A fundamental problem in computer science is searching – find a specific value in a collection.

Example – find the name "John" in an array of names:

"Mary" "Eric" "Nancy" "John" "Jacob"
Searching

- Problem
  - Find a value in an array and return the index where the value is stored.
- Assumption
  - The value appears in the array at most one time
- Input
  - An array and the usage of the array
- Output
  - An index where the value is stored or NOT_FOUND if the value is not found

Searching (String Array)

```c
// NOT_FOUND is an integer constant defined above main()
int SearchStringArray(const string array[], int usage, const string& val)
{
    for (int loc = 0; loc < usage; loc++) {
        // if the array value at location loc is what we're looking for, return the location.
        if (array[loc] == val) {
            return loc;
        }
    }
    // This will only be reached if the value was not found.
    return NOT_FOUND;
}
```
Searching (String Array Usage)

```cpp
int SearchStringArray(const string array[],
    int usage, const string& val);
const int NOT_FOUND = -1;
const int NUM_NAMES = 5;
const string FIND_VAL = "John";

void main() {
    string names[NUM_NAMES] = {
        "Mary", "Eric", "Nancy", "John", "Jacob"
    };
    int index = SearchStringArray(names, NUM_NAMES, FIND_VAL);
    if (index != NOT_FOUND) {
        cout << "Found " << FIND_VAL << " at index " << index << endl;
    } else {
        cout << FIND_VAL << " was not found." << endl;
    }
}
```

---

Searching

- The search presented is called a **linear search**
- In the best situation, the value we are looking for is in __________
- In the worst situation, the value we are looking for is in __________
- So, if the array has \( n \) elements, we have to compare with at most ____ of them
- The number of comparisons grows **linearly** (in a straight line) with the size of the array.
Searching and Parallel Arrays

- Often times we want to find a specific item in a parallel array by searching for only one piece of information.

<table>
<thead>
<tr>
<th>ID</th>
<th>fnames</th>
<th>lnames</th>
<th>ages</th>
<th>hwages</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;Mary&quot;</td>
<td>&quot;Jones&quot;</td>
<td>27</td>
<td>14.25</td>
</tr>
<tr>
<td>1</td>
<td>&quot;Eric&quot;</td>
<td>&quot;Allen&quot;</td>
<td>43</td>
<td>18.72</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Nancy&quot;</td>
<td>&quot;Lane&quot;</td>
<td>51</td>
<td>21.33</td>
</tr>
<tr>
<td>3</td>
<td>&quot;John&quot;</td>
<td>&quot;Steele&quot;</td>
<td>32</td>
<td>15.24</td>
</tr>
<tr>
<td>4</td>
<td>&quot;Jacob&quot;</td>
<td>&quot;Brown&quot;</td>
<td>24</td>
<td>13.56</td>
</tr>
</tbody>
</table>

Searching and Parallel Arrays

- **Problem**
  - Find the employee with the last name "Steele" and print out their information

- **Input**
  
<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary Jones</td>
<td>27</td>
<td>14.25</td>
</tr>
<tr>
<td>Eric Allen</td>
<td>43</td>
<td>18.72</td>
</tr>
<tr>
<td>Nancy Lane</td>
<td>51</td>
<td>21.33</td>
</tr>
<tr>
<td>John Steele</td>
<td>32</td>
<td>15.24</td>
</tr>
<tr>
<td>Jacob Brown</td>
<td>24</td>
<td>13.56</td>
</tr>
</tbody>
</table>
Searching and Parallel Arrays

// Prototypes
void InitStringArray(string array[], int usage, const string& val);
void InitIntArray(int array[], int usage, int val);
void InitDoubleArray(double array[], int usage, double val);
int ReadData(istream& In, string fnames[], string lnames[],
int ages[], double hwages[]);
int SearchStringArray(const string array[], int usage,
const string& val);

// Constants
const int NUM_EMPL = 5;
const int NOT_FOUND = -1;
const string TO_FIND = "Steele";
const string EMPTY_STRING = "";

// Main program
void main() {
    string fnames[NUM_EMPL], lnames[NUM_EMPL];
    int ages[NUM_EMPL];
    double hwages[NUM_EMPL];
    int usage = 0;
    int found = NOT_FOUND;

    // Initialize your arrays
    InitStringArray(fnames, NUM_EMPL, EMPTY_STRING);
    InitStringArray(lnames, NUM_EMPL, EMPTY_STRING);
    InitIntArray(ages, NUM_EMPL, 0);
    InitDoubleArray(hwages, NUM_EMPL, 0.0);
Searching and Parallel Arrays

usage = ReadData(cin, fnames, lnames, ages, hwages);
cout << "Read in " << usage << " employees." << endl;
// Find the employee and print out their information
found = SearchStringArray(lnames, usage, TO_FIND);
if (found != NOT_FOUND) {
cout << "Employee info: " << fnames[found] << " " << lnames[found] << " " << ages[found] << " " << hwages[found] << endl;
} else {
cout << "No employee with last name " << TO_FIND << " was found." << endl;
}
// end of main()

Searching and Parallel Arrays

int ReadData(istream& In, string fnames[], string lnames[],
int ages[], double hwages[])
{
    string fname, lname; // temp first, last name
    int age, ID = 0; // temp age, real employee ID
    double hwage; // temp wage

    In >> fname >> lname >> age >> hwage; // priming read
    while (In && ID < NUM_EMPL) {
        fnames[ID] = fname; // store information
        lnames[ID] = lname; // in parallel arrays
        ages[ID] = age;
        hwages[ID] = hwage;
        ID++;
        In >> fname >> lname >> age >> hwage; // next data set
    }
    return ID;
}