Algo Bucket
-Shivaraj B H

July 15, 2019

Here I will be listing all the important algorithm's that you have to remember before going to any interview or coding competitions. (Not necessary that I mention every single one of them)

Solution :

1. Kadane's algorithm (Effeicient algorithm for maximum subarray sum. It is based on the idea that at any point in the array we can decide whether we have to add the current element to the preivous sum or start fresh from that point.)
2. Johnson and Trotter algorithm (Permutations of a string using mobile integers)
3. Euclidean algorithm (To find GCD)
4. Extended Euclidean algorithm (Find coefficients of Bezout's identity eg: find x,y in ax+by=gcd(a,b))
5. Matrix exponentiation (To find solutions that consists of finding something similar to fibonacci numbers, (Where you add previous numbers (can be previous 2,3..n) to get current number))
5. Partition problem (To find ways to split an array in such a way that the left sum and right sum about that point becomes minimum. There is also a DP approach to find the minimum difference but I haven't been able to figure out the maximum sum (out of left and right sum) that causes the difference to be minimum. I have only written down the recursive approach.)
6. Some simple yet efficient one liners (element swap with XOR, powers of 2 using left shift)
7. Important DSA libraries in python (set(Hashing)))

8 Longest Consecutive Subsequence (O(n) solution using hashing)

9 Sum of all subarrays (Two approaches : One using $2^{n-1}.sum(array)$ and another using the indexes of the elements and it's value.)

10 Properties of Gamma function (For positve real numbers, negative real numbers)

11 Calculate $\binom{n}{k}$ for large values using modulo of $10^9 + 7$ (Prime number) for large values of n and r (This can be used to find the number of ways to move from (0,0) to (n,n) using the formula $\binom{2(n-1)}{(n-1)}$. The basic idea is to find the numerator of combinations and then the modular inverse of the factorial part in the denominator $(n-r)!(r)!$, finally multiply the inverse with the numerator factorial value $(n)!$. Print the modulo of the result.)

12 Fold a paper of dimensions h , w to dimensions h_1 , w_1 (The illustration to reach to the solution for input length reaching upto 10^{15} has been mentioned in the code segment. The algorithm takes on an average 4.23 μs, for h=10,w=20,h_1=3,w_1=2 and 4.27 μs for h=10^5,w=20^5,h_1=3,w_1=2, which is O(1) (If you look at the algorithm used to compute the log base 2, then you will be convinced that it is not O(1) but if you consider the case above there is no such drastic increase in time and hence you can think of the algorithm to be psuedo constant) because log function is approximately a constant time operation.)

13 Karatsuba algorithm for multiplying n-bit numbers, Strassen algorithm for multiplying nxn matrices, change loop order to multiply two matrices (Right utilisation of cache).

14 Divide and conquer algorithms.

15 Number of ways to fill 1 and 2 in a n length tray is the nth fibonacci number, where n is the length of the array. (It is a music theory application to find number of ways to fill long and short syllables, where 1 is short and 2 is long.)

15 Longest monotonically increasing, bitonic sequences. (For LMI, the algorithm I have come up with is a recursive one and it takes less than $O(2^N)$, whereas it would have taken $O(2^N + N)$ to generate all subsequences and then find for the longest and also more auxillary space because we will be storing the subsequences. (Considering you are generating subsequence using recursion itself, which is obsolete.))

16 Longest Increasing Subsequence O(NlogN) solution. I had implemented a recursive approach which was better (Check the code snippet for the code.) than the backtracking recursive approach given here.

17 Quick sort : O(nlogn) time complexity and O(logn) space complexity. Radix sort : O(nlogU) time complexity (Where U is the largest element of the array) and O(logn) auxilary space. Heap sort : O(nlogn) time complexity and O(1) auxilary space.

18 Printing permutations of a string with repetitions allowed. To print permutations of a string we use backtracking (Generally), although we have a better approach as mentioned in (**2**). Using the backtracking approach we only have to edit how many iterations the for loop runs, to print the permutations with repetition.