Algorithms

Algorithms: An ***algorithm*** is a sequence of unambiguous instructions for solving a problem.

Problems:

- No Solution
- One Solution
- **Multiple Solutions**

Multiple Solutions: Which solution amongst multiple is efficient to implement?

How to find efficiency??

Algorithm is associated with two types of efficiencies,

- **Time and Space.**
    - Time Efficiency: How fast an algorithm runs
    - Space efficiency: Extra space the algorithm requires

[One of the above is obsolete concept now. Can you recollect which one? More importantly, why???]

For Time efficiency, Is it running time of the algorithm??

Answer is ***NO!!***

- It becomes machine dependent.
- Program dependent
- Compiler dependent etc.

Basic Operation is the measuring Unit.

What is it??

- **It the operation contributing the most to the total running time of the algorithm. Generally, statements in the innermost loops.**

Ex: 1

```
1.  ALGORITHM Mystery(n)
2.  //Input: A nonnegative integer n
3.  S ←0
4.  for index ←1 to n do
5.     S ←S + index ∗ index
6.  return S
```

Basic Operation: S ←S + index ∗ index (Line no. 5)

Ex: 2

```
1.  ALGORITHM MaxElement(List[0..n-1])
2.  //Determines largest element in Array
3.  //Input: An array list[0..n-1] of real numbers
4.  //Output: Value of largest element
5.  max←list[0]
6.  for index← 0 to n-1 do
7.          if list[index] > max
8.                max←list[index]
9.  return max
```

Basic Operation: if list[index] > max (Line no. 7)

Ex: 3

```
1.  ALGORITHM SelectionSort(List[0..n-1])
2.  //Sorts a given array by selection sort
3.  //Input: An array list[0..n-1] of orderable elements
4.  //Output: Array list[0..n-1] sorted in ascending order
5.  for index← 0 to n-2 do
6.          min←index
7.                  for nextindex←index+1 to n-1 do
8.                      if list[nextindex] < list[min]
9.                               min←nextindex
10.         swap list[index] and list[min]
```

Basic Operation: if list[nextindex] < list[min]

We know how to identify the Basic Operation from a given algorithm. Let's know see how to calculate the efficiency of non-recursive algorithms!!

Ex 1:

```
1.  ALGORITHM MaxElement(List[0..n-1])
2.  //Determines largest element in Array
3.  //Input: An array list[0..n-1] of real numbers
4.  //Output: Value of largest element
5.  currentmax←list[0]
6.  for index← 0 to n-1 do
7.          if list[index] > currentmax
8.                  currentmax←list[index]
9.          if list[index] > currentmax
10.                 currentmax←list[index]
11.
12. return currentmax
```

Basic Operation: if list[index] > currentmax (Line no. 7)

1. **Set up the summation:**

$$C(n) = \sum_{index=0}^{n-1} 2 = 2 \sum_{index=0}^{n-1} 1 = 2n = O(n)$$

2. **Solve the Summation**

$$C(n) = \sum_{index=0}^{n-1} 1 = n-1+1 \in \mathbf{O(n)}$$

$$\sum_{i=l}^{u} 1 = u - l + 1$$

O(n)

Efficiency of given algorithm is O(n).

Ex 2:

```
ALGORITHM SelectionSort(List[0..n-1])
//Sorts a given array by selection sort
//Input: An array list[0..n-1] of orderable elements
//Output: Array list[0..n-1] sorted in ascending order
for index← 0 to n-2 do
        min←index
                for nextindex←index+1 to n-1 do
                  if list[nextindex] < list[min]
                        min←nextindex
        swap list[index] and list[min]
```

1. **Set up the summation:**

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

2. **Solve the Summation**

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

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

$$= \sum^{n-2} n-1-i-1+1$$

i=0

$$= \sum_{i=0}^{n-2} n - 1 - I$$

$$= \sum_{i=0}^{n-2} a-b$$

$$= \sum_{i=0}^{n-2} a - \sum_{i=0}^{n-2} b$$

$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = 1+2+3+\ldots+n=n(n+1)/2$$

$$= \sum_{i=0}^{n-2} n - 1 - \sum_{i=0}^{n-2} i$$

$$= \sum_{i=0}^{n-2} n - 1 - 0+1+2+\ldots+n-2$$

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

$$= \sum_{i=0}^{n-2} n - 1 - (n-2)(n-1)/2$$

$$= (n-1) \sum_{i=0}^{n-2} 1 - (n-2)(n-1)/2$$

= (n-1) (n-2-0+1) - (n-2) (n-1) /2

= (n-1) (n-1) - (n-2) (n-1) /2
= (n-1) [(n-1) – ((n-2)/2)]

= (n-1) ((2n-2-n+2)/2)

= (n-1) (n/2) = $(n^2-n)/2 = n^2/2 - n/2$

$\in O(n^2)$

References:

Text Books:

1. Ananya Levitin – Introduction to design and analysis of Algorithms
2. Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein - Introduction to Algorithms
3. Goodrich and Tamassia - Algorithm Design
4. ELLIS AUTOR HOROWITZ, SARTAJ AUTOR SAHNI – Fundamentals of Computer Algorithms