Searching

Class 27

Introduction

- two of the most fundamental concepts in computer science are, given an array of values:
  - search through the values to see if a specific value is present and, if so, where
  - sort the values in nondecreasing order

Introduction

- two of the most fundamental concepts in computer science are, given an array of values:
  - search through the values to see if a specific value is present and, if so, where
  - sort the values in nondecreasing order
- as a computer scientist, you must be able to understand and program several different algorithms for each of these tasks

Introduction

- two of the most fundamental concepts in computer science are, given an array of values:
  - search through the values to see if a specific value is present and, if so, where
  - sort the values in nondecreasing order

- as a computer scientist, you must be able to understand and program several different algorithms for each of these tasks

- in all of these slides, “array” is a generic term
- it means either an old-fashioned C-array or a modern STL vector

Searching

- an array that contains an arbitrary number of values
- each value is in a specific array location, but the values are not sorted

17	23	5	11	2	29	3

Searching

- an array that contains an arbitrary number of values
- each value is in a specific array location, but the values are not sorted

17	23	5	11	2	29	3

- search for the value 11
- do a comparison on 17, 23, 5, 11 (4 comparisons)
- return 3 (the position where 11 was found)

Searching

- an array that contains an arbitrary number of values
- each value is in a specific array location, but the values are not sorted

17	23	5	11	2	29	3

- search for the value 11
- do a comparison on 17, 23, 5, 11 (4 comparisons)
- return 3 (the position where 11 was found)

- search for the value 7
- do comparisons on 17, 23, 5, 11, 2, 29, 3 (7 comparisons)
- return a not-found indicator

Linear Search

- Program 8-1, page 464, a function to do this

```
int linearSearch(const int arr[], int size, int value);
```

- up to size elements of the array arr are searched for the existence of value
- if value is found within the first size elements of arr, the first position where value is encountered is returned by the function
- if value is not found, the sentinel value -1 is returned

Linear Search

- Program 8-1, page 464, a function to do this

```
int linearSearch(const int arr[], int size, int value);
```

- up to size elements of the array arr are searched for the existence of value
- if value is found within the first size elements of arr, the first position where value is encountered is returned by the function
- if value is not found, the sentinel value -1 is returned

- the search is called linear search because the algorithm searches along the line of array elements, one by one, starting at the beginning, trying to find value

Problems

- there are several problems with Gaddis' approach
- the biggest one is that none of his int data types will be accepted by a modern compiler like clang
- the data type for a size is size_t, and size_t is an unsigned integer; thus, it is impossible to return -1 as a sentinel value for the not-found condition

Problems

- there are several problems with Gaddis' approach
- the biggest one is that none of his int data types will be accepted by a modern compiler like clang
- the data type for a size is size_t, and size_t is an unsigned integer; thus, it is impossible to return -1 as a sentinel value for the not-found condition

- a solution to this problem is to return the size of the array as the not-found sentinel value
- in an array of size, say 10, the largest valid index is 9
- if the searched-for value is not found, we return 10, which is not a valid index, indicating not-found
- look at the complete program, using a vector: program_8_1_modified.cpp

Notice

- several things to note in the modified program:
  - the return type of the function is size_t
  - the index and position are also of type size_t
  - there is no parameter for size because a vector knows how big it is

Linear Search Analysis

- it is traditional to use n to denote the number of elements of an array (this is the same thing as vector's size)

Linear Search Analysis

- it is traditional to use n to denote the number of elements of an array (this is the same thing as vector's size)
- do the following experiment (I encourage you to write this program):

Linear Search Analysis

- it is traditional to use n to denote the number of elements of an array (this is the same thing as vector's size)
- do the following experiment (I encourage you to write this program):
    1. put 1,000 random values, chosen from 0 – 10,000, into a vector

Linear Search Analysis

- it is traditional to use n to denote the number of elements of an array (this is the same thing as vector's size)
- do the following experiment (I encourage you to write this program):
  1. put 1,000 random values, chosen from 0 – 10,000, into a vector
  2. at random, pick a value in the range 0 – 10,000 and

Linear Search Analysis

- it is traditional to use n to denote the number of elements of an array (this is the same thing as vector's size)
- do the following experiment (I encourage you to write this program):
    1. put 1,000 random values, chosen from 0 – 10,000, into a vector
    2. at random, pick a value in the range 0 – 10,000 and
        2.1 see if it is in the vector or not

Linear Search Analysis

- it is traditional to use n to denote the number of elements of an array (this is the same thing as vector's size)
- do the following experiment (I encourage you to write this program):
  1. put 1,000 random values, chosen from 0 – 10,000, into a vector
  2. at random, pick a value in the range 0 – 10,000 and
     2.1 see if it is in the vector or not
     2.2 count how many comparisons it takes to find out

Linear Search Analysis

- it is traditional to use n to denote the number of elements of an array (this is the same thing as vector's size)
- do the following experiment (I encourage you to write this program):
    1. put 1,000 random values, chosen from 0 – 10,000, into a vector
    2. at random, pick a value in the range 0 – 10,000 and
        2.1 see if it is in the vector or not
        2.2 count how many comparisons it takes to find out
    3. repeat step 2 10,000 times

Linear Search Analysis

- it is traditional to use n to denote the number of elements of an array (this is the same thing as vector's size)
- do the following experiment (I encourage you to write this program):
  1. put 1,000 random values, chosen from 0 – 10,000, into a vector
  2. at random, pick a value in the range 0 – 10,000 and
     2.1 see if it is in the vector or not
     2.2 count how many comparisons it takes to find out
  3. repeat step 2 10,000 times
  4. calculate the average number of comparisons from step 2

Linear Search Analysis

- results:

```
Total hits: 930 Total misses: 9070
Minimum comparisons: 1 Maximum comparisons: 1000
Average number of comparisons: 954.747
```

- with 1,000 random values from 0 to 10,000
- the average number of comparisons is about 955
- have to search almost the entire array almost every time

Linear Search

Linear Search Pros

- very easy algorithm to understand
- easy algorithm to code correctly
- the only practical algorithm for unsorted values

Linear Search Cons

- inefficient for sorted values
- must examine $n/2$ elements on average for a value that is in the array
- must examine all n elements for a value not in the array

Linear Search

Linear Search Pros

- very easy algorithm to understand
- easy algorithm to code correctly
- the only practical algorithm for unsorted values

Linear Search Cons

- inefficient for sorted values
- must examine $n/2$ elements on average for a value that is in the array
- must examine all n elements for a value not in the array

- for sorted values, we can do better

Searching a Sorted List

- linear search works perfectly on an unsorted array of values
- for unsorted values, it is the only practical searching algorithm

Searching a Sorted List

- linear search works perfectly on an unsorted array of values
- for unsorted values, it is the only practical searching algorithm

- linear search also works perfectly on a sorted array of values
- however, is it not the only search algorithm for sorted values
- furthermore, it is not a even a good algorithm in this case

Enhanced Linear Search

- if the elements of the array are sorted, we can do much better
- we can stop when any of three conditions is true:
  1. we find the item
  2. we reach the end of the list
  3. we reach an element bigger than the element we're searching for

2	3	5	11	27	23	29

Enhanced Linear Search

- if the elements of the array are sorted, we can do much better
- we can stop when any of three conditions is true:
  1. we find the item
  2. we reach the end of the list
  3. we reach an element bigger than the element we're searching for

2	3	5	11	27	23	29

- search for the value 11
- do a comparison on 2, 3, 5, 11 (4 comparisons)
- return 3 (the position where 11 was found)

Enhanced Linear Search

- if the elements of the array are sorted, we can do much better
- we can stop when any of three conditions is true:
    1. we find the item
    2. we reach the end of the list
    3. we reach an element bigger than the element we're searching for

2	3	5	11	27	23	29

- search for the value 11
- do a comparison on 2, 3, 5, 11 (4 comparisons)
- return 3 (the position where 11 was found)

- search for the value 12
- do comparisons on 2, 3, 5, 11, 27 (just 5 comparisons)
- return a not-found indicator

Binary Search

2	3	5	11	17	23	29

1. divide the range of elements to search into 3:

Binary Search

2	3	5	11	17	23	29

1. divide the range of elements to search into 3:
   1.1 the very middle element

Binary Search

2	3	5	11	17	23	29

1. divide the range of elements to search into 3:
   1.1 the very middle element
   1.2 the elements to the left of middle

Binary Search

2	3	5	11	17	23	29

1. divide the range of elements to search into 3:
   1.1 the very middle element
   1.2 the elements to the **left** of middle
   1.3 the elements to the right of middle

Binary Search

2	3	5	11	17	23	29

1. divide the range of elements to search into 3:
    1.1 the very middle element
    1.2 the elements to the **left** of middle
    1.3 the elements to the right of middle
2. if the range of elements is **empty**, return not-found sentinel

Binary Search

2	3	5	11	17	23	29

1. divide the range of elements to search into 3:
   1.1 the very middle element
   1.2 the elements to the **left** of middle
   1.3 the elements to the right of middle
2. if the range of elements is **empty**, return not-found sentinel
3. else if the searched-for value is the middle element, you've found it and you're done

Binary Search

2	3	5	11	17	23	29

1. divide the range of elements to search into 3:
   1.1 the very middle element
   1.2 the elements to the **left** of middle
   1.3 the elements to the right of middle
2. if the range of elements is **empty**, return not-found sentinel
3. else if the searched-for value is the middle element, you've found it and you're done
4. else if the searched-for value is **smaller** than the middle element, repeat step 1 on the **left half**

Binary Search

2	3	5	11	17	23	29

1. divide the range of elements to search into 3:
   1.1 the very middle element
   1.2 the elements to the **left** of middle
   1.3 the elements to the right of middle
2. if the range of elements is **empty**, return not-found sentinel
3. else if the searched-for value is the middle element, you've found it and you're done
4. else if the searched-for value is **smaller** than the middle element, repeat step 1 on the **left half**
5. else repeat step 1 on the right half

Binary Search

2	3	5	11	17	23	29

- search for 11

Binary Search

2	3	5	11	17	23	29

- search for 11
- do a comparison on 11 (1 comparison)

Binary Search

2	3	5	11	17	23	29

- search for 11
- do a comparison on 11 (1 comparison)
- return 3 (the position where 11 was found) — done

Binary Search

2	3	5	11	17	23	29

- search for 11
- do a comparison on 11 (1 comparison)
- return 3 (the position where 11 was found) — done

- search for 7

Binary Search

2	3	5	11	17	23	29

- search for 11
- do a comparison on 11 (1 comparison)
- return 3 (the position where 11 was found) — done

- search for 7
- do a comparison on 11, 3, 5 (3 comparisons)

Binary Search

2	3	5	11	17	23	29

- search for 11
- do a comparison on 11 (1 comparison)
- return 3 (the position where 11 was found) — done

- search for 7
- do a comparison on 11, 3, 5 (3 comparisons)
- report not-found — done

Binary Search Implementation

- as with linear search, Gaddis' implementation won't compile on clang
- see program_8_2_modified.cpp

Binary Search Analysis

- do the same simulation with binary search as with linear search
- 10,000 times, search for a random number 0 – 1,000
- results:

```
Total hits: 971 Total misses: 9029
Minimum comparisons: 1 Maximum comparisons: 10
Average number of comparisons: 9.8768
```

- remember the linear results:

```
Total hits: 930 Total misses: 9070
Minimum comparisons: 1 Maximum comparisons: 1000
Average number of comparisons: 954.747
```

- on average linear search takes a hundred times as many steps as binary search!
- binary search never takes more than ten comparisons

Binary Search Analysis

- let's dig a little deeper into exactly how long the two searches take

Binary Search Analysis

- let's dig a little deeper into exactly how long the two searches take
- linear search starts with a search space of size n

Binary Search Analysis

- let's dig a little deeper into exactly how long the two searches take
- linear search starts with a search space of size n
- with each comparison, the search space decreases by one

Binary Search Analysis

- let's dig a little deeper into exactly how long the two searches take
- linear search starts with a search space of size n
- with each comparison, the search space decreases by one
- if the value is not in the array, clearly it takes n steps to decrease the search space to size zero and determine not-found

Binary Search Analysis

- let's dig a little deeper into exactly how long the two searches take
- linear search starts with a search space of size n
- with each comparison, the search space decreases by one
- if the value is not in the array, clearly it takes n steps to decrease the search space to size zero and determine not-found

- binary search also starts with a search space of size n

Binary Search Analysis

- let's dig a little deeper into exactly how long the two searches take
- linear search starts with a search space of size n
- with each comparison, the search space decreases by one
- if the value is not in the array, clearly it takes n steps to decrease the search space to size zero and determine not-found

- binary search also starts with a search space of size n
- with each comparison, the search space decreases by half

Binary Search Analysis

- let's dig a little deeper into exactly how long the two searches take
- linear search starts with a search space of size n
- with each comparison, the search space decreases by one
- if the value is not in the array, clearly it takes n steps to decrease the search space to size zero and determine not-found

- binary search also starts with a search space of size n
- with each comparison, the search space decreases by half
- how many steps does it take to repeatedly divide a number by 2 until you get to zero?

Binary Search Analysis

- let's dig a little deeper into exactly how long the two searches take
- linear search starts with a search space of size n
- with each comparison, the search space decreases by one
- if the value is not in the array, clearly it takes n steps to decrease the search space to size zero and determine not-found

- binary search also starts with a search space of size n
- with each comparison, the search space decreases by half
- how many steps does it take to repeatedly divide a number by 2 until you get to zero?
- the term for this is logarithm

Logarithms

- logarithms are particularly important in algorithm analysis
- computer scientists deal mainly with base-2 logarithms
- the simplest working definition of a base-2 logarithm is

Base-2 Logarithm

How many times can you divide a number n by 2 using integer division before the result is 1 or 0?

- the answer to this question is approximately the base-2 logarithm of n
- you can estimate base-2 logarithms directly from the powers of 2 table

Powers of 2

you should learn all the powers of 2 from 0 to 10

$$
\begin{aligned}
2^0 &= 1\\
2^1 &= 2\\
2^2 &= 4\\
2^3 &= 8\\
2^4 &= 16\\
2^5 &= 32\\
2^6 &= 64\\
2^7 &= 128\\
2^8 &= 256\\
2^9 &= 512\\
2^{10} &= 1024
\end{aligned}
$$

Binary Search

- remember the simulation results searching in an array with 1,000 elements
- the most comparisons ever needed was 10
- the base-2 logarithm of 1,000 is approximately 10
- the number of comparisons needed for binary search for an array of size n is at most $\log_2 n$

Binary Search

- remember the simulation results searching in an array with 1,000 elements
- the most comparisons ever needed was 10
- the base-2 logarithm of 1,000 is approximately 10
- the number of comparisons needed for binary search for an array of size n is at most $\log_2 n$

- compare this to linear search, which needs at most 1,000 comparisons for a 1,000-element array

Binary Search

Binary Search Pros

- much more efficient than linear search
- requires at most $\log_2 n$ comparisons
- easy algorithm to understand and code correctly

Binary Search Cons

- requires array elements to already be sorted

Binary Search

Binary Search Pros

- much more efficient than linear search
- requires at most $\log_2 n$ comparisons
- easy algorithm to understand and code correctly

Binary Search Cons

- requires array elements to already be sorted

- clearly, sorting is an issue, and we turn to that next