Brute Force

- For some problems trying everything is an option
  - If we want to find the longest path in a graph ...
  - If we want to generate every subset of a set ...
  - If we don't know how to do something smart ...

- Brute force typically means "try everything", generate all possibilities and test
  - For example: try all subsets
  - See CodeBloat weekly problem

- Sometimes smarter solutions exist
  - Dynamic programming, heuristic methods
Generate all subsets

- **Straightforward recursively, see subsets.cpp**
  - Start with an empty subset
    - For each element in original set, add it to subset, recurse
    - Don't add it to the subset, recurse
  - Base case is when all elements have been considered
    - See code for details, note similarity to backtracking!

- **Can also try everything using bits and masks**
  - int values represented as a sequence of bits *binary digits*
  - We can use this representation to generate and test
Bits, a simple primer/introduction

- **N** bits can represent $2^N$ values, typically we ignore the actual value of the bits, but we can exploit bit patterns in an int

<table>
<thead>
<tr>
<th>Bits</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1 = $2^0$</td>
</tr>
<tr>
<td>010</td>
<td>2 = $2^1$</td>
</tr>
<tr>
<td>011</td>
<td>3 = $2^1 + 2^0$</td>
</tr>
<tr>
<td>100</td>
<td>4 = $2^2$</td>
</tr>
<tr>
<td>101</td>
<td>5 = $2^2 + 2^0$</td>
</tr>
<tr>
<td>110</td>
<td>6 = $2^2 + 2^1$</td>
</tr>
<tr>
<td>111</td>
<td>7 = $2^2 + 2^1 + 2^0$</td>
</tr>
</tbody>
</table>

- **If we can find the 1's we can see which elements in a subset**
  - Example: for 100, element 2, for 010, element 1, and for 101, elements 2 and 0
Shift, mask, brute-force

```cpp
int size = myElements.size();
int subSize = 1 << myElements.size();
for(int count = 0; count < subSize; count++){
    mask = count;
    tvector<string> subs;
    for(int k=0; k < size; k++){
        if ( (mask & 1) == 1){
            subs.push_back(myElements[k]);
        }
        mask = mask >> 1;
    }
    // process subs
}
```