Big O Notation

Order of Magnitude

The mathematical function that approximates the relationship between the size of the job (expressed by \( N \)) and the work that is required to finish the job.
Common Notations

- $O(1)$: Constant
- $O(N)$: Linear
- $O(N^2)$: Quadratic
- $O(2^N)$: Exponential
- $O(N \log_2(N))$: “$n \log n$”
- $O(\log_2(N))$: Logarithmic
Example of $O(N)$

```cpp
void DisplayList(apvector& arry, int N)
{
    int k;
    for( k = 0; k < N; k++)
        cout << arry[k] << endl;
}
```

The work required to display the contents of the array is always linearly proportional to $N$, regardless the size of $N$. 
Example of $O(N^2)$

```cpp
global DisplayList(apmatrix& arry, int N) {
    int row, col;
    for (row = 0; row < N; row++)
        for (col = 0; col < N; col++)
            cout << arry[row][col] << endl;
}
```

The work required to display the contents of the matrix will always be quadratically proportional to $N$ (the number of rows and columns) regardless the size of $N$. 
Function displays contents of matrix, minus top and bottom rows.

```c++
void DisplayList(apmatrix& arry, int N)
{
    int row, col;
    for( row = 2; row < N-2; row++)
        for (col = 2; col < N-2; col++)
            cout << arry[row][col] << endl;
}
```

The function is still proportional to the size of the matrix. Big O is still $O(N^2)$. 
Constants are ignored by Big-O

Any coefficient is changed to 1.

\[
\begin{align*}
F(2N) & \text{ becomes } O(N) \\
F(N/2) & \text{ becomes } O(N) \\
F(N) = (N^2/2) & \text{ becomes } O(N^2)
\end{align*}
\]
Graphing Functions

F(N) = N^2 and F(N) = N^2/2
Understanding O(1)

A function of Order 1 or O(1) runs in constant time. Regardless of the size of $N$, the work stays constant.

It is a common error to think that $O(1)$ means (1) operation like 1 comparison.
Consider displaying the data field of a first node in a linked list. Now, what about the work to display the data field of the 4th node of a linked list.

In both cases, the size of the list is irrelevant.
O(2^N) Examples

Tower of Hanoi
Recursive Fibonacci Function
Generating permutations of $n$ symbols
Binary Searches

Binary search is proportional to the log (base 2) of the size of the array.

The log base 2 of N is the number of doublings it takes to get N, starting with 1. For example, the log of 2 is 1, because only one doubling is required: 1 times 2 equals 2. The log of 8 is 3, because three doublings are required: 1 times 2 equals 2; 2 times 2 equals 4; 4 times 2 equals 8.
$O(\log_2(N))$ Examples

Binary search of sorted array
Searching a full binary tree
Splits in a merge sort
O(N \log_2 (N)) Examples

- Merge Sort
- Heap Sort
- Quick Sort
- Binary Tree Sort
Best/Average/Worst Case

Best Case
Best case times are usually ignored. It is possible that a search routine finds the required info with the first comparison. It is also possible that the data is already sorted. In practical terms, this rarely happens.
**Average Case**

When an algorithm is assigned a Big-O function, such as Quick Sort, $O(N \log(N))$, it is the average situation that is used. The average case usually identifies the algorithm.
Worst Case

Computer scientists are very concerned about worst case scenarios. Not all algorithms have worst case situations, such as the *merge sort*. Other algorithms, like the *quick sort*, can switch from $O(N \log(N))$ to $O(N^2)$ in a worst case scenario when the data is already sorted.

This kind of information can be used to prevent problems. Storing data to an external file from a *binary search tree* is one example.