Chapter 8

Tuesday, November 14, 2017 1:03 PM

Chapter 8: Searching and Sorting Arrays

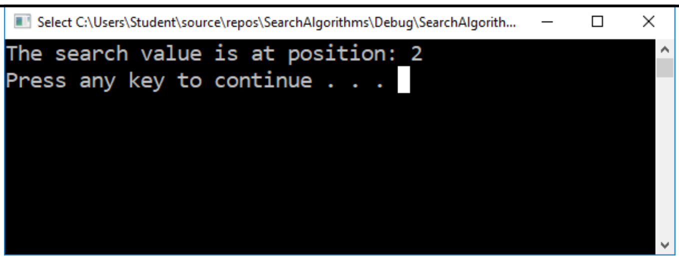
Searching Arrays

- Linear Search
- Binary Search

Linear Search

- Uses a loop to step through each element in an array
- Compares each element with the search value
- Stops when the value is found or the end or array is reached
- Return index of search value or -1 if search value isn't present

```
#include <iostream>
#include <string>
using namespace std;

int linearSearch(string target, string array[], int size);


int main(){
    string books[] = {"To Kill a Mockingbird",
        "The King James Bible", "A Brief History of Time" };
    int result = linearSearch("A Brief History of Time", books, 3);
    cout << "The search value is at position: " << result << endl;
    system("pause");
    return 0;
 }

int linearSearch(string target, string array[], int size) {
    for (int i = 0; i < size; i++) {
        if (array[i] == target)
            return i;
    }
    return -1;
}
```


Select C:\Users\Student\source\repos\SearchAlgorithms\Debug\SearchAlgorith...

```
The search value is at position: 2
Press any key to continue . . .
```

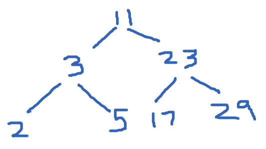
Binary Search

- Faster than linear search
- Requires that the elements are sorted before the search occurs

$\log_2(n)$

numlist

2	3	5	11	17	23	29
0	1	2	3	4	5	6



- Start with the middle element of the array
- If that element is the desired value
  - Search is over
- Otherwise,
  - If the middle element is greater than the desired value,
    - Search the lower half of the array by setting last = middle - 1
  - If the middle element is less than the desired value,
    - Search the upper half of the array by setting first = middle + 1

Search for the 5:

2	3	5	11	17	23	29
0	1	2	3	4	5	6

first	last	middle = (first + last) / 2	numlist[middle]
0	6	3	11
0	2	1	3
2	2	2	5

return 2;

Search for the 7:

2	3	5	11	17	23	29
0	1	2	3	4	5	6

first	last	middle = (first + last) / 2	numlist[middle]
0	6	3	11
0	2	1	3
2	2	2	5

If first == last && numlist[middle] != targetValue
    return -1

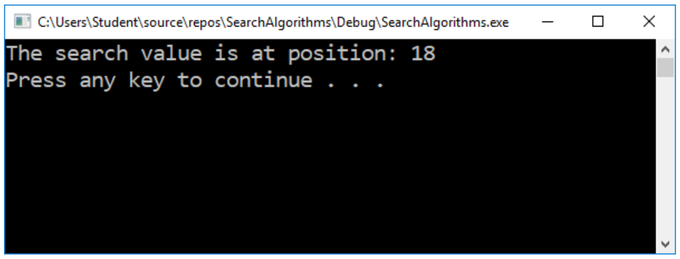
Search for the 536:

101	142	147	189	199	207	222	234	289	296	310	319	388	394	417	429	447	521	536	600
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

first	last	middle = (first + last) / 2	numlist[middle]
0	19	9	296
10	19	14	417
15	19	17	521
18	19	18	536

return 18;

```
#include <iostream>
#include <string>
using namespace std;

int binarySearch(int value, int array[], int size);

int main(){
     int numlist[] = {101, 142, 147, 189, 199, 207, 222, 234, 289, 296, 310, 319, 388, 394, 417, 429, 447, 521, 536, 600};
     int result = binarySearch(536, numlist, 19);
     cout << "The search value is at position: " << result << endl;
     system("pause");
     return 0;
 }

int binarySearch(int value, int array[], int size) {
     int first = 0;
     int last = size - 1;
     int position = -1; // We'll change this if we find the value
int binarySearch(int value, int array[], int size) {
     int first = 0;
     int last = size - 1;
     int position = -1; // We'll change this if we find the value
     bool found = false; // We'll change this if we find the value
     int middle = (first + last) / 2;

     //Search for the value
     for (int i = 0; i < size; i++) {
               //if we still haven't found it
         if (!found) {
             //calculate middle as (first + last) /2
             middle = (first + last) / 2;
             //if value is found at middle, set found to true, position to middle
             if (array[middle] == value) {
                 found = true;
                 position = middle;
             }
             //if value is in lower half, set last = middle - 1
             else if (array[middle] > value)
                 last = middle - 1;
             //if value is in upper half, set first = middle + 1
             else if (array[middle] < value)
                 first = middle + 1;

         }

     }
     return position;
}
```


C:\Users\Student\source\repos\SearchAlgorithms\Debug\SearchAlgorithms.exe

```
The search value is at position: 18
Press any key to continue . . .
```

Sorting Algorithms

- Bubble Sort
- Selection Sort

7	2	3	8	9	1

- Take each pair of elements
- Compare them to each other
- If they are out of order,
  - Swap them
- Continue comparing pairs of elements until a pass is made in which no elements were swapped

7	2	3	8	9	1
2	7	3	8	9	1
2	3	7	8	9	1
2	3	7	8	9	1
2	3	7	8	9	1
2	3	7	8	1	9
2	3	7	8	1	9
2	3	7	8	1	9
2	3	7	8	1	9

<--First Pass Complete. It had swaps, so we will continue

2	3	7	8	1	9	
2	3	7	1	8	9	<--Second Pass Complete. It had swaps, so we will continue
2	3	7	1	8	9	
2	3	7	1	8	9	
2	3	7	1	8	9	
2	3	1	7	8	9	
2	3	1	7	8	9	<--Third Pass Complete. It had swaps, so we will continue
2	3	1	7	8	9	
2	3	1	7	8	9	
2	1	3	7	8	9	
2	1	3	7	8	9	
2	1	3	7	8	9	<--Fourth Pass Complete. It had swaps, so we will continue
2	1	3	7	8	9	
1	2	3	7	8	9	
1	2	3	7	8	9	
1	2	3	7	8	9	
1	2	3	7	8	9	
1	2	3	7	8	9	<--Fifth Pass Complete. It had swaps, so we will continue
1	2	3	7	8	9	
1	2	3	7	8	9	
1	2	3	7	8	9	
1	2	3	7	8	9	
1	2	3	7	8	9	<--Sixth Pass Complete. It had no swaps, so the algorithm terminates.

Bubble Sort

Time Complexity: n^2

Compare 1st 2 elements

If out of order, exchange

Move down 1 element, compare 2nd and 3rd elements,

- Exchange if necessary
- Continue until end of array

Pass through array again, exchanging as necessary using boolean flag variable, e.g. swapMade

Repeat until pass made with no exchanges

Boolean flag swapped;

Swap function, takes in the array, the two positions to be swapped, swaps them

showArray, takes in the array, displays its elements

bubbleSort function will use swap function to make the exchanges

```
#include <iostream>
using namespace std;

void swap(int arr[], int i, int j);
void bubbleSort(int array[], int size);

int main() {
    const int SIZE = 6;
    int values[] = { 7, 2, 3, 8, 9, 1 };
    cout << "The unsorted values are: \n";
    for (int value : values) {
        cout << value << " ";
    }
    bubbleSort(values, SIZE);
    cout << "The sorted values are: \n";
    for (int value : values) {
        cout << value << " ";
    }
    bubbleSort(values, SIZE);
    cout << "The sorted values are: \n";
    for (int value : values) {
        cout << value << " ";
    }
    system("pause");
    return 0;
}
void bubbleSort(int array[], int size) {
    //bubbleSort
    bool swapped;
    do {
        swapped= false;
        //Compare the first 2 elements
        //If out of order, exchange
        //Move down 1 elem., compare 2nd and 3rd elems
        //    Exchange those if necessary
        //    Continue until end of array
        for (int count = 0; count < size - 1; count++) {
            if (array[count] > array[count + 1]) {
                swap(array, count, count + 1);
                swapped = true;
            }
        }
    } while (swapped);
    //Pass through array again, exchanging as needed
    //Keep track of swaps using swapped
    //Repeat until pass made with no exchanges
}

void swap(int arr[], int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;

}
```

Selection Sort (Also has Time Complexity: n^2)

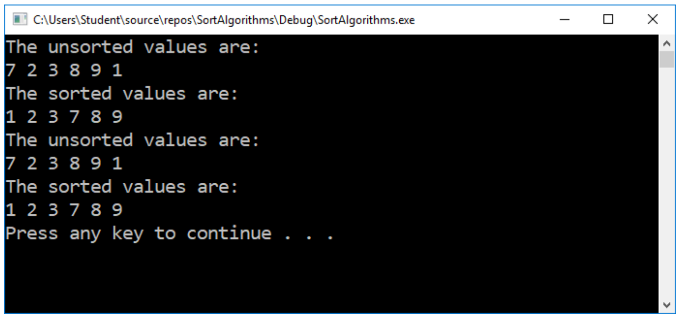
- The smallest value in the array is located and moved to element 0
- Then, the next smallest value is located and moved to element 1
- This process continues until all elements are in the proper order

2	13	5	23	11	3	19
11	13	5	23	2	3	19
2	13	5	23	11	3	19
2	13	5	23	11	3	19
2	3	5	23	11	13	19
2	3	5	23	11	13	19
2	3	5	11	23	13	19
2	3	5	11	23	13	19
2	3	5	11	13	23	19
2	3	5	11	13	19	23

```
#include <iostream>
using namespace std;

void swap(int arr[], int i, int j);
void bubbleSort(int array[], int size);
int findMin(int a[], int start, int size);
void selectionSort(int ARR[], int size);

int main() {
    const int SIZE = 6;
    int values[] = { 7, 2, 3, 8, 9, 1 };
    cout << "The unsorted values are: \n";
    for (int value : values) {
```

```
int main() {
    const int SIZE = 6;
    int values[] = { 7, 2, 3, 8, 9, 1 };
    cout << "The unsorted values are: \n";
    for (int value : values) {
        cout << value << " ";
    }
    bubbleSort(values, SIZE);
    cout << "\nThe sorted values are: \n";
    for (int value : values) {
        cout << value << " ";
    }


    int values2[] = { 7, 2, 3, 8, 9, 1 };
    cout << "\nThe unsorted values are: \n";
    for (int value : values2) {
        cout << value << " ";
    }
    selectionSort(values2, SIZE);
    cout << "\nThe sorted values are: \n";
    for (int value : values2) {
        cout << value << " ";
    }
    cout << endl;
    system("pause");
    return 0;
}
void bubbleSort(int array[], int size) {
    //bubbleSort
    bool swapped;
    do {
        swapped= false;
        //Compare the first 2 elements
        //If out of order, exchange
        //Move down 1 elem., compare 2nd and 3rd elems
        //    Exchange those if necessary
        //    Continue until end of array
        for (int count = 0; count < size - 1; count++) {
            if (array[count] > array[count + 1]) {
                swap(array, count, count + 1);
                swapped = true;
            }
        }
    } while (swapped);
    //Pass through array again, exchanging as needed
    //Keep track of swaps using swapped
    //Repeat until pass made with no exchanges
}

void selectionSort(int ARR[], int size) {
    //find minimum starting at 0, 1, 2, ...
    //swap the minimum with 1st element
    //find the minimum starting at 1, 2, 3, ...
    //swap the minimum with the 2nd element
    for (int count = 0; count < size - 1; count++) {
        //try to find the lower number
        int index = findMin(ARR, count, size);
        //put it in a certain position
        swap(ARR, count, index);

    }


}



/*findMin function
* Return index of the smallest value in an array
* starting at a certain position
*/
int findMin(int a[], int start, int size) {
    int min = a[start]; //temporarily make 1st value min
    int minIndex = start;
    for (int i = start; i < size; i++) {
        if (a[i] < min) {
            min = a[i];
            minIndex = i;
        }
    }
    return minIndex;
}
```

```
        }
    }
    return minIndex;
}

void swap(int arr[], int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;

}
```



C:\Users\Student\source\repos\SortAlgorithms\Debug\SortAlgorithms.exe

```
The unsorted values are:
7 2 3 8 9 1
The sorted values are:
1 2 3 7 8 9
The unsorted values are:
7 2 3 8 9 1
The sorted values are:
1 2 3 7 8 9
Press any key to continue . . .
```