Linear Search
vs
Binary Search

Computer Science Department
University of Central Florida

COP 3502 – Computer Science I

Linear Search

- Searching from C-Programming class
  - In COP 3223, we studied how to find a value in an array
    - Look at each value in the array
      - Compare it to what we're looking for
    - If we see the value we are searching for,
      - Return that we've found it!
    - Otherwise, if we've iterated through the entire array and haven't located the value,
      - Return that the value isn't in the array

Linear Search

- Searching from C-Programming class
  - Your code should look something like this:

```
int search(int array[], int len, int value) {

        int i;
        for (i=0; i<len; i++) {
                if (array[i] == value)
                        return 1;
        }
        return 0;
}
```

Linear Search

- Searching from C-Programming class
  - Analyze code:
    - Clearly, if the array is unsorted, this algorithm is optimal
      - They ONLY way to be sure that a value isn't in the array is to look at every single spot of the array
      - Just like you can't be sure that you DON'T have some piece of paper or form unless you look through ALL of your pieces of paper
  - But we ask a question:
    - Could we find an item in an array faster if it were already sorted?

Binary Search

- Number Guessing Game from childhood
  - Remember the game you most likely played as a child
    - I have a secret number between 1 and 100.
    - Make a guess and I'll tell you whether your guess is too high or too low.
    - Then you guess again. The process continues until you guess the correct number.
    - Your job is to MINIMIZE the number of guesses you make.

Binary Search

- Number Guessing Game from childhood
  - What is the first guess of most people?
    - 50.
  - Why?
    - No matter the response (too high or too low), the most number of possible values for your remaining search is 50 (either from 1-49 or 51-100)
    - Any other first guess results in the risk that the possible remaining values is greater than 50.
      - Example: you guess 75
      - I respond: too high
      - So now you have to guess between 1 and 74
        - 74 values to guess from instead of 50

Binary Search

- Number Guessing Game from childhood
  - Basic Winning Strategy
    - Always guess the number that is halfway between the lowest possible value in your search range and the highest possible value in your search range
- Can we now adapt this idea to work for searching for a given value in an array?

Binary Search

- Array Search
  - We are given the following sorted array:

index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
value	**2**	**6**	**19**	**27**	**33**	**37**	**38**	**41**	**118**

  - We are searching for the value, 19
  - So where is halfway between?
    - One guess would be to look at 2 and 118 and take their average (60).
    - But 60 isn't even in the list
    - And if we look at the number closest to 60
      - It is almost at the end of the array

Binary Search

- Array Search
  - We quickly realize that if we want to adapt the number guessing game strategy to searching an array, we MUST search in the middle INDEX of the array.
  - In this case:
    - The lowest index is 0
    - The highest index is 8
    - So the middle index is 4

Binary Search

- Array Search
  - Correct Strategy
    - We would ask, “is the number I am searching for, 19, greater or less than the number stored in index 4?
      - Index 4 stores 33
    - The answer would be “less than”
    - So we would modify our search range to in between index 0 and index 3
      - Note that index 4 is no longer in the search space
    - We then continue this process
      - The second index we’d look at is index 1, since (0+3)/2=1
      - Then we’d finally get to index 2, since (2+3)/2 = 2
      - And at index 2, we would find the value, 19, in the array

Binary Search

- Binary Search code:

```
int binsearch(int a[], int len, int value) {

    int low = 0, high = len-1;
    while (low <= high) {
        int mid = (low+high)/2;
        if (value < a[mid])
            high = mid-1;
        else if (value > a[mid])
            low = mid+1;
        else
            return 1;
    }

    return 0;
}
```

Binary Search

- Binary Search code:

  - At the end of each array iteration, all we do is update either low or high
  - This modifies our search region
    - Essentially halving it

Binary Search

- Efficiency of Binary Search
  - Analysis:
    - Let's analyze how many comparisons (guesses) are necessary when running this algorithm on an array of n items

      First, let's try n = 100

      - After 1 guess, we have 50 items left,
      - After 2 guesses, we have 25 items left,
      - After 3 guesses, we have 12 items left,
      - After 4 guesses, we have 6 items left,
      - After 5 guesses, we have 3 items left,
      - After 6 guesses, we have 1 item left
      - After 7 guesses, we have 0 items left.

Binary Search

- Efficiency of Binary Search
  - Analysis:
    - Notes:
      - The reason for the last iteration is because the number of items left represent the number of other possible values to search
        - We need to reduce this to 0.
      - Also, when n is odd, such as when n=25
        - We search the middle element, # 13
        - There are 12 elements smaller than 13
        - And 12 elements bigger than 13
        - This is why the number of items is slightly less than ½ in those cases

Binary Search

- Efficiency of Binary Search
  - Analysis:
    - General case:
    - After 1 guess, we have n/2 items left
    - After 2 guesses, we have n/4 items left
    - After 3 guesses, we have n/8 items left
    - After 4 guesses, we have n/16 items left
    - …
    - After k guesses, we have $n/2^k$ items left

Binary Search

- Efficiency of Binary Search
  - Analysis:
    - General case:
    - So, after k guesses, we have $n/2^k$ items left
    - The question is:
      - How many k guesses do we need to make in order to find our answer?
      - Or until we have one and only one guess left to make?
    - So we want to get only 1 item left
    - If we can find the value that makes the above fraction equal to 1, then we know that in one more guess, we'll narrow down the item

Binary Search

- Efficiency of Binary Search
  - Analysis:
    - General case:
    - So, after k guesses, we have $n/2^k$ items left
      - Again, we want only 1 item left
      - So set this equal to 1 and solve for k

$$\frac{n}{2^k} = 1 \qquad n = 2^k \qquad k = \log_2 n$$

    - This means that a binary search roughly takes $\log_2 n$ comparisons when searching in a sorted array of n items

Binary Search

- Efficiency of Binary Search
  - Analysis:
    - Runs in logarithmic (log n) time
    - This is MUCH faster than searching linearly
    - Consider the following chart:

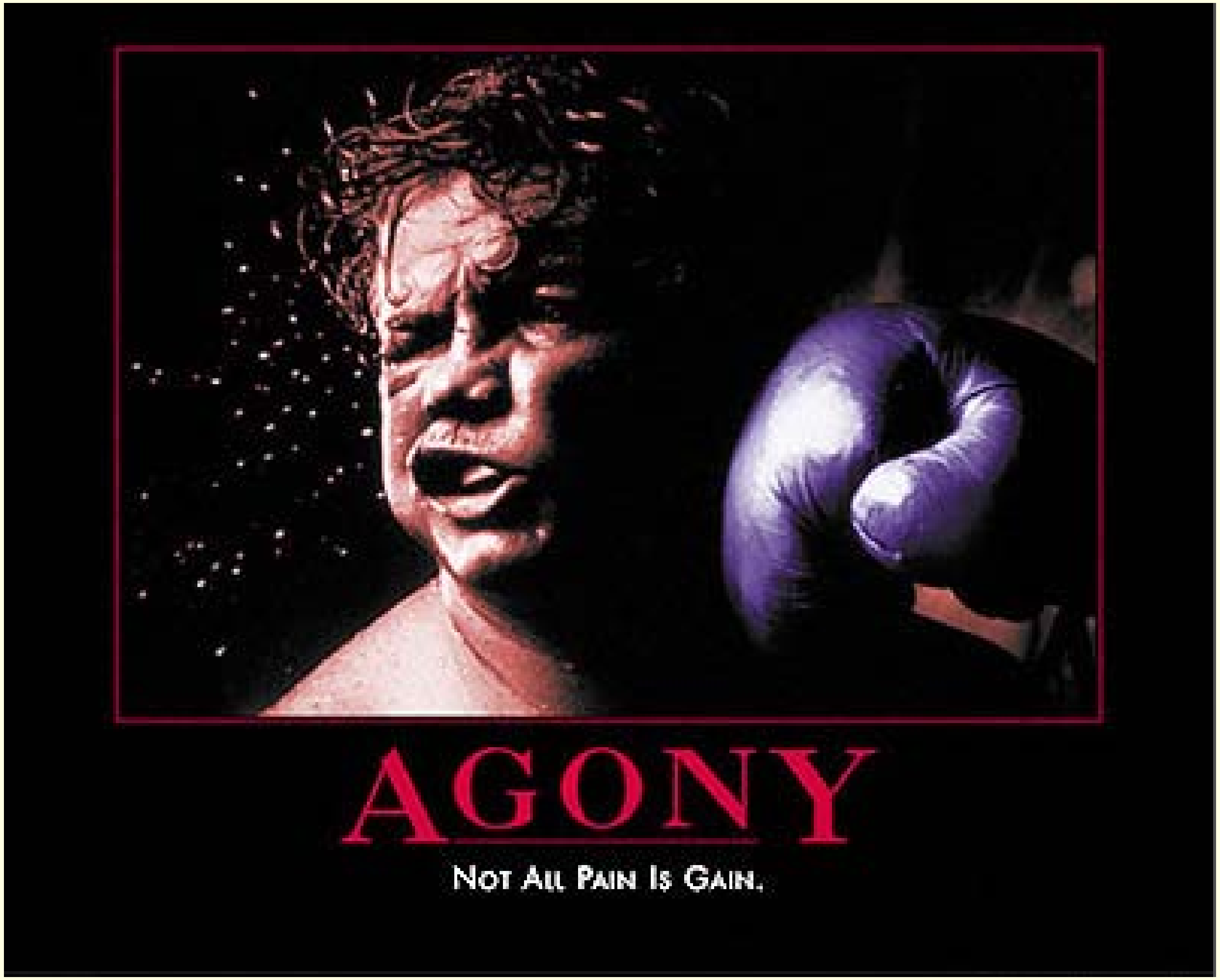
n	**log n**
8	3
1024	10
65536	16
1048576	20
33554432	25
1073741824	30

    - Basically, any log n algorithm is SUPER FAST.

Binary Search

WASN'T THAT INCREDIBLE!

Daily Demotivator



Linear Search vs Binary Search

Computer Science Department
University of Central Florida

COP 3502 – Computer Science I