
Headstart
Coding Club, IIT Guwahati
< WEEK 3 >

CONTENTS

1. **Binary Search** *(Expected Time: 1-2 days)*
2. **Bitwise Operations** *(Expected Time: 0-1 days)*
3. **Two Pointers** *(Expected Time: 1-2 days)*
4. **Number Theory** *(Expected Time: 2-3 days)*

Note:

- The articles mentioned contain theoretical concepts and should be read by everyone. The codes might be given in C++, but we can correspondingly develop the python code from the theory given.
- The articles from USACO Guide can also be accessed in both python and C++ by changing the language using an option on the webpage
- Python users can checkout our **GitHub repo** which contains the code of some useful functions to be covered ahead.

BINARY SEARCH

(Expected Time: 1-2 days)

This is one of the most useful and important algorithms you will ever come across. It is used to solve problems which have **some monotonic features.**

The main idea of the algorithm is to **divide the problem into half at each stage**.
Let's take an example of a classical problem. Suppose you are given an array of elements whose elements are sorted in increasing order and you are asked the smallest number greater than k which is present in the array.

Now take some time and think of some strategies.

One possible strategy can be:
Iterate over the array from 1 to N, once you get to a position whose value is greater than k then this is the answer.

Optimal Strategy: **Binary search**

You can divide the array into two halves **1 to N/2** and **N/2 to N**. Now let's check in which half is our answer present. If it's in the left half then a[N/2] has to be greater than k. If not, then it has to be in the right half.

Thus, just by checking the value of **a[N/2]** we can get rid of one half.

An important point to note is that after every query the length of the array in which we search becomes half.
So **N -> N/2 -> N/4 -> N/8 ..... 1** , and we only need to ask **log(N)** queries.

So, Binary search is only this much (just think about removing one half).

Here are some great resources which will teach you how to think about binary search and how to implement it:-

- Binary Search tutorial (C++ and Python)
- All 5 steps of codeforces edu Courses - Codeforces

STL STUFF RELATED TO BINARY SEARCH

In almost all Containers in C++ you can use inbuilt binary search which works in **O(log(n))**.

These two you will use most frequently.

- **Vectors** :- You need to first sort the vector for these to work correctly.
  - **Binary Search**
  - **Lower Bound**
  - **Upper Bound**

- **Sets / Multisets** :
  - **Find (works like binary search)**
  - **Lower Bound**
  - **Upper Bound**

Note that both have different syntax. If you use the syntax for vectors in Set then its time complexity won't be **O(log(n))**, it will be **O(n)**, and will probably give TLE. You have to be very careful about the syntax.

PYTHON

These two you will use most frequently.

- **Lists** :- You need to first sort the list for these to work correctly.
  Bisect module:
  The bisect module (as discussed in week 1), is an inbuilt library that allows you to apply binary search over a list in **O(log(n))** time.
  These modules are a little uncommon among python users so the best place to learn them is from their **official documentation**

- **Sets** :- Unfortunately, Sets in python don't work the same way as sets in cpp. So it becomes really complex to apply binary search over sets in python. It can be done by using a separate library called **Itertools** which has an attribute '**iter'** that can be used to iterate over a set. However, this method is extremely tedious. The best method would be to make your own class object "set" using Red black trees which is kind of complex at this stage. Read **this** for more information

- **Multisets** :- Like cpp, python has **multisets** as well and just like python sets, it isn't possible to apply binary search over python multisets. So the best method is to create your own Class object "Multiset" using **Red black trees** as suggested above.

Practice Problems

- **Maximum Median**
- **Codeforces EDU (Step 1,2,3)**

Some additional problems

- **Alyona and a Narrow Fridge**
- **Sagheer and Nubian Market**
- **K-th Not Divisible by n**
- **Valhalla Siege**
- **The Doctor Meets Vader**
- **Concert Tickets (Imp)**
- **Cellular Network**
- **Sum of Three Values**
- **Array Division (Tough)**
- **Elemental Decompress**

You can relate Binary exponentiation to binary search also. First try to write a code yourself to calculate a to the power of b. If you can't come up with a solution, you may refer to this video **Binary Exponentiation**. I advise you to devote, at the very least, half an hour before watching the video.

BIT OPERATIONS

(Expected Time: 0-1 days)

In Competitive Programming, writing numbers in their binary form might help you. It's really necessary to know different properties of binary representation of numbers and how to perform binary operations.

While we see integers in decimal system, they are still stored in binary form. We can apply operations like and, or, xor etc to each bit of a number and use it for our benefit.

For example, you might know that a set of n elements (array of n elements for our purpose) has 2^n subsets. All of these can be easily accessed using binary representation as 0 or 1 at a bit position can indicate the presence or absence of that element in the subset.

- **CPH (pg 96 to 99):** This covers the essentials of how bit operations work and how they can be useful. This is a **must read**.
- It is fine if you find the concepts difficult to grasp, the following video can be helpful- **Bitwise Operations tutorial #1 | XOR, Shift, Subsets**
- **Bitwise Operators in Python** (Uses the same idea as the above, just that the code will be in py).

Practice Problems

- **Petr and a Combination Lock**
- **Absolute Maximization**
- **NIT orz!**
- **Vampiric Powers**
- **Pairs Of Integers**

Additional Questions

- **Even-Odd XOR**
- **Orray**
- **Odd Topic**

Additional Resources

- **Bitwise Hacks for Competitive Programming**
- **C++ Bitsets in Competitive Programming**
- Codeforces Blog: **Part_1** , **Part_2**
- **USACO Guide** **(Hard)**

TWO POINTERS

(Expected Time: 1-2 days)

The name “Two Pointer” suggests the use of two different pointers, and it is exactly that, but without actually using C++ pointers (Well, you can use, but why would you want to?). It’s just a fancy name for a technique which uses two variables in a single loop. The variables are generally used to keep track of indices of an array or string, and generally in a sorted array.

This technique is all about iterating two monotonic pointers across an array to search for a pair of indices satisfying a given condition in linear time. This technique comes handy in other problem types like **sliding-window** and sub-array problems.

The best place to learn this technique is from **Codeforces EDU Two-pointers-CF**

Some other resources

- **USACO guide**
- **CPH**

Practice Problems

CSES (Important)

- **Sum of Two Values**
- **Sum of Three Values**
- **Maximum Subarray Sum**
- **Subarray Sums I**

Codeforces

- **Codeforces edu Two Pointers - Step 1**

Additional Question

- **Codeforces edu Two Pointers - Step 2**

NUMBER THEORY

(Expected Time: 2-3 days)

Another important part of CP is number theory, and recently questions based on maths and number theory have become quite frequent. A basic grasp on these topics will be very useful.

CALCULATING GCD

- GCD of two numbers can be calculated using the Euclidean Algorithm, i.e., **gcd(a,b)=gcd(a, a-b)**.
- This can be further optimized as we can reduce repeated subtractions. Hence, **gcd(a,b)=gcd(a, a%b)**.
- Its time complexity is **O(logn)**.
- You can also use the inbuilt functions

  C++ : *__gcd(a,b)* Python(use import math) : *math.gcd(a,b)*

Resources

- CPH, page **200-201**
- **Euclidean algorithm** (Ignore Binary GCD)
- ***Extended Euclidean Algorithm**

PRIME NUMBERS AND THEIR TESTS

- To check whether a number is prime or not, we need not check its divisibility by all numbers from 1 to n.
- We can just check for numbers **<= sqrt(n)** because if there exists a factor greater then sqrt(n), there will also exist a factor less than sqrt(n).
- It's time complexity is **O(sqrt(n))**

**Optional topics ! Study only if you have time .*

Resources

- CPH, page **199**
- **Primality test: algorithm**
- ***Extended Primality Tests**

PRE PROCESSING PRIMES TILL INTEGER

- When we need to check for primality of multiple numbers in the same code, it can be better to preprocess and store whether a number is prime or not.
- This can be done in **O(n(log(log(n)))** time for preprocessing of **Sieve of Eratosthenes**. Then we can access whether a number is prime or not later in O(1) time. {Might be useful if you have queries of numbers .
- The problem with this is that it may **not** be possible for large n (>=10^7)
- CPH, page 200
- **Sieve of Eratosthenes** (Ignore Segmented Sieve)

INTEGER FACTORIZATION

- **CPH, page 197-199** (Give it a quick read, it mostly contains mathematics that you must have studied before like number of factors of a number etc)
- Prime Factorization of a number can be done using trial division method in **O(sqrt(n))** time.
- However, again for multiple queries, it may not be optimal. So we can precompute the smallest prime factor **(spf)** of every number from 1 to n. This takes **O(nloglogn)** time.
- Suppose we have a number x, one of its prime factors will be spf(x). And rest of the prime factors will be contained in x/spf(x). We can do this recursively to find all prime factors in O(logn) time for each query.
- **Precomputing SPF**
- ***Integer factorization**

**Optional topics ! Study only if you have time .*

MODULAR ARITHMETIC

An essential part of CP is working with remainders of integers with a particular number, instead of working with integers themselves. This is done to prevent integer overflows in built-in data types.

A introduction to modular arithmetic along with sufficient practice problems can be found here :

- CPH, page 201
- **Modular Arithmetic**

Note: In C++, -8%7=-1 and not 6. You have to add 7 at the end and take mod again. In general, n mod m= (n%m+m)%m.

BONUS TIP FOR PYTHON USERS

The python inbuilt function "**pow(a,b,mod)**" calculates (a^b)%mod in **O(logb)** complexity itself. So python users can directly use this function instead of writing the code for binary exponentiation, **however, it is recommended that you know how to do the binary exponentiation manually.**

**Optional topics ! Study only if you have time .*

Congratulations on completing Week 3 of our Competitive Programming course this semester! We hope you've found this journey of problem-solving and algorithmic thinking both enlightening and rewarding.

As we wrap up this course for the current semester, we want to take a moment to express our appreciation for your hard work, dedication, and enthusiasm. You've demonstrated remarkable progress, and we're confident that your newfound skills will serve you well in your programming endeavors.

What's Next ?

While this semester's course may be coming to an end, we're excited to announce that we will be offering another such course on Competitive Programming, next semester. You will be introduced to Recursion, Dynamic Programming and many more topics which are present in one form or other in almost all questions.

General Advice

- It's important to follow the course and participate in contests regularly. If you couldn't solve a problem during the contest, give it another try after the contest ends. Consistent practice and "upsolving" can significantly improve your competitive programming skills.
- If you turn to reading editorial, read hints/partial solution and give it a try again. Implementing solutions on your own guarantees that you properly understood the problem and the solution.
- Maintain healthy competition, Discuss your solutions with your friends after contest ends. Doing CP with friends not only makes it more exciting and fun but also helps in discovering different ways to approach a problem.