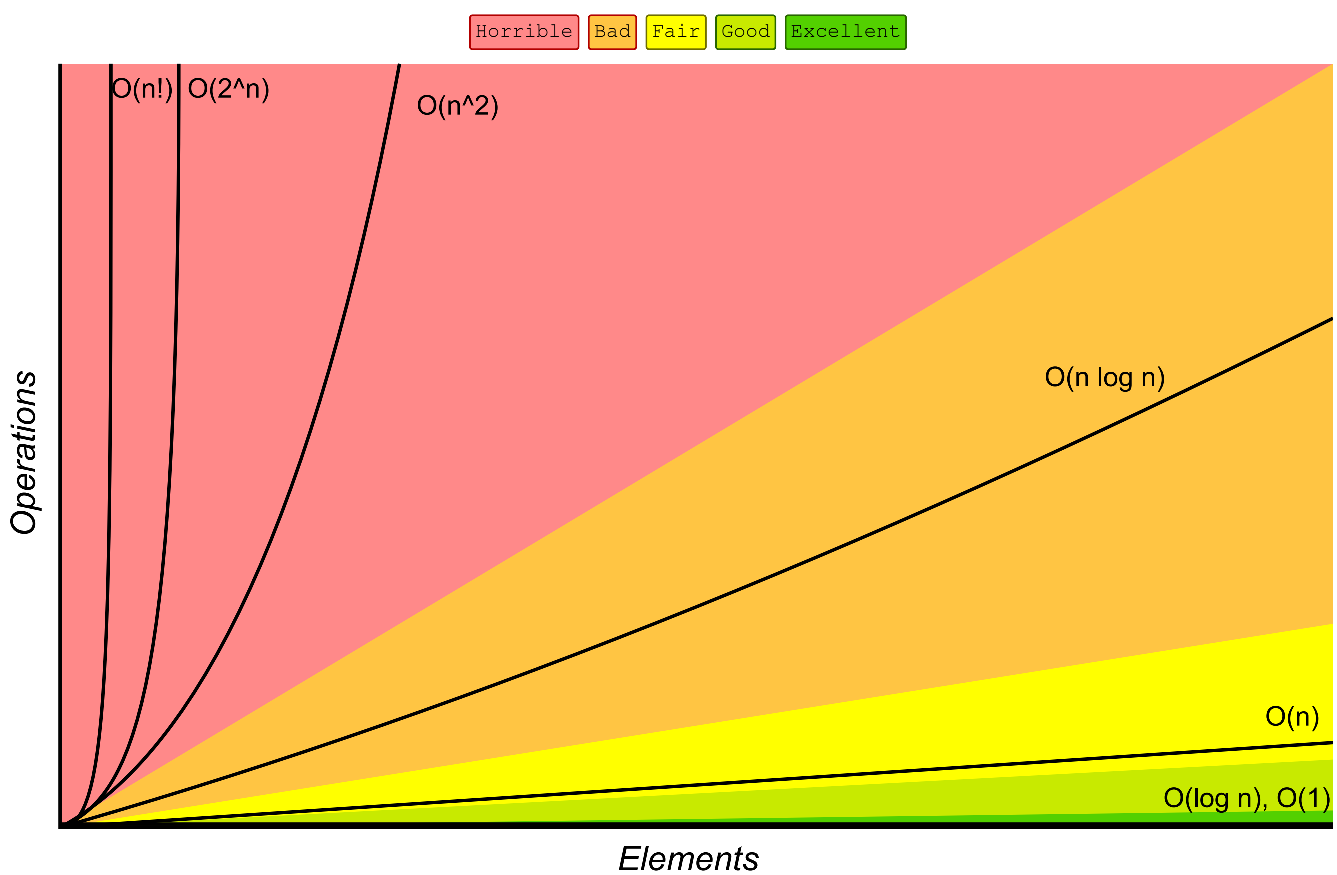
Know Thy Complexities!

Hi there! This webpage covers the space and time Big-O complexities of common algorithms used in Computer Science. When preparing for technical interviews in the past, I found myself spending hours crawling the internet putting together the best, average, and worst case complexities for search and sorting algorithms so that I wouldn't be stumped when asked about them. Over the last few years, I've interviewed at several Silicon Valley startups, and also some bigger companies, like Google, Facebook, Yahoo, LinkedIn, and Uber, and each time that I prepared for an interview, I thought to myself "Why hasn't someone created a nice Big-O cheat sheet?". So, to save all of you fine folks a ton of time, I went ahead and created one. Enjoy! - Eric

Big-O Complexity Chart



Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

Learn More

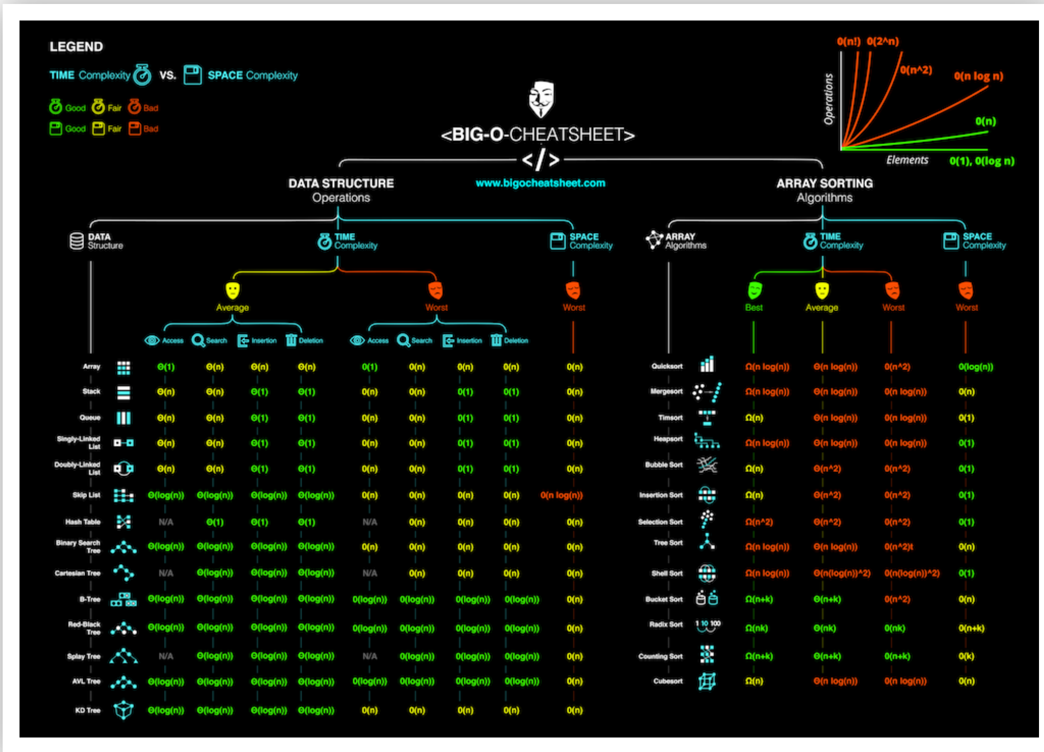
[Cracking the Coding Interview: 150 Programming Questions and Solutions]

[Introduction to Algorithms, 3rd Edition]

[Data Structures and Algorithms in Java (2nd Edition)]

[High Performance JavaScript (Build Faster Web Application Interfaces)]

Get the Official Big-O Cheat Sheet Poster



Contributors

Eric Rowell | Quentin Pleple | Michael Abed | Nick Dizazzo | Adam Forsyth | Felix Zhu | Jay Engineer | Josh Davis | Nodir Turakulov | Jennifer Hamon | David Dorfman | Bart Massey | Ray Pereda | Si Pham | Mike Davis | mcverry | Max Hoffmann | Bahador Saket | Damon Davison | Alvin Wan | Alan Briolat | Drew Hannay | Andrew Rasmussen | Dennis Tsang | Vinnie Magro | Adam Arold | Alejandro Ramirez | Aneel Nazareth | Rahul Chowdhury | Jonathan McElroy | steven41292 | Brandon Amos | Joel Friedly | Casper Van Gheluwe | Eric Lefevre-Ardant | Oleg | Renfred Harper | Piper Chester | Miguel Amigot | Apurva K | Matthew Daronco | Yun-Cheng Lin | Clay Tyler | Orhan Can Ozalp | Ayman Singh | David Morton | Aurelien Ooms | Sebastian Paaske Torholm | Koushik Krishnan | Drew Bailey | Robert Burke

Make this Page Better

[Edit these tables!]

484 Comments **Big-O Cheat Sheet** 1 **Login**

Recommend 312 Tweet Share **Sort by Best**

Join the discussion…

LOG IN WITH

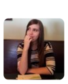
OR SIGN UP WITH DISQUS ?

Name

Michael Mitchell • 7 years ago

This is great. Maybe you could include some resources (links to khan academy, mooc etc) that would explain each of these concepts for people trying to learn them.

366 | 1 • Reply • Share ›



Amanda Harlin Michael Mitchell • 7 years ago

Yes! Please & thank you

80 ^ | ∨ • Reply • Share ›

Asim Ahmad ➦ Amanda Harlin • 2 years ago
Can you Explain the Above Algorithm.??
^ | ∨ 4 • Reply • Share ›

Anonymous ➦ Asim Ahmad • 2 years ago
Mr.
you can learn these algorithms easily in google by searching
Don't always ask or wait for someone to post things for you go out and search on internet
You will find everything you want to learn

If you are a beginner in Data structures and algorithms then visit mycodeschool youtube channel and learn there
if you want more then email me at rise.d1105@gmail.com I will help you as much as I can
25 ^ | ∨ 6 • Reply • Share ›

Cam Cecil ➦ Michael Mitchell • 7 years ago
This explanation in 'plain English' helps:
http://stackoverflow.com/qu...
35 ^ | ∨ 1 • Reply • Share ›

Richard Wheatley ➦ Cam Cecil • 4 years ago
this is plain english.
13 ^ | ∨ 2 • Reply • Share ›

Arjan Nieuwenhuizen ➦ Michael Mitchell
• 7 years ago • edited
Here are the links that I know of.

#1) http://aduni.org/courses/al...
#2) http://ocw.mit.edu/courses/...
#3) https://www.udacity.com/cou...

probably as good or maybe better # 2, but I have not had a chance to look at it.
http://ocw.mit.edu/courses/...

Sincerely,
Arjan

p.s.
https://www.coursera.org/co...
This course has just begun on coursera (dated 1 July 2013), and looks very good.
21 ^ | ∨ • Reply • Share ›

fireheron ➦ Arjan Nieuwenhuizen • 6 years ago
Thank you Arjan. Espaecially the coursera.org one ;-)

5 ^ | v • Reply • Share ›

@hangtwentyy ➦ fireheron • 5 years ago

also this! http://opendatastructures.org

7 ^ | v • Reply • Share ›

yth ➦ @hangtwentyy • 5 years ago

thank you for sharing this.

1 ^ | v • Reply • Share ›

Eduardo Sánchez ➦ Michael Mitchell • 4 years ago

There is an amazing tutorial for Big O form Derek Banas in Youtube, that guy is amazing explaining!!!

see more

10 ^ | v • Reply • Share ›

Sudhanshu Mishra ➦ Eduardo Sánchez • 4 years ago

Cool! This is a more than adequate introduction! Thanks a ton for sharing!

1 ^ | v • Reply • Share ›

Mohammed Hameed ➦ Eduardo Sánchez • 6 months ago

Thanks...

^ | v • Reply • Share ›

CodeMunkey ➦ Michael Mitchell • 3 years ago

Not sure if this helps, but here's a more visual learner for some of these algorithms - if you're interested.
http://visualgo.net

2 ^ | v • Reply • Share ›

Divyendra Patil ➦ Michael Mitchell • 2 years ago

www.codenza.us

1 ^ | v • Reply • Share ›

Abby Jones ➦ Michael Mitchell • 5 months ago

Abby Jones ↪ Michael Mitchell • 3 months ago

Fabulous idea!

^ | v • Reply • Share ›

Jeshika Morneau ↪ Michael Mitchell • a year ago

Or you could have supplied them in your comment instead.

^ | v • Reply • Share ›

nate lipp ↪ Michael Mitchell • 3 years ago

This is a well put together introduction

https://www.interviewcake.c...

^ | v • Reply • Share ›

Nhập Hàng Ngoại ↪ Michael Mitchell • 3 years ago

http://fashionfor.life/t-sh...

see more

^ | v 7 • Reply • Share ›

Blake Jennings • 7 years ago

i'm literally crying

100 ^ | v • Reply • Share ›

friend ↪ Blake Jennings • 3 years ago

you give me a big o

5 ^ | v • Reply • Share ›

Gokce Toykuyu • 7 years ago

Could we add some tree algorithms and complexities? Thanks. I really like the Red-Black trees ;)

89 ^ | v • Reply • Share ›

ericdrowell Mod ↪ Gokce Toykuyu • 7 years ago

Excellent idea. I'll add a section that compares insertion, deletion, and search complexities for specific data structures

31 ^ | v • Reply • Share ›

31 ^ | v • Reply • Share ›

yash bedi ➦ ericdrowell • 3 years ago
its been 4 years you haven't added that section :)
^ | v • Reply • Share ›

Elliot Géhin ➦ yash bedi • 3 years ago
It's up there Yash, bottom of the first table
1 ^ | v • Reply • Share ›

Jonathan Neufeld ➦ Gokce Toykuyu • 3 years ago
Where I come from we use trees on a regular rotation
3 ^ | v • Reply • Share ›

Valentin Stanciu • 7 years ago
1. Deletion/insertion in a single linked list is implementation dependent. For the question of "Here's a pointer to an element, how much does it take to delete it?", single-linked lists take O(N) since you have to search for the element that points to the element being deleted. Double-linked lists solve this problem.
2. Hashes come in a million varieties. However with a good distribution function they are O(logN) worst case. Using a double hashing algorithm, you end up with a worst case of O(loglogN).
3. For trees, the table should probably also contain heaps and the complexities for the operation "Get Minimum".
62 ^ | v • Reply • Share ›

Alexis Mas ➦ Valentin Stanciu • 6 years ago
If you a list: A B C D, When you want to delete B, you can delete a node without iterating over the list.

1. B.data = C.data
2. B.next = C.next
3. delete C

If you can't copy data between nodes because its too expensive then yes, it's O(N)
6 ^ | v 1 • Reply • Share ›

Miguel ➦ Alexis Mas • 5 years ago
You still have to find the position in the list, which can only be done linearly.
7 ^ | v • Reply • Share ›

Guest ➦ Miguel • 5 years ago • edited
You still have to find the position in the list, which can only be done linearly.
3 ^ | v • Reply • Share ›

Alexis Mas ➦ Miguel • 5 years ago
Yes of course, If you need to search the node it's O(n), otherwise you can delete it as I stated before.
1 ^ | v • Reply • Share ›

Guest ➦ Alexis Mas • 5 years ago

No need to find the position if you can delete it as Alexis mentioned

2 ^ | ˅ 1 • Reply • Share ›

OmegaNemesis28 ➦ Alexis Mas • 5 years ago • edited

To get to B - you HAVE to iterate over the list though. You can't just manipulate B without a pointer. So unless you do book-keeping and have pointers to specific nodes you intend to delete/manipulate, LinkLists are O(n) insert and delete.

3 ^ | ˅ • Reply • Share ›

Alexis Mas ➦ OmegaNemesis28 • 5 years ago

Strictly speaking no, you don't. let's say you have this function.

public void delete(Node node)

That function doesn't care how did you got that node.

Did you got my point?

When you have a pointer to a node, and that node needs to be deleted you don't need to iterave over the list.

1 ^ | ˅ 2 • Reply • Share ›

Sam Lehman ➦ Alexis Mas • 5 years ago

But in order to get to that pointer, you probably need to iterate through the list

2 ^ | ˅ • Reply • Share ›

OmegaNemesis28 ➦ Alexis Mas • 5 years ago • edited

But that is MY point :p

You have to have the node FIRST. You have to iterate through the list before you can do that, unless you do book-keeping and happen to have said node. Reread what I said. "have pointers to specific nodes" Most of the time, you do not with LinkedLists. If you have a Linked List and want to delete index 5, you have to iterate to 5 and such. Your example was ABCD, our points are that you typically don't have the pointer to B just offhand. You have to obtain it first which will be O(n)

2 ^ | ˅ • Reply • Share ›

Chris B ➦ OmegaNemesis28 • 4 years ago

Search and insert/delete are different operations. Insert/delete on an unsorted linked list is O(1). The fact that you might have to first search for the element that you want to delete is not considered relevant, as that functionality is covered by the O(n) search operation, not the O(1) insert/delete operations. A real world example of linked list insert/delete can be found in list_del and list_add of the Linux kernel source, those functions are only 2 and 4 lines of code, so should be easy to understand: http://lxr.free-electrons.c...

2 ^ | ∨ • Reply • Share ›

Pingu App ➦ Alexis Mas • 5 years ago

What if B is the last element in the list?
How would B's predecesor know that its next field should point to NULL and not to a futurely invalid memory address?

2 ^ | ∨ • Reply • Share ›

Alexis Mas ➦ Pingu App • 5 years ago

In that case you can't deleted that way, you're forced to have a pointer to the previous item.

1 ^ | ∨ • Reply • Share ›

pvlbzn ➦ Alexis Mas • 3 years ago • edited

And you will introduce the side effect which will be hell to debug. Consider:
Singly linked list { A:1, B:2, C:3, D:4 } where is X:Y, y is a value, function `delete` which works as you described, function `get` which returns pointer to the node by index.

```

// Take needed node C
node_t* node = get(list, 2)
print(node->value) // prints 3

// Delete B
delete(list, 1)

// Try to access C again
print(node->value) // well, enjoy your O(1)
```