

Introduction to Algorithms and Data Structures

Lesson 4: Searching (2) Block search

Professor Ryuhei Uehara,
School of Information Science, JAIST, Japan.

uehara@jaist.ac.jp

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

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 2

Data in the array in increasing order

- $s[] =$

3	9	12	25	29	33	37	65	87
---	---	----	----	----	----	----	----	----

- This is something like dictionary and address book...

Q: Do you use sequential search algorithm when you check dictionary?



Algorithm 2: m-block method

Idea of m-block method

- (0) Divide the array into m blocks B_0, B_1, \dots, B_{m-1}
- (1) Check the biggest item in each block,
and find the block B_j that can contain x
- (2) Perform sequential search in B_j

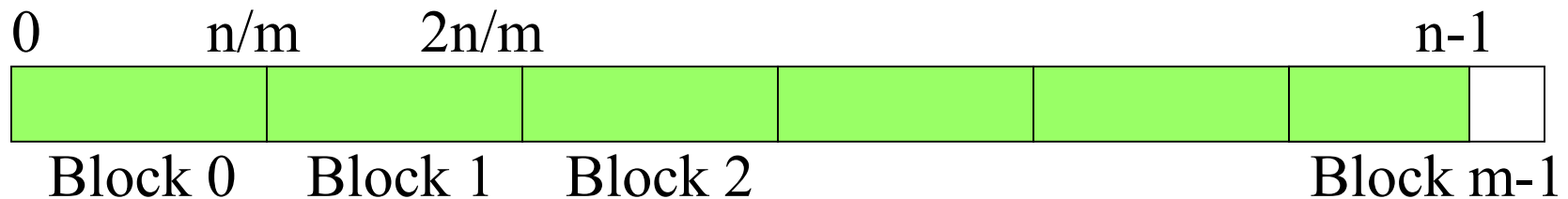
Algorithm 2: m-block method

Idea of m-block method

- (0) Divide the array into m blocks B_0, B_1, \dots, B_{m-1}
- (1) Check the biggest item in each block, and find the block B_j that can contain x
- (2) Perform sequential search in B_j

Simple implementation:

divide into the blocks of same size except the last one.



- Each block has length k , where $k = \lceil n/m \rceil$
- Block B_j has items from $s[jk]$ to $s[(j+1)k-1]$: $B_j = [jk, (j+1)k-1]$

Algorithm 2: m-block method

Idea of m-block method

- (0) Divide the array into m blocks B_0, B_1, \dots, B_{m-1}
- (1) Check the biggest item in each block, and find the block B_j that can contain x
- (2) Perform sequential search in B_j

```
j=0;
while(j<=m-2)
    if x<=s[(j+1)*k-1] then exit from loop
    else j=j+1;
```

If the program exits from the loop, the variable j indicates the index of the block, and j indicates the last one otherwise.

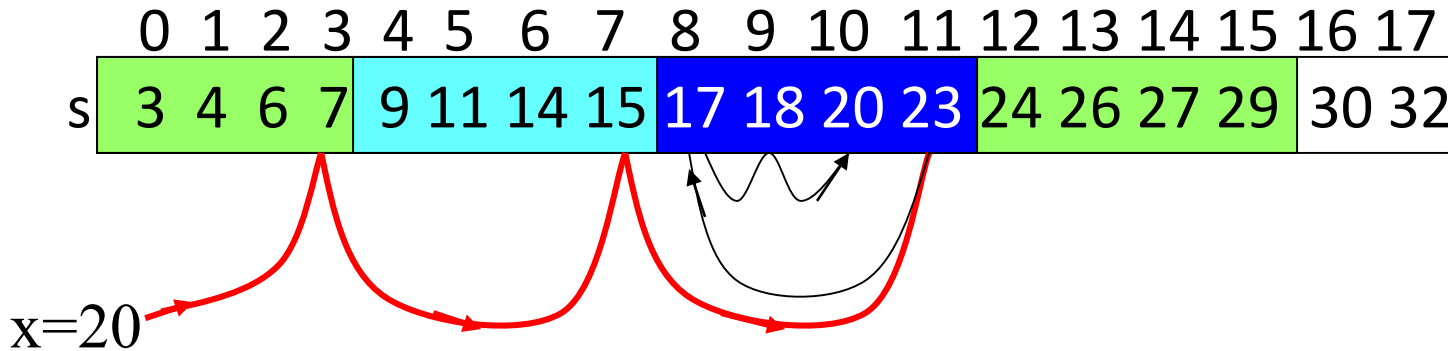
Algorithm 2: m-block method

Idea of m-block method

- (0) Divide the array into m blocks B_0, B_1, \dots, B_{m-1}
- (1) Check the biggest item in each block,
and find the block B_j that can contain x
- (2) Perform sequential search in B_j

```
i=j*k; t = min{ (j+1)*k-1, n-1 };  
while( i < t )  
    if  $x \geq s[i]$  then exit from the loop;  
    else  $i=i+1$ ; //next item in the block  
if  $x == s[i]$  then return  $i$  and halt;  
else return -1 and halt.
```

Example and time complexity



- # of comparisons \leq # of blocks + length of block = $m + n/m$
- What the value of m that minimize $m + n/m$?
 - Let $f(m) = m + n/m$, and take the differential for m
 - $f'(m) = 1 - n/m^2 = 0 \rightarrow m = \sqrt{n}$
 - When $m = \sqrt{n}$, # of comparisons $\leq \sqrt{n} + n/\sqrt{n} = 2\sqrt{n}$
- Time complexity: $O(\sqrt{n})$

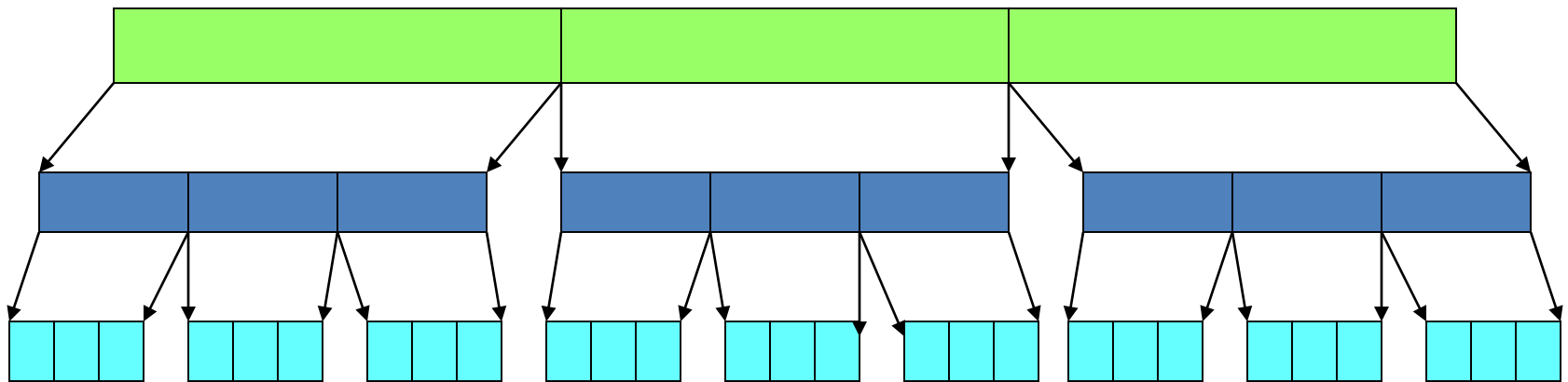
For example, when $n=1000000$,
 Linear search takes $n/2=500000$ comparisons, but
 Block search takes $\sqrt{1000000}=1000$ comparisons!!

Algorithm 3: Double m-block method

In the m-block method, we use sequential search in each block.

➡ We can use m-block method again in the block!!

Recursive call: basic and **strong** idea



Idea of double m-block method

Divide search area into m blocks, and repeat the same process for the block that contains x , and repeat again and again up to the block has length at most some constant N

```
double-m-block-search(int left, int right) {
```

Some constant

```
    Length L = right - left + 1
```

```
    if L > Lmin then
```

Length of the block

```
        k = L/m;
```

```
        for j = 0 to m-2 do
```

```
            if  $x \leq s[\text{left} + (j+1)k - 1]$  then exit the loop;
```

```
        endfor
```

```
        f = left + jk; t = min{left + (j+1)k - 1, n-1};
```

```
        double-m-block-search(f, t);
```

Recursive call

```
    else
```

```
        i = left;
```

```
        while (i < right)
```

```
            if  $x \leq s[i]$  then exit the loop else  $i = i + 1$ ;
```

```
            if  $x == s[i]$  then return i;
```

```
            else return -1;
```

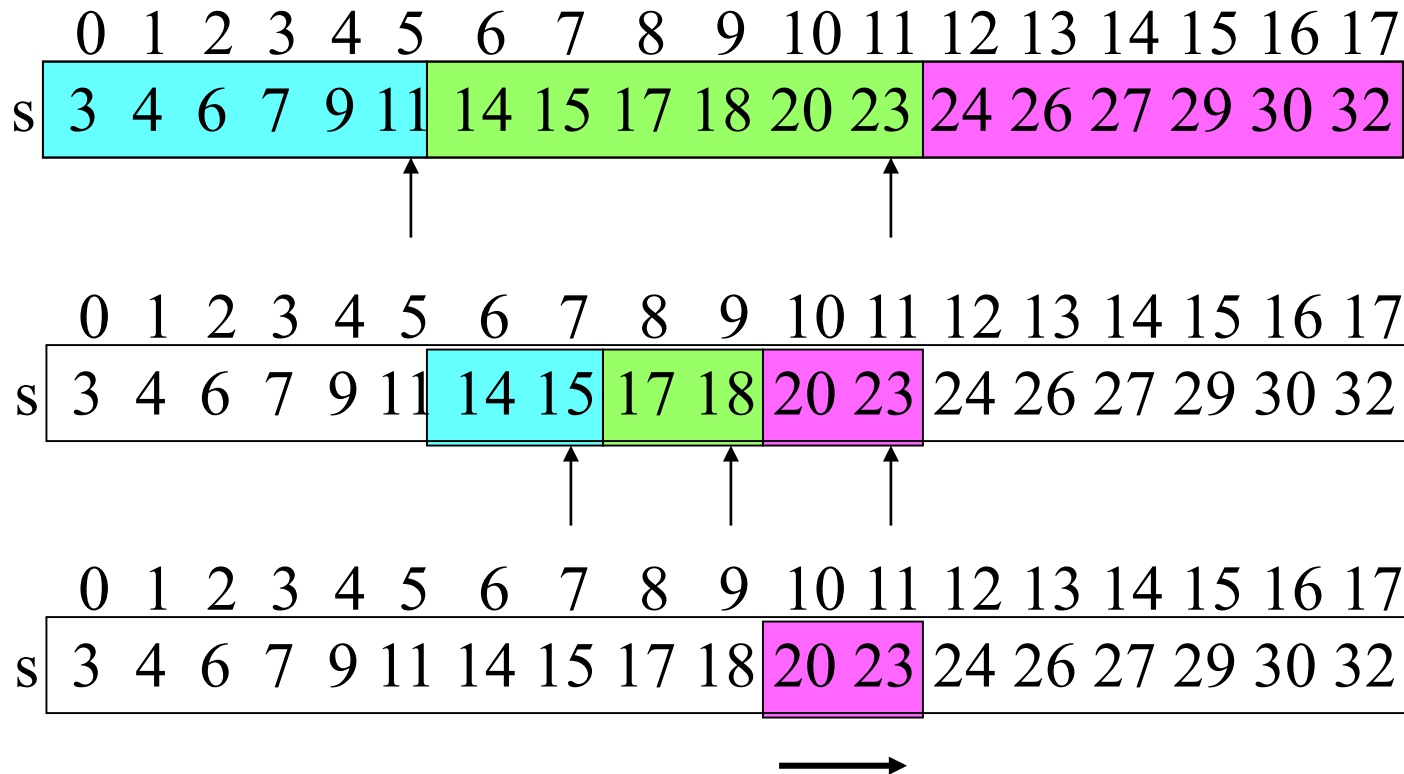
```
    endif
```

sequential search if the
interval is short enough

```
}
```

Example:

find 20 ($x=20$) for block size 3



Analysis of time complexity

- Length of search space

$$n \rightarrow \left\lceil \frac{n}{m} \right\rceil \rightarrow \left\lceil \frac{\left\lceil \frac{n}{m} \right\rceil}{m} \right\rceil \rightarrow \left\lceil \frac{\left\lceil \frac{\left\lceil \frac{n}{m} \right\rceil}{m} \right\rceil}{m} \right\rceil \rightarrow \dots$$

- Let n_i be the length after the i -th call

$$n_1 = \left\lceil \frac{n}{m} \right\rceil \leq \frac{n}{m} + 1$$

$$n_2 = \left\lceil \frac{n_1}{m} \right\rceil \leq \frac{n}{m^2} + \frac{1}{m} + 1$$

...

$$n_i \leq \frac{n}{m^i} + \sum_{j=0}^{i-1} \frac{1}{m^j} \leq \frac{n}{m^i} + 2$$

Analysis of time complexity

- The length n_i after the i -th recursive call:

$$n_i \leq n/m^i + 2$$

- How many recursive calls made?

$$n_i \leq L_{\min} \iff L_{\min} \geq \frac{n}{m^i} + 2 \iff i \geq \log_m \frac{n}{L_{\min} - 2}$$

- Each recursive call make at most $m-1$ comparisons, so the total number of comparisons is $\leq (m - 1) \log_m \frac{n}{L_{\min} - 2} + L_{\min}$

- The time complexity is $O(\log n)$

Analysis of time complexity:

The best value of m

- $$T(n, m) = (m - 1) \log_m \frac{n}{L_{\min} - 2} + L_{\min}$$
$$= \frac{m - 1}{\log_2 m} \log_2 \frac{n}{L_{\min} - 2} + L_{\min}$$
- To make $T(n, m)$ the minimum, smaller m is better because $m-1$ grows faster than $\log_2 m$ (which will be checked in the big-O notation).
- Therefore, $m=2$ is the optimal



We will have “binary search”