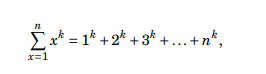
**Competitive Programming**

1. A.P: Arithmetic Progression, to get the sum of numbers where the diff is constant.





and for xk



There’s a general formula for sums, Faulhaber’s formula.

General formula for AP series:



1. G.P: A geometric progression is a series when ratio between any 2 consecutive numbers is constant.





1. Harmonic Sum: …
2. Set Theory:
3. log:
4. function:
5. logic:
6. Time Complexity(TC): Denoted by O(…).

… can be

1: Constant time, no matter input size, output will take same time.

log2n: meaning the alg. halves the inp size at each step.

n1/2

n: linear time, for big inputs this is usually the most efficient alg as this means 1 loop for input, 1 for processing and 1 loop for output and all go n times.

nlogn: Indicates sorting alg, as the alg takes log n time on each element.

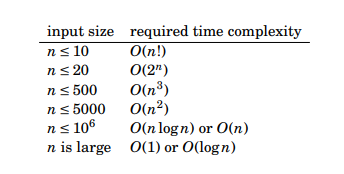
nk: k loops

2n: Indicates alg iterates through all subsets of the input set.

n!: Indicates alg iterates through all permutations of the input.

An alg is polynomial if at most it takes O(nk) time.

* 1. Inp Size and Required time complexity of the alg:



1. Sort: We use sort(begin iterator, end iterator); For user-defined types we overload ‘<’ operator. We can provide a bool function as predicate or as address to create custom comparator.

struct P {

int x, y;

bool operator<(const P &p) {

if (x != p.x) return x < p.x;

else return y < p.y;

}

};

1. Searching: To find an element we can iterate through the array, O(n) TC but if the array is sorted, using Binary search we can do it in O(logn) time.

Efficient BS:

int k = 0;

for (int b = n/2; b >= 1; b /= 2) {

while (k+b < n && array[k+b] <= x) k += b;

}

if (array[k] == x) {

// x found at index k

}

* 1. lower\_bound: returns a pointer to first array element whose value is atleast x.
  2. upper\_bound: returns a ptr to the first array element whose value is larger than x.
  3. equal\_range: returns both of those ptrs.
  4. Binary search can be used to find the lowest/highest value of a function as well.

1. Comparison: For 2 arrays

A = [5,2,8,9,4] and B = [3,2,9,5],

* 1. Going through each element in an array and then looking for it in another takes O(n2) time.
  2. Putting array A in a set and then going through each element of B and checking if it exists in set of A takes O(nlogn) time.
  3. The same as above but with unordered set takes O(n).
  4. Sorting both arrays and checking values at the same indices for both of them takes O(nlogn), nlogn for sorting and n for the comparison. This is the most efficient solution as sorting is done once but the rest of the alg is in linear time.

1. Complete Search: A general method to solve almost any algorithm problem, we look for all possible solutions using brute-force type approach and then select the best ones from them. It’s time taking and is a very general approach hence we may prefer a Greedy Algorithm or Dynamic Programming to solve a problem.