CS 162C++
Recursion Solutions

Converting a string to an integer

A recursive version of converter is:

```
int converter(string input, int result = 0)
{
   if (input.length() > 0)
   {
      result *= 10;
      result += input[0] - '0';
      return converter(input.substr(1), result);
   }

   return result;
}
```

Reversing a string

A recursive version of reverser is:

```
string reverser(string input)
{
   int length = static_cast<int>(input.length());

   if (length == 0)
      return "";

   length--;

   return input[length] + reverser(input.substr(0, length));
}
```

Linear Search

A recursive version of Linear search is:

```
bool linearFind(int array[], int length, int value)
{
   length--;

   if (length < 0)
      return false;

   if (array[length] == value)
         return true;

   return linearFind(array, length, value);
}
```

Binary Search

A recursive version of Binary search is:

```
bool binFind(int array[], int low, int high, int value)
{
   if (low > high)
      return false;

   int middle = (low + high) / 2;

   if (array[middle] == value)
      return true;

   if (array[middle] > value)
      high = middle - 1;
   else
      low = middle + 1;

   return binFind(array, low, high, value);
}
```

Display all Subsets

A classic recursive problem is to display all the subsets of a given set of characters. For example:

```
    // recursive method to generate substrings of given string
    // builds each substring in output
void subSets(string input, string output)
{
        // if done with this substring, display it
    if (input.length() == 0)
    {
        cout << output << " ";
        return;
    }
        // create the two possible substrings
        // one with and one without the first char
    subSets(input.substr(1), output + input[0]);
    subSets(input.substr(1), output);
}
```

Display all permutations

Here is a brute force version of a recursive permutation program:

```
    // recursive method to generate permutations of given string
    // builds each permutation in output
    // this is a brute force approach similar to subSets example
void permutations(string input, string output)
{
        // if done with input, output output
    if (input.length() == 0)
    {
        cout << output << " ";
        return;
    }
        // create all possible options for next element
        // removing each option from the remaining string
    for (int i = 0; i < input.length(); i++)
    {
        string temp = input;
        permutations(temp.erase(i, 1), output + input[i]);
    }
}
```

A more elegant solution is Heap's algorithm.