degree n for a given x value using a brute-force approach and the Horner's approach, respectively. A brute-force approach is to compute each term in the polynomial and add them together. The Horner's approach was introduced in Section 6.7.

$f(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + ... + a_1 x^1 + a_0$

A brute-force for approach to evaluate a polynomial f(x) of degree n will take n+(n-1)+…+2+1=O(n^2) time. The Horner's method takes O(n) time.

Hide Answer

Section 22.5

▼22.5.1

What is dynamic programming? Give an example of dynamic programming.

See the definition and example in the text.

Hide Answer

▼22.5.2

Why is the recursive Fibonacci algorithm inefficient, but the nonrecursive Fibonacci algorithm efficient?

The recursive Fibonacci algorithm is inefficient, because the subproblems in the recursive Fibonacci algorithm overlaps, which causes redundant work. The non- recursive Fibonacci algorithm is dynamic algorithm that avoids redundant work.

Hide Answer

Section 22.6

▼22.6.1

Prove that the following algorithm for finding the GCD of the two integers m and n is incorrect.

```
int gcd = 1;
for (int k = Math.min(Math.sqrt(n), Math.sqrt(m)); k >= 1; k--) {
  if (m % k == 0 && n % k == 0) {
    gcd = k;
    break;
  }
}
```

To prove this is wrong, all you need is to give a counter example to show the algorithm does not work. Try n = 64 and m = 48. The algorithm will produce the gcd 8, but the actual gcd is 16.

Hide Answer

Section 22.7

▼22.7.1

Prove that if n is not prime, there must exist a prime number p such that p <= sqrt(n) and p is a factor of n.

If n is not a prime, then there exists two number n1 and n2 such that n1 * n2 = n. Assume n1 <= n2, n1 <= srqt(n). If n1 is not a prime, you can continue the same process to find the factors of n1, until a factor is a prime.

Hide Answer

▼22.7.2

Describe how the sieve of Eratosthenes is used to find the prime numbers.

See the text.

Hide Answer

Section 22.8

▼22.8.1

What is the divide-and-conquer approach? Give an example.

See the definition and example in the text.

Hide Answer

▼ **22.8.2**

What is the difference between divide-and-conquer and dynamic programming?

See the text.

Hide Answer

▼ **22.8.3**

Can you design an algorithm for finding the minimum element in a list using divide-and-conquer? What is the complexity of this algorithm?

Yes. Finding the minimum in the first half and the second half of the list and return the minimum of these two. So, the time complexity is O(n) = 2 * O(n/2) + O(1) = O(n).

Hide Answer

Section 22.9

▼ **22.9.1**

What is backtracking? Give an example.

See the definition and example in the text.

Hide Answer

▼ **22.9.2**

If you generalize the Eight Queens problem to the n-Queens problem in an n-by-n chessboard, what will be the complexity of the algorithm?

O(n!)

Hide Answer

Section 22.10

▼ **22.10.1**

What is a convex hull?

See the text.

Hide Answer

▼ **22.10.2**

Describe the gift-wrapping algorithm for finding a convex hull. Should list H be implemented using an ArrayList or a LinkedList?

See the text.

Hide Answer

▼22.10.3

Describe Graham's algorithm for finding a convex hull. Why does the algorithm use a stack to store the points in a convex hull?

See the text.

Hide Answer