Algorithms

Chapter-1: The Role Of Algorithms in Computing:

Algorithm:

A computational procedure that takes some values or set of values and working on those inputs & finally produces output or set of outputs is called Algorithm.An algorithm is nothing but a set of logical statements which ensures a valid output for each valid input.

Application of algorithms:

Applications of algorithms are immense. below are the some important sectors.

- In sorting procedure,
- manipulation of big data in the internet website,
- data analysis in genome Sequencing,
- public key Cryptography and digital signature,
- number theory manipulation,
- maximum profit calculation in commercial Enterprises,
- finding the best route or for finding the shortest path,
- minimum cost calculation,
- maximum profit calculation
- in hashing techniques
- In searching techniques
- hardware design

Data structure:

Data Structure is a way to store and organise data in order to facilitate access and modification. No single Data Structure works well for all purposes and so it is important to know the strength and limitations of several of them.

Technique:

Necessary techniques are applied in an algorithm in such a way so that the efficiency of the algorithms is the most and the algorithm gives the appropriate result or correct result.

Hard problems:

In general, the efficiency of an algorithm is determined by how long time it will be executed for each case. Sometimes, there is no efficient solution. So, this type of hard problem can be solved using NP complete technique.

Parallelism:

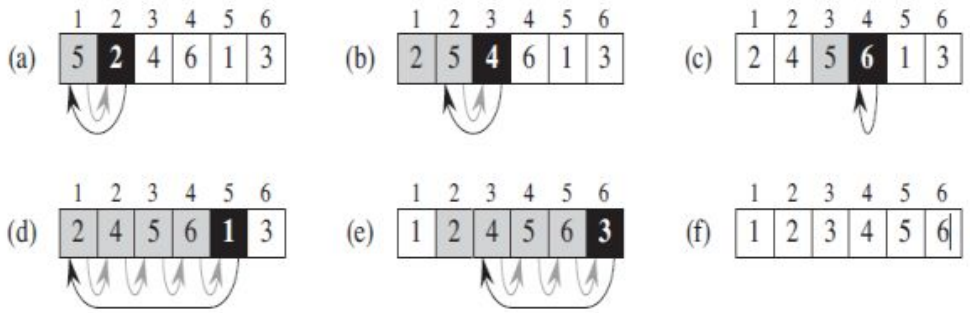
Now a days almost every machine or every computer has more than 1 processing core. So, for better efficiency of our algorithm, we should develop our algorithm keeping in mind that in a single second more than one tasks can be completed or executed. This type of algorithm is called multithreaded algorithm which takes the advantages of multiple cores.

Efficiency:

The efficiency of an algorithm depends on how much time it will take to be executed. go for an example, Insertion sort takes N^2 time to sort N items, while merge sort takes NlogN times to sort N items. Means merge sort is faster than insertion sort. The efficiency also depends upon the computer architecture. if we want to sort 10 million numbers in two different computers having different clock speed. Let's say, computer A can execute 10 billions instructions per second while computer B execute 10 million of instructions per second.So, computer A is 1000 times faster than computer B. Now for sorting 10 million numbers in computer A with Insertion sort algorithm, 20000 seconds will take. But in computer B , 1163 seconds will be needed to sort 10 million numbers applying merge sort algorithm. That means computer B runs 17 times faster than computer A.

Insertion Sort:

Insertion sort Is a sorting technique algorithm please works fast & perfectly for a small number of items. It's complexity is N^2



Pseudocode:

```
1   for j = 2 to A.length
2       key = A[j]
3       // Insert A[j] into the sorted sequence A[1 .. j - 1]
4       i = j - 1
5       while i > 0 and A[i] > key
6           A[i + 1] = A[i]
7           i = i - 1
8       A[i + 1] = key
```

Analysis of Insertion sort:

In general, the time taken by an algorithm grows with the size of the input. Means sorting a thousand numbers takes longer time than sorting four numbers.

Let's say , all the statements with the algorithms will cost totally c and the number of items is equal to n. so, the first statement will cost c1, the second statement will cost c2 and so on. now according to the Insertion sort algorithm, the first statement will the executed for n Times, the second statement will be executed for n -1 times and so on. the statement will be executed hot the summation of of t_j times from j=2 to n and the followings:

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] = key$	c_8	$n-1$

The total running time will be: $c_1n+c_2(n-1)+c_4(n-1)+c_5$ $SUM^{j=n}_{j=2}(t_j)$ $+c_6$ $SUM^{j=n}_{j=2}(t_j-1)+c_7$ $SUM^{j=n}_{j=2}(t_j-1)$ $+c_8(n-1)$

The running time may depend on the input size.

Worst case and average case analysis:

Worst case will happen only when the input array is already sorted in descending order. and best case will happen only when the input array is already sorted in ascending order. We can express the best case running time as the linear function an+ b . the worst case running time can be represented as the quadratic function an^2+bn+c .

To insert the last element, we need at most n-1 comparisons and at most n-1 swaps. To insert the second to last element, we need at most n−2 comparisons and at most n−2 swaps, and so on.The number of operations needed to perform insertion sort is therefore: 2*(1+2+....+n-2+n-1). To calculate the recurrence relation for this algorithm, use the following summation:

$$\sum_{q=1}^{p} = \{(p(p+1))/2\}.$$

It follows that

$\{2(n-1)(n-1+1)/2\}=n(n-1)$.

Use the master theorem to solve this recurrence for the running time. As expected, the algorithm's complexity is $O(n^2).O(n^2)$

When analyzing algorithms, the average case often has the same complexity as the worst case. So insertion sort, on average, takes $O(n^2)$ time

The divide and Conquer approach:

Divide: The main problem is divided into a number of some problems that is smaller distance of the main problem.

Conquer: Solving the problems using recursive manner is called the conquering.

Combine: Lastly the recursive solutions are combined into the solutions of the original problem. Merge sort algorithm is the example of divide and conquer approach.

the main array is divided into several similar sub-arrays.The sub-arrays are solved using recursive method . After after solving the sub-array, their combined to get the solution of the main problem.

Chapter-3: Growth Of Functions

Asymptotic notation: Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as $f(n)$ and may be for another operation it is computed as $g(n^2)$. This means the first operation running time will increase linearly with the increase in n and the running time of the second operation will increase exponentially when n increases. Similarly, the running time of both operations will be nearly the same if n is significantly small.

Θ-notation:

The Θ-notation asymptotically bounds a function from above and below. When we have only an asymptotic tight bound, we use Θ-notation.

O-notation:

The Θ-notation asymptotically bounds a function from above and below. When we have only an asymptotic upper bound, we use O-notation. We use O-notation to give an upper bound on a function, to within a constant factor.

Ω-notation:

Just as O-notation provides an asymptotic upper bound on a function, Ω-notation provides an asymptotic lower bound.

o-notation:

The asymptotic upper bound provided by O-notation may or may not be asymptotically tight. We use o-notation to denote an upper bound that is not asymptotically tight.

ω-notation:

By analogy, ω-notation is to Ω-notation as o-notation is to O-notation. We use ω-notation

to denote a lower bound that is not asymptotically tight.

Standard notations and common functions:

Monotonicity: A function f(n) is monotonically increasing if m<=n Implies f(m)<=f(n). Similarly, it is monotonically decreasing if m<=n implies f(m)>=f(n)

Floor and ceil:

The **floor function** is the function that takes as input a real number x and gives as output the greatest integer less than or equal to x.
.Similarly, the **ceiling function** maps x to the least integer greater than or equal to x

Polynomials:

A polynomial is an expression consisting of variables (also called indeterminates) and coefficients, that involves only the operations of addition, subtraction, multiplication, and non-negative integer exponents of variables. An example of a polynomial of a single indeterminate, x, is x2 − 4x + 7. An example in three variables is x3 + 2xyz2 − yz + 1.

Factorial:

The factorial of a number N is the multiplication of the numbers from 1 to N.
Factorial of 3 is :1*2*3 = 6

Fibonacci Number:

Fibonacci Numbers is the series of the summation of the two previous numbers starting from 0 & 1
Example: 0 1 1 2 3 5 8 13……..