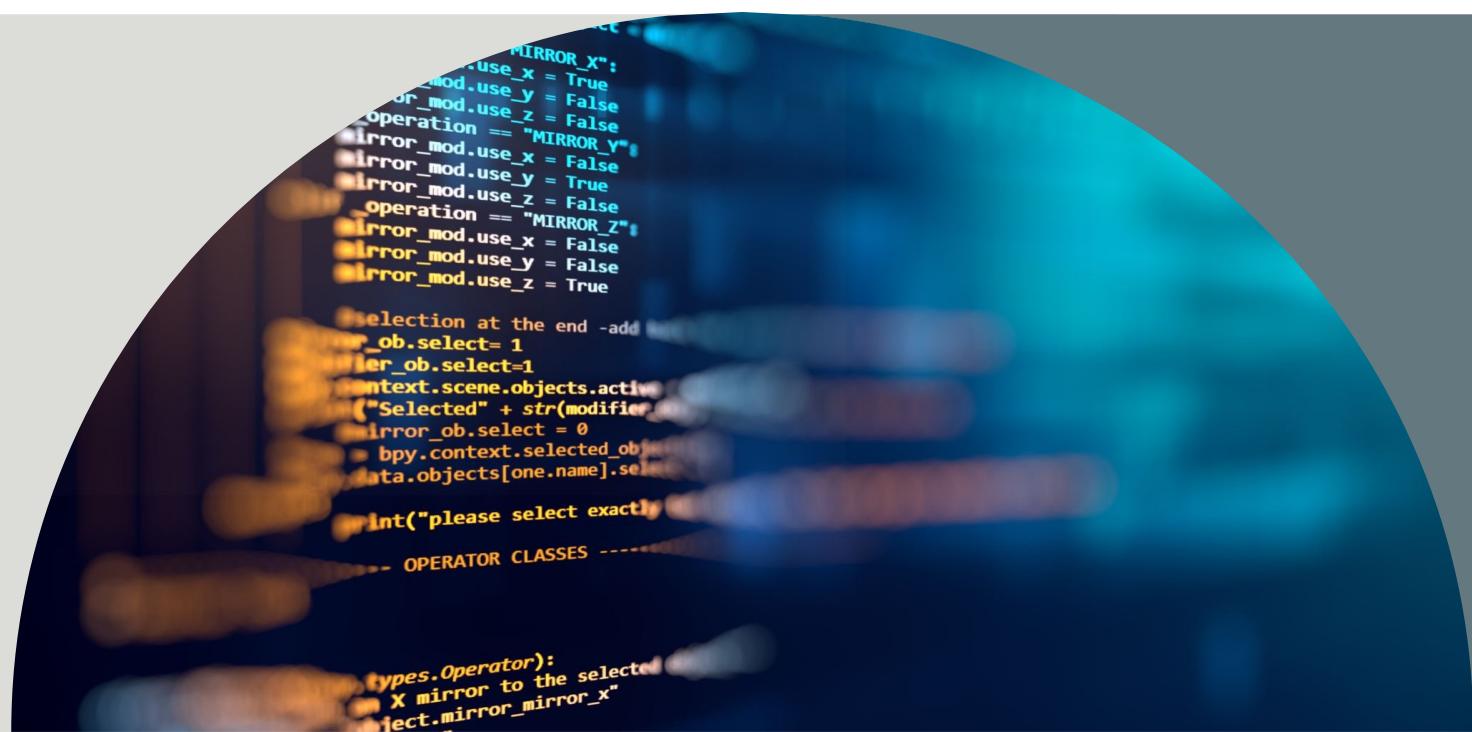
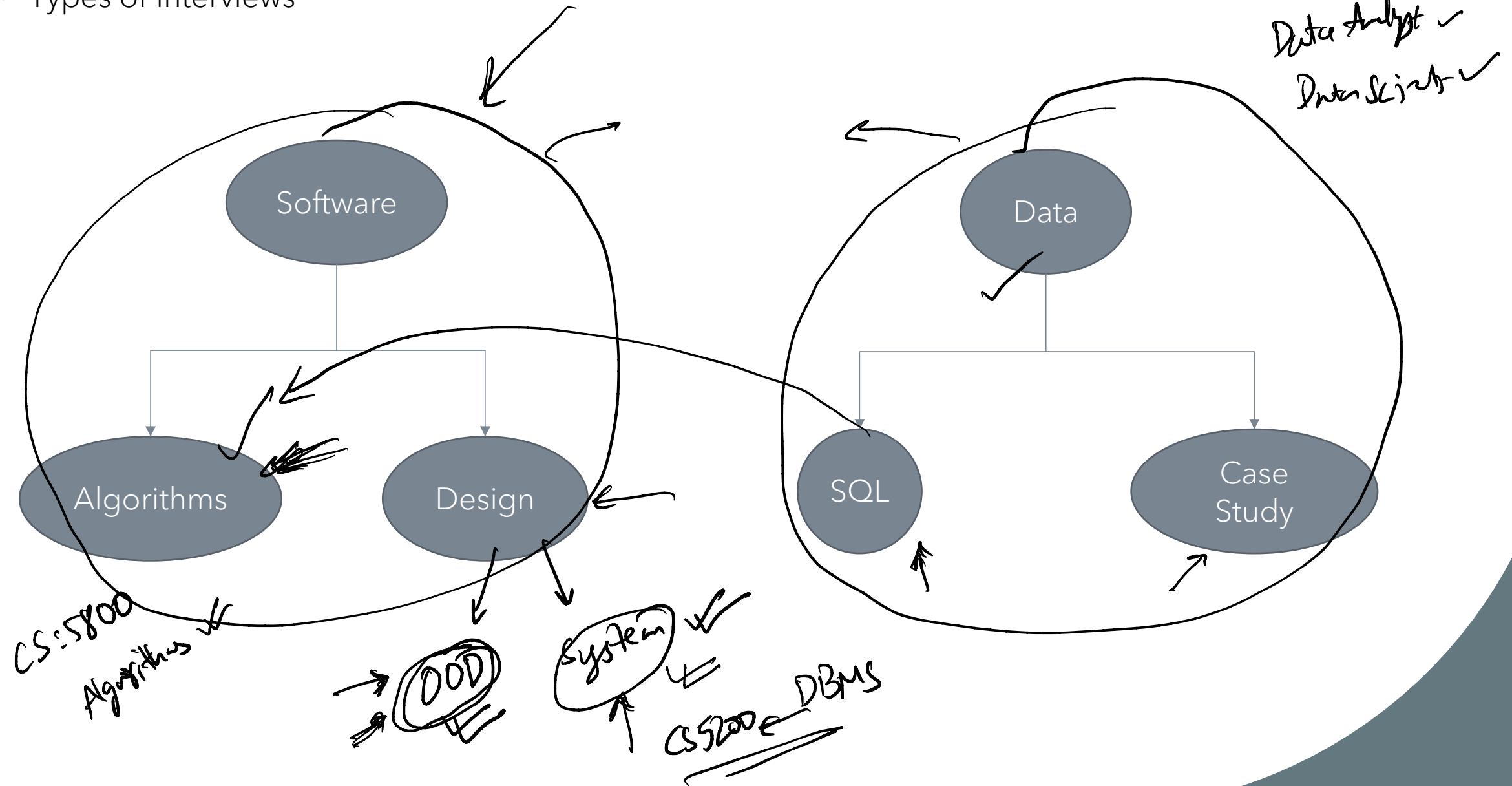


# DATA CLUB - MEET I

Topic: Complexity



- Types of interviews



- Types of job roles (for Applied Mathematicians)

→ 1. Research Engineer (Algorithms, Distributed Systems)

→ 2. Machine Learning Engineer (Algorithms, OOPS, Distributed Systems, ML System Design)

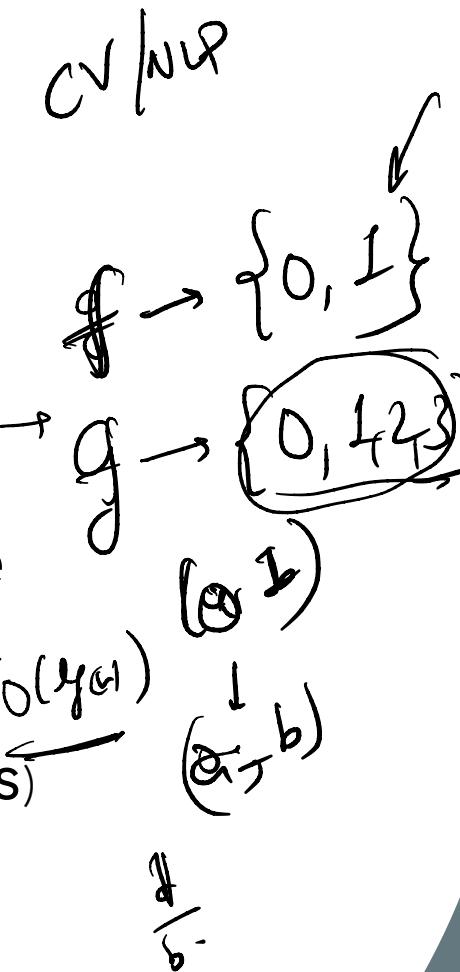
→ 3. Quantitative Researcher (Algorithms, Mathematics, Case Studies)  
 $a = [0, 12, 2, 3, 3, 3, 4, 5]; \quad O(n^2) = O(n^2)$

→ 4. Software Engineers (Algorithms, OOPS, Distributed Systems)

→ 5. Data Scientist (Algorithms, SQL Case Studies)

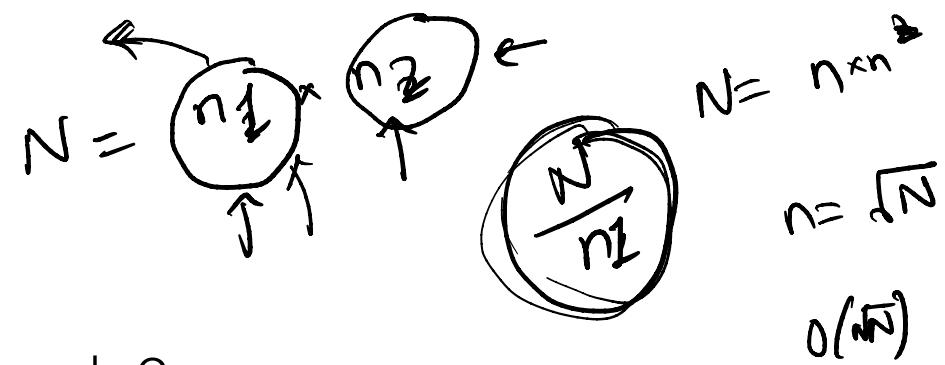
→ 6. Data Analyst (Algorithms, SQL Case Studies)

$S1 = \{ \dots \}$   
 $S2 = \{ \dots \}$   
 $\text{return } S1 \cdot \text{intersection}(S2)$





1. Top-down learning
2. Learn by examples
3. More practice
4. Interactive
5. NOT EASY



Let's assume we ask 2 interviewees A and B to write a program to detect if a number  $N \geq 2$  is prime.

<i>A</i>	<i>B</i>
<pre>i = 2 while i &lt; N     if N is divisible by i         N is not prime     add 1 to i</pre>	<pre>i = 2 while i &lt;= square root of N     if N is divisible by i         N is not prime     add 1 to i</pre>
$\checkmark$ $\uparrow$ $O(n)$	$\checkmark$ $\uparrow$ $O(\sqrt{n})$

Let's assume that the operation N is divisible by i takes 1 ms.

$N = \underline{1000033}$  ( Prime number )

Time taken by A's program = 1 ms \* number of divisions

= 1 ms \* 1000033

= approximately 1000 seconds or 16.7 mins.

$O(n)$

Time taken by B's program = 1ms \* number of divisions

= 1ms \* square root of 1000033

= approximately 1000ms = 1 second.

$N = \underline{1500450271}$  ( Prime number )

Time taken by A's program = 1 ms \* number of divisions

= 1 ms \* 1500450271

= approximately 1000000 seconds or 11.5 days

Time taken by B's program = 1ms \* number of divisions

= 1ms \* square root of 1500450271

= approximately 40000ms = 40 seconds.

- Computer scientists have developed a convenient notation for hiding the constant factor.
- We write  $O(n)$  (read: "order  $n$ ") instead of " $cn$  for some constant  $c$ ."
- Thus, an algorithm is said to be  $O(n)$  or *linear time* if there is a fixed constant  $c$  such that for all sufficiently large  $n$ , the algorithm takes time at most  $cn$  on inputs of size  $n$ . An algorithm is said to be  $O(n^2)$  or *quadratic time* if there is a fixed constant  $c$  such that for all sufficiently large  $n$ , the algorithm takes time at most  $cn^2$  on inputs of size  $n$ .
- $O(1)$  means *constant time*.
- *Polynomial time* means  $n^{O(1)}$ , or  $n^c$  for some constant  $c$ . Thus, any constant, linear, quadratic, or cubic ( $O(n^3)$ ) time algorithm is a polynomial-time algorithm.
- One important advantage of big-O notation is that it makes algorithms much easier to analyze, since we can conveniently ignore low-order terms. For example, an algorithm that runs in time

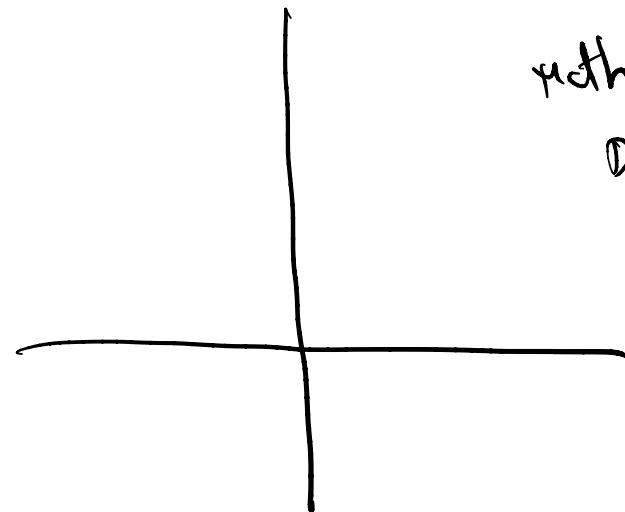
$10n^3 + 24n^2 + 3n \log n + 144$  is still a cubic algorithm, since

$$\begin{aligned}
 & 10n^3 + 24n^2 + 3n \log n + 144 \\
 & \leq 10n^3 + 24n^3 + 3n^3 + 144n^3 \\
 & \leq (10 + 24 + 3 + 144)n^3 \\
 & = O(n^3).
 \end{aligned}$$

Donald Knuth

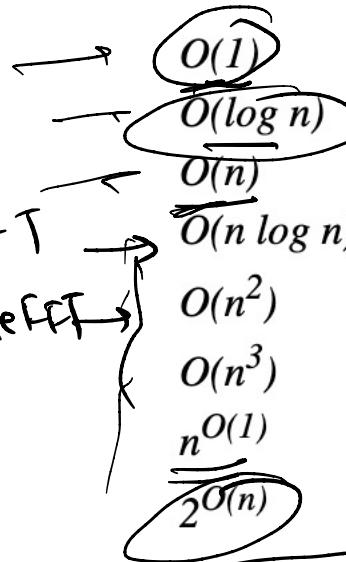
Some common orders of growth seen often in complexity analysis are,

$$f(n) = f(n-1) + f(n-2)$$



with FFT

Discrete FFT



constant

logarithmic

linear

" $n \log n$ "

quadratic

cubic

polynomial

exponential

min heap

OK)

Find minimum in min-heap

search for  $n$  elts in sorted array

search for elts in unsorted array

Merge sort

Insertion sort

Bubble sort

Fibonacci recursive

1)

What is the time, space complexity of following code :

```
int a = 0, b = 0;
for (i = 0; i < N; i++) {
    a = a + rand();
}
for (j = 0; j < M; j++) {
    b = b + rand();
}
```

Assume that `rand()` is  $O(1)$  time,  $O(1)$  space function.

Select your answer from the following options:

- $O(N * M)$  time,  $O(1)$  space
- $O(N + M)$  time,  $O(N + M)$  space
- $O(N + M)$  time,  $O(1)$  space
- $O(N * M)$  time,  $O(N + M)$  space
- $O(N * M)$  time,  $O(N * M)$  space

2)

What is the time, space complexity of following code :

```
int a = 0, b = 0;
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        a = a + j;
    }
}
for (k = 0; k < N; k++) {
    b = b + k;
}
```

**Select your answer from the following options:**

- O( $N * N$ ) time, O(1) space
- O( $N$ ) time, O( $N$ ) space
- O( $N$ ) time, O( $N$ ) space
- O( $N * N$ ) time, O( $N$ ) space
- O( $N * N * N$ ) time, O(1) space

3) What does it mean when we say that an algorithm X is asymptotically more efficient than Y?

- a) X will always be a better choice for all inputs
- b) X will always be a better choice for large inputs
- c) X will always be a better choice for small inputs
- d) Y will always be a better choice for small inputs

4)

What is the time complexity of the following code :

```
int a = 0, i = N;  
while (i > 0) {  
    a += i;  
    i /= 2;  
}
```

Select your answer from the following options:

- O(N)
- O(Sqrt(N))
- O(N / 2)
- O(log N)
- O(log(log N))

5)

What is time complexity of following code :

```
int count = 0;
for (int i = N; i > 0; i /= 2) {
    for (int j = 0; j < i; j++) {
        count += 1;
    }
}
```

Select your answer from the following options:

- $O(N * N)$
- $O(N * \log N)$
- $O(N * \log(\log(N)))$
- $O(N)$
- $O(N * \text{Sqrt}(N))$

6)

What is the time complexity of the following code:

```
int i, j, k = 0;
for (i = n/2; i <= n; i++) {
    for (j = 2; j <= n; j = j * 2) {
        k = k + n/2;
    }
}
```

Select your answer from the following options:

- O(n)
- O(nLogn)
- O(n^2)
- O(n^2 / Logn)
- O(n^2Logn)

7) Which of the following is not bounded by  $O(n^2)$ ?

- $(15^{10}) * n + 12099$
- $n^{1.98}$
- $n^3 / (\sqrt{n})$
- $(2^{20}) * n$

Select your answer from the following options:

8) Which of the given options provides the increasing order of complexity of functions  $f_1, f_2, f_3$  and  $f_4$ :

$$f_1(n) = 2^n$$

$$f_2(n) = n^{(3/2)}$$

$$f_3(n) = n \log n$$

$$f_4(n) = n^{\log n}$$

Select your answer from the following options:

- $f_3, f_2, f_4, f_1$
- $f_3, f_2, f_1, f_4$
- $f_2, f_3, f_1, f_4$
- $f_2, f_3, f_4, f_1$

9)

In a competition, four different functions are observed. All the functions use a single for loop and within the for loop, same set of statements are executed.

Consider the following for loops:

- A) `for(i = 0; i < n; i++)`
- B) `for(i = 0; i < n; i += 2)`
- C) `for(i = 1; i < n; i *= 2)`
- D) `for(i = n; i > -1; i /= 2)`

If n is the size of input(positive), which function is the most efficient? In other words, which loop completes the fastest.

Select your answer from the following options:

- A
- B
- C
- D

10)

Select your answer from the following options:

O(sqrt N)

O(log N)

O(log^2 N)

What is the worst case time complexity of the following code :

O(N)

/\*

O(N \* log N)

\* V is sorted  
\* V.size() = N

O(N \* sqrt N)

\* The function is initially called as searchNumOccurrence(V, k, 0, N-1)  
\*/

```
int searchNumOccurrence(vector<int> &V, int k, int start, int end) {  
    if (start > end) return 0;  
    int mid = (start + end) / 2;  
    if (V[mid] < k) return searchNumOccurrence(V, k, mid + 1, end);  
    if (V[mid] > k) return searchNumOccurrence(V, k, start, mid - 1);  
    return searchNumOccurrence(V, k, start, mid - 1) + 1 + searchNumOccurrence(V, k, mid + 1, end);  
}
```

11)

Select your answer from the following options:

- O( $2^{(R+C)}$ )
- O( $R \cdot C$ )
- O( $R + C$ )
- O( $R^2 + C^2$ )
- O( $R \cdot C \cdot \log(R \cdot C)$ )

What is the worst case time complexity of the following code:

```
int findMinPath(vector<vector<int>> &V, int r, int c) {  
    int R = V.size();  
    int C = V[0].size();  
    if (r >= R || c >= C) return 100000000; // Infinity  
    if (r == R - 1 && c == C - 1) return 0;  
    return V[r][c] + min(findMinPath(V, r + 1, c), findMinPath(V, r, c + 1));  
}
```

Assume R = V.size() and C = V[0].size().

12)

## Select your answer from the following options:

O( $2^{R+C}$ )

What is the worst case time complexity of the following code:

O( $R \cdot C$ )

```
int memo[101][101];
int findMinPath(vector<vector<int> >& V, int r, int c) {
    int R = V.size();
    int C = V[0].size();
    if (r >= R || c >= C) return 100000000; // Infinity
    if (r == R - 1 && c == C - 1) return 0;
    if (memo[r][c] != -1) return memo[r][c];
    memo[r][c] = V[r][c] + min(findMinPath(V, r + 1, c), findMinPath(V, r, c + 1));
    return memo[r][c];
}
```

O( $R + C$ )

O( $R^2 + C^2$ )

O( $R \cdot C \cdot \log(R \cdot C)$ )

Callsite :  
memset(memo, -1, sizeof(memo));  
findMinPath(V, 0, 0);

Assume R = V.size() and C = V[0].size() and V has positive elements

13)

Select your answer from the following options:

- O(n)
- O(n^2)
- O(nlogn)
- O(n(logn)^2)
- Can't say. Depends on the value of arr.

What is the time complexity of the following code :

```
int j = 0;
for(int i = 0; i < n; ++i) {
    while(j < n && arr[i] < arr[j]) {
        j++;
    }
}
```