6/2/14

Big O Notation : O notation is used to measure the performance of any algorithm. Performance of any algorithm depends upon the volume of input data. It is proportional to the input data. O notation is used to define the order of growth for any algorithm.

For a particular input size n we know how a particular algorithm behaves but when n increases and it becomes larger, then the same algorithm behaves differently. So the O notation defines the order of growth of any algorithm which is useful to measure the performance of algorithm.

Let us take two functions $f(n)$ and $g(n)$. Here, $g(n)$ is upper bound of $f(n)$, for some constant multiplier. So we can say $f(n) = O(g(n))$

and $f(n) <= c\,g(n) \ \forall\, n >= N$.

or we can say for any value of $n >= N$, $f(n)$ is bounded by multiplier of $g(n)$.

Let us take a function $f(n) = n^2 + 10n$ and $g(n) = n^2$

Here, $\forall\, n >= 10$ value of $f(n)$ will always less than $2n^2$ which is $g(n)$.

So, we can say that $f(n) = O(n^2)$.

Let us take some situations and behaviours of algorithm in order.

$O(1) \rightarrow$ when we get data in first time itself. eg Hash Table.

$O(n) \rightarrow$ when all the elements of list will be traversed. eg best case of bubble sort.

$O(n^2) \longrightarrow$ when the full list will be traversed for each element. eg. worst case of bubble sort.

$O(\log n) \rightarrow$ when we divide list half each time and traverse the middle element. eg. Binary Searching, Binary Tree traversal.

$O(n \log n) \rightarrow$ when we divide list half each time and traverse that half portion. eg. Best case of Quick Sort.

Design of Algorithm :

For designing of any algorithm some important things should be considered. Runtime, space and simplicity of algorithm. Some common approaches for designing algorithm are —
- (i) Greedy Algorithm.
- (ii) Divide and Conquer
- (iii) Non recursive algorithm
- (iv) Randomized.
- (v) Modular Programming Approach.

i) Greedy Algorithm :

This algorithm works in steps. In each step it selects the best available option until all options finished. This approach is widely used in so many places for designing algorithm.
eg. shortest Path algorithm.

ii) Divide and Conquer Algorithm :

Here we divide the big problem into some same type of small problems and we design the algorithm to combine the implementation of these small problems for implementing big problem.
eg. Quick sort, where we divide the initial list into several small lists, after sorting those smaller lists we combine them and get the initial list sorted.

iii) Non- recursive Algorithm :

As we know that recursion is very powerful technique which is supported by C but some compilers don't support this feature. In some places, recursion is not as effective and it can be avoided with iteration concept easily. So we have a need to design the algorithm with non-recursive approach.

iv) Randomized Algorithm :
In this algorithm, we use the feature of random number instead of fixed no. Performance of some algorithm depends upon the input data. It gives different results with different input data.
Let us take a case of quick sort. It behaves $O(n^2)$ in worst case. We take the pivot element for dividing the list. Suppose element of list are already in sorted order and we are taking the pivot as the first element then it will behave as $O(n^2)$. Here we can take any element list randomly for point, so it can improve the performance of algorithm. In any way it will not behave worst than $O(n^2)$, because it is a worst case behaviours of quick sort. Suppose with this random selection of pivot it behaves as average case of quick sort then it's performance will be of $O(n\log n)$.

v) Modular programming approach :
Here we divide the big problem into smaller ones which are totally different from each other. Then we combine the solution of all smaller problems and we get the solution of big problem. Actually here we can use different algorithm design for all small problems. This approach gives different modules for different problems and makes it easier to handle the big problem.

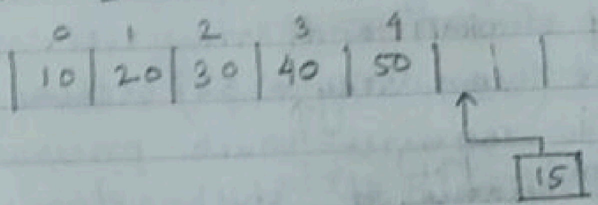
7/2/14

Array Implementation of Lists

3 operations in list – ① Insertion ② Deletion ③ Searching

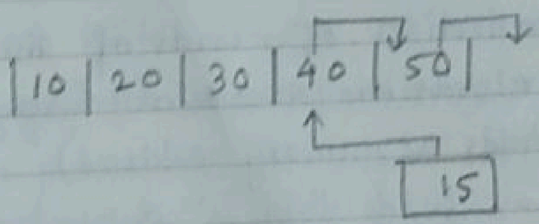
① Insertion : Two ways – (i) At end
(ii) In between.

(i) index = Total no. of elements (i.e. 5)
a[index] = value of inserted element



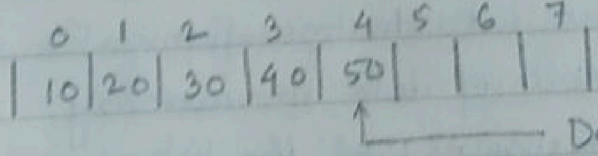
element inserted at 6th position.

(ii) shift right one position



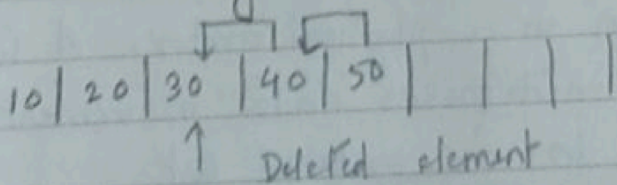
element inserted at 4th position.

② Deletion : Two ways – (i) last element
(ii) in between.

i)


Deleted element

ii) shift left one position from the next element to the last element of the array. and decrease the Total no. of elements by 1.



Deleted element

③ Searching : (i) Traverse all elements of the array list.
(ii) Compare each element of array with the given list