Complexity of Algorithms

Let n be the size of input to an algorithm, and k some constant. The following are common rates of growth.

- Constant: $\Theta(k)$, for example $\Theta(1)$
- Linear: $\Theta(n)$
- Logarithmic: $\Theta(\log_k n)$
- $n \log n$: $\Theta(n \log_k n)$
- Quadratic: $\Theta(n^2)$
- Polynomial: $\Theta(n^k)$
- Exponential: $\Theta(k^n)$

Classification of algorithms - $\Theta(1)$

- Operations are performed k times, where k is some constant, independent of the size of the input n.
- This is the best one can hope for, and most often unattainable.
- **Examples:**

```
int Fifth_Element(int A[],int n) {
    return A[5];
}


int Partial_Sum(int A[],int n) {
    int sum=0;
    for(int i=0;i<42;i++)
       sum=sum+A[i];
    return sum;
```

Classification of algorithms - $\Theta(n)$

- Running time is linear
- As n increases, run time increases in proportion
- Algorithms that attain this look at each of the n inputs at most some constant k times.
- **Examples:**

```
void sum_first_n(int n) {
   int i,sum=0;
   for (i=1;i<=n;i++)
      sum = sum + i;
    }

void m_sum_first_n(int n) {
     int i,k,sum=0;
     for (i=1;i<=n;i++)
         for (k=1;k<7;k++)
                 sum = sum + i;
     }
```

Classification of algorithms - $\Theta(\log n)$

- A logarithmic function is the inverse of an exponential function, i.e. $b^x = n$ is equivalent to $x = \log_b n$)
- Always increases, but at a slower rate as n increases. (Recall that the derivative of $\log n$ is $\frac{1}{n}$, a decreasing function.)
- Typically found where the algorithm can systematically ignore fractions of the input.
- **Examples:**

```
int binarysearch(int a[], int n, int val)
{
int l=1, r=n, m;
   while (r>=1) {
     m = (l+r)/2;
     if   (a[m]==val) return m;
     if   (a[m]>val) r=m-1;
     else l=m+1; }
return -1;
}
```

Classification of algorithms - $\Theta(n \log n)$

- Combination of $O(n)$ and $O(\log n)$
- Found in algorithms where the input is recursively broken up into a constant number of subproblems of the same type which can be solved independently of one another, followed by recombining the sub-solutions.
- **Example:** Quicksort is $O(n \log n)$.

Perhaps now is a good time for a reminder that when speaking asymptotically, the base of logarithms is irrelevant. This is because of the identity

$$\log_a b \log_b n = log_a n.$$

Classification of algorithms - $\Theta(n^2)$

- We call this class quadratic.
- As n doubles, run-time quadruples.
- However, it is still polynomial, which we consider to be good.
- Typically found where algorithms deal with all pairs of data.
- **Example:**

```
int *compute_sums(int A[], int n) {
    int M[n][n];
    int i,j;
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            M[i][j]=A[i]+A[j];
    return M;
    }
```

- More generally, if an algorithm is $\Theta(n^k)$ for constant k it is called a polynomial-time algorithm.

Classification of algorithms - $\Theta(2^n)$

- We call this class exponential.
- This class is, essentially, as bad as it gets.
- Algorithms that use brute force are often in this class.
- Can be used only for small values of n in practi
- **Example:** A simple way to determine all n bit numbers whose binary representation has k non-zero bits is to run through all the numbers from 1 to 2^n, incrementing a counter when a number has k nonzero bits. It is clear this is exponential in n.