Competitive Programming and Mathematics Society

Intro to Competitive Programming

CPMSoc

Welcome

- Mathematics workshops will run every even-numbered week (4, 6, 8, ...)
- Programming ones will run every odd-numbered week (3, 5, 7, ...)
- Slides will be uploaded on our website (unswcpmsoc.com)

Workshop Overview

- Key concepts
- Application to Binary Search
- Demonstration
- Solve problems together

Conceptual toolbox

- Time Complexity
- Precomputation
- Solution space
- Invariants

Time Complexity

- Big "O" Notation
- Asymptotic Time Complexity
- Worst-Case Time Complexity

If you change the input size, how does it change the runtime of your algorithm?

Counting Dogs

let's say you want to find how many people on a street own a dog.

Counting Dogs

let's say you want to find how many people on a street own a dog.

Approach: Knock at each house

Counting Dogs

let's say you want to find how many people on a street own a dog.

Approach: Knock at each house
Time complexity:

- Double the houses => Double the time

Counting Dogs

let's say you want to find how many people on a street own a dog.

Approach: Knock at each house
Time complexity:

- Double the houses => Double the time
- Grows "Linearly" or $O(n)$ (like a line)

Counting Dogs 2

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

Counting Dogs 2

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

Approach: For each dog-owner, ask every other dog owner if they have the same breed.

Counting Dogs 2

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

Approach: For each dog-owner, ask every other dog owner if they have the same breed.
Time complexity:

- Double the houses => Quadruple the time

Counting Dogs 2

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

Approach: For each dog-owner, ask every other dog owner if they have the same breed.
Time complexity:

- Double the houses => Quadruple the time
- Grows "Quadratically" or $O(n^2)$ (like a square)

Higher or Lower

Let's play a game of Higher or Lower.

What is the optimal strategy?

Higher or Lower

Let's play a game of Higher or Lower.

- I have a number between 1 and 1000.

What is the optimal strategy?

Higher or Lower

Let’s play a game of Higher or Lower.

- I have a number between 1 and 1000.
- After each guess you make, I’ll tell you if my number is higher or lower than your guess.

What is the optimal strategy?

Higher or Lower

Observations:

-

Higher or Lower

Observations:

- The best strategy is to pick the midpoint every time

Higher or Lower

Observations:

- The best strategy is to pick the midpoint every time
- The number of remaining options halves each time

Higher or Lower

Observations:

- The best strategy is to pick the midpoint every time
- The number of remaining options halves each time
- Never need more than $\log_2(\text{number of options})$ moves

Higher or Lower

Observations:

- The best strategy is to pick the midpoint every time
- The number of remaining options halves each time
- Never need more than $\log_2(\text{number of options})$ moves

Time complexity:

- Double the number of options => 1 extra guess

Higher or Lower

Observations:

- The best strategy is to pick the midpoint every time
- The number of remaining options halves each time
- Never need more than $\log_2(\text{number of options})$ moves

Time complexity:

- Double the number of options => 1 extra guess
- Grows "Logarithmically" or $O(\log n)$ (basically nothing)

Mathematically

"Rules":

-

Mathematically

"Rules":

- Constants dont matter: $O(n+3) = O(n)$

Mathematically

"Rules":

- Constants dont matter: $O(n + 3) = O(n)$
- Coefficients dont matter: $O(42n) = O(n)$

Mathematically

"Rules":

- Constants dont matter: $O(n+3) = O(n)$
- Coefficients dont matter: $O(42n) = O(n)$
- Small things dont matter: $O(n^2+n) = O(n^2)$

Time Complexity

- Time complexity captures relationship of input size => runtime

Time Complexity

- Time complexity captures relationship of input size => runtime
- There are more subtleties and nuances to the topic

Time Complexity

- Time complexity captures relationship of input size => runtime
- There are more subtleties and nuances to the topic
- Which you'll learn in COMP2521 and COMP3821

Time Complexity

- Time complexity captures relationship of input size => runtime
- There are more subtleties and nuances to the topic
- Which you'll learn in COMP2521 and COMP3821
- I'll focus on how you'd practically use it

Will it run in One Second?

Will it run in One Second?

A good rule of thumb is the Magic Number of 100 Million or 10^8.

Will it run in One Second?

A good rule of thumb is the Magic Number of 100 Million or 10^8.

Plug the maximum value of n into the time complexity equation, and if it goes over 100 Million then your algorithm probably won't run in time.

Will it run in One Second?

A good rule of thumb is the Magic Number of 100 Million or 10^8.

Plug the maximum value of n into the time complexity equation, and if it goes over 100 Million then your algorithm probably won't run in time.

- For example, an $O(n^2)$ algorithm when $n \leq 100,000$
- $(100,000)^2 > 10^8$ So it's too slow.

Will it run in One Second?

A good rule of thumb is the Magic Number of 100 Million or 10^8.

n	Possible Complexities
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(n \cdot 2^n)$
$n \leq 400$	$O(n^3)$
$n \leq 10^4$	$O(n^2)$
$n \leq 10^5$	$O(n\sqrt{n})$ or $O(n \log^2 n)$
$n \leq 10^6$	$O(n \log n)$
$n \leq 10^7$	$O(n)$
$n \leq 10^9+$	$O(\log n)$ or $O(1)$

Precomputation

Precomputation

- Process the data beforehand to save time later

Precomputation

- Process the data beforehand to save time later
- Structure your data, specific for the queries you need

Counting Dogs 2, Revisted

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

Counting Dogs 2, Revisted

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

Approach 2: Collect a list of the breeds of each dog. For each dog you have, count how many other dogs of the same breed there are in the list

Counting Dogs 2, Revisted

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

Approach 2: Collect a list of the breeds of each dog. For each dog you have, count how many other dogs of the same breed there are in the list

Still need to look at every dog to check each dog!

Counting Dogs 2, Revisted

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

Approach 2: Collect a list of the breeds of each dog. For each dog you have, count how many other dogs of the same breed there are in the list

Still need to look at every dog to check each dog!

- Double the dogs => Quadruple the time

Counting Dogs 2, Revisted

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

Approach 2: Collect a list of the breeds of each dog. For each dog you have, count how many other dogs of the same breed there are in the list

Still need to look at every dog to check each dog!

- Double the dogs => Quadruple the time
- Grows "Quadratically" or $O(n^2)$

Counting Dogs 2, Revisted

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

We want it to be easy to find duplicates

Counting Dogs 2, Revisted

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

We want it to be easy to find duplicates

Sort the list! This takes $O(n \log n)$ time

Counting Dogs 2, Revisted

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

We want it to be easy to find duplicates

Sort the list! This takes $O(n \log n)$ time

- Count the length of each same-breed run.

Counting Dogs 2, Revisted

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

We want it to be easy to find duplicates

Sort the list! This takes $O(n \log n)$ time

- Count the length of each same-breed run.

Counting Dogs 2, Revisted

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

We want it to be easy to find duplicates

Sort the list! This takes $O(n \log n)$ time

- Count the length of each same-breed run.
- This runs in $O(n)$ time.

Counting Dogs 2, Revisted

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

We want it to be easy to find duplicates

Sort the list! This takes $O(n \log n)$ time

- Count the length of each same-breed run.
- This runs in $O(n)$ time.

Time complexity:

- Double the dogs => Double the time (plus a bit)

Counting Dogs 2, Revisted

Problem: you want to figure out what's the chance that two random dog owners on the street have the same breed of dog.

We want it to be easy to find duplicates

Sort the list! This takes $O(n \log n)$ time

- Count the length of each same-breed run.
- This runs in $O(n)$ time.

Time complexity:

- Double the dogs => Double the time (plus a bit)
- Grows "log-linearly" $O(n \log n)$

Finding a Value

Problem: Given a list of the phone numbers stored on your phone (q of them), and a phonebook of length n, find out how many numbers from your list appear in the phonebook.

Finding a Value

Problem: Given a list of the phone numbers stored on your phone (q of them), and a phonebook of length n, find out how many numbers from your list appear in the phonebook.

Approach 1: For each phone number, search through the entire phonebook until you find it.

Finding a Value

Problem: Given a list of the phone numbers stored on your phone (q of them), and a phonebook of length n, find out how many numbers from your list appear in the phonebook.

Approach 1: For each phone number, search through the entire phonebook until you find it.

Time complexity is $O(nq)$.

Finding a Value

Problem: Given a list of the phone numbers stored on your phone (q of them), and a phonebook of length n, find out how many numbers from your list appear in the phonebook.

Approach 2: Sort the phonebook and use higher-or-lower to search for each of your numbers.

Finding a Value

Problem: Given a list of the phone numbers stored on your phone (q of them), and a phonebook of length n, find out how many numbers from your list appear in the phonebook.

Approach 2: Sort the phonebook and use higher-or-lower to search for each of your numbers.

Time complexity is $O(n \log n + q \log n)$.

Binary Search Code

CPMSOC

```
int binary_search(int arr[100000], int val) {
    int left = 0;
    int right = 100000;
    while (right - left > 1) {
        int mid = (left + right) / 2;
        if (arr[mid] <= val) left = mid;
        else right = mid;
    }
    return left;
}
```

Invariants

Invariants

- `arr[L]` $<$ `val` is true at every iteration.
- `arr[R]` $\geq$ `val` is true at every iteration.

Invariants

- `arr[L]` $<$ `val` is true at every iteration.
- `arr[R]` $\geq$ `val` is true at every iteration.
- After each iteration, the remaining interval halves

Medusa’s Snakes

Medusa has snakes instead of hair. Each of her snakes DNA is represented by an uppercase string of letters. Each letter is one of S, N, A, K or E. Your extensive research shows that a snakes venom level depends on its DNA. A snake has venom level x if its DNA:

- has exactly 5x letters
- begins with x copies of the letter S
- then has x copies of the letter N
- then has x copies of the letter A
- then has x copies of the letter K
- ends with x copies of the letter E.

By deleting zero or more letters from the DNA, what is the maximum venom level this snake could have?
The length of the DNA is at most 100 000.

Toolbox - data structures

- Arrays
- Sorted arrays
- Linked lists
- Graphs
- Hash table
- Cache/memoization

Toolbox - algorithms

- Binary search
- Graph search algorithms
- Greedy algorithms
- Recursion/divide and conquer
- Dynamic programming
- String algorithms
- Sorting and searching

Attendance form :D

Feedback form :D

Further events

Please join us for:

- Social session tomorrow 4pm
- Math workshop next week
- Programming workshop in two weeks