SEARCHING (Linear/Binary)

Searching Algorithms

- method of locating a specific item of information in a larger collection of data.
- two popular search algorithms:
  - *Linear Search*
  - *Binary Search:*
- LINEAR SEARCH
  - uses a loop to sequentially step through the array, starting at front and comparing items
  - if reach the end and that item does not match, the item is not found
  - ALGORITHM:
    - Set position number to 0
    - For each array element:
      - Compare item to array element at position number
      - if match return the subscript
      - Move to next element

SAMPLE CODE (LINEAR SEARCH)

```
void main()
{
	int MyArray = {87, 85, 77, 100, 82};
	int itemtofind;
	cout << "Enter an array item to look for:";
	cin >> itemtofind;
	int found = LinearSearch(MyArray, 5, itemtofind);
	if (found == -1)
		cout << "That item is not in the array!";
	else
		cout << itemtofind << " is located at position " << found;
} [trace this]
```

LINEAR SEARCH (cont.)

- **Code:**

```
int LinearSearch(int a[], int listSize, int item)
 {
      for (int pos=0; pos < listSize; pos++)
      {
           if (a[pos] = = item)
                return pos;
      }
      return -1;
}
```

Advantages/Disadvantages of Linear Search:

- Advantages:
  - easy to understand and implement
- Disadvantages:
  - not efficient (what if item is last one?)
  - in average case, n/2 comparisons will be made
  - in worst case, n comparision will be made
  - thus, time complexity of Linear Search: O(n)

BINARY SEARCH

- more efficient than linear search;
- **recursive** algorithm
  - keep dividing array in half, and checking half of it for item
  - if get down to 1 element, and that doesn't match, them item not there
  - ALGORITHM:
    - Set first to 0
    - Set last to last subscript
    - Set Found to False
    - Set middle to middle element of current list
    - if middle matches the item return middle
    - else if middle > then item set last to middle - 1
      - Call binary search with new last
    - else set first to middle + 1
      - call binary search with new first

CODE FOR BINARY SEARCH

```
int BinarySearch(int a[], int first, int last, int item)
{
        if (last < First)
            return -1;
        else {
            int middle = (last + first) / 2;
            if (item == a[middle])
                return middle;
            else if (a[middle] > item)
                return BinarySearch(a,first,middle-1,item);
            else
                return BinarySearch(a,middle+1,last,item);
        }
}
```

CODE USING BINARY SEARCH

```
void main()
{
    int MyArray={101, 114, 150, 190, 180, 266, 274, 383};
    int itemtofind;
    cout << "Enter an item to search for:";
    cin >> itemtofind;
    int foundit = BinarySearch(MyArray, 0, 7, itemtofind);
    if (foundit == -1)
        cout << "That item is not in list" << endl;
    else
        cout << "The item is located at position " << itemtofind;
}
```

BINARY SEARCH SUMMARY

- list is halved each time BS is called
  - maximum number of comparisons:
    - 1) if n = 1, algorithm invoked 2 times
    - 2) if n > 1, algorithm invoked 2_m times
      - where m is size of sequence being searched
  - 3) thus, total number of invocations:

    $a_n = 1 + a_{n/2}$

    which is called the **recurrance relation**
  - 4) Thus, by solving the recurrance relation, we get:
    - $a_{2k} = 1 + a_{2k-1}$
    - $2_{k-1} < n <= 2_k$
    - $k-1 < \log n <= k$
    - so, Algorithm complexity: O(log n)
    - [see discrete textbook for more details]