SoC 2023: Competitive Programming

Week-2: Sorting, Searching, and Number Theory

Mentor: Virendra Kabra

Summer 2023

Contents

1 Sorting

1.1 C++

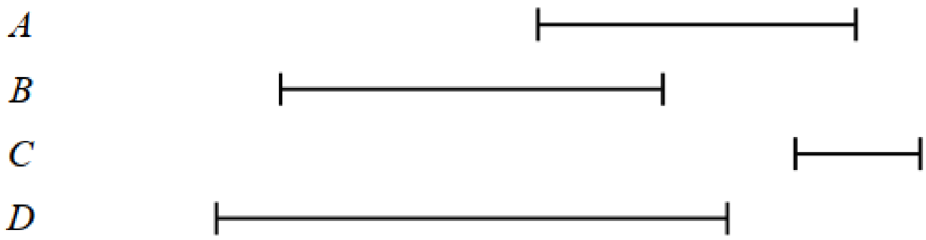
- To sort vectors or strings, use `sort` from the STL. References: GFG, cplusplus.com.
- For n items, number of operations is $O(n \log n)$.
- To sort a vector of custom structs, use a comparator function. Example in file.

1.2 Algorithms

- Comparison-based: Bubblesort $O(n^2)$, Mergesort $O(n \log n)$, Quicksort - average $O(n \log n)$, worst $O(n^2)$
- Counting sort: If all elements are in an interval of size $O(n)$, maintain a frequency array or `unordered_map`, and finally list elements in order. Complexity $O(n)$.

1.3 Examples

- Find number of unique elements in an array. $O(n \log n)$ with sorting or `set`, $O(n)$ with `unordered_set`.
- Interval scheduling. Given a list of intervals with respective start and end times, report the maximum number of non-overlapping intervals.

  Here, $\{B, C\}$ or $\{D, C\}$ are optimal. $\{A, C\}$ is not valid, while $\{A\}$ is sub-optimal.

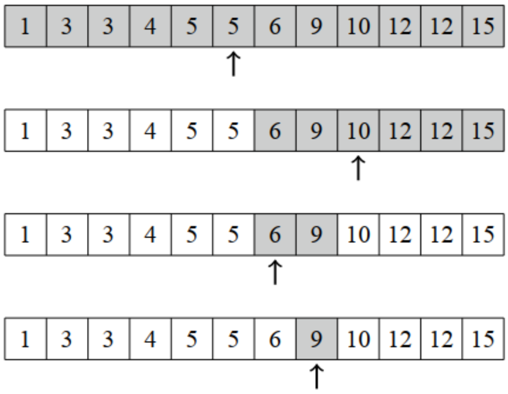
  A "greedy" solution: always choose the next interval with smallest finish time (this requires sorting). The algorithm is greedy in the sense of local optimization. It is also globally optimal: for any schedule that you pick, we can replace the first interval with an interval that ends earlier and repeat the process.

- Find the maximum number of overlapping intervals: Sort all start and end times in a single vector, with information if it is start/end. Iterate over and maintain a counter: $+1$ for start, -1 for end. Max counter value is the answer.

2 Binary Search

2.1 Introduction

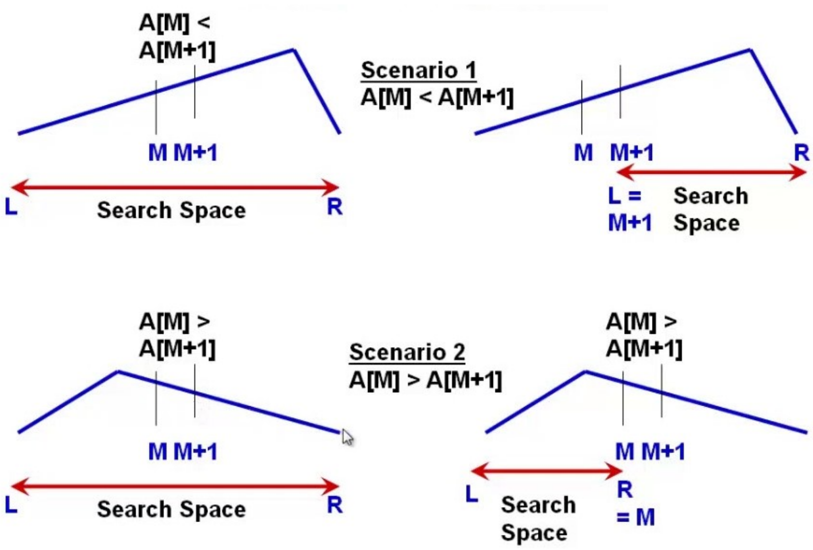
- Search for an element in a **sorted** array. Search range halves in every iteration, so going from space of size n to that of size 1 takes $O(\log_2 n)$ iterations.



- Code in file.

2.2 Examples

- Find the maximum value in a unimodal array. Use the interval-halving idea with a different condition.



- Painters' Partition Problem

  We have to paint n boards of lengths $\{A_1, A_2, \ldots, A_n\}$. k painters are available and each takes 1 unit time to paint 1 unit of board. Find the minimum time to get the job done under the constraint that any painter will only paint continuous sections of boards.
  Example: Lengths $\{9, 4, 7, 10, 5\}$ with $k = 3$. Allocations $\{9\}, \{4, 7\}, \{10, 5\}$ and $\{9, 4\}, \{7\}, \{10, 5\}$ are optimal, while $\{9, 4\}, \{7, 10\}, \{5\}$ isn't.

  Maximum time taken by any painter is the answer. Assuming any number of painters, an initial *range* on this is $\left[\max_i A_i, \sum_i A_i\right]$ - with n and 1 painters respectively.
  A number t is a candidate answer if we can assign contiguous segments to $\leq k$ painters, each of length $\leq t$. If t works, then any number $\geq t$ works. So, we have an array like the following

Candidate	max A_i	2	3	...	answer	...	$\sum A_i$
Works?	N	N	N	N	Y	Y	Y

Candidates are sorted, so we can use binary search. To check if a candidate works, need to iterate over the array in $O(n)$. The overall complexity is $O(n \log(\sum A_i - \max A_i))$.

Code in file. We can start with a much larger initial interval such as $[0, \text{INT\_MAX}]$; idea remains the same.

3 Number Theory

3.1 Factors

- Factors of a number n: Iterate from 1 to $\sqrt{n}$. If `n%i == 0`, then i and n/i are factors.
- Primes: Sieve of Eratosthenes. For example, we need primes from L to R. Check the reference for a simple implementation.

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~	11	~~12~~	13	~~14~~	15	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~
2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~	11	~~12~~	13	~~14~~	~~15~~	~~16~~

- Prime decomposition of n: Use the sieve to get primes in $[2, \sqrt{n}]$ and test with each prime. Implementation.

3.2 Combinatorics

- Modular arithmetic: $a \equiv (a\%m) \pmod{m}$. m is usually a large prime to ease later calculations.
- Binary exponentiation: $a^b \pmod{m}$ in $O(\log_2 b)$. Code in file.
- Inverse Modulo: With Euler's Totient function ϕ, $a^{\phi(m)} \equiv 1 \pmod{m}$. For prime m, this is $a^{m-1} \equiv 1 \pmod{m}$. Further, if $\gcd(a, m) = 1$, we get $a^{m-2} \equiv a^{-1} \pmod{m}$.
  So $a^{-1} \pmod{m}$ is equivalent to $a^{m-2}\%m$ - use binary exponentiation.
- $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. Modulo prime m, we use precomputed factorials and inverse modulo.
- Resources: Binary Exponentiation, Modular Inverse, Binomial Coefficients

4 Todos

- First 5 problems from CSES (Sorting and Searching)
- Codeforces: 1612C, 1613C, 1610C