MA2252 Introduction to computing

lectures 11-12

Searching algorithms and complexity

Matias Ruiz

October 2023

Searching: searching algorithms

Given an array `arr` and an element `key`, find (one or many) `k` such that `a(k) = key`.

Example:

```
arr = ['dog' 'horse' 'rabit' 'dragon' 'cat']

result = mySrearchingAlgorithm(arr,'dragon')

result = 4
```

Searching: Sequential Search

An intuitive algorithm: Loop through the array, comparing one-by-one.

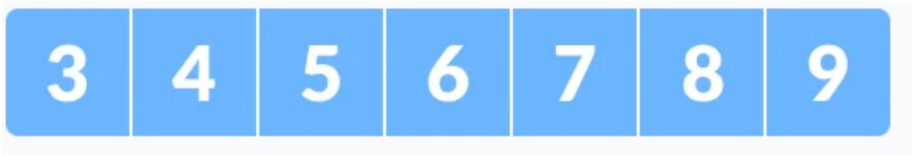
```
function index = sequentialSearch(arr, key)
len = length(arr);
index = 0;
for i = 1:len
    if arr(i) == key
        index = i;
        break
    end
end
```

Note: Doesn't need to be a double array. Work's with any variable type.

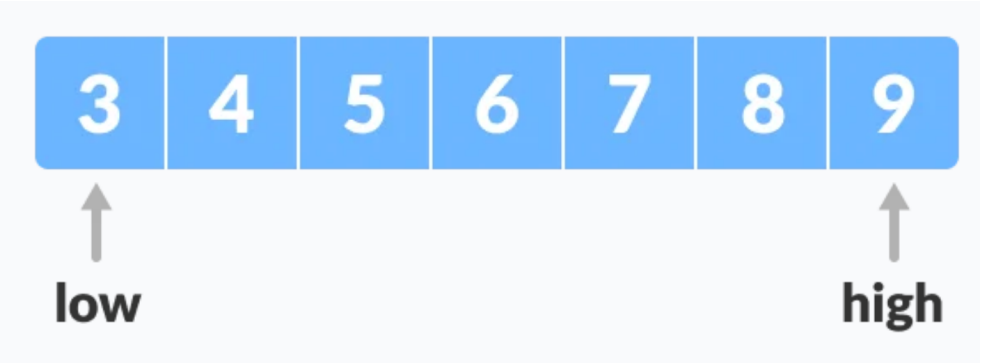
Searching: binary search

Assumes that the array is sorted

1. Example for the following array with `key=4`:



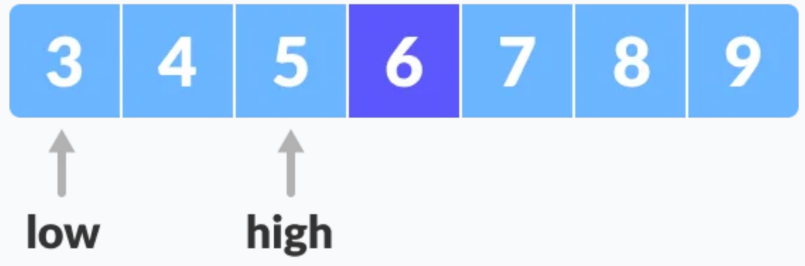
2. Set two pointers, *low* and *high*, at the lowest and highest positions respectively



3. Find the middle element of the array

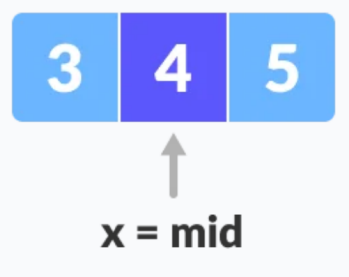
Searching: binary search

4 Modify the high pointer to `mid-1`



5 Repeat the previous steps using the same logic

6 The `key` is found



Searching: binary search

Two possible implementations

- Recursive
- Iterative

Searching: Binary Search (iterative)

```
function outind = binarySearch(arr, key)
low = 1;
high = length(arr);
outind = -1;
while low <= high && outind == -1
    mid = floor((low + high)/2);
    if arr(mid) == key
        outind = mid;
    elseif key < arr(mid)
        high = mid - 1;
    else
        low = mid + 1;
    end
end
end
```

Complexity

Complexity: efficiency of a program

How much time does it take to my program to complete its task?

It's hard to answer exactly $\Rightarrow$ depends on the coding style, programing lenguage, your computer, etc.

Better question: How does the runtime of this function grows as the input parameter grows?

Answer: The 'big' O notation, also known as Landau notation.

Complexity: big O notation

Say we have two algorithms taking in total

$$T1 = a_1 n + b_1 \text{seconds},$$

and

$$T2 = a_2 n + b_2 \text{ secs},$$

How does the scaling of these algorithms behave?

Say we have another algorithm that takes

$$T3 = a_3 n^2 + b_3 \text{ secs}.$$

How does this last algorithm compare to the first two?

Complexity: big O (and small o) notation

Definition (big O)
Given two functions $f(n)$ and $g(n)$, we say that $f(n) = O(g(n))$ if there exist a constant $N > 0$ and a constant $C > 0$ such that for every $n > N$,

$$|f(n)| \leq C|g(n)|.$$

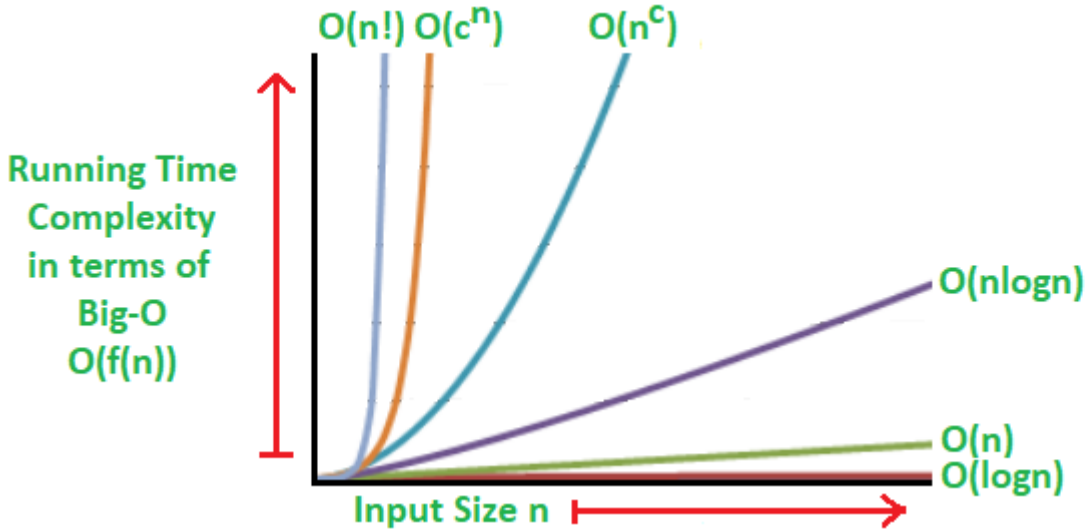
Definition (small o)
Given two functions $f(n)$ and $g(n)$, we say that $f(n) = o(g(n))$ if for all $C > 0$ there exist an N such that for every $n > N$,

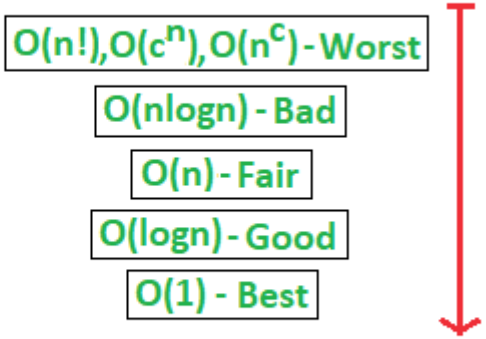
$$|f(n)| \leq C|g(n)|.$$

Complexity: big O notation (and other)

Common types of complexity:

- $O(1)$ = constant
- $O(\log n)$ = logarithmic
- $O(n)$ = linear
- $O(n^2)$ = quadratic
- $O(n^c)$ = polynomial
- $O(c^n)$ = exponential





Complexity: $O(1)$ operations

The following constitute $O(1)$ operations:

- Addition, substraction, multiplication, and division between two numbers.
- Variable assignement.
- Function call

Complexity: big O notation

How to compute it? $\Rightarrow$ calculate the number of $O(1)$ operations.
Example: What is the complexity of the following algorithm

```
function out = f(n)

out = 0;
for i=1:n
    for j= 1:n
        out = out + i*j;
end

end
```

additions: n^2, subtractions: 0, multiplications: n^2, divisions: 0,
assignments: $2n^2 + 1$, function calls: 0, total: $4n^2 + 1 = O(n^2)$.

Complexity: another example

Example: What is the complexity of the following algorithm?

```
function out = divideByTwo(n)

out = 0;
while n>1
    n = n/2;
    out = out + ;
end

end
```

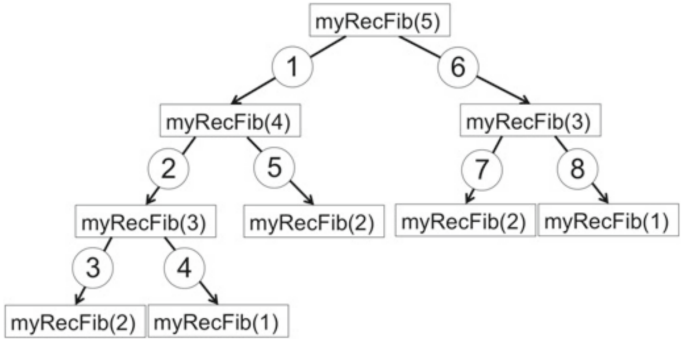
The total number of operations is proportional to the number of iterations $I(n)$, which satisfies $\frac{n}{2^{I(n)}} = 1 \Rightarrow I(n) = \log n$. The complexity is $O(\log n)$.

Complexity: complexity of recursive Fibonacci?

```
function F = recursiveFibonacci(n)

    if n==1
        F = 1;
    elseif n==2
        F = 1;
    else
        F = recursiveFibonacci(n-1) +
            recursiveFibonacci(n-2);
    end

end
```



Complexity: complexity of iterative Fibonacci?

```
function out = iterativeFibonacci(n)

F = ones(n,1);
for i = 3:n
    F(i) = F(i-1) + F(i-2);
end

out = F(n);
end
```