

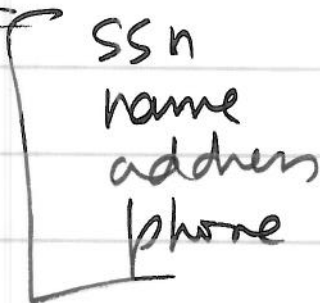
- Assign 3 is hosted
- Scores Assign 3 => weekend
- Midterm Oct 15th

Structures

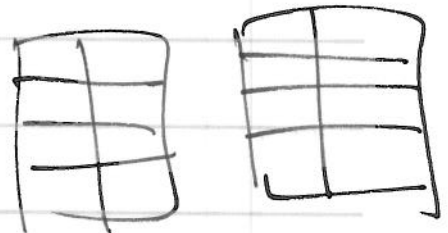
↳ used to model real-world objects

struct => capture the fields of a type

Person



Person A, B



```

struct Person
{
  string ssn;
  string name;
  int age;
};

```

user-defined type

data members

```

Person x, y;
x.ssn = "000";
x.name = "Joe";

```

```

x.age = 20;
y.ssn = "111";
y.name = "Jane";
y.age = 21;

```

```

x = y; // LEGAL

```

```

[
  x.ssn = y.ssn
  x.name = y.name
  x.age = y.age
] struct assignment

```

structures — reference
 \ by value

```
void display (Person p)
```

```
{
  cout << "Person data << endl
        << p.ssn << p.name
        << p.age;
}
```

```
Person x; Person y;
display(x); // x copied
display(y); // y copied
```

x

name	Joe
SSN	000
age	20

y

name	Jane
SSN	111
age	21

// reference parameters
 void swap (Person &a, Person &b)

```
{
  Person temp = a;
  a = b;
  b = temp;
}
```

}

swap (x, y)

x

name	Jane
SSN	111
age	21

y

name	Joe
SSN	000
age	20

```

struct Person
{
  string ssn;
  string name;
  int age;
};

```

List of objects

```

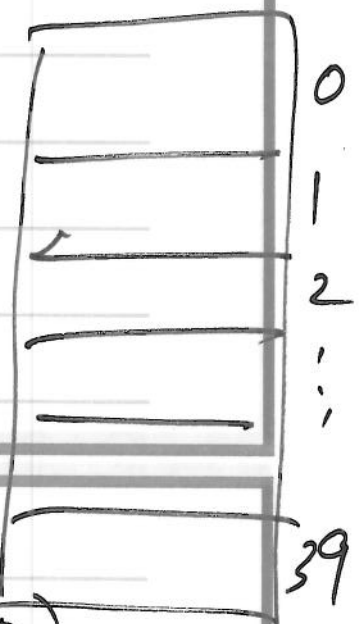
Person x, y;

```

```

Person csci455[40];

```



```

for (int i = 0; i < 40; i++)

```

```

{
  csci455[i].age = 0;
  csci455[i].ssn = "";
  csci455[i].name = "";
}

```

Searching for information

db struct

KEY => attribute by
value which you
wish to search

Toyota

Can db[MAXSIZE];

SEQUENTIAL SEARCH [to implement Always possible]

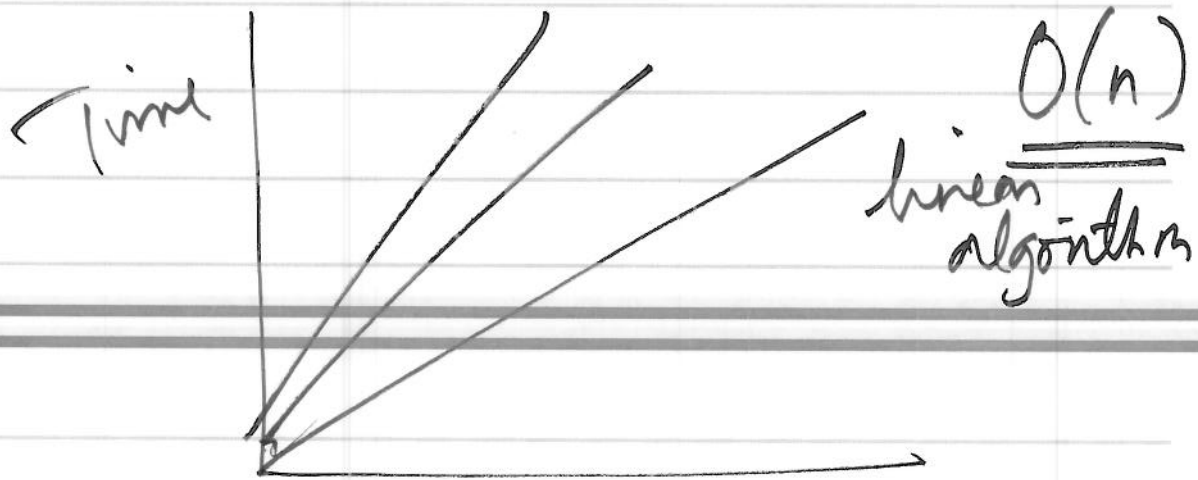
```

for (int i = 0; i < MAXSIZE; i++)
{
  if (db[i].manufacturer ==
      uppercase[db[i].manufacturer] ==
      uppercase("Toyota"))
  {
    cout << "Toyota found"
          << "Price = "
          << db[i].price;
  }
}

```

Big "Oh" notation

quantify complexity of algorithm
as a function of input size



→ size of input

Simplest
Best

Constant Time

$A[!]$ ⇒ independent of
size of array

Constant
 $O(\log n) - O(\log n)$

$O(n)$
 $O(n \log n)$ $O(n \log n) \Leftarrow$

$O(n^2)$ \Rightarrow deteriorates much faster as n increases

correct but inefficient

$O(n^3)$
 $O(2^n)$
 (e^n) } Exponential algorithms

Sequential Search $O(n)$ } Simplest-
 └ Best Case 1
 └ Worst Case max size $O(n)$
 └ Average Case max size / 2 $O(n)$
 $O(n)$

Analysis of Algorithms

$O(n)$

Analyze look for loops

for ($i = 0$; $i < n$ — .)

$O(n)$

nested loop can be $O(n^2)$

BINARY SEARCH] $O(\log_2^n)$

