Introduction to Algorithms and Data Structures

Lesson 3: Searching (1) Sequential search

Professor Ryuhei Uehara,
School of Information Science, JAIST, Japan.
uehara@jaist.ac.jp
http://www.jaist.ac.jp/~uehara

How to tackle the problem

- Consider data structure and how to store data
  - Data are in an array in any ordering
  - Data are in an array in increasing order
- Search algorithm: The way of searching
  - Sequential search
  - m-block method
  - Double m-block method
  - Binary search
- Analysis of efficiency
  - Big-O notation

Search Problem

- Problem: *S* is a given set of data. For any given data *x*, determine efficiently if *S* contains *x* or not.

- Efficiency: Estimate the time complexity by $n = |S|$, the size of the set *S*
  - In this problem, "checking every data in S" is enough, and this gives us an upper bound $O(n)$ in the worst case.

Roughly, "the running time is proportional to *n*."

Data structure 1
Data are stored in arbitrary ordering

- Each element in the set *S* is stored in an array s from s[0] to s[*n*-1] in any arbitrary ordering.

s[]=	37	12	25	9	87	33	65	3	29

Sequential search

- Input: any natural number *x*
- Output:
  - If there is i such that s[i] == *x*, output i
  - Otherwise, output -1 (for simplicity)

```
for (i=0; i<n; ++i)
   if(x==s[i]) return i;
return -1;
```

In the worst case, we need n comparisons.
Thus, the running time is proportional to n.
→ O(n) time algorithm

Precise time complexity of sequential search

- At most 3n + 2 steps

Initialization of i takes 1 operation

```
for (i=0; i<n; ++i)
    if(x==s[i]) return i;
return -1;
```

For the number of loops ≦ n,

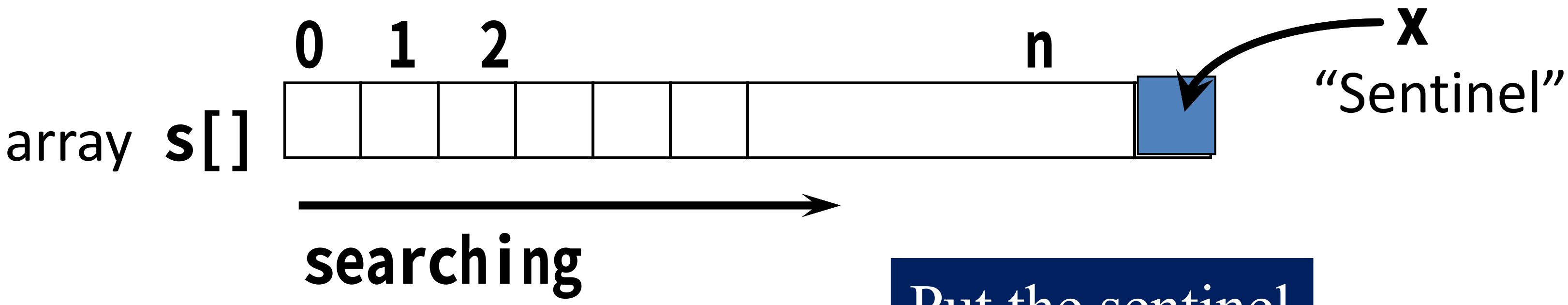
comparison ×2 (==, <)

increment ×1 (++)

Return takes 1 operation

Programming tips 1: simplify by using “sentinel”

Before searching, push *x* itself at the end of the array;
Then you definitely have **x==s[i]** for some **0<=i<=n**
So you do not need the check **i<n any more.**




```
s[n] = x;
i = 0;
while(x != s[i])
 i = i+1;
if(i < n) return i;
else      return -1;
```

Put the sentinel

Simple loop!
→ 2 operations

At most 2n+4 operations
$=O(n)$

Analysis of the number of comparisons

```
s[n] = x;
i = 0;
while(x!=s[i])
 i = i+1;
if(i < n)
  return i;
else
  return -1;
```

- The best case: 1 time
  - In the case of s[0] == x
- The worst case: n times
  - x is not in s[0]…s[n-1]
- The average case: $\sum_{i=1}^{n+1} \frac{i}{n} = \frac{n+2}{2}$
  - The expected value of # of comparisons
  - The i-th element is compared with probability 1/n
  - The number of comparisons when x is equal to the i-th element is i.

Randomized algorithm

Flip a fair coin, and
- “H”: search from s[0] forwardly
- “T”: search from s[n−1] backwardly

Intuition:

For any (sometimes fixed or unbalanced) input, the average case occurs on average.

RANDOMIZED ALGORITHM

The behavior depends on random numbers.
The worst case occurs with low probability.

Data structure 2
Data in the array in increasing order

- s[]=

3	9	12	25	29	33	37	65	87

- Q: Any improvement in sequential algorithm?

```
s[n]=x;
i = 0;
while(x!=s[i])
 i = i+1;
if(i < n) return i;
else      return -1;
```

We can stop when s[i] is greater than x

x!=s[i] ➔ x>s[i]

Data structure 2
Data in the array in increasing order

- s[]=

3	9	12	25	29	33	37	65	87

- Q: Any improvement in sequential algorithm?

```
s[n]=x;
i = 0;
while(s[i]<x)
 i = i+1;
if(i < n) return i;
else      return -1;
```

We can stop when s[i] is greater than x

x!=s[i] ➔ x>s[i]

Data structure 2
Data in the array in increasing order

- s[]=

3	9	12	25	29	33	37	65	87

- Q: Any improvement in sequential algorithm?

```
s[n]=x;
i = 0;
while(s[i]<x)
 i = i+1;
if(i < n) return i;
else      return -1;
```

We can stop when s[i] is greater than x
x!=s[i] ➔ x>s[i]

It may stop even if i<n
i<n ➔ s[i]==x

Data structure 2
Data in the array in increasing order

- s[]=

3	9	12	25	29	33	37	65	87

- Q: Any improvement in sequential algorithm?

```
s[n]=x;
i = 0;
while(s[i]<x)
 i = i+1;
if(s[i]==x) return i;
else        return -1;
```

We can stop when s[i] is greater than x
x!=s[i] → x>s[i]

It may stop even if i<n
i<n → s[i]==x

Data structure 2
Data in the array in increasing order

- s[]=

3	9	12	25	29	33	37	65	87

- Q: A … equential algorithm?

When x is not in s[], it returns n
`s[n]=x` ➔ `s[n]=x+1`

```
s[n]=x;
i = 0;
while(s[i]<x)
 i = i+1;
if(s[i]==x) return i;
else       return -1;
```

We can stop when s[i] is greater than x
`x!=s[i]` ➔ `x>s[i]`

It may stop even if `i<n`
`i<n` ➔ `s[i]==x`

Data structure 2
Data in the array in increasing order

- s[]=

3	9	12	25	29	33	37	65	87

- Q: A … equential algorithm?

When x is not in s[], it returns n
s[n]=x → s[n]=x+1

```
s[n]=x;
i = 0;
while(s[i]<x)
 i = i+1;
if(s[i]==x) return i;
else        return -1;
```

We can stop when s[i] is greater than x
x!=s[i] → x>s[i]

It may stop even if i<n
i<n → s[i]==x

Data structure 2
Data in the array in increasing order

- s[]=

3	9	12	25	29	33	37	65	87

- Q: A … equential algorithm?

When x is not in s[], it returns n
s[n]=x ➔ s[n]=x+1

```
s[n]=x+1;
i = 0;
while(s[i]<x)
 i = i+1;
if(s[i]==x) return i;
else       return -1;
```

We can stop when s[i] is greater than x
x!=s[i] ➔ x>s[i]

It may stop even if i<n
i<n ➔ s[i]==x

Data structure 2
Data in the array in increasing order

- s[]=

3	9	12	25	29	33	37	65	87

  – Exit from loop when: s[i]≧x
  – Check after loop: s[i]==x
  – Sentinel: greater than x, e.g., x+1

```
s[n]=x+1;
i = 0;
while(s[i]<x)
 i = i+1;
if(s[i]==x) return i;
else        return -1;
```

Q. Improve of comparison?

A. Average is better.
But the same in
the worst case