

Binary Search

The Vanilla Problem

Given: an **n**-character string consisting of a block of 'T' characters followed by a block of 'F' characters.

Find the index of the **last** 'T', or return -1 if one doesn't exist

The Vanilla Problem

Given: an **n**-character string consisting of a block of 'T' characters followed by a block of 'F' characters.

Find the index of the **last** 'T', or return -1 if one doesn't exist

Examples:

- TTTTFFFFFF: 3
- TTT: 2
- FFFFFF: -1

The Vanilla Problem

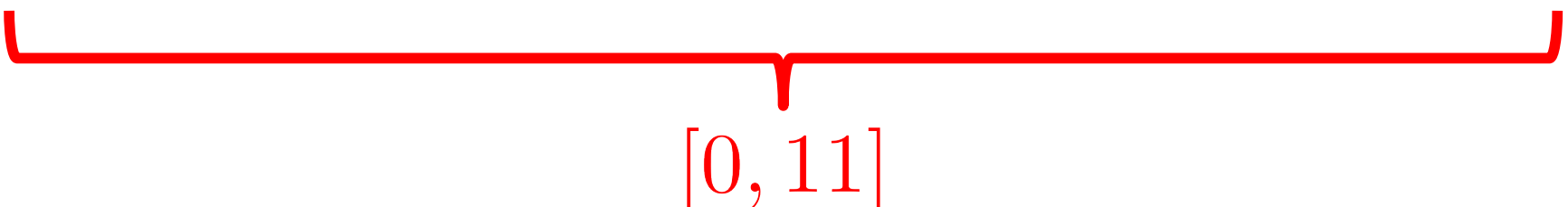
Given: an **n**-character string consisting of a block of 'T' characters followed by a block of 'F' characters.

Find the index of the **last** 'T', or return -1 if one doesn't exist

Trivial $O(n)$ solution using linear search. Can we do better than $O(n)$?

Divide and Conquer

i	0	1	2	3	4	5	6	7	8	9	10	11
s[i]	T	T	T	T	T	T	T	T	F	F	F	F



[0, 11]

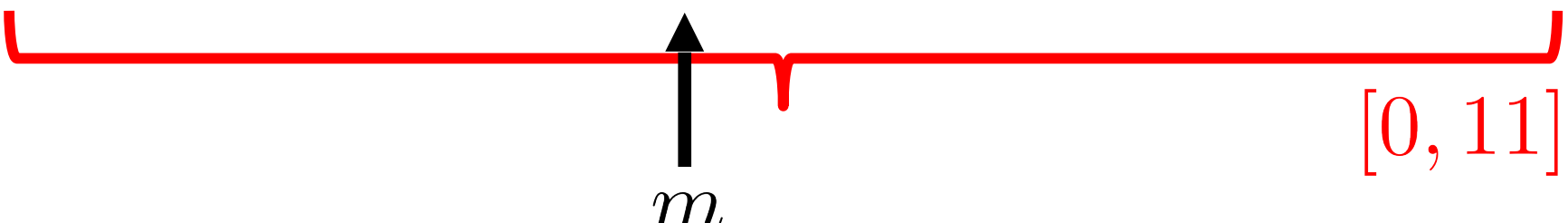
main idea: bracket the answer in interval

- **might** the answer
- $s[0] = T$ **cannot** be the answer
- answer **must** be in

[0, 11]

Divide and Conquer

i	0	1	2	3	4	5	6	7	8	9	10	11
s[i]	T	T	T	T	T	T	T	T	F	F	F	F



A red bracket is drawn below the array, spanning from index 0 to index 11. The label $[0, 11]$ is written in red to the right of the bracket. A black arrow points upwards from the label m to the element at index 5.

main idea: bracket the answer in $[a, b]$

compute **pivot** $m = \lfloor (a + b) / 2 \rfloor$

Divide and Conquer

i	0	1	2	3	4	5	6	7	8	9	10	11
s[i]	T	T	T	T	T	T	T	T	F	F	F	F

m

$[5, 11]$

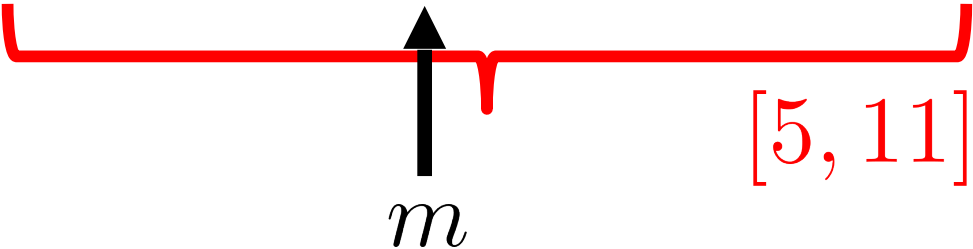
main idea: bracket the answer in $[a, b]$

compute **pivot** $m = \lfloor (a + b) / 2 \rfloor$

- if $s[m] = \text{T}$, set $a = m$

Divide and Conquer

i	0	1	2	3	4	5	6	7	8	9	10	11
s[i]	T	T	T	T	T	T	T	T	F	F	F	F



[5, 11]

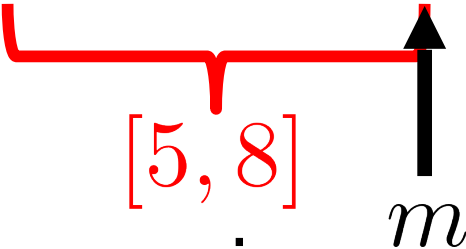
main idea: bracket the answer in $[a, b]$

compute **pivot** $m = \lfloor (a + b) / 2 \rfloor$

- if $s[m] = \text{T}$, set $a = m$

Divide and Conquer

i	0	1	2	3	4	5	6	7	8	9	10	11
s[i]	T	T	T	T	T	T	T	T	F	F	F	F



main idea: bracket the answer in

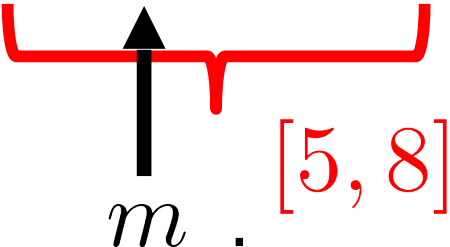
$[a, b]$

compute pivot $m = \lfloor (a + b) / 2 \rfloor$

- if $s[m] = T$, set $a = m$
- $s[m] = F$ $b = m$

Divide and Conquer

i	0	1	2	3	4	5	6	7	8	9	10	11
s[i]	T	T	T	T	T	T	T	T	F	F	F	F



m

$[5, 8]$

main idea: bracket the answer in

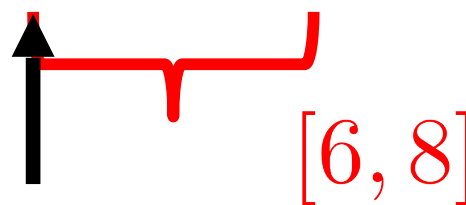
$[a, b]$

compute pivot $m = \lfloor (a + b) / 2 \rfloor$

- if $s[m] = T$, set $a = m$
- $s[m] = F$ $b = m$

Divide and Conquer

i	0	1	2	3	4	5	6	7	8	9	10	11
s[i]	T	T	T	T	T	T	T	T	F	F	F	F



main idea: bracket the answer in

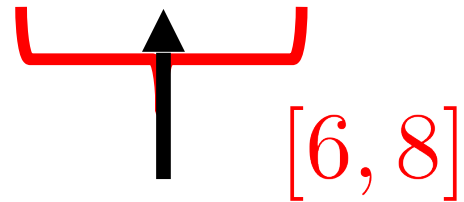
$[a, b]$

compute pivot $m = \lfloor (a + b) / 2 \rfloor$

- if $s[m] = T$, set $a = m$
- $s[m] = F$ $b = m$

Divide and Conquer

i	0	1	2	3	4	5	6	7	8	9	10	11
s[i]	T	T	T	T	T	T	T	T	F	F	F	F




main idea: bracket the answer in $[a, b]$

compute pivot $m = \lfloor (a + b) / 2 \rfloor$

- if $s[m] = T$, set $a = m$
- $s[m] = F$ $b = m$

Divide and Conquer

i	0	1	2	3	4	5	6	7	8	9	10	11
s[i]	T	T	T	T	T	T	T	T	F	F	F	F



main idea: bracket the answer in

$[a, b]$

compute pivot $m = \lfloor (a + b) / 2 \rfloor$

- if $s[m] = T$, set $a = m$
- $s[m] = F$ $b = m$

Implementation

search(s, low, hi)

invariant: $s[\text{low}] = T, s[\text{hi}] = F$

Implementation

```
search(s, low, hi)
while(true) {
    if(hi - low == 1)
        return low
    mid = (low + hi)/2;
    if(s[mid] == 'T') low = mid;
    else hi = mid;
}
```

invariant: $s[\text{low}] = \text{T}, s[\text{hi}] = \text{F}$

Implementation

```
search(s, low, hi)
while(true) {
    if(hi - low == 1)
        return low
    mid = (low + hi)/2;
    if(s[mid] == 'T') low = mid;
    else hi = mid;
}
```

invariant: $s[\text{low}] = \text{T}, s[\text{hi}] = \text{F}$

what's the bug?

Implementation

```
search(s, low, hi)
```

```
while(true) {
```

```
    if(hi - low == 1)
```

```
        return low
```

```
    mid = (low + hi)/2;
```

```
    if(s[mid] == 'T') low = mid;
```

```
    else hi = mid;
```

```
}
```

invariant: $s[\text{low}] = T, s[\text{hi}] = F$

better: $\text{low} + (\text{hi} - \text{low})/2$

Binary Search: Practical Issues

Time complexity: $O(\log n)$

Binary Search: Practical Issues

Time complexity: $O(\log n)$

- each iteration halves the interval

Binary Search: Practical Issues

Time complexity: $O(\log n)$

- each iteration halves the interval

Instead of a string **s**, can binary search on any predicate **bool P(int i)**

- “string” is given by **s[i] = P(i)**

Binary Search: Practical Issues

Time complexity: $O(\log n)$

- each iteration halves the interval

Instead of a string **s**, can binary search on any predicate **bool P(int i)**

- “string” is given by **s[i] = P(i)**
- ...but obviously, only evaluate P lazily

Binary Search: Practical Issues

Time complexity: $O(\log n)$

- each iteration halves the interval

Instead of a string **s**, can binary search on any predicate **bool P(int i)**

- “string” is given by $\mathbf{s}[i] = P(i)$
- ...but obviously, only evaluate P lazily
- important: what must be true about P ?

Binary Search: Practical Issues

Can take advantage of library implementations:

Java: `Collections.binarySearch(list, value)`

C++: `std::lower_bound(vec.begin(),
vec.end(), value)`

Python: `bisect.bisect_left(arr, value)`

Swimming Pool

Your backyard contains bushes at each integer lattice point (i,j) , and you want to build an elliptic pond of size r given by

What is the largest pond (value of r) that requires cutting down $\leq k$ bushes?

Bounds: $0 \leq k \leq 1000$

The answer should have relative error no greater than $1e-6$

Binary Search on Doubles

Same algorithm: bracket answer in $[a, b]$

Binary Search on Doubles

Same algorithm: bracket answer in $[a, b]$

Terminate when $|b - a| < \epsilon$

Binary Search on Doubles

Same algorithm: bracket answer in $[a, b]$

Terminate when $|b - a| < \epsilon$

How to pick initial **b**?

Binary Search on Doubles

Same algorithm: bracket answer in $[a, b]$

Terminate when $|b - a| < \epsilon$

How to pick initial **b**?

- start with arbitrary value (e.g. **b** = 1)
- keep doubling until **P(b)** = F

Maximum Average

Given an array of **n** integers ($n < 10^6$) and an integer **k**, find the subarray of size $\geq k$ with highest average

Maximum Average

Given an array of **n** integers (**n** < 10⁶) and an integer **k**, find the subarray of size **>= k** with highest average

Hint:

$$\frac{a_1 + a_2 + \cdots + a_m}{m} \geq t \Leftrightarrow (a_1 - t) + (a_2 - t) + \cdots + (a_m - t) \geq 0$$