[**When is it practical to use DFS vs BFS?**](http://stackoverflow.com/questions/3332947/when-is-it-practical-to-use-dfs-vs-bfs)

That heavily depends on the structure of the tree and the number of elements in it. If you know a solution is not far from the root of the tree, a breadth first search (BFS) might be better. If the tree is very wide, a BFS might need too much memory, so it might be impractical. If solutions are frequent but located deep in the tree, DFS might be better.

Breadth First Search is generally the best approach when the depth of the tree can vary, and you only need to search part of the tree for a solution. For example, finding the shortest path from a starting value to a final value is a good place to use BFS.

Depth First Search is commonly used when you need to search the entire tree. It's easier to implement than BFS.

**What is RESTful programming?**

REST: Representational State Transfer

It’s an architectural style that specifies constraints such as the uniform interface. The data and functionalities are accessed using URIs – typically the links on the web – which contain the information about the operations to perform. Resources are manipulated using create, read, update, and delete operations in HTTP. Clients and servers exchange representations of resources by using a standardized interface and protocol.

**What are the differences between GET and POST?**

Get Requests

* Request data from a specified resource.
* Remains in browser history.
* Can be bookmarked.
* Should never be used when dealing with sensitive data.
* Have length restrictions.

Post Requests

* Submits data to be processed.
* Never cached.
* Cannot be bookmarked.
* Do not have restrictions on data length.
* Safer than GET because parameters are not stored in browser history.

**What is SOA?**

An architecture style that principles in independence of products and technologies. It logically represents a business activity with a specified outcome. It provides business outcomes to the customers while acting as a black box to them abstracting away the underlying complexity of what’s within the system. Breakdown service components into blocks and construct service units by flexibly assembling all the required pieces.

**What is MSA?**

MSA is the opposite of monolithic architecture. A monolith architecture is always built as a single unit. The problem with it is that all business processes end up being tied to one another that even a modification on a small section of the application might result in deploying the entire unit. This impacts the system’s scalability as well. Even if you need to scale specific functions in an app, monolithic architecture might result in scaling the entire app. MSA was introduced to solve such problems. By diving specific components into modules, we can readily scale only the parts of the services desired in the application.

Q) How would you design the news feed feature in Facebook?

public interface Iterator<E> {

/\*\*

\*

\* @return {@code true} if the iteration has more elements

\*/

boolean hasNext();

/\*\*

\* @return the next element in the iteration

\* @throws NoSuchElementException if the iteration has no more elements

\*/

E next();

}

Write an iterator that wraps a list of iterators, e.g. [1, 2, 3, 4], [5, 6, 7, 8], [9] and outputs their elements one after the other (in a round robin fashion), e.g. [1, 5, 9, 2, 6, 3, 7, 4, 8]

public class Test {

List<Iterator> iters;

int cnt;

public Test(List<Iterator<E>> iterators) {

iters = iterators;

}

public boolean hasNext() {

var i = cnt % iters.Count;

if (iters[i].hasNext()) return true;

return false;

}

public E next() {

var i = cnt++ % iters.Count;

if (iters[i].hasNext()) {

var result = iters[i].next();

if (iters[i].hasNext() == false) {

}

return result;

}

return null;

}

}

**-- Question 2**

You are given two operations.

Op1: insert an integer element

Op2: Get the median --> The number in the very middle when everything is sorted.

How would you implement the two operations?

Op1: Just insert everything at the end of the array. Complexity: O(1)

Op2: Sort the entire array then get the middle index. Complexity: O(n log n)

Follow Up I:

Let's say you are using quick sort to implement Op2.

How does quick sort work? --> by partitioning.

When we use quick sort, do we need to look at the entire array everytime?

For example, when there are 10 elements and the partioning returned back index 7,

we don't need to look at the last three elements because we already know they are not the medians.

Using this algorithm, we would need to look on average n elements in the first partitioning,

n/2 elements in the second partitioning, n/4 elements in the third partitioning, and so on.

Using geometric series, 1 + 1/2 + 1/4 + ... approaches 2.

Therefore, n + n/2 + n/4 + ... approaches 2n, which results in the complexity of O(2n).

Follow Up II:

How would you implement the two operations so that getting the median results in O(1)?

Ans) We could implement the algorithm by this time, sorting the array at every insertion operation,

and getting the median by looking up the element in the middle index.

Q) What's the time complexity of insertion in this case?

- We are using sorted array here. Searching for the location to insert an element takes O(log n).

Once we have found the location, we need to shift the remaining elements to the back to make space for the element.

This takes O(n/2) in average. Therefore, the time complexity for inserting and sorting the elements is O(n/2).

Follow Up III:

How would you implement the operations using a binary search tree?

What happens when 1, 2, 3, 4, ... are added to the tree? The tree deforms into a linked-list like structure.

You would need to balanace the tree when this happens. When and how would you do it?

**-- Question 2**

public int countValues(TreeNode root, int min, int max) {

if (root == null) return 0;

if (root.val < min) return countValues(root.right, min, max);

if (root.val > max) return countValues(root.left, min, max);

if (root.val == min) return 1 + countValues(root.right, min, max);

if (root.val == max) return 1 + countValues(root.left, min, max);

return 1 + countValues(root.left, min, max) + countValues(root.right, min, max);

}