Maximum Subarray Sum

How to present for a Google/Fb/Amazon/Microsoft interview:

Vyshnav Ramesh · Follow
4 min read · Aug 20, 2019

1

Maximum Subarray Sum (also known as Kadane's Algorithm / Largest Sum Continuous Subarray) - is the task of finding the largest possible sum of a subarray, within an array of numbers.

Application:

1. One of the obvious application is **business analysis** where you need to find out the duration of time where the company experienced the maximum growth or minimum growth which helps company to find what they did good or bad during those periods to repeat or prevent them in future for benefit of company.
2. **Stocks:** Suppose you have the daily stock prices of a company and you want to know which period the company had maximum growth and then correlate that period with any current affairs or news in that period.
3. For **computer vision** , maximum subarray algorithms are used in

bitmap images to detect the highest score subsequence which represents the brightest area in an image. The image is a two dimensional array of positive values that corresponds to the brightness of a pixel. The algorithm is evaluated after normalizing every value in the array by subtracting them from the mean brightness.

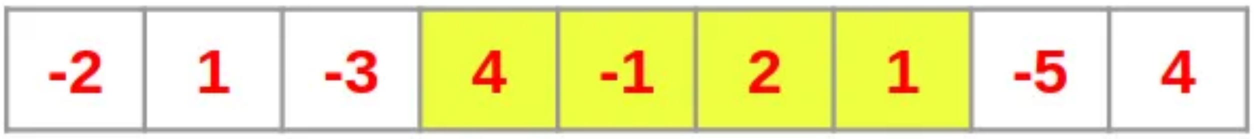
How does it work?

1. **Brute Force or Dynamic Programming.**

> *How should I explain dynamic programming to a 4-year-old?*
>
> **writes down "1+1+1+1+1+1+1+1 =" on a sheet of paper**
> *"What's that equal to?"*
> **counting* "Eight!"*
> **writes down another "1+" on the left**
> *"What about that?"*
> [illegible]

> *"You just added one more"*
> *"So you didn't need to recount because you remembered there were eight!*
> Dynamic Programming *is just a fancy way to say 'remembering stuff to save time later'"*

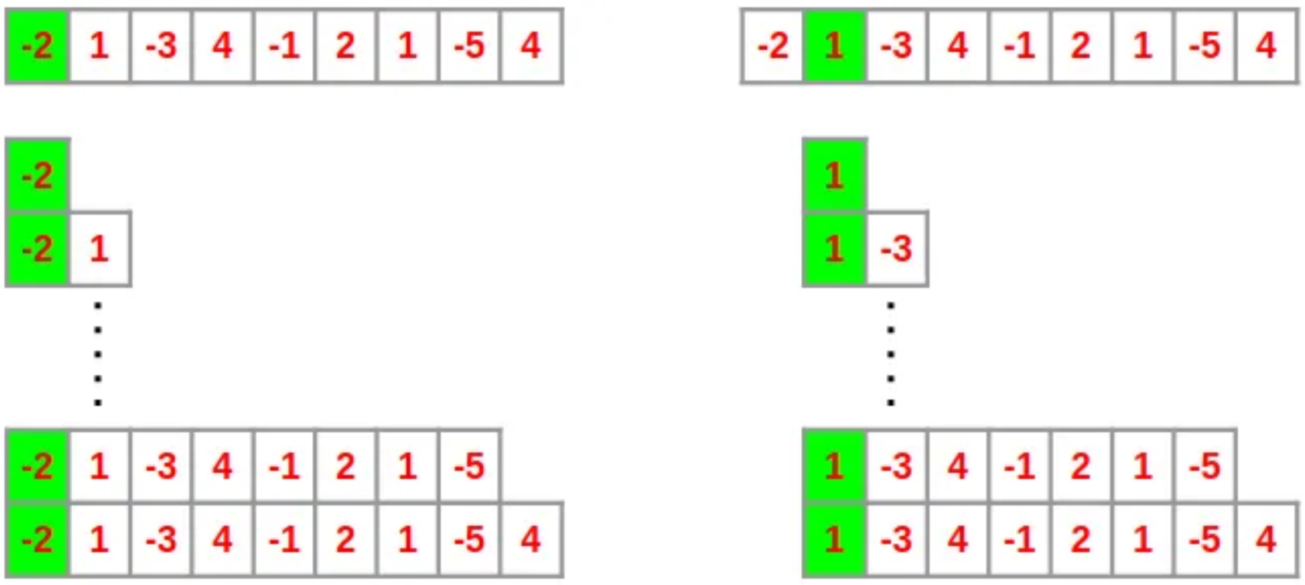
2. Explanation with a Problem



Maximum Sum Subarray (In Yellow)

a. Brute Force Approach:

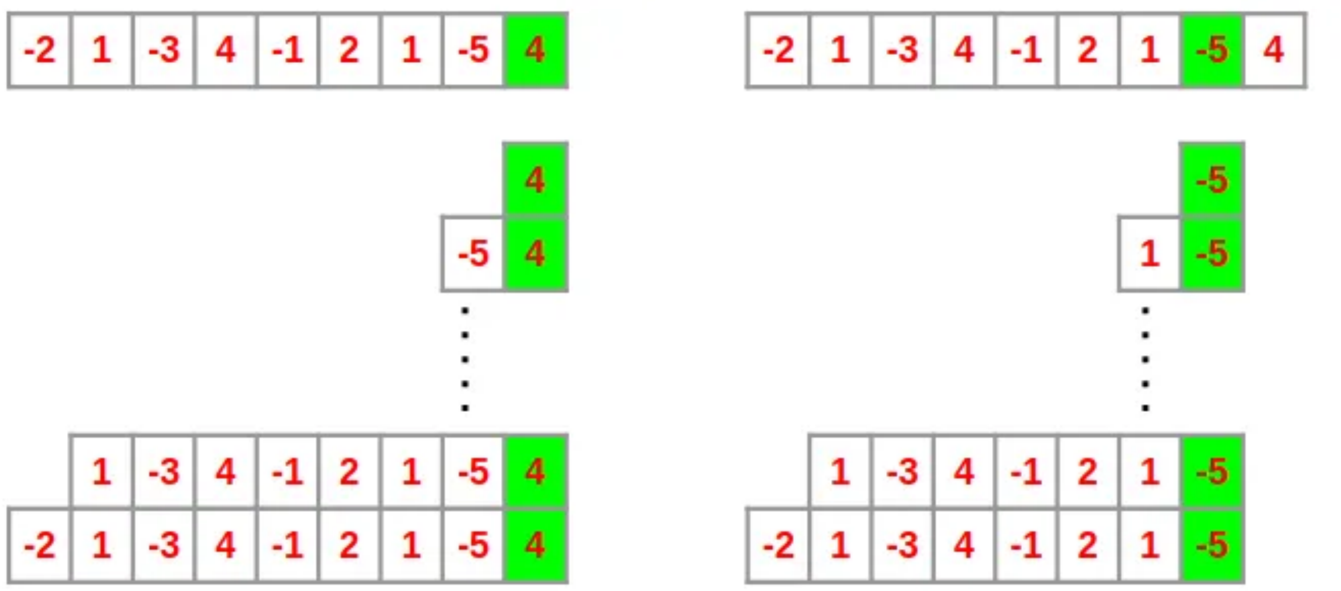
One very obvious but not so good solution is to calculate the sum of every possible subarray and the maximum of those would be the solution. We can start from index *0* and calculate the sum of every possible subarray starting with the element *A[0],* as shown in the figure below. Then, we would calculate the sum of every possible subarray starting with *A[1], A[2]* and so on up to *A[n-1]*, where ***n*** denotes the size of the array (n = 9 in our case). Note that every single element is a subarray itself.



We will call the maximum sum of subarrays starting with element *A[i]* the ***local_maximum*** at index ***i***. Thus after going through all the indices, we would be left with *local_maximum* for all the indices. Finally, we can find the maximum of these *local_maximum*s and we would get the final solution, *i.e.* the maximum sum possible. We would call this the *global_maximum*. The time complexity of this solution is ***O(n^2)*** which is not very good.

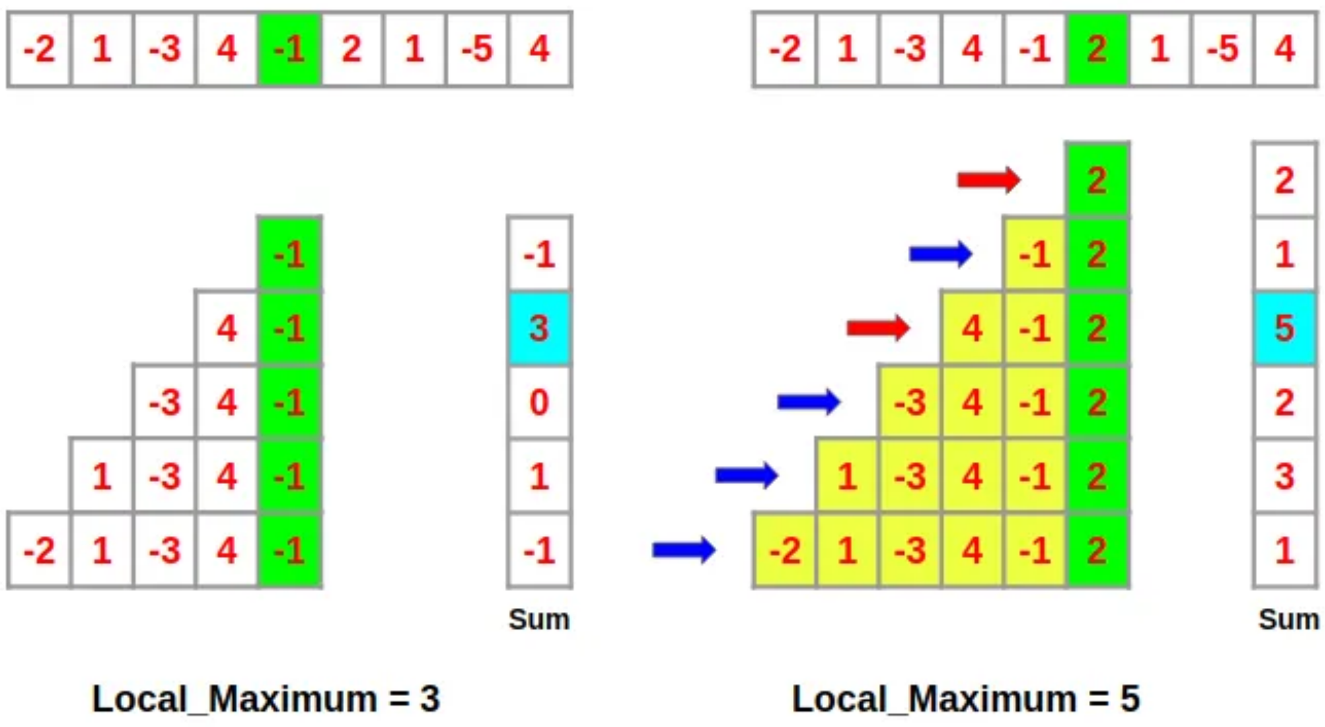
b. Dynammic Programming Approach:

We would start from the last element and calculate the sum of every possible subarray ending with the element ***A[n-1]***, as shown in the figure below. Then, we would calculate the sum of every possible subarray ending with ***A[n-2]***, ***A[n-3]*** and so on up to ***A[0]***.



Backward Brute Force Approach: Iteration 0 (left) and Iteration 1 (right)

Now let's focus on the subarrays ending with the element ***A[4]*** (=-1) and ***A[5]*** (=2) shown in the figure below.



From the figure above, we see that the *local_maximum[4]* is equal to 3 which is the sum of the subarray [4, -1]. Now have a look at the subarrays ending with *A[5]*. You'll notice that these subarrays can be divided into two parts, the subarrays ending with *A[4]* (highlighted with yellow) and the single element subarray *A[5]* (in green).

Let's say somehow I know the *local_maximum[4]*. Then we see that to calculate the *local_maximum[5],* we don't need to compute the sum of all subarrays ending with *A[5]* since we already know the result from arrays ending with *A[4]*. Note that if array [4, -1] had the maximum sum, then we only need to check the arrays highlighted with the red arrows to calculate *local_maximum[5]*. And this leads us to the principle on which Kadane's Algorithm works.

> local_maximum at index i is the maximum of A[i] and the sum of A[i] and local_maximum at index i-1.

This way, at every index ***i***, the problem boils down to finding the maximum of just two numbers, ***A[i]*** and ***(A[i] + local_maximum[i-1])***. Thus the maximum subarray problem can be solved by solving these sub-problems of finding ***local_maximums*** recursively. Also, note that ***local_maximum[0]***would be ***A[0]*** itself.

Using the above method, we need to iterate through the array just once, which is a lot better than our previous brute force approach. Or to be more precise, the time complexity of Kadane's Algorithm is ***O(n)***.

Programming Data Structures Algorithms

Written by Vyshnav Ramesh

Follow

6 Followers

SDE - Java at ITC India

More from Vyshnav Ramesh

Vyshnav Ramesh

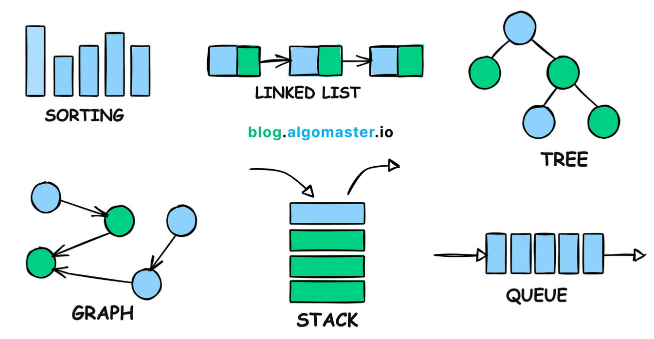
Internal Implementation of Java collections

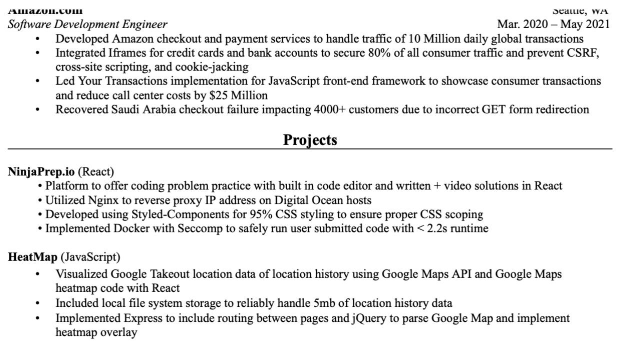
2 days back, I was asked to design HashMap in an interview for SDE-Java.

Dec 30, 2019 7 1

See all from Vyshnav Ramesh

Recommended from Medium





Amazon.com Seattle, WA
Software Development Engineer Mar. 2020 – May 2021
- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by $25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

Projects

NinjaPrep.io (React)
- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

HeatMap (JavaScript)
- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay

Ashish Pratap Singh in AlgoMaster.io

How I Mastered Data Structures and Algorithms

Getting good at Data Structures and Algorithms (DSA) helped me clear...

Jul 23 1.3K 13

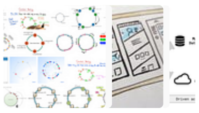
Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at...

1-page. Well-formatted.

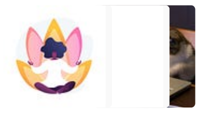
Jun 1 20K 377

Lists



General Coding Knowledge
20 stories · 1561 saves

Coding & Development
11 stories · 802 saves



Stories to Help You Grow as a Software Developer
19 stories · 1349 saves

ChatGPT
21 stories · 792 saves

Md Shadekur Rah... in coding interview prepar...

My Interview Experience at Meta [E4 Offer]

A recruiter reached out to me via email at the end of Nov 2023 to see if I am...

Jul 4 21

Swarnavo Pramanik

Design (LLD) a system for online food ordering and delivery like...

Online food order design requirements:

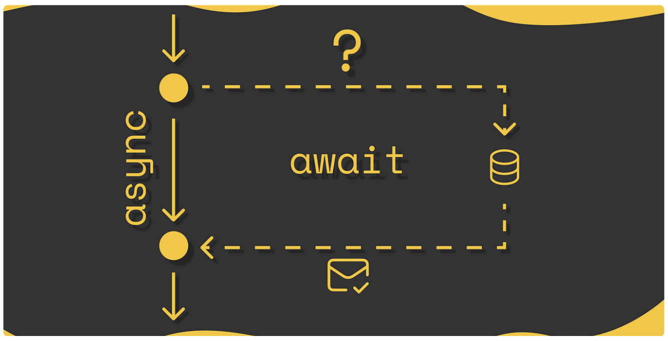
Apr 2 2

DesignNerds

Code refactoring challenge_

Background

Aug 3



Vyacheslav Efimov in Towards Data Science

Intuitive Explanation of Async / Await in JavaScript

Designing asynchronous pipelines for efficient data processing

3d ago 82

See more recommendations