ALGORITHMS AND DATA STRUCTURES

SEARCH ALGORITHMS

THE PROBLEM OF SEARCHING

- can be defined as the process of finding an element in an array
- More formally, the problem of searching is stated as follows
  - **Input**: A sequence of n number $A = \{a_1, ..., a_n\}$ and a value/ key x
  - **Output**: An index i such that A[i] = x or the special value NIL if x does not appear in A

THE PROBLEM OF SEARCHING

- can be defined as the process of finding an element in an array
- More formally, the problem of searching is stated as follows
  - **Input**: A sequence of n number A = {a_1, ..., a_n} and a value/key x
  - **Output**: An index i such that A[i] = x or the special value NIL if x does not appear in A
- Our task now would be to find several algorithms, hopefully with different complexities, that solve the searching problem

SEARCH ALGORITHMS

- A *search algorithm* is any algorithm which solves the problem of search
- A *search algorithm* retrieves information stored within some data structure, either with discrete or continuous values

TYPES OF SEARCH ALGORITHMS

- can be classified based on their mechanism of searching
  - **Sequential search** algorithms check every record for the one associated with a target key in a linear fashion
  - **Interval searches**, repeatedly target the center of the search structure and divide the search space in half
  - **Hashing algorithms** directly map values/keys to records based on a hash function.
- Last two require data to be (re)arranged in some way

LINEAR SEARCH ALGORITHM

- Considering the array of elements A, what would be the simplest approach to solving such problem?
- A possible algorithm is: “Begin at the beginning, and go on till you either find the key you're looking for or reach the end. If you find the key, you’re done; if you reach the end, then the key does not appear in the array”
- Because this process goes from beginning to end in a straight line, this algorithm is called ***linear search***

LINEAR SEARCH ALGORITHM

- Pseudocode for linear search (first attempt):

```
FUNCTION linear_search:
  INPUT: integer array A[n], key x
  OUTPUT: integer i
  USAGE: idx = linear_search(A, x)
BEGIN
  FOR i: 1, A.length
    IF A[i] == x THEN
      RETURN A[i]
    END
  END
  RETURN ??????
END // linear_search
```

What to return when value is not found?

LINEAR SEARCH ALGORITHM

- Pseudocode for linear search:

```
FUNCTION linear_search:
  INPUT: integer array A[n], key x
  OUTPUT: integer i
  USAGE: idx = linear_search(A, x)
BEGIN
  FOR i: 1, A.length
    IF A[i] == x THEN
      RETURN i
    END
  END
  RETURN -1
END // linear_search
```

Why not return value A[i] instead of i?

BINARY SEARCH ALGORITHM

- If the array is too large, linear search becomes inefficient because it has to go through every value

BINARY SEARCH ALGORITHM

- If the array is too large, linear search becomes inefficient because it has to go through every value
- Now we are posed with the questions:
  - However, is searching every value really necessary?
  - Do we need conditions on the array in order to be able to skip some of the values?

BINARY SEARCH ALGORITHM

- If the array is too large, linear search becomes inefficient because it has to go through every value
- Now we are posed with the questions:
  - However, is searching every value really necessary?
  - Do we need conditions on the array in order to be able to skip some of the values?
- The binary search algorithm answers such questions

BINARY SEARCH ALGORITHM

- Binary search takes advantage of some prior knowledge on the structure of the array
  - So we can skip the reading in of some values in the array
- Binary search requires that the array must be sorted
- *How can we take advantage of the fact that the input is ordered?*
  - … [*Any ideas?*]

BINARY SEARCH ALGORITHM

- searches in a sorted array by repeatedly dividing the search interval in half. The algorithm executes the following steps:
    1. Begin with an interval covering the entire array
    2. If the search key matches the middle element, we're done
    3. Else if the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half
    4. Otherwise narrow it to the upper half
    5. Repeatedly check until key is found or interval is empty

BINARY SEARCH ALGORITHM

`key = Miami`

	cityNames		cityNames		cityNames
0	Atlanta	0	~~Atlanta~~	0	~~Atlanta~~
1	Boston	1	~~Boston~~	1	~~Boston~~
2	Chicago	2	~~Chicago~~	2	~~Chicago~~
3	Denver	3	~~Denver~~	3	~~Denver~~
4	Detroit	4	~~Detroit~~	4	~~Detroit~~
5	Houston	5	~~Houston~~	5	~~Houston~~
6	Los Angeles	6	Los Angeles	6	Los Angeles
7	Miami	7	Miami	7	Miami
8	New York	8	New York	8	~~New York~~
9	Philadelphia	9	Philadelphia	9	~~Philadelphia~~
10	San Francisco	10	San Francisco	10	~~San Francisco~~
11	Seattle	11	Seattle	11	~~Seattle~~

BINARY SEARCH ALGORITHM

- Pseudocode for binary search

```
FUNCTION binary_search:
  INPUT: integer array A[n], key x
  OUTPUT: integer i
  USAGE: idx = binary_search(A, x)
BEGIN
  lh = 1; rh = A.length
  WHILE lh <= rh
    mid = (lh + rh) / 2
    IF x == A[mid] THEN RETURN mid
    ELSE IF x > A[mid]   lh = mid + 1
    ELSE                 rh = mid - 1
  END
  RETURN -1
END // binary_search
```

EFFICIENCY OF SEARCH ALGORITHMS

- We can measure the computational complexity of search algorithms in two ways:

EFFICIENCY OF SEARCH ALGORITHMS

- We can measure the computational complexity of search algorithms in two ways:
  - Using the rules of computation within the RAM model or

EFFICIENCY OF SEARCH ALGORITHMS

- We can measure the computational complexity of search algorithms in two ways:
  - Using the rules of computation within the RAM model or
  - Counting number of times relational operators are called (comparison model)

EFFICIENCY OF SEARCH ALGORITHMS

- We can measure the computational complexity of search algorithms in two ways:
  - Using the rules of computation within the RAM model or
  - Counting number of times relational operators are called (comparison model)
- We will use the comparison model later on. For now, we stick to what we know

EFFICIENCY OF SEARCH ALGORITHMS

- Calculate T(N) for both, linear and binary, search algorithms
- Discuss best, average, and worst cases using asymptotic analysis

LINEAR VS BINARY SEARCH

- Binary search is much more efficient than linear search
- Binary search requires the array to be sorted, which is in itself quite a challenging task
- Linear search is much simpler to code. Straightforward write a loop and you're pretty much done

LINEAR VS BINARY SEARCH

- Binary search is much more efficient than linear search
- Binary search requires the array to be sorted, which is in itself quite a challenging task
- Linear search is much simpler to code. Straightforward write a loop and you're pretty much done

- A new question is then: How to order or sort an array of elements? **Sorting problem! (next classes)**

LINEAR VS BINARY SEARCH

- Comparison of linear and binary search algorithms performance

N (linear search)	lg N (binary search)
10	1
100	7
1,000	10
1,000,000	20
1,000,000,000	30

OTHER SEARCH ALGORITHMS

- Interesting search algorithms worth looking into
- The following algorithms require the array to be ordered

```
Algorithm                      Complexity
=======================        =======================
Ternary search                 O(lg N)
Jump search                    O(sqrt(N))
Interpolation search           O(lg lg N)
Exponential search             O(lg N)
Fibonacci search               O(fib(N))      [Hard!]
```