Homework 3

1. Design and analyze a **brute-force** algorithm for the following problem:
    - input: a positive integer *N*;
    - output: the sum $1^4 + 2^4 + \cdots + N^4$.

A1: Code done with C++

We can see that the first term is 1^4, which is N^4 where N = 1. So, the base case has N = 1.

```
Int bruteSum (unsigned int N)

{

        //Sum initialized

        Sum = 0;

        //The term initialized, to be easier to see

        Term = 0;

        For (unsigned int i = 1; i <= N; ++i)

        {

                //Updates the current term, which is i to the power of 4

                Term = i * i * i * i;        // Not using any external functions, is just i ⁴

                //Adds the current term to the overall sum

                Sum += Term;

        }

        //Returns the sum

        Return Sum;

}
```

Analysis: I will be analyzing only the number of multiplication operations for worst case analysis. The for loop turns into a summation. There are three multiplication operations in one line of the for loop. The for loop runs at i = 1 to i <= N. So, it becomes ${}_{i=1}\Sigma^{n}$ (3) = 3(${}_{i=1}\Sigma^{n}$ (1)) = 3(n - 1 + 1) = 3n

2. Design and analyze an **exhaustive-search** algorithm for the following problem:

- input: an array *A[lo..hi]*;
- output: a value in *A[lo..hi]* that occurs most often.
- example: [1 3 1 3 2 3 3] -> 3

A2:

```
Int frequentValue (int[] myArray)

{

    //Maximum frequency counter and the element itself, current counter

    freqCounter = 0;

    freqElem = 0;

    currCounter = 0;

    //Initialize freqElem and freqCounter in case there is 1 element only

    freqElem = myArray[i];

    freqCounter += 1;

    //Individually picks each element

    For (int i = 0, i <= myArray.size() - 1; ++i)

    {

        currCounter = 0;

        For (int j = 0; j <= myArray.size() - 1; ++j)

        {

            //Looks through the array again and counts how many times the current element i appears

            If (myArray[i] == myArray[j])

            {

                currCounter += 1;

            }

        }
```

```
        //Then check if the current counter beats the freqCounter

        If (currCounter > freqCounter)

        {

            freqCounter = currCounter;    // Update the freqCounter

            freqElem = myArray[i];        // Update freqElem to the picked element

        }

    }

    //Then return the freqElem

    Return freqElem;

}
```

Analysis: I will be analyzing the number of element array comparisons. It occurs only once in the inner for loop. So, it will be

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(1) = \sum_{i=0}^{n-1}((n-1)-0+1) = \sum_{i=0}^{n-1}(n) = n[(n-1)-(0)+1] = n[n] = n^2$$

3. Design and analyze a **decrease-conquer** algorithm for the following problem:
    - input: an array *A[lo..hi]* and a positive integer *k*;
    - output: the k^{th} smallest value in the array.

    To get started, think about the base case.

A3:

```
Int decreaseSmallest (int[] myArray, unsigned int k)
{
    //Base case, 1 element
    If (myArray.size() == 1)
    {
        Return myArray[0];
    } else  // Recursive case, size >= 1
    {
        //Decrease the array by 1 element each time
        Int newArray[myArray.size() - 1];
        For (int i = 0; i <= myArray.size() - 2; ++i)
        {
            newArray[i] = myArray[i];
```

```
            }
            //Variable to store the minimum
            Int smallNum = decreaseSmallest(newArray, k);
            //Compares with the last element of the array
            If (smallNum < myArray[myArray.size() - 1])
            {
                //Then smallNum is the kth smallest element in the array
                Return smallNum;
            } else if (myArray[myArray.size() - 1] < smallNum)
            {
                //Then there is a new smallest kth element
                Return myArray[myArray.size() - 1];
            } else
            {
                //If there are more elements than k
                If (myArray.size() > k)
                {
                    //Stop comparing and disregard the other values
                    Return smallNum;
                }
            }
        }
}
```

Analysis: I will be analyzing comparisons with array elements. Since the comparisons are only possible through if statements, I will analyze the path that runs through the worst case scenario. The first if statement might run as a check, and then if it fails, it will run through the other checks, until the else statement at the bottom, making three comparisons.

$M(1) = 0$

$M(n) = 3 + M(n - 1)$

By the method of unrolling, we will solve the recurrence.

$\rightarrow 3 + M(n - 1)$

$\rightarrow 3 + [3 + M(n - 2)]$

$\rightarrow 3 + 3 + [3 + M(n - 3)]$

….

${}_{I = 1}\Sigma^{n}$ $(3) + M(n - n)$

$\rightarrow 3((n) - (1) + 1) + M(0)$

$\rightarrow 3n + 0 \rightarrow 3n = M(n)$