TSiege / **The Technical Interview Cheat Sheet.md**

Last active 5 hours ago • Report gist

This is my technical interview cheat sheet. Feel free to fork it or do whatever you want with it. PLEASE let me know if there are any errors or if anything crucial is missing. I will add more links soon.

`The Technical Interview Cheat Sheet.md`

Studying for a Tech Interview Sucks, so Here's a Cheat Sheet to Help

This list is meant to be a both a quick guide and reference for further research into these topics. It's basically a summary of that comp sci course you never took or forgot about, so there's no way it can cover everything in depth. It also will be available as a gist on Github for everyone to edit and add to.

Data Structure Basics

###**Array** ####Definition:

- Stores data elements based on an sequential, most commonly 0 based, index.
- Based on tuples from set theory.
- They are one of the oldest, most commonly used data structures.

####What you need to know:

- Optimal for indexing; bad at searching, inserting, and deleting (except at the end).
- **Linear arrays**, or one dimensional arrays, are the most basic.
  - Are static in size, meaning that they are declared with a fixed size.
- **Dynamic arrays** are like one dimensional arrays, but have reserved space for additional elements.
  - If a dynamic array is full, it copies it's contents to a larger array.
- **Two dimensional arrays** have x and y indices like a grid or nested arrays.

####Big O efficiency:

- Indexing: Linear array: O(1), Dynamic array: O(1)
- Search: Linear array: O(n), Dynamic array: O(n)
- Optimized Search: Linear array: O(log n), Dynamic array: O(log n)
- Insertion: Linear array: n/a Dynamic array: O(n)

###**Linked List** ####Definition:

- Stores data with **nodes** that point to other nodes.
  - Nodes, at its most basic it has one datum and one reference (another node).
  - A linked list *chains* nodes together by pointing one node's reference towards another node.

####What you need to know:

- Designed to optimize insertion and deletion, slow at indexing and searching.
- **Doubly linked list** has nodes that reference the previous node.
- **Circularly linked list** is simple linked list whose **tail**, the last node, references the **head**, the first node.
- **Stack**, commonly implemented with linked lists but can be made from arrays too.
  - Stacks are **last in, first out** (LIFO) data structures.
  - Made with a linked list by having the head be the only place for insertion and removal.

- **Queues**, too can be implemented with a linked list or an array.
  - Queues are a **first in, first out** (FIFO) data structure.
  - Made with a doubly linked list that only removes from head and adds to tail.

####Big O efficiency:

- Indexing: Linked Lists: O(n)
- Search: Linked Lists: O(n)
- Optimized Search: Linked Lists: O(n)
- Insertion: Linked Lists: O(1)

###**Hash Table or Hash Map** ####Definition:

- Stores data with key value pairs.
- **Hash functions** accept a key and return an output unique only to that specific key.
  - This is known as **hashing**, which is the concept that an input and an output have a one-to-one correspondence to map information.
  - Hash functions return a unique address in memory for that data.

####What you need to know:

- Designed to optimize searching, insertion, and deletion.
- **Hash collisions** are when a hash function returns the same output for two distinct inputs.
  - All hash functions have this problem.
  - This is often accommodated for by having the hash tables be very large.
- Hashes are important for associative arrays and database indexing.

####Big O efficiency:

- Indexing: Hash Tables: O(1)
- Search: Hash Tables: O(1)
- Insertion: Hash Tables: O(1)

###**Binary Tree** ####Definition:

- Is a tree like data structure where every node has at most two children.
  - There is one left and right child node.

####What you need to know:

- Designed to optimize searching and sorting.
- A **degenerate tree** is an unbalanced tree, which if entirely one-sided is a essentially a linked list.
- They are comparably simple to implement than other data structures.
- Used to make **binary search trees**.
  - A binary tree that uses comparable keys to assign which direction a child is.
  - Left child has a key smaller than it's parent node.
  - Right child has a key greater than it's parent node.
  - There can be no duplicate node.
  - Because of the above it is more likely to be used as a data structure than a binary tree.

####Big O efficiency:

- Indexing: Binary Search Tree: O(log n)
- Search: Binary Search Tree: O(log n)
- Insertion: Binary Search Tree: O(log n)

Search Basics

###**Breadth First Search** ####Definition:

- An algorithm that searches a tree (or graph) by searching levels of the tree first, starting at the root.
  - It finds every node on the same level, most often moving left to right.
  - While doing this it tracks the children nodes of the nodes on the current level.
  - When finished examining a level it moves to the left most node on the next level.
  - The bottom-right most node is evaluated last (the node that is deepest and is farthest right of it's level).

####What you need to know:

- Optimal for searching a tree that is wider than it is deep.
- Uses a queue to store information about the tree while it traverses a tree.
  - Because it uses a queue it is more memory intensive than **depth first search**.
  - The queue uses more memory because it needs to stores pointers

####Big O efficiency:

- Search: Breadth First Search: O(|E| + |V|)
- E is number of edges
- V is number of vertices

###**Depth First Search** ####Definition:

- An algorithm that searches a tree (or graph) by searching depth of the tree first, starting at the root.
  - It traverses left down a tree until it cannot go further.
  - Once it reaches the end of a branch it traverses back up trying the right child of nodes on that branch, and if possible left from the right children.
  - When finished examining a branch it moves to the node right of the root then tries to go left on all it's children until it reaches the bottom.
  - The right most node is evaluated last (the node that is right of all it's ancestors).

####What you need to know:

- Optimal for searching a tree that is deeper than it is wide.
- Uses a stack to push nodes onto.
  - Because a stack is LIFO it does not need to keep track of the nodes pointers and is therefore less memory intensive than breadth first search.
  - Once it cannot go further left it begins evaluating the stack.

####Big O efficiency:

- Search: Depth First Search: O(|E| + |V|)
- E is number of edges
- V is number of vertices

####Breadth First Search Vs. Depth First Search

- The simple answer to this question is that it depends on the size and shape of the tree.
  - For wide, shallow trees use Breadth First Search
  - For deep, narrow trees use Depth First Search

####Nuances:

- Because BFS uses queues to store information about the nodes and its children, it could use more memory than is available on your computer. (But you probably won't have to worry about this.)
- If using a DFS on a tree that is very deep you might go unnecessarily deep in the search. See xkcd for more information.
- Breadth First Search tends to be a looping algorithm.

- Depth First Search tends to be a recursive algorithm.

Efficient Sorting Basics

###**Merge Sort** ####Definition:

- A comparison based sorting algorithm
  - Divides entire dataset into groups of at most two.
  - Compares each number one at a time, moving the smallest number to left of the pair.
  - Once all pairs sorted it then compares left most elements of the two leftmost pairs creating a sorted group of four with the smallest numbers on the left and the largest ones on the right.
  - This process is repeated until there is only one set.

####What you need to know:

- This is one of the most basic sorting algorithms.
- Know that it divides all the data into as small possible sets then compares them.

####Big O efficiency:

- Best Case Sort: Merge Sort: O(n)
- Average Case Sort: Merge Sort: O(n log n)
- Worst Case Sort: Merge Sort: O(n log n)

###**Quicksort** ####Definition:

- A comparison based sorting algorithm
  - Divides entire dataset in half by selecting the average element and putting all smaller elements to the left of the average.
  - It repeats this process on the left side until it is comparing only two elements at which point the left side is sorted.
  - When the left side is finished sorting it performs the same operation on the right side.
- Computer architecture favors the quicksort process.

####What you need to know:

- While it has the same Big O as (or worse in some cases) many other sorting algorithms it is often faster in practice than many other sorting algorithms, such as merge sort.
- Know that it halves the data set by the average continuously until all the information is sorted.

####Big O efficiency:

- Best Case Sort: Merge Sort: O(n)
- Average Case Sort: Merge Sort: O(n log n)
- Worst Case Sort: Merge Sort: O(n^2)

###**Bubble Sort** ####Definition:

- A comparison based sorting algorithm
  - It iterates left to right comparing every couplet, moving the smaller element to the left.
  - It repeats this process until it no longer moves and element to the left.

####What you need to know:

- While it is very simple to implement, it is the least efficient of these three sorting methods.
- Know that it moves one space to the right comparing two elements at a time and moving the smaller on to left.

####Big O efficiency:

- Best Case Sort: Merge Sort: O(n)

- Average Case Sort: Merge Sort: O(n^2)
- Worst Case Sort: Merge Sort: O(n^2)

####Merge Sort Vs. Quicksort

- Quicksort is likely faster in practice.
- Merge Sort divides the set into the smallest possible groups immediately then reconstructs the incrementally as it sorts the groupings.
- Quicksort continually divides the set by the average, until the set is recursively sorted.

Basic Types of Algorithms

###**Recursive Algorithms** ####Definition:

- An algorithm that calls itself in its definition.
  - **Recursive case** a conditional statement that is used to trigger the recursion.
  - **Base case** a conditional statement that is used to break the recursion.

####What you need to know:

- **Stack level too deep** and **stack overflow**.
  - If you've seen either of these from a recursive algorithm, you messed up.
  - It means that your base case was never triggered because it was faulty or the problem was so massive you ran out of RAM before reaching it.
  - Knowing whether or not you will reach a base case is integral to correctly using recursion.
  - Often used in Depth First Search

###**Iterative Algorithms** ####Definition:

- An algorithm that is called repeatedly but for a finite number of times, each time being a single iteration.
  - Often used to move incrementally through a data set.

####What you need to know:

- Generally you will see iteration as loops, for, while, and until statements.
- Think of iteration as moving one at a time through a set.
- Often used to move through an array.

####Recursion Vs. Iteration

- The differences between recursion and iteration can be confusing to distinguish since both can be used to implement the other. But know that,
  - Recursion is, usually, more expressive and easier to implement.
  - Iteration uses less memory.
- **Functional languages** tend to use recursion. (i.e. Haskell)
- **Imperative languages** tend to use iteration. (i.e. Ruby)
- Check out this Stack Overflow post for more info.

####Pseudo Code of Moving Through an Array (this is why iteration is used for this)

```
Recursion                         | Iteration
----------------------------------|----------------------------------
recursive method (array, n)       | iterative method (array)
  if array[n] is not nil          |   for n from 0 to size of array
    print array[n]                |     print(array[n])
    recursive method(array, n+1)  |
  else                            |
    exit loop                     |
```

###**Greedy Algorithm** ####Definition:

- An algorithm that, while executing, selects only the information that meets a certain criteria.
- The general five components, taken from Wikipedia:
  - A candidate set, from which a solution is created.
  - A selection function, which chooses the best candidate to be added to the solution.
  - A feasibility function, that is used to determine if a candidate can be used to contribute to a solution.
  - An objective function, which assigns a value to a solution, or a partial solution.
  - A solution function, which will indicate when we have discovered a complete solution.

####What you need to know:

- Used to find the optimal solution for a given problem.
- Generally used on sets of data where only a small proportion of the information evaluated meets the desired result.
- Often a greedy algorithm can help reduce the Big O of an algorithm.

####Pseudo Code of a Greedy Algorithm to Find Largest Difference of any Two Numbers in an Array.

```
greedy algorithm (array)
  var largest difference = 0
  var new difference = find next difference (array[n], array[n+1])
  largest difference = new difference if new difference is > largest difference
  repeat above two steps until all differences have been found
  return largest difference
```

This algorithm never needed to compare all the differences to one another, saving it an entire iteration.

TSiege commented on May 2, 2014 Owner

For more checkout my blog Coderall

Ruckt commented on May 5, 2014

First line - remove a - be a both a quick.

Merge v Quick - 3rd the should be something else - Merge Sort divides the set into the smallest possible groups immediately then reconstructs the incrementally as it sorts the groupings.

Line 185 should be - "It repeats this process until it no longer moves an element to the left."

To plug my own blog - if you want an example of a recursive algorithm, here's an explanation of the backbone of the simple app i'm about to submit - http://ruckt.info/how-to-implement-a-recursive-algorithm/

afeld commented on Aug 19, 2014

I only got through Search Basics, and it's been a while since my Data Structures and Algorithms class, but some clarifications:

- Include that arrays can be many-dimensional to represent matrices
- Clarify that "Optimized Search" for Arrays is binary search
- "Doubly linked list has nodes that **also** reference the previous node."
- Insertion for linked lists is O(n)
- Might want to point out that hash pairs don't generally have an order
- You might want to introduce trees before binary trees
- You should introduce the concept of worst-case performance
- BFS isn't `O(|E| + |V|)`

- I don't think the recommendations for BFS vs DFS are correct
- "you probably won't have to worry about [running out of memory]"... except in an interview

taycaldwell commented on Aug 24, 2015

What about worst case space complexity?

Data Structures:
Array: O(n)
Linked List: O(n)
Hash Table: O(n)
BST: O(n)

Sorting:
Quicksort: O(log(n))
Merge Sort: O(n)
Bubble Sort: O(1)

randomascii commented on Aug 24, 2015

I thought this seemed odd. I mean yes, you could check for an already sorted list and then do nothing, but that is a very narrow special case. If you do that check at all levels then you make the whole thing slower, which seems like a poor tradeoff. I's say it's just O(n log n) always.

> Best Case Sort: Merge Sort: O(n)

This is an error. This is from the Quick Sort section (and Bubble Sort section). In fact, the results for all of the sorts are tagged with "Merge Sort". Oops.

> Worst Case Sort: Merge Sort: O(n^2)

emilyst commented on Aug 24, 2015

> Hash collisions are when a hash function returns the same output for two distinct outputs.

Should probably read:

> Hash collisions are when a hash function returns the same output for two distinct **inputs**.

gregory-nisbet commented on Aug 24, 2015

You should clarify the difference between amortized constant time and constant time, e.g.

> Indexing: Hash Tables: O(1)
> Search: Hash Tables: O(1)
> Insertion: Hash Tables: O(1)

The claim about the complexity of insertion into a hash table is not accurate if we consider one insertion at a time, in which case the worst case performance is O(n) (since we occasionally need to construct a hash table [some fixed multiple]-times larger than the previous one. Listing O(n) as the time complexity of insertion without elaborating is probably very misleading though.

ksylvest commented on Aug 24, 2015

This isn't right:

> Hash functions accept a key and return an output unique only to that specific key.

Hash functions accept arbitrary sized data and map it to fixed size data. The mapping is not guaranteed to be unique (and should never be assumed to be).

gregory-nisbet commented on Aug 24, 2015

Right, hash functions on data of arbitrary length cannot possibly be injective, but one could probably make a useful comment about SHA-256 having no known collisions and why that's important.

ksylvest commented on Aug 24, 2015

@gregory-nisbet Yup, it is correct that cryptographic hashing functions exist that have no known collisions.

However, hash functions are also used for ADTs like hash tables or binary search trees. Many implementations use use integer keys (about 4 billion possibilities on 32 bit systems). Collisions in this case are **extremely** likely (i.e. with a hundred thousand objects I believe a collision is more likely than not). Implementation of many of the basic ADTs require knowing / addressing collisions.

gregory-nisbet commented on Aug 24, 2015

We have a closed form for the probability of a such a collision (assuming data chosen randomly). It's the number of injective functions from `[m]` to `[n]` (`product(n, n-1, ... n-m+1)`) divided by the number of functions `n^m` ... Who's the target audience here? (edit : mixed up variables, counted injective functions up to permutation)

rsiemens commented on Aug 24, 2015

Small, but important typo on line 61:

> - **Hash collisions** are when a hash function returns the same output for two distinct outputs.

I believe this should be

> - **Hash collisions** are when a hash function returns the same output for two distinct `inputs` .

rjarteta commented on Aug 24, 2015

Can you do an extended document for OOP Basics, is like a nightmare when you don't have the basics covered.
Awesome work btw!

bullwinklescousin commented on Aug 25, 2015

Thank you!

TimoFreiberg commented on Aug 25, 2015

In

> Search: Depth First Search: O(|E| + |V|)
> E is number of edges
> V is number of vertices

If I remember correctly, E is the set of edges and |E| is the size of that set, so you should probably put pipes around the E and V in the last two lines.

whbboyd commented on Aug 25, 2015

This is missing one of the most important distinctions between breadth-first and depth-first searches:

- Breadth-first search is guaranteed to find a shortest possible path through a graph. Depth-first search is not (and usually does not).

In the specific case of searching a tree for a specific node, there is only one possible path, so both will return the same result, but search algorithms are very rarely used this way; typically, they operate on arbitrary graphs.

Additionally worth knowing is that most practically-used searching and pathfinding algorithms (e.g. Dijkstra's algorithm, A*) are specializations of breadth-first search.

javabuddy commented on Aug 25, 2015

Awesome cheat-sheet, but nothing beats practice, here are some String and Array related problems to check your knowledge and skill
http://goo.gl/a5LqjS

kmh287 commented on Aug 25, 2015

In the bullet under dynamic arrays:

"it's contents" should be changed to "its contents"

gmfawcett commented on Aug 25, 2015

"Iteration uses less memory" -- that's not true when the compiler or interpreter supports tail-call elimination, or can otherwise optimize recursive functions. For example, a good C compiler will translate tail-recursive functions into efficient code with no recursive calls (only local jumps). [edit: I assume you're talking about memory on the stack; otherwise I'm not sure what kind of memory usage you're claiming is only present in recursive functions.]

c4milo commented on Aug 25, 2015

> Hash collisions are when a hash function returns the same output for two distinct outputs.
> All hash functions have this problem.
> This is often accommodated for by having the hash tables be very large.

I think it is better to mention the techniques to deal with collisions: chaining and open addressing.

PeterDowdy commented on Aug 25, 2015

For quicksort, I think

> Divides entire dataset in half by selecting the average element and putting all smaller elements to the left of the average.

should read

> Divides entire dataset in half by selecting the median element and putting all smaller elements to the left of the average.

Determining the average element is an O(n) operation, whereas determining the median element is an O(1).

ahausmann commented on Aug 25, 2015

@**TSiege** Here are a few corrections:

1. DFS and BFS have no defined order in which the elements are searched.
2. Quicksort picks a random element from the dataset as the pivot element, then sorts all elements smaller before that and all greater than the pivot after it.
   Then quicksort is executed on the part left of the pivot and right of it.
3. Stackoverflow errors are thrown if the stack is full, not when the RAM is full. The stack usually has a constant size (defined by the OS and/or language).

@**PeterDowdy** For determining the median element in O(1) the dataset must first be sorted

pnathan commented on Aug 25, 2015

You really don't want DFS for infinite trees. ;)

ParoXoN commented on Aug 25, 2015

@TSiege:
Re: Greedy algos

> Used to find the optimal solution for a given problem.

I always thought greedy algorithms were used to find *expedient* answers, rather than optimal ones. E.g. a Traveling Salesman greedy algo just takes the locally minimum available distance at each city. It's not generally optimal, but it's quick and (often) decent.

Greedy algorithms *do* find the optimal solution if the general solution is the result of optimally solving the sub-problems, but many problems aren't structured this way.

ballance commented on Aug 25, 2015

I suggest adding some info about data streams, such as the following:

Stream I/O

Definition:

- A stream is an abstract representation of a sequence of bytes
- It is usually buffered to prevent representing the entire sequence of bytes in memory at one time
- Generally one-way and forward-only
- Has significant advantages over a standard array of bytes

What you need to know:

- Can be stored:
  - In memory
  - Transferred over a network
  - On disk
  - In a database
- Advantages over byte arrays
  - Efficient use of memory
  - Smaller memory footprint
- Uses
  - Transferring files between persistence locations
  - Compression / Decompression
  - Encryption / Decryption
  - Throttling
- Can be chained
- Some are one-way (NetworkStream, CipherStream, etc), others can seek in either direction

brianrodri commented on Aug 25, 2015

Merging can be done with any n number of splits, it does not need at most 2. This is true with many of the restrictions you've placed on things, I'd suggest clarifying that they "tend to have at most 2".

PeterDowdy commented on Aug 25, 2015

@ahausman sorry, I meant median position. I should just say "middle element" like a good communicator :)

sairion commented on Aug 26, 2015

"Big O Efficiency" sounds weird to me, I think it should be either "Computational Complexity" or "Time and Space Complexity".

KodeSeeker commented on Aug 26, 2015

Definitely switch to Time Complexity over Big-O Complexity. Big-O is generally worse case time complexity.

Also, worse case time complexity for merge sort is O(n* log n) and not O(n^2).

PH111P commented on Aug 26, 2015

You could add some stuff about Dynamic Programming. I think it's quite important and cool to know; especially if you can demonstrate some of its applications (e.g. the Floyd-Warshall algorithm, the DP solution for the Knapsack problem etc.).

Further, I suggest that you move the subsection “Greedy Algorithms” to a new section, as greedy algorithms can be coded either as a recursive or as an iterative algorithm (Quite often even both is possible). DP would fit into that new section as well, as would the missing “Complete Search” and “Divide and Conquer” approaches to solve a problem.

Depending on your expected use-cases, some string algorithms (e.g. KMP, Aho-Corasick, …) could be added as well, though that might be a little bit too specific.

BTW, Iterative Deepening DFS could be added as something “between” BFS and DFS.

pilkch commented on Aug 27, 2015

@Ruckt There are simpler recursive examples that don't do string manipulation, such as factorials :)

mjgpy3 commented on Aug 28, 2015

Under Quicksort's computational complexity, the headers still say "Merge Sort" (probably a copy/paste error from the Merge Sort section)

joeylmaalouf commented on Oct 19, 2015

> Hash collisions are when a hash function returns the same output for two distinct ~~outputs~~ inputs.

dmh2000 commented on Jan 23, 2016

when would you use merge sort vs quicksort? Merge sort is stable, quicksort is not.

btamada commented on Feb 1, 2016

> Big O efficiency:
> Insertion: Linked Lists: O(1)

I believe this should be O(N) as worse case scenario you have to insert a node at the end of the Linked List, which will require you to traverse the whole list (N).

jackyliang commented on Feb 12, 2016

Could you add one for basic concepts of OOP?

KKevinLL commented on Feb 16, 2016

There are some typos in Sorting basics where I think u copied and pasted so all three of them have mergeSort: O(...) under their big O efficiency(runtime are correct just some typos in names)

daveweber commented on Mar 16, 2016

Heaps would be a great topic to add

bencmbrook commented on Mar 31, 2016

This is great! Thanks. Found a typo:

> Hash collisions are when a hash function returns the same output for two distinct outputs.

two distinct **inputs**

Also, the definition of hash functions (especially where it says one-to-one correspondence) is a bit confusing: it seems like there are, by definition, no collisions.

sundbry commented on May 7, 2016 • edited

What is an array? you mean what you practiced during the summer before 101?

sundbry commented on May 7, 2016

Your notes generalize cheats, and unsourced opinions, such as the opinion when to use DFS/BFS. Good luck.

sundbry commented on May 7, 2016

If you spend 30 minutes of the same time reading an actual book on computer science, you'll start a much more thorough education.

ikeyany commented on Jun 12, 2016

Hey there's a typo under Quick Sort--for Big O comparison, it's labeled as Merge Sort instead of Quick Sort.

ruizz commented on Jul 20, 2016

@**sundbry** Wow, it's like you missed entire the point of the document.

BryanByteZero commented on Oct 2, 2016

Thanks will help this little noob here!

MaraJade commented on Oct 19, 2016

@**TSiege** your blog link doesn't work anymore.

anshusharma11 commented on Nov 16, 2016

I am looking for study buddy to conduct mock interviews. I have upcomming interview at a big tech company. Anyone interested, plz let me know anshusharma11@gmail.com

abhinavsv3 commented on Dec 27, 2016

Great stuff!

KingGeneral commented on Jan 26

wow, still helpful anyway :3

BazzalSeed commented on Feb 16

Worst case Space complexity for HashTable would be O(hashcode_range)
O(N) should be the best case?

ppaulojr commented on Mar 10

You forgot to add the time complexity of the AKS: O (log(n) ^ 3) if Agrawal's conjecture holds!

#justkidding

Nice Gist

ntratcliff commented on Apr 3

The formatting on your headers seems to be broken. I think you just need to add a space between your the hash and the first character of your header. Fixed on my fork.

vuonghv commented on Apr 5

@TSiege Thank you for your excellent post. But the formatting on the markdown header is incorrect, please fix it.

rmortimer commented on Apr 6

You wrote the merge sort efficiency under the quicksort section

pmkary commented on Apr 9

It's funny how these interviews has made the whole computer science into a few very silly questions. It's even offensive when you think that of all the huge hard courses that we have, this is what you need to know. Just imagine what others from other industries may think when you say: "oh yes it's just 100 questions and you're a developer". I hope there be a day where these stupid interviews be removed from the hiring process and people look at GitHub and CVs, something that actually shows years of hard work and the knowledge not just these questions.

Mohamed3on commented on Apr 12

Hi. I've updated the markdown of your document as it wasn't working correctly (The headers were displayed as ###header). Take a look here:
https://gist.github.com/Mohamed3on/d6c6189f140e7f2edb66436083ec9771
feel free to pull the document to update the original one (I made no changes to the content).

aripakman commented on Apr 30

Best Case Sort: Merge Sort: O(n) is wrong, should be O(n log n)

LawnboyMax commented on May 6 • edited

I suggest adding an OOP basics section.

Sample question: When should you use an abstract class and when should you use an interface?

Answer:

```
Consider using abstract classes if any of these statements apply to your situation:

   - You want to share code among several closely related classes.
  - You expect that classes that extend your abstract class have many common methods or fields, or require access
modifiers other than public (such as protected and private).
   - You want to declare non-static or non-final fields. This enables you to define methods that can access and modify
the state of the object to which they belong.

Consider using interfaces if any of these statements apply to your situation:

   - You expect that unrelated classes would implement your interface. For example, the interfaces Comparable and
Cloneable are implemented by many unrelated classes.
  - You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.
  - You want to take advantage of multiple inheritance of type.
```

Taken from: https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html

kchia commented on Jun 20

Best case runtime for merge sort should be O(n log n)

lifeboardshortcut commented on Jun 27

Formatted programming interview cheat sheet - https://www.amazon.com/dp/0692907815

wcpines commented on Jul 16

I just forked it and fixed the headings. You can find it here. Guessing I'm not the first nor the last to offer :P

mega0319 commented on Jul 29

Very helpful. Thanks! Might be worth showing some of the data structures in actual code. I have started working on this in JS and so far I have completed linked lists and binary search trees. Will be working on hash tables later today. Here's a link to my repo.

https://github.com/mega0319/data-structures

harrisonlingren commented on Aug 2 • edited

I forked this to an actual repo here. Fixed headings (again) and added a ToC. Gave you credit at the top and linked back to this Gist.

leitasat commented on Aug 14

> Hash functions accept a key and return an output unique only to that specific key.

Please make it more precise as it might be a first place for somebody to look it up. The sentence contradicts the very idea of a hash function which is used to compress a huge key space (e.g. all strings up to certain length) via mapping it to range `[0, 1, ... , n - 1]`

GraniteConsultingReviews commented on Aug 27

This algorithm is very hard to understand but i m trying please make this simple.