# Introduction to Algorithms and Data Structures

# Lesson 5: Searching (3)

## Binary Search and Hash method


Professor Ryuhei Uehara,

School of Information Science, JAIST, Japan.

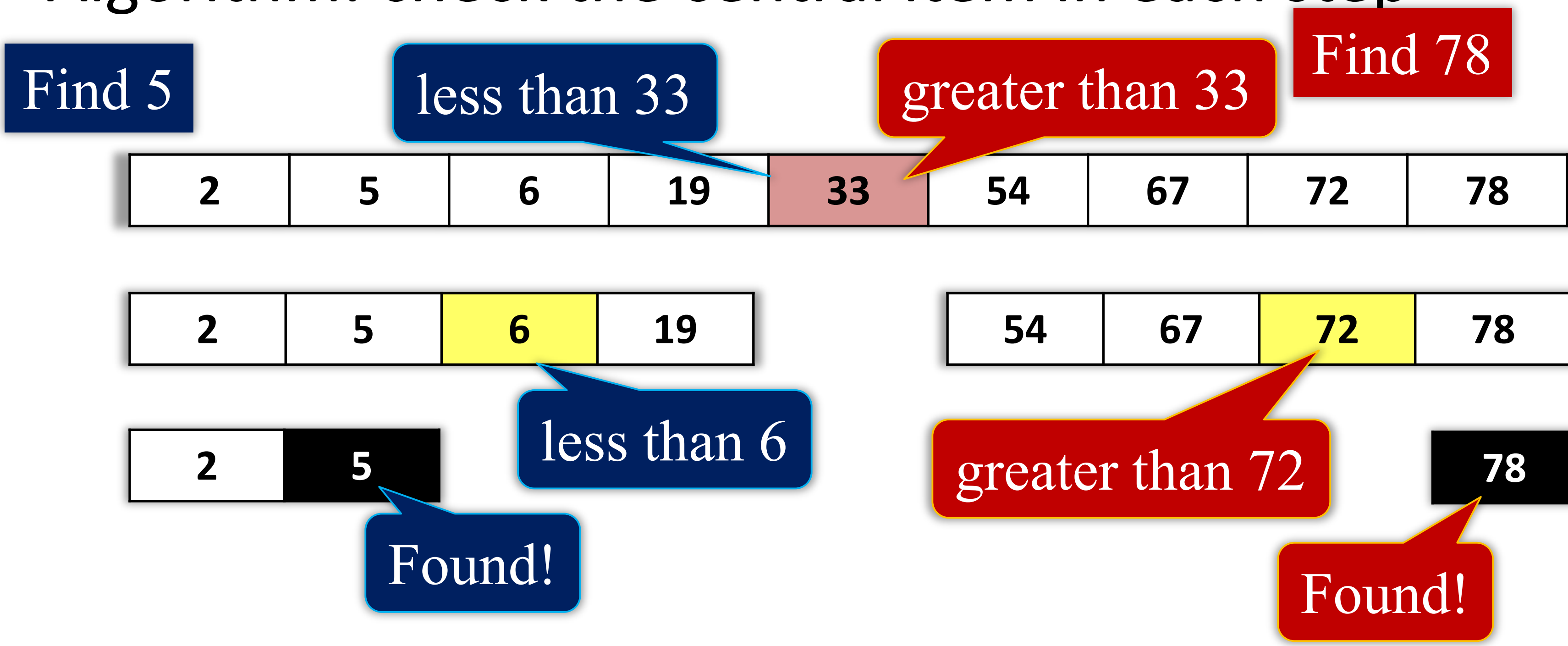
uehara@jaist.ac.jp

http://www.jaist.ac.jp/~uehara

# Binary search

Input: Array s[] such that data are in increasing order

Algorithm: check the central item in each step




– Divide at <u>center</u> in each step!

# Binary Search

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 6 | 19 | 33 | 54 | 67 | 72 | 78 |

| | | | |
|---|---|---|---|
| 2 | 5 | 6 | 19 |

| | |
|---|---|
| 2 | 5 |

- In the interval [left, right], compare the central item s[mid] with desired value x
  - x > s[mid] ➔ Search in the right half
    left = mid+1; (right is not changed)
  - x < s[mid] ➔ Search in the left half
    (left is not changed), right = mid-1
  - x = s[mid] ➔ Found!
- Repeat above until interval becomes empty

# Binary Search Algorithm

```
BinarySearch(x,s[]){
  left=0; right=n-1;
  do{
    mid = (left+right)/2;
    if x < s[mid] then
      right = mid-1;
    else
      left = mid+1;
  }while(x != s[mid] && left≦right);
  if x == s[mid] then return mid;
  else return -1;
}
```

Search interval

Find the center of the interval

In former half?

Move right endpoint to center

Move left endpoint to center

Exit loop when
- x equals s[mid], or
- Interval becomes empty

# Time complexity of binary search

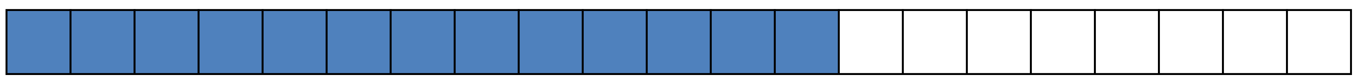
- Search space becomes in half in each loop, with $n/2^k = 1$, $k = \log_2 n$, where
  - n: number of data
  - k: number of loops

```
left=0; right=n-1;
do{
  mid = (left+right)/2;
  if x < s[mid] then right = mid-1;
  else left = mid+1;
}while(x != s[mid] && left≦right);
if x == s[mid] then return mid;
else return -1;
```

Therefore, time complexity is O(log n)

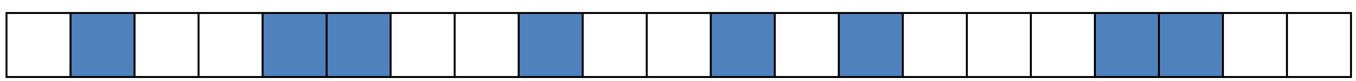
# Hash Method

- Management of data so far:
  Data are in order in the array



Data are packed in order

- Hash method: Data are distributed in the array



How can we decide the index of the data x?

Compute by a hash function

Data x $\longrightarrow$ index (position) in the array

# How to store data in hash

1. Compute “hash” value j for a data x
2. From the j-th element in the array, find the first empty element, and put x at the index (there may be other data that has the same hash value)

```
Initialize hash table htb[0]…htb[m-1] by 0;
for i=0 to n-1 do{
   Let x be the i-th data;
   j = hash(x);            //compute hash function
   while(htb[j] != 0)      //find the empty entry
      j = (j+1) % m;       //     from htb[j]
   htb[j] = x;             //store x there
}
```

We denote the size of hash table by m, and h[j]=0 means that it is “empty”

**Example:**

Set S = {3, 4, 6, 7, 9, 11, 14, 15, 17, 18, 20, 23, 24, 26, 27, 29, 30, 32}

Hash function hash(x) = x mod 27

(the size of hash table is 27)

| | | | |
|---|---|---|---|
| 3 → 3 | 11 → 11 | 20 → 20 | 29 → 2 |
| 4 → 4 | 14 → 14 | 23 → 23 | 30 → 3 |
| 6 → 6 | 15 → 15 | 24 → 24 | 32 → 5 |
| 7 → 7 | 17 → 17 | 26 → 26 | |
| 9 → 9 | 18 → 18 | 27 → 0 | |

Hash value is on the right hand

If we use this hash function, red numbers are in collision

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| htb | 27 | 0 | 29 | 3 | 4 | 30 | 6 | 7 | 32 | 9 | 0 | 11 | 0 | 0 |

| | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| htb | 14 | 15 | 0 | 17 | 18 | 0 | 20 | 0 | 0 | 23 | 24 | 0 | 26 |

# Hash method: Searching

- For a given data x, compute the hash function and obtain the value j
  - If it is the same value of x, halt.
  - If it is not equal to x and not 0, check the next
  - If it is 0, we have no data x in the table

```
Search_In_Hash(x){
  j = hash(x);
  while( htb[j] != 0 and htb[j] != x )
     j = (j+1) % m;     //move to next
  if htb[j] == x then return j;
  else return -1;
}
```

# Hash method: Example of searching

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| htb | 27 | 0 | 29 | 3 | 4 | 30 | 6 | 7 | 32 | 9 | 0 | 11 | 0 | 0 |

| | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| htb | 14 | 15 | 0 | 17 | 18 | 0 | 20 | 0 | 0 | 23 | 24 | 0 | 26 |

Case x=14 : Since hash(14)=14, it finds at htb[14].

Case x=32 : Since hash(32)=5, it searches from htb[5], and finds after checking 30, 6, and 7.

Case x=41 : Since hash(41)=14, it searches from htb[14], and finds 0 after checking 14 and 15. It reports x=41 not found.

# Performance of hash

- The number t of table accesses depends on the occupation ratio (or load ratio) $\alpha = n/m$.
  - When it finds: $$t \cong \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$$
  - When it fails: $$t \cong \frac{1}{2}\left(1 + \left(\frac{1}{1-\alpha}\right)^2\right)$$

Note: It is independent from n, the size of data.
When hash table is large, each access is a constant time.

  - Practical Tips: it works well for two primes p, q, and set hash(x) = p x + q (mod n)