ARRAYS SEARCHING

NATIONAL UNIVERSITY OF TECHNOLOGY (NUTECH)

DR. SAMAN RIAZ

LECTURE # 4

ANALYSIS OF ARRAYS

- How to insert an item?
  - How many steps in terms of N (number of elements in array)?
    - N steps at maximum (move items to insert at given location)

- How to delete an item?
  - How many steps in terms of N (number of elements in array)?
    - N steps at maximum (move items back to take place of deleted item)

ANALYSIS OF ARRAYS

- How to search an item?
  - Linear Search
  - Binary Search

SEARCHING ARRAYS

REFERENCE: DATA STRUCTURES

(BY LIPSCHUTZ P4.12 – 4.18)

INTRODUCTION TO SEARCH ALGORITHMS

- A search algorithm is a method of locating a specific item of information in a larger collection of data. This section discusses two algorithms for searching the contents of an array.
  - Linear Search
  - Binary Search

THE LINEAR SEARCH

- This is a very simple algorithm.
- It uses a loop to sequentially step through an array, starting with the first element.
- It compares each element with the value being searched for and stops when that value is found or the end of the array is reached.

LINEAR SEARCH

```
// The searchList function performs a linear search on an
// integer array. The array list, which has a maximum of numElems
// elements, is searched for the number stored in value. If the
// number is found, its array subscript is returned. Otherwise,
// -1 is returned indicating the value was not in the array.

int searchList(int list[], int numElems, int value)
{
  int index = 0;       // Used as a subscript to search array
  int position = -1;   // To record position of search value
  bool found = false;  // Flag to indicate if the value was found
  while (index < numElems && !found)
  {
          if (list[index] == value)
          {
          found = true;
          position = index;
         }
         index++;
   }
   return position;
}
```

CALLING SEARCHLIST

```
// This program demonstrates the searchList function, which
// performs a linear search on an integer array.

#include <iostream>

// Function prototype
int searchList(int [], int, int);

const int arrSize = 5;

void main(void)
{
  int tests[arrSize] = {87, 75, 98, 100, 82};
  int result;
```

PROGRAM CONTINUES

```
  result = searchList(tests, arrSize, 100);
  if (result == -1)
       cout << "You did not earn 100 points on any test\n";
  else
  {
       cout << "You earned 100 points on test ";
       cout << (result + 1) << endl;
  }
}
```

PROGRAM OUTPUT

You earned 100 points on test 4

EFFICIENCY OF THE LINEAR SEARCH

- The advantage is its simplicity.
  - It is easy to understand
  - Easy to implement
  - Does not require the array to be in order
- The disadvantage is its inefficiency
  - If there are 20,000 items in the array and what you are looking for is in the 19,999th element, you need to search through the entire list.

BINARY SEARCH

- The binary search is much more efficient than the linear search.
- It requires the list to be in order.
- The algorithm starts searching with the middle element.
  - If the item is less than the middle element, it starts over searching the first half of the list.
  - If the item is greater than the middle element, the search starts over starting with the middle element in the second half of the list.
  - It then continues halving the list until the item is found.

BINARY SEARCH

```
// The binarySearch function performs a binary search on an integer array. Array,
// which has a maximum of numElems elements, is searched for the number
// stored in value. If the number is found, its array subscript is returned.
// Otherwise, -1 is returned indicating the value was not in the array.

int binarySearch(int array[], int numelems, int value)
{
   int first = 0, last = numelems - 1, middle, position = -1;
   bool found = false;
   while (!found && first <= last)
   {        middle = (first + last) / 2;  // Calculate mid point
            if (array[middle] == value)   // If value is found at mid
          {   found = true;
                position = middle;
          }
            else if (array[middle] > value) // If value is in lower half
               last = middle - 1;
            else
                 first = middle + 1;             // If value is in upper half
   }
   return position;
}
```

CALLING BINARYSEARCH

```
// This program demonstrates the binarySearch function, which
// performs a binary search on an integer array.
#include <iostream>

// Function prototype
int binarySearch(int [], int, int);

const int arrSize = 20;

void main(void)
{
int empIDs[arrSize] = {101, 142, 147, 189, 199, 207, 222,
                       234, 289, 296, 310, 319, 388, 394,
                       417, 429, 447, 521, 536, 600};
```

PROGRAM CONTINUES

```
  int result, empID;

  cout << "Enter the Employee ID you wish to search for: ";
  cin >> empID;
  result = binarySearch(empIDs, arrSize, empID);
  if (result == -1)
       cout << "That number does not exist in the array.\n";
  else
  {
       cout << "That ID is found at element " << result;
       cout << " in the array\n";
  }
}
```

PROGRAM OUTPUT WITH EXAMPLE INPUT

Enter the Employee ID you wish to search for: **199**

That ID is found at element 4 in the array.

BINARY SEARCH

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
list	4	8	19	25	34	39	45	48	66	75	89	95

Sorted list for binary search

key = 89

Iteration	first	last	mid	list[mid]	
1	0	11	5	39	
2	6	11	8	66	
3	9	11	10	89	⇐ Value is found

key = 34

Iteration	first	last	mid	list[mid]	
1	0	11	5	39	
2	0	4	2	19	
3	3	4	3	25	
4	4	4	4	34	⇐ Value is found

EFFICIENCY OF THE BINARY SEARCH

- Much more efficient than the linear search.
- How long does this take (worst case)?

- If the list has 8 elements
  - It takes 3 steps ($2^3 = 8$)
- If the list has 16 elements
  - It takes 4 steps ($2^4 = 16$)
- If the list has 64 elements
  - It takes 6 steps ($2^6 = 64$)

- Similarly, if the list has *n* elements
  - It takes $\log_2 n$ steps