Multiple Choice Questions Related To Testing Knowledge about Time and Space Complexity Of A Program

■ tutorial

admin Jan '18

Hi fellow programmers,

We are trying to create a multiple choice quiz for space and time complexity of the programs related questions. Here are a set of 20 questions we collected. Please feel free to give your answers to these questions. Any feedback about the set of questions. Please also feel propose to any more set of MCQs that you would like to add here, there might be some interesting questions that you might have encountered during the programming and would like to add here 🙂

Q1.

Average case time complexity of quicksort?

A. $\mathcal{O}(n)$
B. $\mathcal{O}(n \log n)$
C. $\mathcal{O}(n^2)$
D. $\mathcal{O}(n^3)$

(B) $O(n\log n)$

Q2.

Worst case time complexity of quicksort?

A. $\mathcal{O}(n)$
B. $\mathcal{O}(n \log n)$
C. $\mathcal{O}(n^2)$
D. $\mathcal{O}(n^3)$

(C) $O(n^2)$

Q3.

Time complexity of binary search?

A. $\mathcal{O}(1)$
B. $\mathcal{O}(\log n)$
C. $\mathcal{O}((\log n)^2)$

(B) $O(\log n)$

Answer Key

(1.)	B
(2)	C
(3)	B
(4)	B
(5)	C
(6)	D
(7)	C
(8)	C
(9)	C
(10)	B
(11)	A
(12)	C
(13)	C
(14)	B
(15)	B
(16)	B
(17)	C
(18)	D
(19)	B
(20)	B
(21)	C

D. $\mathcal{O}(n)$

Q4.

```
def f()
    ans = 0
    for i = 1 to n:
        for j = 1 to log(i):
            ans += 1
    print(ans)
```

Time Complexity of this program:

A. $\mathcal{O}(n)$
B. $\mathcal{O}(n \log n)$
C. $\mathcal{O}(n^2)$
D. $\mathcal{O}(n^3)$

(B) $O(n\log n)$

Q5.

```
def f():
    a = 0
    for i = 1 to n:
        a += i;
    b = 0
    for i = 1 to m:
        b += i;
```

(C) $O(n+m)$

Time Complexity of this program:

A. $\mathcal{O}(n)$
B. $\mathcal{O}(m)$
C. $\mathcal{O}(n+m)$
D. $\mathcal{O}(n*m)$

Q6.

```
def f():
    a = 0
    for i = 1 to n:
        a += random.randint();
        b = 0
        for j = 1 to m:
            b += random.randint();
```

Time Complexity of this program:

(D) $o(n * m)$

A. $\mathcal{O}(n)$
B. $\mathcal{O}(m)$
C. $\mathcal{O}(n+m)$
D. $\mathcal{O}(n*m)$

Q7.

```
def f():
    int a[n][n]
    // Finding sum of elements of a matrix
    sum = 0
    for i = 1 to n:
        for j = i to n:
            sum += a[i][j]
    print(sum)
```

(C) $o(n^2)$

Time Complexity of this program:

A. $\mathcal{O}(n)$
B. $\mathcal{O}(n \log n)$
C. $\mathcal{O}(n^2)$
D. $\mathcal{O}(n^3)$

Q8.

(C) $o(n^2)$

```
def f():
    int a[n][n]
    sum = 0
    // Finding sum of elements of a matrix
    for i = 1 to n:
        for j = i to n:
            sum += a[i][j]
    print(sum)

    for i = 1 to n:
        sum -= a[i][i]
```

Time Complexity of this program:

A. $\mathcal{O}(n)$
B. $\mathcal{O}(n \log n)$
C. $\mathcal{O}(n^2)$
D. $\mathcal{O}(n^3)$

Q9.

```
def f():
    ans = 0
    for i = 1 to n:
        for j = n to i:
            ans += (i * j)
    print(ans)
```

Time Complexity of this program:

A. $\mathcal{O}(n)$
B. $\mathcal{O}(n \log n)$
C. $\mathcal{O}(n^2)$
D. $\mathcal{O}(n^3)$

(C) $O(n^2)$

Q10.

```
def f():
    int a[N + 1][M + 1][K + 1]
    sum = 0
    for i = 1 to N:
        for j = i to M:
            for k = j to K:
                sum += a[i][j]
    print(sum)
```

Time Complexity of this program:

A. $\mathcal{O}(N + M + K)$
B. $\mathcal{O}(N * M * K)$
C. $\mathcal{O}(N * M + K)$
D. $\mathcal{O}(N + M * K)$

(B) $O(N * M * K)$

Q11.

```
def f(n):
    ans = 0
    while (n > 0):
        ans += n
        n /= 2;
    print(ans)
```

(A) $O(\log n)$

Time Complexity of this program:

A. $\mathcal{O}(\log n)$

B. $\mathcal{O}(n)$
B. $\mathcal{O}(n \log n)$
C. $\mathcal{O}(n^2)$

Q12.

```
// Find the sum of digits of a number in it
def f(n):
    ans = 0
    while (n > 0):
        ans += n % 10
        n /= 10;
    print(ans)
```

(C) $O(\log_{10} n)$

Time Complexity of this program:

A. $\mathcal{O}(\log_2 n)$
B. $\mathcal{O}(\log_3 n)$
C. $\mathcal{O}(\log_{10} n)$
D. $\mathcal{O}(n)$

Q13.

```
def f():
    ans = 0
    for (i = n; i >= 1; i /= 2):
        for j = m to i:
            ans += (i * j)
    print(ans)
```

(C) $O(m \log n)$

Time Complexity of this program:

A. $\mathcal{O}(n+m)$
B. $\mathcal{O}(n*m)$
C. $\mathcal{O}(m \log n)$
D. $\mathcal{O}(n \log m)$

Q14.

```
def f():
    ans = 0
    for (i = n; i >= 1; i /= 2):
        for (j = 1; j <= m; j *= 2)
            ans += (i * j)
```

(B) $O(\log m \log n)$

```
    print(ans)
```

Time Complexity of this program:

A. $\mathcal{O}(n * m)$

[✓ B. $\mathcal{O}(\log m \log n)$]

C. $\mathcal{O}(m \log n)$

D. $\mathcal{O}(n \log m)$

Q15.

```
// Finding gcd of two numbers a, b.
def gcd(a, b):
    if (a < b) swap(a, b)
    if (b == 0) return a;
    else return gcd(b, a % b)
```

(B) O(log n)

Time Complexity of this program:

Let $n = \max\{a, b\}$

A. $\mathcal{O}(1)$

B. $\mathcal{O}(\log n)$

C. $\mathcal{O}(n)$

D. $\mathcal{O}(n^2$

Q16.

```
// Binary searching in sorted array for fin
def exists(a, x):
    // Check whether the number x exists in
    lo = 0, hi = len(a) - 1
    while (lo <= hi):
        mid = (lo + hi) / 2
        if (a[mid] == x): return x;
        else if (a[mid] > x): hi = mid - 1;
        else lo = mid + 1;
    return -1; // Not found.
```

(B) O(log n)

Time Complexity of this program:

Let $n = len(a)$

A. $\mathcal{O}(1)$

B. $\mathcal{O}(\log n)$

C. $\mathcal{O}(n)$

D. $\mathcal{O}(n^2$

Q17.

// Given a sorted array a, find the number of occurrence of number x in the entire array.

(C) O(n)

```
def count_occurrences(a, x, lo, hi):
    if lo > hi: return 0
    mid = (lo + hi) / 2;
    if a[mid] < x: return count_occurrences
    if a[mid] > x: return count_occurrences
    return 1 + count_occurrences(a, x, lo,

// in the main function, we call it as
count_occurrences(a, x, 0, len(a) - 1)
```

Time Complexity of this program:

Let $n = len(a)$

A. $\mathcal{O}(1)$
B. $\mathcal{O}(\log n)$
C. $\mathcal{O}(n)$
D. $\mathcal{O}(n^2$

Q18.

(D) O(2^n)

```
// Finding fibonacci numbers.
def f(n):
    if n == 0 or n == 1: return 1
    return f(n - 1) + f(n - 2)
```

Time Complexity of this program:

A. $\mathcal{O}(\log n)$
B. $\mathcal{O}(n)$
C. $\mathcal{O}(n^2)$
D. $\mathcal{O}(2^n)$

Q19.

```
Create array memo[n + 1]

// Finding fibonacci numbers with memoizati
```

```
def f(n):
    if memo[n] != -1: return memo[n]
    if n == 0 or n == 1: ans = 1
    else: ans = f(n - 1) + f(n - 2)
    memo[n] = ans
    return ans

// In the main function.
Fill the memo array with all values equal t
ans = f(n)
```

Time Complexity of this program:

(B) $O(n)$

A. $\mathcal{O}(\log n)$
B. $\mathcal{O}(n)$
C. $\mathcal{O}(n^2)$
D. $\mathcal{O}(2^n)$

Q20.

```
def f(a):
    n = len(a)
    j = 0
    for i = 0 to n - 1:
        while (j < n and a[i] < a[j]):
            j += 1
```

(B) $O(n)$

Time Complexity of this program:

A. $\mathcal{O}(\log n)$
B. $\mathcal{O}(n)$
C. $\mathcal{O}(n \log n)$
D. $\mathcal{O}(n^2)$

Q21.

```
def f():
    ans = 0
    for i = 1 to n:
        for j = i; j <= n; j += i:
            ans += 1
    print(ans)
```

(C) $O(n\log n)$

Time Complexity of this program:

A. $\mathcal{O}(\log n)$ B. $O(n)$ C. $O(n\log n)$ D. $O(n^2)$