Common interview topics:

Data structure:

Linked List

Trees

Binary tree: also referred to as height-balanced binary tree.

Graphs

Stack & Queue (could be implemented by array or linked-list)

Heaps: min-heap and max-heap

Implemented by an array

HeapifyUp or heapifyDown

Used to implement priority queue

Vector / ArrayLists

Array which can grow, each grows need copy the data.

HashTable

Resolve hash collision. Allow storing more items using array or sorted binary tree.

Algorithm:

Breadth-First Search:

1 **procedure** BFS(*G*, *root*) **is**

2 let *Q* be a queue

3 label *root* as discovered

4 *Q*.enqueue(*root*)

5 **while** *Q* is not empty **do**

6 *v* := *Q*.dequeue()

7 **if** *v* is the goal **then**

8 **return** *v*

9 **for all** edges from *v* to *w* **in** *G*.adjacentEdges(*v*) **do**

10 **if** *w* is not labeled as discovered **then**

11 label *w* as discovered

12 *w*.parent := *v*

13 *Q*.enqueue(*w*)

Depth-First Search

**procedure** DFS\_iterative(*G*, *v*) **is**

let *S* be a stack

*S*.push(*v*)

**while** *S* is not empty **do**

*v* = *S*.pop()

**if** *v* is not labeled as discovered **then**

label *v* as discovered

**for all** edges from *v* to *w* **in** *G*.adjacentEdges(*v*) **do**

*S*.push(*w*)

Binary Search

Merge Sort

Quick Sort

**algorithm** quicksort(A, lo, hi) **is**

**if** lo < hi **then**

p := partition(A, lo, hi)

quicksort(A, lo, p - 1)

quicksort(A, p + 1, hi)

**algorithm** partition(A, lo, hi) **is**

pivot := A[hi]

i := lo

**for** j := lo **to** hi **do**

**if** A[j] < pivot **then**

swap A[i] with A[j]

i := i + 1

swap A[i] with A[hi]

**return** i

Dijkstra’s:

1 **function** Dijkstra(*Graph*, *source*):

2

3 create vertex set Q

4

5 **for each** vertex *v* in *Graph*:

6 dist[*v*] ← INFINITY

7 prev[*v*] ← UNDEFINED

8 add *v* to *Q*

10 dist[*source*] ← 0

11

12 **while** *Q* is not empty:

13 *u* ← vertex in *Q* with min dist[u]

14

15 remove *u* from *Q*

16

17 **for each** neighbor *v* of *u*: *// only v that are still in Q*

18 *alt* ← dist[*u*] + length(*u*, *v*)

19 **if** *alt* < dist[*v*]:

20 dist[*v*] ← *alt*

21 prev[*v*] ← *u*

22

23 **return** dist[], prev[]

Concepts:

Bit Manipulation

Memory (Stack vs Heap)

1. stack: quick to allocate, automatically to free. Limited by size
2. heap: allow big allocation, can cause fragmentation. Slow to allocate.

Recursion

Dynamic Programming

Big O Time & Space

Computer Architecture:

Caching, cachelines.

CPU pipeline

GPU pipeline

Dynamic Programming:

1) "careful brute force"

2) guessing + recursion + memorization

3) shortest path in some DAG

time = #subproblems \* time/subproblem

My own experience:

1. Implement Motion matching:
   1. Feature vectors generation
   2. Kmean clustering to reduce search space.
2. Owning weapon, camera system.
3. Providing IK solutions (need review)
   1. Creating jacobian matrix: each joint may have 3 degrees of freedom.
   2. Calculate the derivative of end-vector to joint.
   3. An iterative process.
   4. Apply joint limit and effort by how hard to reach it.
4. Implement mesh ray casters which utilize GPU.
   1. Need kd-tree to partition the scene
   2. Need solution for animated character.

Questions for interviewer:

1) What's your job in the company.

2) How does your team/studio collaborate with other teams/studios.

3) Is there any risk joining Tencent due to bad Sino-America relationship?

4) Can I change studio internally?