Binary Search
Not your parents’ binary search

Ethan Arnold, Arnav Sastry

CS 104C

Spring 2017
How to write binary search

- **Java:**

  ```java
  Collections.binarySearch(list, value);
  ```

- **C++:**

  ```cpp
  lower_bound(vec.begin(), vec.end(), value);
  ```

- **Python:**

  ```python
  bisect.bisect_left(arr, value)
  ```
How to write binary search

- **Java**: Collections.binarySearch(list, value);
How to write binary search

- **Java**: `Collections.binarySearch(list, value);`
- **C++**: `lower_bound(vec.begin(), vec.end(), value);`
How to write binary search

- **Java**: Collections.binarySearch(list, value);
- **C++**: lower_bound(vec.begin(), vec.end(), value);
- **Python**: bisect.bisect_left(arr, value)
Why are we covering binary search?

Problem:
Given a boolean array $A$ (with length $n \leq 10^6$), what is the largest subarray of 0s you can find if you are allowed to change at most $0 \leq k \leq n$ values in the array?

Idea:
Write a function $p: \mathbb{Z} \geq 0 \rightarrow \{T, F\}$ with $p(m) = T$ iff there is a subarray of length $m$ with at most $k$ 1s.

Claim:
If $p(m) = T$, then $p(m-1) = T$; if $p(m) = F$, then $p(m+1) = F$ (these are actually equivalent statements).

Why is this true?
If we have a window of size $m$ with $\leq k$ 1s, then we can easily find a window of size $m-1$ with $\leq k$ 1s — just shrink that same window.

If every window of size $m$ has $> k$ 1s, then every window of size $m+1$ will have $> k$ 1s.

Why is this helpful?
Why are we covering binary search?

- **Problem:** Given a boolean array $A$ (with length $1 \leq n \leq 10^6$), what is the largest subarray of 0s you can find if you are allowed to change at most $0 \leq k \leq n$ values in the array?
Why are we covering binary search?

- **Problem:** Given a boolean array $A$ (with length $1 \leq n \leq 10^6$), what is the largest subarray of 0s you can find if you are allowed to change at most $0 \leq k \leq n$ values in the array?

- **Idea:** Write a function $p : \mathbb{Z}_{\geq 0} \rightarrow \{ T, F \}$ with $p(m) = T$ iff there is a subarray of length $m$ with at most $k$ 1s.
Why are we covering binary search?

- **Problem:** Given a boolean array $A$ (with length $1 \leq n \leq 10^6$), what is the largest subarray of 0s you can find if you are allowed to change at most $0 \leq k \leq n$ values in the array?

- **Idea:** Write a function $p : \mathbb{Z}_{\geq 0} \rightarrow \{T, F\}$ with $p(m) = T$ iff there is a subarray of length $m$ with at most $k$ 1s.

- **Claim:** If $p(m) = T$, then $p(m - 1) = T$; if $p(m) = F$, then $p(m + 1) = F$ (these are actually equivalent statements).
  - Why is this true?
Why are we covering binary search?

- **Problem:** Given a boolean array $A$ (with length $1 \leq n \leq 10^6$), what is the largest subarray of 0s you can find if you are allowed to change at most $0 \leq k \leq n$ values in the array?

- **Idea:** Write a function $p : \mathbb{Z}_{\geq 0} \rightarrow \{ T, F \}$ with $p(m) = T$ iff there is a subarray of length $m$ with at most $k$ 1s.

- **Claim:** If $p(m) = T$, then $p(m - 1) = T$; if $p(m) = F$, then $p(m + 1) = F$ (these are actually equivalent statements).

  - Why is this true?
    - If we have a window of size $m$ with $\leq k$ 1s, then we can easily find a window of size $m - 1$ with $\leq k$ 1s — just shrink that same window.
    - If every window of size $m$ has $> k$ 1s, then every window of size $m + 1$ will have $> k$ 1s.
Why are we covering binary search?

- **Problem:** Given a boolean array $A$ (with length $1 \leq n \leq 10^6$), what is the largest subarray of 0s you can find if you are allowed to change at most $0 \leq k \leq n$ values in the array?

- **Idea:** Write a function $p : \mathbb{Z}_{\geq 0} \rightarrow \{T, F\}$ with $p(m) = T$ iff there is a subarray of length $m$ with at most $k$ 1s.

- **Claim:** If $p(m) = T$, then $p(m - 1) = T$; if $p(m) = F$, then $p(m + 1) = F$ (these are actually equivalent statements).
  
  - Why is this true?
  - If we have a window of size $m$ with $\leq k$ 1s, then we can easily find a window of size $m - 1$ with $\leq k$ 1s — just shrink that same window.
  - If every window of size $m$ has $> k$ 1s, then every window of size $m + 1$ will have $> k$ 1s.

- Why is this helpful?
**Problem**: Given a boolean array $A$ (with length $1 \leq n \leq 10^6$), what is the largest subarray of 0s you can find if you are allowed to change at most $0 \leq k \leq n$ values in the array?
Problem: Given a boolean array $A$ (with length $1 \leq n \leq 10^6$), what is the largest subarray of 0s you can find if you are allowed to change at most $0 \leq k \leq n$ values in the array?

Say we want to know if it’s possible for a fixed size $m \leq n$. 
Sliding Window Technique

- **Problem**: Given a boolean array $A$ (with length $1 \leq n \leq 10^6$), what is the largest subarray of 0s you can find if you are allowed to change at most $0 \leq k \leq n$ values in the array?
- Say we want to know if it’s possible for a fixed size $m \leq n$.
- Find the sum of the first $m$ elements.
Problem: Given a boolean array $A$ (with length $1 \leq n \leq 10^6$), what is the largest subarray of 0s you can find if you are allowed to change at most $0 \leq k \leq n$ values in the array?

Say we want to know if it’s possible for a fixed size $m \leq n$.

Find the sum of the first $m$ elements.

“slide” the window by adding the next element and subtracting the first element.
Formalization

- Consider a predicate function $p$ defined over the natural numbers.
- For any natural number $i$, $p(i) = T$ or $p(i) = F$. 
Formalization

- Consider a predicate function \( p \) defined over the natural numbers
- For any natural number \( i \), \( p(i) = T \) or \( p(i) = F \)
- \( p \) is also a sequence of \( T \) followed by a sequence of \( F \). Mathematically, it is \textit{monotonic}
Formalization

- Consider a predicate function $p$ defined over the natural numbers
- For any natural number $i$, $p(i) = T$ or $p(i) = F$
- $p$ is also a sequence of $T$ followed by a sequence of $F$. Mathematically, it is *monotonic*

$$
\begin{array}{ccccccccccc}
i & 0 & 1 & \cdots & j - 1 & j & j + 1 & \cdots & n - 2 & n - 1 \\
p(i) & T & T & \cdots & T & T & F & \cdots & F & F \\
\end{array}
$$
While the first binary search was published in 1946...
Binary search history

- “While the first binary search was published in 1946...
- the first binary search that works correctly for all values of n did not appear until 1962.”
Binary search history

▶ “While the first binary search was published in 1946...
▶ the first binary search that works correctly for all values of n did not appear until 1962.”
▶ A bug was discovered in Java’s binary search as recently as 2006!
Bug free binary search

**Function:** BinarySearch\( (p, \text{max\_input}) \)

- \( lo = 0 \)
- \( hi = \text{max\_input} \)
- assert \( p(lo) == T \)
- assert \( p(hi) == F \)
- while \( lo + 1 < hi \)
  - \( mid = \lfloor \frac{lo+hi}{2} \rfloor \)
  - if \( p(mid) == T \), then \( lo = mid \)
  - else, \( hi = mid \)

What can you guarantee about \( lo \) and \( hi \) throughout the program?

What is special about \( lo \) and \( hi \) at the end of the while loop?
Bug free binary search

- **Function**: BinarySearch($p$, $max\_input$)
  - $lo = 0$
  - $hi = max\_input$
  - assert $p(lo) == T$
  - assert $p(hi) == F$
  - while $lo + 1 < hi$
    - $mid = \lfloor \frac{lo + hi}{2} \rfloor$
    - if $p(mid) == T$, then $lo = mid$
    - else, $hi = mid$

- What can you guarantee about $lo$ and $hi$ throughout the program?
Bug free binary search

- **Function:** BinarySearch($p$, $max\_input$)
  - $lo = 0$
  - $hi = max\_input$
  - assert $p(lo) == T$
  - assert $p(hi) == F$
  - while $lo + 1 < hi$
    - $mid = \left\lfloor \frac{lo+hi}{2} \right\rfloor$
    - if $p(mid) == T$, then $lo = mid$
    - else, $hi = mid$

- What can you guarantee about $lo$ and $hi$ throughout the program?
- What is special about $lo$ and $hi$ at the end of the while loop?
Finding high

- Say we have a $p(i)$ function defined on all the natural numbers. For binary search to work, we need a $lo$ and a $hi$.
- How can we find that $hi$?
Finding high

- Say we have a $p(i)$ function defined on all the natural numbers. For binary search to work, we need a $lo$ and a $hi$.
- How can we find that $hi$?
- Start with $lo = 0$, $hi = 1$,
- Keep doubling $hi$ until $p(lo)! = p(hi)$. 

Escaping the natural numbers

- There is no need for $p(i)$ to be restricted to the natural numbers.
Escaping the natural numbers

▶ There is no need for \( p(i) \) to be restricted to the natural numbers
▶ Any totally ordered set can be used as a domain
Escaping the natural numbers

- There is no need for \( p(i) \) to be restricted to the natural numbers
- Any totally ordered set can be used as a domain
- This includes the real numbers!
Maximum average

- **Problem:** Given a list of $1 \leq n \leq 10^6$ integers, find a non-empty subarray with maximum average

- Idea: Let $p : \mathbb{R} \to \{T, F\}$ with $p(x) = T$ iff there exists a non-empty subarray with average at least $x$

- How can we compute this?

- Question: Can we binary search on $x$ (the answer)?

- How many iterations of binary search do we need?

- The online judge will tell us, but usually 200 or so is plenty for double-precision floats
Maximum average

- **Problem:** Given a list of $1 \leq n \leq 10^6$ integers, find a non-empty subarray with maximum average
  - The subarray must have at least $k$ elements
Maximum average

- **Problem:** Given a list of $1 \leq n \leq 10^6$ integers, find a non-empty subarray with maximum average
  - The subarray must have at least $k$ elements
- **Idea:** Let $p : \mathbb{R} \to \{ T, F \}$ with $p(x) = T$ iff there exists a non-empty subarray with average at least $x$
Problem: Given a list of $1 \leq n \leq 10^6$ integers, find a non-empty subarray with maximum average

- The subarray must have at least $k$ elements

Idea: Let $p : \mathbb{R} \rightarrow \{T, F\}$ with $p(x) = T$ iff there exists a non-empty subarray with average at least $x$

- How can we compute this?
Maximum average

- **Problem:** Given a list of $1 \leq n \leq 10^6$ integers, find a non-empty subarray with maximum average
  - The subarray must have at least $k$ elements
- **Idea:** Let $p : \mathbb{R} \rightarrow \{T, F\}$ with $p(x) = T$ iff there exists a non-empty subarray with average at least $x$
  - How can we compute this?
- **Question:** Can we binary search on $x$ (the answer)?
Maximum average

- **Problem:** Given a list of $1 \leq n \leq 10^6$ integers, find a non-empty subarray with maximum average
  - The subarray must have at least $k$ elements
- **Idea:** Let $p : \mathbb{R} \to \{T, F\}$ with $p(x) = T$ iff there exists a non-empty subarray with average at least $x$
  - How can we compute this?
- **Question:** Can we binary search on $x$ (the answer)?
  - How many iterations of binary search do we need?
Maximum average

- **Problem:** Given a list of \(1 \leq n \leq 10^6\) integers, find a non-empty subarray with maximum average
  - The subarray must have at least \(k\) elements
- **Idea:** Let \(p : \mathbb{R} \rightarrow \{T, F\}\) with \(p(x) = T\) iff there exists a non-empty subarray with average at least \(x\)
  - How can we compute this?
- **Question:** Can we binary search on \(x\) (the answer)?
  - How many iterations of binary search do we need?
  - The online judge will tell us, but usually 200 or so is plenty for double-precision floats