2. Bipartite Checking

Algorithm to determine if $G$ is bipartite

- $G$ is undirected, connected.

BFS

odd: blue
even: gold

Claim: If $G$ is bipartite, this demonstrates it
3. Bipartite Checking

If $G$ is not bipartite, can we prove from algorithm?

\[
\text{find } e = (u, v) \text{ monochromatic}
\]

\[
u, v \text{ same layer}
\]

4. Big Extra Credit Question

- **Input:** any simple graph $G$
- **Output:** chromatic number of $G$
- To claim extra credit:
  - Prove running time, polynomial in $n$ and $m$ 
    $O(n^{100000})$ okay
    $O(2^n)$ is not.
  - Prove it is correct

- Worth an A+ in this class
  Actually worth more than that.
Running time of linear search

```cpp
int linSearch(const std::vector<int> & numbers, int target)
{
    int i;
    int n = numbers.size();
    for(i=0; i < n; i++)
        if( numbers[i] == target )
            return i;
    throw ElementNotFoundException("msg");
}
```

growth rate

worst? $O(n)$

---

Running time of linear search

```cpp
int linSearch(const std::vector<int> & numbers, int target)
{
    int i;
    int n = numbers.size();
    for(i=0; i < n; i++)
        if( numbers[i] == target )
            return i;
    throw ElementNotFoundException("msg");
}
```

▶ Does this time change if the vector is sorted?
6. Running time of binary search

```cpp
int binarySearch(const std::vector<int> & numbers, int target){
    return binarySearch(numbers, target, 0, numbers.size() - 1);
}

int binarySearch(const std::vector<int> & numbers, int target, int low, int high)
{
    if( low > high )
        throw ElementNotFoundException("msg");
    int mid = (low + high) / 2;
    if( numbers[mid] == target )
        return mid;
    else if( target < numbers[mid])
        return binarySearch(numbers, target, low, mid-1);
    else
        return binarySearch(numbers, target, mid+1, high);
}
```

What is binary search doing?

```
\[
\begin{array}{c}
[1, 1000) \rightarrow [501, 1000] \\
[751, 1000] \\
[751, 874] \\
[813, 874] \\
[813, 843] \\
[829, 843] \\
[829, 835] \\
\end{array}
\]
```

8 32 !
Lower Bounds?

- Potential new solution to \texttt{find-min}
- Arbitrary input vector of size $n$
- Better than $\mathcal{O}(n)$ time.
- Believable? Why or why not?

Which is better? Why?

- Input: an array $A$ of size $n$.
- Algorithm 1: $20n$ operations
- Algorithm 2: $n^2$ operations
- Which is better?