Searching Algorithms

Linear (or Serial) Search

```
// Search for x in an array of n elements starting at A[ first ]
// Return position where found, or -1 if not found
j = 0; found = false;
while ((j < n) && ! found)
  if (A[ first + j ] == x)
     found = true;
  else j++;


if (found)
  return( first + j );   // return position where x was found
else return( -1 );
```

Runtime analysis: ???

Binary Search

// Search for x in a **sorted** array of n elements starting at A[first]

// Return position where found, or -1 if not found

```
int search(const int A[ ], const int first, const int n, const int x) {
   if (n == 0)
      return( -1 );  // x is not here
  else {
          int middle = first + n/2;
          if (x == A[ middle ])
            return( middle );
          else if (x < A[ middle ])
                    return(search(A, first, n/2, x));
               else return(search(A, middle+1, (n-1)/2, x);
  }
}
```

Runtime analysis: ???

Hashing

Basic Idea:

- Maintain a collection of **records** in a **table** (i.e., array), where each has a unique **key**
- **Hash function** maps a key value to an array index
- Can be O(1) for searches

Example: Student records, key is student ID, data[10], h(ID) = ID % 10

Perfect hashing: every key produces a different index value when hashed

Collision: 2 different records hash to the same index

Open-Address Hashing

(1) Compute index = h(key)
(2) If data[index] available, then store record there and you're done
(3) If data[index] not available, try data[index + 1], etc. until you find available spot, wrapping around to data[0] if necessary (this is called **linear probing**)

Note: Requires that array be initialized so we can test to see if position is available

Required Functions:

Constructor: initialize array entries to **NEVER_USED**, and set numUsed to 0

Insert: add new record, or replace old record with same key[1]

IsPresent: true if record with specified key exists; otherwise, false

Find: given a key, return true (and copy of that record) if found; otherwise, false

Remove: if record with specified key exists, set that entry to **PREVIOUSLY_USED**

Note: **NEVER_USED** vs. **PREVIOUSLY_USED** important for Find ***…why???***

[1] If array is full, then resize array and rehash all existing entries to place them in larger array