Complexity of Algorithm

- Algorithm is step by step procedure to solve a problem
- Once an algorithm is written it should be checked for its efficiency.
- That is how much time it takes and how much space it requires.
- The exact time can be determined by implementing the algorithm in a programming language and running in a machine.
- But this type of analysis will be confined to the machine and language used.
- The analysis should be general so that it can be applicable to all machine and all implementation.

- Suppose M is an algorithm and n is its size of input. The time and space used by the algorithm are two measures for efficiency of M.
- The time is measured by counting the number of key operations.
- In sorting and searching algorithms for example, the number of comparisons. That is because key operations are so defined that the time for other operations is much less than or at most proportional to the time for key operations.
- The space is measured by counting the maximum of memory needed by the algorithm.
- The complexity of an algorithm M is the function f(n), which gives the running time and/or storage space requirement of the algorithm M in terms of size n of input data.

- Worst Case: The Maximum value of f(n) for any possible input
- Best Case:minimum possible value of f(n)
- Average Case: the expected value of f(n)
- The analysis of average case assumes certain probabilistic distribution of the input data.
- Linear Search(Data[n], item)
- K=1, loc=0
- While(loc==0 && k<=n){
- If(item==Data[k] loc=k;
- K=k+1}
- If(loc==0) write(“not found”);
- Else write(“item found at ”, loc);

- Worst Case f(n) = n
- Average Case: Assume that item does appear in data and is equaly likely to occure in any position in the array. So the number of comparisons will be 1, 2, 3, 4, …., n each withprobability 1/n then
- F(n)=1*1/n + 2*1/n + …+ n*1/n=(n+1)/2
- Big O Notation
- We know the f(n) the standard functions are
- log n, n log n, n^2, n^3, …, 2^n, n!, n^n

Theorem: Suppose f(n) and g(n) are functions defined on +ve integers with property that f(n) is bounded below by some multiple of g(n) for almost all n. i.e suppose there exist a positive integer n0 and positive number M such that for all n>n0 |f(n)|<= M*|g(n)|

- Then we may write f(n)= O(g(n)

- Example: P(n) = 8n3 - 13n^2 + 21n – 45

  <= 8n3 + 13n^2 + 21n + 45 for all n

  <= 8n3 + 13n^3 + 21n^3 + 45 n^3 for all n

  <=87 * n^3 for all n

So P(n)=O(n3)

Linear Search: O(log n)

Binary Search: O(log n)

Bubble Sort: O(n^2)

Merge Sort: O(n log n)

Quick Sort: O(n logn n)

Time, space tradeoff

- Suppose a file of records contains name, phoneno, address and much additional information.
- Suppose you want to sort (arrange the records in alphabetical order) by name.
- It will take more time if there are large number of records, because sorting will need interchange of records.
- The sorting time can be decreased if we take another file called index file with two fields (name and pointer). The pointer will contain the address of the actual record.

Name	Pointer
rama	
hari	
gopal	
madhu	

Phone no	Name	Address	Other fields…
8895955560	rama	berhampur	
8895882345	hari	cuttack	
8895432689	gopal	puri	
8895763669	madhu	Balesore	

- Now instead of sorting the main file, if we sort the index file it will take less time., because only two fields will be interchanged while sorting. But in this case extra space is used.
- Clearly this trade off of space for time is not worth expense.
- Instead of sorting each time, we can maintain the index file as sorted file by adding one more pointer field in each record. This pointer will point to the next index record in sequence.
- In this case we have to insert the new index record such that the ordering of the index file will not be altered.
- This Solution will be more appropriate.
- But this will take some more space.
- That means when we want to decrease the execution time, we have to sacrifice some space.