CSE 2320

Week of 06/08/2020

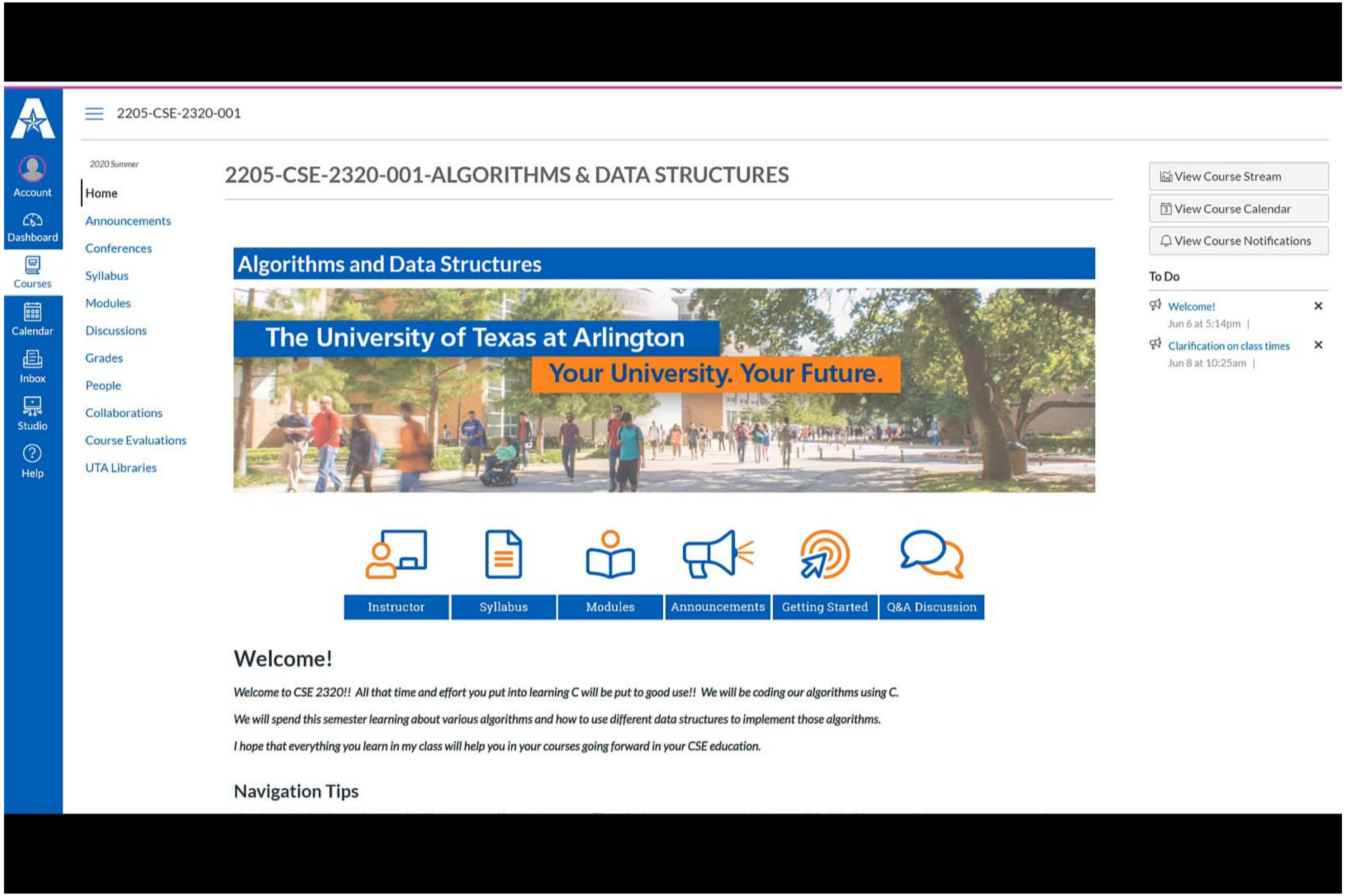
Instructor : Donna French

Tools Needed for this Class

- Text editor or IDE that recognizes the C language (syntax highlighting)
- File Transfer program (FileZilla)
- Terminal emulator (PuTTY or SSH)
- Oracle VM

We will be using BOTH Omega and the Oracle VM to compile our programs.

Installing the VM



The Role of Algorithms in Computing

Informal definition

An algorithm is any well-defined computational procedure that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**.

An algorithm is a sequence of computational steps that transform the **input** to the **output**.

The Role of Algorithms in Computing

An algorithm is also a tool for solving a well-specified computational problem.

The statement of the problem specifies in general terms the desired input/output relationship.

The algorithm describes a specific computational procedure for achieving that input/output relationship.

The Role of Algorithms in Computing

Informal definition of problem :

Sort a sequence of numbers into nondecreasing order.

Formal definition of problem :

Input : A sequence of n numbers $\langle a_1, a_2, \ldots, an \rangle$.

Output : A permutation (reordering) $\langle a'1, a'2, \ldots a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

The Role of Algorithms in Computing

Input : A sequence of n numbers $\langle a_1, a_2, \ldots, an \rangle$.

Output : A permutation (reordering) $\langle a'1, a'2, \ldots a'_n \rangle$ of the input sequence such that $a'_1 \le a'_2 \le \ldots \le a'_n$.

Given the input sequence of

$$\langle 31, 41, 59, 26, 41, 58 \rangle$$

our sorting algorithm would return the output as

$$\langle 26, 31, 41, 41, 58, 59 \rangle$$

The Role of Algorithms in Computing

This input sequence

$$\langle 41, 31, 58, 26, 41, 59 \rangle$$

or this input sequence

$$\langle 31, 41, 59, 26, 41, 58 \rangle$$

or this input sequence

$$\langle 58, 59, 41, 41, 26, 31 \rangle$$

would all result in the same output

$$\langle 26, 31, 41, 41, 58, 59 \rangle$$

Each of these are an ***instance*** of the sorting problem.

An ***instance of a problem*** consists of the input needed to compute a solution to the problem.

The Role of Algorithms in Computing

Sorting is a fundamental operation in computer science.

Which sorting algorithm is best for a given application depends on many factors…

- the number of items to be sorted
- the extent to which the items are already somewhat sorted
- possible restrictions on the item values
- the architecture of the computer
- type of storage device (memory, disk, tape…)

The Role of Algorithms in Computing

So what makes an algorithm "correct"?

An algorithm is called **correct** if, for every input instance, it halts with the correct output.

A correct algorithm solves the given computational problem.

An incorrect algorithm might not halt at all on some input instances or it might halt with an incorrect answer.

The Role of Algorithms in Computing

What kinds of problems are solved by algorithms?

Lots and lots and lots of them!!!

- Sorting
- Internet – managing and manipulating large volumes of data
- Internet – routing data
- Internet – searching
- Cryptography and digital signatures
- Allocation of resources
- GPS routing

The Role of Algorithms in Computing

Two characteristics common to many algorithmic problems

1. Have many candidate solutions
   a. most of which do not solve the problem
   b. finding a solution that does or is the best can be very challenging

2. Have practical applications
   a. You want your GPS to not only find a route but find the best route – which may not be the shortest because of other factors

The Role of Algorithms in Computing

We are going to use various data structures to store the data our algorithms will be manipulating.

A data structure is a way to store and organize data in order to facilitate access and modifications.

No single data structure works well for all purposes.

We will learn the strengths and limitations of several.

The Role of Algorithms in Computing

Other than speed, what other measures of efficiency might one use in a real-world setting?

Resources

Memory space

For example, using recursion may be the most efficient approach timewise but does the machine running the algorithm have the resources/memory space to execute the recursion? Remember all of those function execution environments?

The Role of Algorithms in Computing

Select a data structure that you have seen previously and discuss its strengths and limitations.

Arrays vs Linked Lists

Arrays require contiguous memory but allow for random access.

Linked lists do not require contiguous memory but are limited to sequential access.

An array's size is set when it is created – a linked list can grow.

The Role of Algorithms in Computing

Come up with a real-world problem in which only the best solution will do. Then come up with one in which a solution that is "approximately" the best is good enough.

Launching a probe to Mars

vs

Driving to Grandma's house for Sunday dinner

The Role of Algorithms in Computing

Why study algorithms?

Computers are fast but not infinitely fast.

Memory may not be expensive, but it is not free.

Computing time is a bounded resource and so is space in memory.

We need to use these resources efficiently.

The Role of Algorithms in Computing

Efficiency

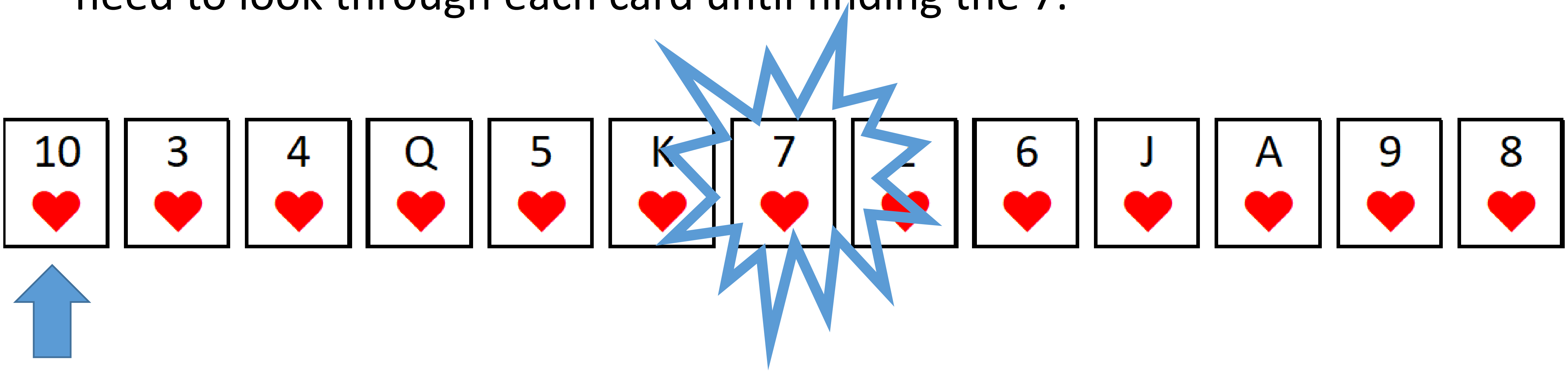
Different algorithms devised to solve the same problem often differ dramatically in their efficiency.

These differences can be much more significant than differences due to hardware and software.

Linear Search Algorithm

A **linear search** or **sequential search** is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched.

For example, if you have a set of cards and need to find the 7, you will need to look through each card until finding the 7.

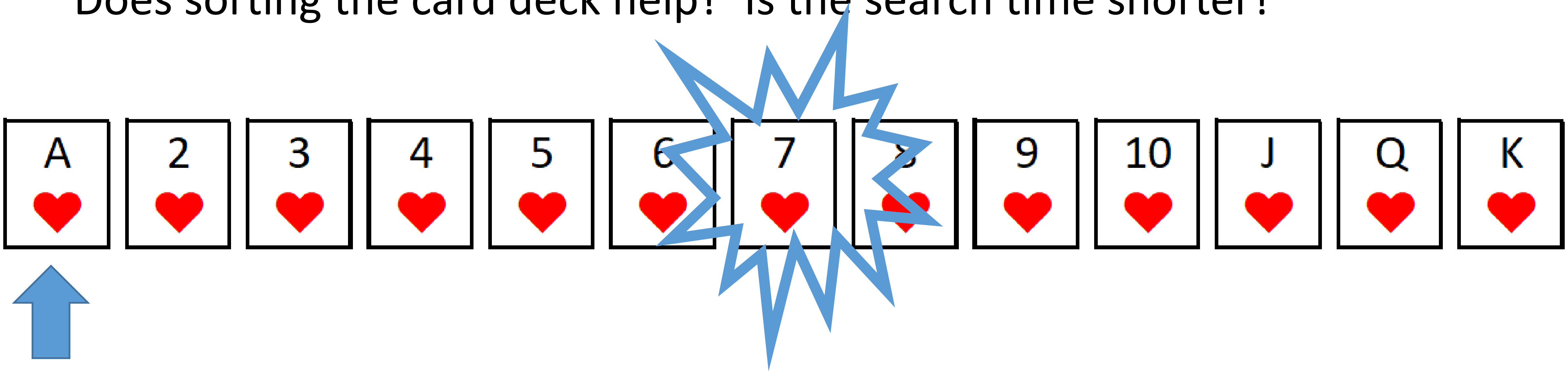


Linear Search Algorithm

The card you are looking for might be the first card or it might be the last card or it could be anywhere in between.

You would have to compare each card one by one.

Does sorting the card deck help? Is the search time shorter?



Linear Search Algorithm

Best case scenario – the card we are searching for is the first card.

Worst case scenario – the card we are searching for is the last card.

Sorting the information does not change either of these scenarios.

What happens when the card we are searching for is not in the set?

When you look up a word in the dictionary, do you start with the first word on the first page and keep checking every word until you find the one you want?

Hope you are not looking for "zyzzya".

The **Oxford English Dictionary** has a new "**last word**" – "zyzzyva", which is a genus of tropical weevil native to South America typically found in palm trees.

C:\Users\frenc\Desktop\VM2320\LinearSearch.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

new 7 | InsertionSort.c | new 1 | MergeSort.c | new 2 | new 3 | MergeSortFile.c | MSFile.txt | RandFileGen.c | Cards.txt | LinearSearch.c

```c
/* C program for Linear Search */
#include<stdlib.h>
#include<stdio.h>

/* Function to print an array */
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n\n");
}

int main(int argc, char *argv[])
{
    int arr[] = {12, 11, 5, 13, 7, 6};
    int arr_size = sizeof(arr)/sizeof(arr[0]);
    int i = 0;
    int FoundIt = 0;
    int SearchValue = atoi(argv[1]);

    printArray(arr, arr_size);

    while (i < arr_size)
    {
        if (arr[i++] == SearchValue)
            FoundIt = 1;
    }

    if (FoundIt)
        printf("Found %d in array after %d checks\n", SearchValue, i);
```

C source file | length : 707 lines : 37 | Ln : 26 Col : 37 Sel : 0 | 0 | Windows (CR LF) | UTF-8 | INS

Type here to search | 6:13 PM 6/7/2020

Linear Search Algorithm

Adding the logic to break out of the search when the value is found helps with the number of searches but only if the search item is found early in the search.

Linear Search is not very useful except for very small sets of data or when sorting would actually take more time.

For example, in class, after everyone turns in their written quiz, I have a stack of unsorted papers. You approach me after class and ask to see your paper to make sure you put on ID on it. To find your quiz, I have to perform a linear search.

It would not be beneficial to sort the stack alphabetically by last name and then find your paper.

Linear Search Algorithm

But, when I store the papers in my filing cabinet, I do alphabetize them first.

Why?

One reason is because it is easier for me to enter them into the gradebook when they are alphabetized because the gradebook is alphabetized.

Also, once the papers are sorted (alphabetized), I can perform a better, much more efficient search if someone asks to see their paper…

Binary Search Algorithm

Binary search is an efficient algorithm for finding an item from a sorted list of items.

It works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one.

One of the most common ways to use binary search is to find an item in an array.

C:\Users\frenc\Desktop\VM2320\BinarySearchI.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

new 7 | InsertionSort.c | new 1 | MergeSort.c | new 2 | new 3 | MergeSortFile.c | MSFile.txt | RandFileGen.c | Cards.txt | LinearSearch.c | BinarySearchI.c | concomDemo.c

```c
// C program to implement iterative Binary Search
#include <stdio.h>
#include <stdlib.h>

/* Function to print an array or a portion of an array */
void printArray(int A[], int start, int end)
{
    int i;
    for (i=start; i <= end; i++)
    {
        if (i != end)
            printf("%d,", A[i]);
        else
            printf("%d}\n", A[i]);
    }
}


// Return index of SearchValue or -1 if SearchValue not found
int binarySearch(int SortedArray[], int left, int right, int SearchValue)
{
    int middle = 0;

    while (left <= right)
    {
        middle = (right + left) / 2;

        // Check if SearchValue is present at mid
        if (SortedArray[middle] == SearchValue)
            return middle;

```

C source file   length : 1,800   lines : 75   Ln : 1   Col : 1   Sel : 0 | 0   Windows (CR LF)   UTF-8   INS

Type here to search   9:31 PM 6/7/2020

```c
int main(void)
{
    //conditional compile
    #ifdef ARRAYSIZE11
        int SortedArray[] = {2, 3, 4, 10, 11, 15, 40, 42, 47, 49, 50};
    #else
        int SortedArray[] = {2, 3, 4, 10, 11, 15, 40, 42, 47, 49};
    #endif

    int NumberOfElements = sizeof(SortedArray) / sizeof(SortedArray[0]);
    int SearchValue = 0;

    printf("Enter search value ");
    scanf("%d", &SearchValue);

    printf("\nSearch array -> {");
    printArray(SortedArray, 0, NumberOfElements-1);

    int result = binarySearch(SortedArray, 0, NumberOfElements - 1, SearchValue);

    (result == -1) ? printf("Element is not present in SortedArray\n")
                   : printf("Element is present at index %d\n", result);
    return 0;
}
```

```c
// Return index of SearchValue or -1 if SearchValue not found
int binarySearch(int SortedArray[], int left, int right, int SearchValue)
{
    int middle = 0;

    while (left <= right)
    {
        middle = (right + left) / 2;

        // Check if SearchValue is present at mid
        if (SortedArray[middle] == SearchValue)
            return middle;

        // If SearchValue greater, ignore left half
        if (SortedArray[middle] < SearchValue)
            left = middle + 1;

        // If SearchValue is smaller, ignore right half
        else
            right = middle - 1;

        if (left <= right)  // print array
        {
            printf("Now searching this portion of the array -> {");
            printArray(SortedArray, left, right);
        }
    }

    // Did not find it
    return -1;
}
```

Binary Search Algorithm

```
Enter search value 4

Search array -> {2,3,4,10,11,15,40,42,47,49}
Now searching this portion of the array -> {2,3,4,10}
Now searching this portion of the array -> {4,10}
Element is present at index 2

Enter search value 15

Search array -> {2,3,4,10,11,15,40,42,47,49}
Now searching this portion of the array -> {15,40,42,47,49}
Now searching this portion of the array -> {15,40}
Element is present at index 5

Enter search value 49

Search array -> {2,3,4,10,11,15,40,42,47,49}
Now searching this portion of the array -> {15,40,42,47,49}
Now searching this portion of the array -> {47,49}
Now searching this portion of the array -> {49}
Element is present at index 9
```

Binary Search Algorithm

```
Enter search value 5

Search array -> {2,3,4,10,11,15,40,42,47,49,50}
Now searching this portion of the array -> {2,3,4,10,11}
Now searching this portion of the array -> {10,11}
Element is not present in SortedArray

Enter search value 2

Search array -> {2,3,4,10,11,15,40,42,47,49,50}
Now searching this portion of the array -> {2,3,4,10,11}
Now searching this portion of the array -> {2,3}
Element is present at index 0

Enter search value 50

Search array -> {2,3,4,10,11,15,40,42,47,49,50}
Now searching this portion of the array -> {40,42,47,49,50}
Now searching this portion of the array -> {49,50}
Now searching this portion of the array -> {50}
Element is present at index 10
```

```c
// Return index of SearchValue or -1 if SearchValue not found
int binarySearch(int SortedArray[], int left, int right, int SearchValue)
{
    int middle = 0;

    if (left <= right)
    {
        middle = (right + left) / 2;

        // Check if SearchValue is present at mid
        if (SortedArray[middle] == SearchValue)
            return middle;

        // If SearchValue greater, ignore left half
        if (SortedArray[middle] < SearchValue)
            return binarySearch(SortedArray, middle + 1, right, SearchValue);

        // If SearchValue is smaller, ignore right half
        else
            return binarySearch(SortedArray, left, middle - 1, SearchValue);

        if (left <= right)  // print array
        {
            printf("Now searching this portion of the array -> {");
            printArray(SortedArray, left, right);
        }
    }

    // Did not find it
    return -1;
}
```

R
E
C
U
R
S
I
O
N

Binary Search Algorithm

Linear search on an array of *n* elements might have to make as many as *n* guesses.

What about a binary search?

If we have 4 elements in an array, the 1st guess, if incorrect, cuts the searchable array down to half of the original size which would leave 2 elements to search.

The 2nd guess, if incorrect, cuts the searchable array down to 1 element.

The next guess is either right or wrong and no more searching is required.

4 elements required, at most, 3 guesses and 2 "divide in half" actions.

Binary Search Algorithm

If we have 8 elements in an array, the 1st guess, if incorrect, cuts the searchable array down to half of the original size which would leave 4 elements to search.

The 2nd guess, if incorrect, cuts the searchable array down to 2 elements.

The 3rd guess, if incorrect, cuts the searchable array down to 1 element.

The next guess is either right or wrong and no more searching is required.

8 elements required, at most, 4 guesses and 3 "divide in half" actions.

Binary Search Algorithm

If we have 16 elements in an array, the 1st guess, if incorrect, cuts the searchable array down to half of the original size which would leave 8 elements to search.

The 2nd guess, if incorrect, cuts the searchable array down to 4 elements.

The 3rd guess, if incorrect, cuts the searchable array down to 2 elements.

The 4th guess, if incorrect, cuts the searchable array down to 1 element.

The next guess is either right or wrong and no more searching is required.

16 elements required, at most, 5 guesses and 4 "divide in half" actions.

Binary Search Algorithm

If we have 32 elements in an array, the 1st guess, if incorrect, cuts the searchable array down to half of the original size which would leave 16 elements to search.

The 2nd guess, if incorrect, cuts the searchable array down to 8 elements.

The 3rd guess, if incorrect, cuts the searchable array down to 4 elements.

The 4th guess, if incorrect, cuts the searchable array down to 2 elements.

The 5th guess, if incorrect, cuts the searchable array down to 1 element.

The next guess is either right or wrong and no more searching is required.

32 elements required, at most, 6 guesses and 5 "divide in half" actions

Binary Search Algorithm

4 elements -> 3 guesses -> and 2 "divide in half" actions

8 elements -> 4 guesses -> and 3 "divide in half" actions

16 elements -> 5 guesses -> and 4 "divide in half" actions

32 elements -> 6 guesses -> and 5 "divide in half" actions

See the pattern?

Every time the size of the array doubled, we added one more guess.

Binary Search Algorithm

Let *m* represent the number of times the array was halved and *m*+1 represent the number of guesses. Let *n* represent the number of elements in the array.

n **elements**	***m*** **halving actions**	***m*+1** **total guesses**
4	2	3
8	3	4
16	4	5
32	5	6

We can describe the number of guesses, in the worst case, as "the number of times we can repeatedly halve, starting at *n*, until we get the value 1, plus one."

Binary Search Algorithm

"the number of times we can repeatedly halve, starting at *n*, until we get the value 1, plus one."

n **elements**	***m*** **halving actions**	***m*+1** **total guesses**
4	2	3
8	3	4
16	4	5
32	5	6

Does this sound familiar? Does this pattern look familiar?

Binary Search Algorithm

"the number of times we can repeatedly halve, starting at *n*, until we get the value 1, plus one."

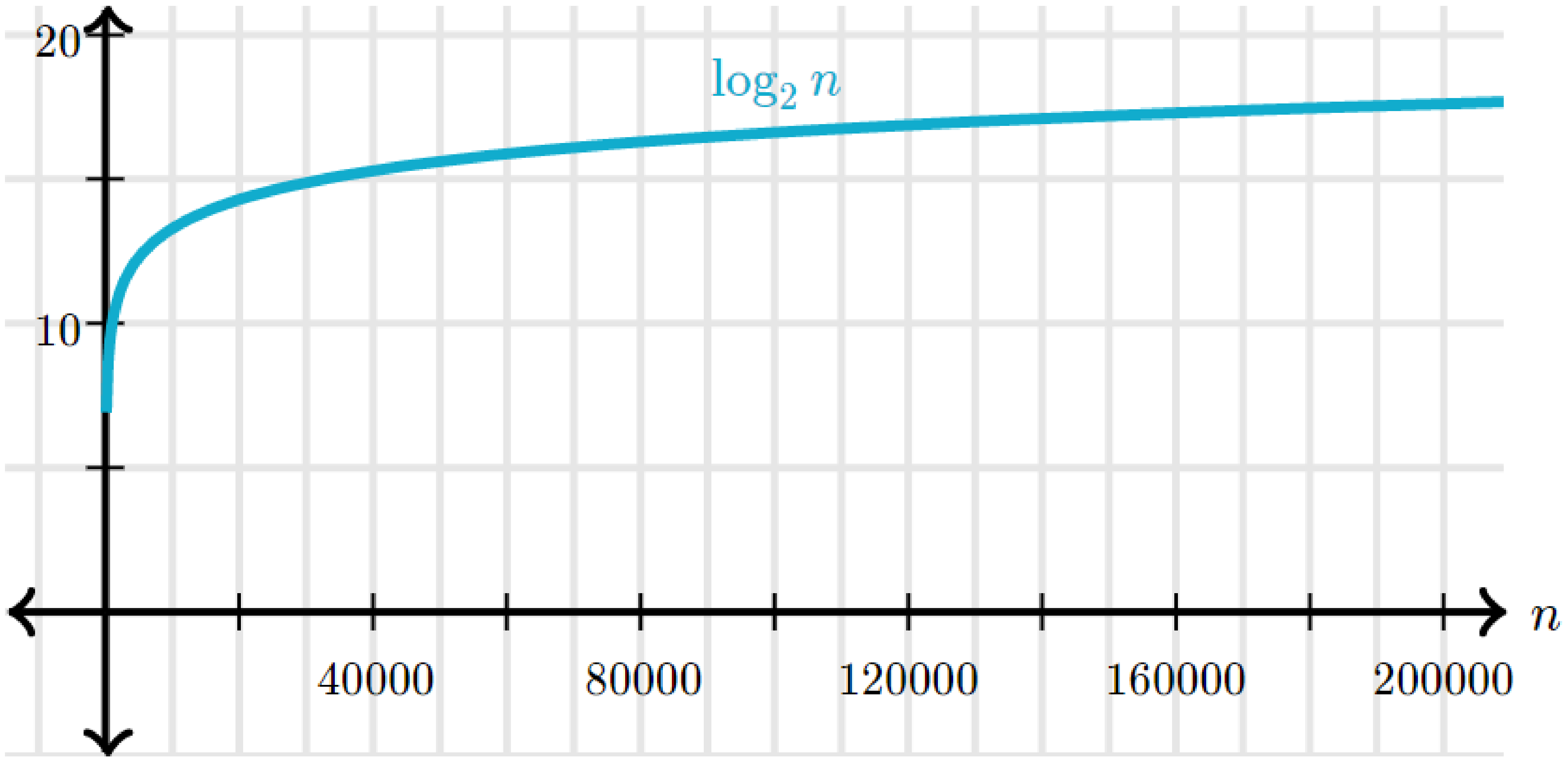
n elements	*m* halving actions	*m*+1 total guesses
4	2	3
8	3	4
16	4	5
32	5	6

n	$\log_2 n$
1	0
2	1
4	2
8	3
16	4
32	5
64	6

This is describing the mathematical function base 2 logarithm of *n*

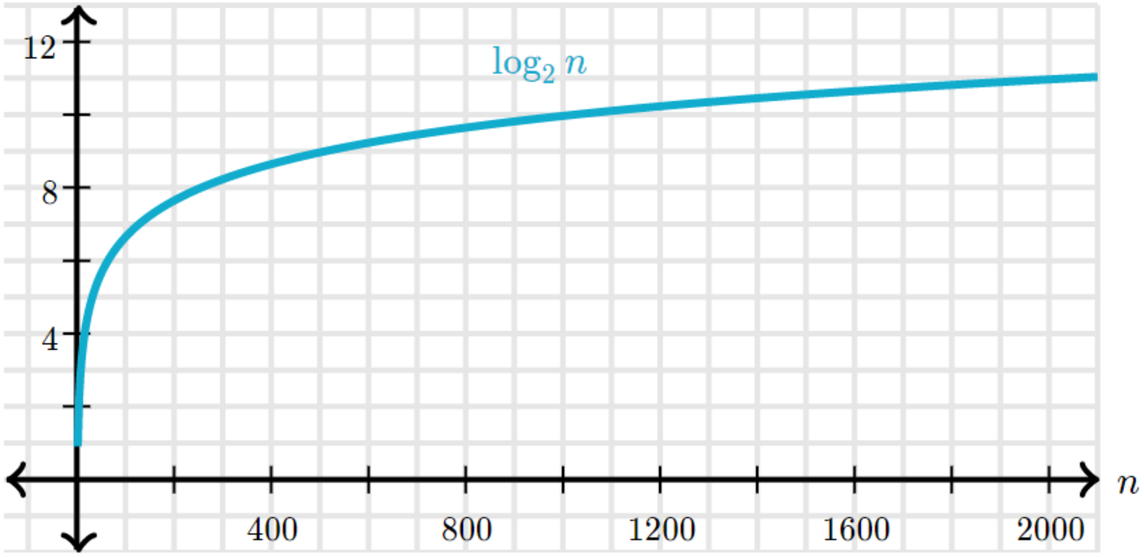
Binary Search Algorithm

n	$\log_2 n$
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024	10
1,048,576	20
2,097,152	21



Binary Search Algorithm

n	$\log_2 n$
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024	10
1,048,576	20
2,097,152	21

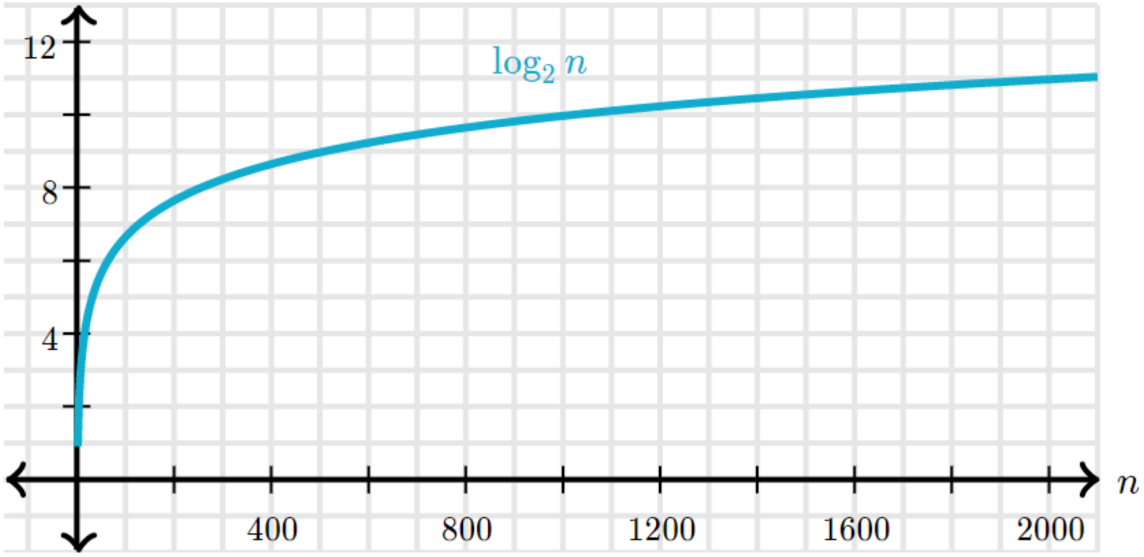


The logarithm function grows very slowly.

Logarithms are the inverse of exponentials, which grow very rapidly.

Binary Search Algorithm

n	$\log_2 n$
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024	10
1,048,576	20
2,097,152	21



Remember that $\log_2 n = x$ means $n = 2^x$

$\log_2 128 = 7$ because $2^7 = 128$

$\log_2 2097152 = 21$ because $2^{21} = 2{,}097{,}152$

Binary Search Algorithm

It is easy to calculate the runtime of a binary search algorithm where the number of elements being searched, *n*, is exactly a power of 2.

If *n* is 128, how many guesses will a binary search require (worst case)?

For a *n* of 128, a binary search will require at most

$\log_2 128 + 1$

A total of 8 guesses (7 + 1).

Binary Search Algorithm

What if *n* isn't a power of 2?

We would look at the closest lower power of 2.

For an *n* of 1000 (remember that *n* is the number of elements in the array), what is the closest **lower** power of 2?

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
1	2	4	8	16	32	64	128	256	512	1024

Binary Search Algorithm

For an *n* of 1000 (remember that *n* is the number of elements in the array), what is the closest **lower** power of 2?

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
1	2	4	8	16	32	64	128	256	512	1024

It might be tempting to go with 2^{10} since 1024 is SO much closer to 1000 than 512 (2^9).

Why is that not a good plan?

Binary Search Algorithm

2^9 (512) vs 2^{10} (1024) when n = 1000

Since $\log_2 512$ is 9 and $\log_2 1024$ is 10, we can estimate that $\log_2 1000$ is between 9 and 10.

The actual value of $\log_2 1000$ is approximately 9.97.

Since we don't have 0.97th of a guess, we take just the 9 and add that one final guess to get a total number of guesses of 10 (worst case).

Binary Search Algorithm

Did we need to get out the calculator to get the exact value of $\log_2 n$ in this case?

What if we had just used 1024 because it closest to 1000?

$\log_2 1024$ is 10. Adding that one final guess would make the total number of guesses 11.

$\log_2 512$ is 9. Adding that one final guess would make the total number of guesses 10.

Binary Search Algorithm

1000

1st guess -> 500

2nd guess -> 250

3rd guess -> 125

4th guess -> 62

5th guess -> 31

6th guess -> 15

7th guess -> 7

8th guess -> 3

9th guess -> 1

Final guess

TOTAL -> 10

Using 1024 for our estimate leads to a number of guesses of 11

Using 512 for our estimate leads to a number of guess of 10.

10 is correct.

This is why we would use the closest **lower** power of 2.

Binary Search Algorithm

The Tycho-2 Catalog is an astronomical catalog of more than 2.5 million of the brightest stars.

2,539,913 of the brightest stars in the Milky Way to be more precise.

Each star is numbered using its Guide Star region number (0001-9537) and a five-digit star number within each region, separated by a decimal point.

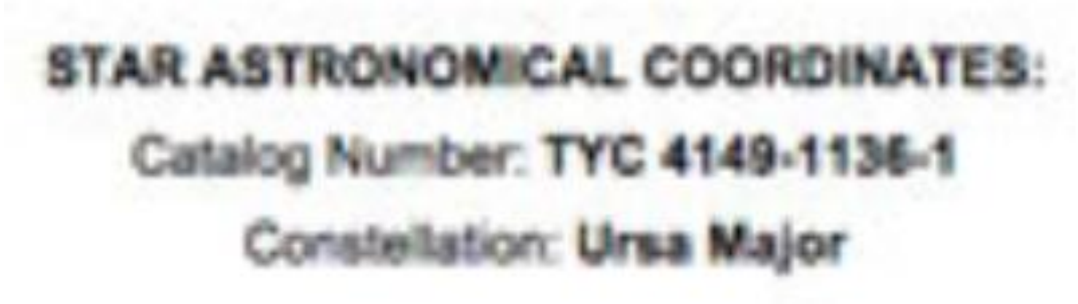
Binary Search Algorithm

Let's say that someone pays to have a star named for you. You are presented with a certificate like this.

How many guesses would it take (worst case) to find your star in the Tycho-2 Catalog which has 2,539,913 entries?

Binary Search Algorithm

Your certificate listed the Tycho-2 catalog number.


STAR ASTRONOMICAL COORDINATES:

Catalog Number: TYC 4149-1136-1

Constellation: Ursa Major

What is the worst case scenario for a linear search?

2,539,913

Binary Search Algorithm

What is the worst case scenario for a binary search?

n is 2,539,913 which is not a power of 2.

What is the closest **lower** power of 2?

2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}
2048	4096	8192	16384	32768	65536	131072	262144	524288	1048576	2097152

Do we need to calculate 2^{22}?

Not really – we know that would go way beyond 2,539,913 and we want the closest **lower** power so 2^{21} is what we need.

Binary Search Algorithm

So 2^{21} is the closest lower power of 2.

$\log_2 2097152$ is 21

Adding the 1 final guess gives us

22 total guesses is the worst case scenario.

22 is just a little bit better than searching 2,539,913 entries for our star!!