**A Survey on Mathematical Background and Asymptotic Notation**

**Sumaiya Tabassum Kona**

**Nura Alahi Sagor**

**Irfan Ahmed Shaad**

**Department of Computer Science and Engineering**

**Bangladesh University of Business and Technology (BUBT)**

**Dhaka, Bangladesh**

**Abstract:**

We want you to reason with mathematics. We are not trying to get everyone to give formalized proofs in the sense of contemporary mathematics; ‘proof’ in this course means ‘convincing argument.’ We expect you to use correct reasoning and to give careful explanations. The survey brings out these issues in the way we ﬁnd best for most students, but the mathematical questions also interest some students. This paper of "mathematical background" shows how to ﬁll in the mathematical details of the main topics from the rest of the course. These proofs are completely rigorous in the sense of modern mathematics – technically bulletproof. We start this chapter with a summary of basic concepts and notations for sets and ﬁrst-order logic formulas, which will be used in the rest of the course. Our aim is to try to make the paper self-contained; readers who are familiar with set theory. Almost all the theorems in this paper are well-known old results of a carefully studied subject.

We hope our fresh look at the foundations of mathematical background will stimulate your interest. Decide for yourself what’s the

best way to understand this wonderful subject. Give your own proofs.

**Introduction:**

In arithmetic, a set is a very much well defined gathering of unmistakable items, considered as a protest in its own privilege. For instance, the numbers 2, 4 and 6 are particular articles when considered independently, yet when they are considered all things considered they frame a solitary collectively of size three, composed {2, 4, 6}.

George Cantor (1895), the originator of set hypothesis, gave the accompanying meaning of a set toward the start of his "Beiträge zur Begründung der transfiniten Mengenlehre".

"A set is an assembling into an entire of unequivocal, unmistakable objects of our recognition or of our idea which are called components of the set." **[1]**

Sets are mainly indicated with capital letters.

Cantor's definition ended up being deficient; rather, the idea of a "set" is taken as a primitive thought in proverbial set hypothesis, and the properties of sets are characterized by a gathering of adages. The most fundamental properties are that a set can have components, and that two sets are equivalent (one and the same) if and just if each component of each set is a component of alternate; this property is known as the extensionality of sets.

There are two methods for portraying or indicating the individuals from a set. One path is by intentional definition, utilizing a manage or semantic portrayal:

A is the set whose individuals are the initial four positive numbers.

B is the arrangement of shades of the French banner.

The second path is by augmentation - that is, posting every individual from the set. An extensional definition is signified by encasing the rundown of individuals in wavy sections:

A = {4, 2, 1, 3}

B = {blue, white, red}

**Classifications of sets:**

**i) Natural Numbers:**

A Natural number is a number that happens ordinarily and clearly in nature. In that capacity, it is an entire, non-negative number. The arrangement of common numbers, meant N, can be characterized in both of two ways:

N = {0, 1, 2, 3, ......}

N = {1, 2, 3, 4, ......}

The set N, regardless of whether it incorporates zero, is a denumerable set. Denumerability refers to the fact that, even though there might be an infinite number of elements in a set, those elements can be denoted by a list that implies the identity of every element in the set. For instance, it is instinctive from either the rundown {1, 2, 3, 4, ...} or the rundown {0, 1, 2, 3, ...} that 356,804,251 is a characteristic number, yet 356,804,251.5, 2/3, and - 23 are most certainly not. **[2]**

In like manner dialect, words utilized for checking are "cardinal numbers" and words utilized for requesting are "ordinal numbers".

Properties of the regular numbers, for example, detachability and the dissemination of prime numbers, are contemplated in number hypothesis. Issues concerning checking and requesting, for example, apportioning and counts, are examined in combinatorics. **[3]**

**ii) Integers:**

A whole number (articulated IN-tuh-jer) is an entire number (not a partial number) that can be certain, negative, or zero.

Cases of whole numbers are: - 5, 1, 5, 8, 97, and 3,043.

Cases of numbers that are not whole numbers are: - 1.43, 1 3/4, 3.14, .09, and 5,643.1.

The arrangement of whole numbers, indicated Z, is formally characterized as takes after:

Z = {..., - 3, - 2, - 1, 0, 1, 2, 3, ...}

The set Z is a denumerable set. Denumerability alludes to the way that, despite the fact that there may be an unending number of components in a set, those components can be indicated by a rundown that suggests the personality of each component in the set. For instance, it is instinctive from the rundown {..., - 3, - 2, - 1, 0, 1, 2, 3, ...} that 356,804,251 and - 67,332 are numbers, however 356,804,251.5, - 67,332.89, - 4/3, and 0.232323 ... are definitely not.

The components of Z can be combined off balanced with the components of N, the arrangement of characteristic numbers, without any components being let alone for either set. Let N = {1, 2, 3, ...}. At that point the blending can continue along these lines:

Z 0 1 - 1 2 - 2 3 - 3 4 - 4 5

| | | | | | | | | | |

N 1 2 3 4 5 6 7 8 9 10

Z is a subset of the arrangements of levelheaded and genuine numbers and, similar to the normal numbers, is countably limitless. **[4]**

Z is a completely requested set without upper or lower bound. The requesting of Z is given by: -3 < -2 < -1 < 0 < 1 < 2 < 3 < … A number is sure on the off chance that it is more prominent than zero and negative in the event that it is under zero. Zero is characterized as neither negative nor positive. **[5]**

**iii) Rational Numbers:**

A rational number is a number determined by the ratio of some integer p to some nonzero natural number q. The set of rational numbers is denoted Q, and represents the set of all possible integer-to-natural-number ratios p/q **[6].** In mathematical expressions, unknown or unspecified rational numbers are represented by lowercase, italicized letters from the late middle or end of the alphabet, especially r, s, and t, and occasionally u through z.

A rational number is a number that can be written as a ratio. That means it can be written as a fraction, in which both the numerator (the number on top) and the denominator (the number on the bottom) are whole numbers.

* The number 8 is a rational number because it can be written as the fraction 8/1.
* Likewise, 3/4 is a rational number because it can be written as a fraction.
* Even a big, clunky fraction like 7,324,908/56,003,492 is rational, simply because it can be written as a fraction.

Every whole number is a rational number, because any whole number can be written as a fraction. For example, 4 can be written as 4/1, 65 can be written as 65/1, and 3,867 can be written as 3,867/1. **[7]**

**iv) Real Numbers:**

The Real numbers incorporate all the objective numbers, for example, the whole number -5 and the part 4/3, and all the unreasonable numbers. Real numbers can be considered as focuses on an interminably long line called the number line or real line, where the focuses comparing to whole numbers are similarly dispersed. Any real number can be controlled by a perhaps unbounded decimal portrayal, for example, that of 8.632, where each back to back digit is measured in units one tenth the span of the past one. The real line can be considered as a piece of the intricate plane, and complex numbers incorporate real numbers. **[8]**

In the event that x and z are real numbers to such an extent that x < z, then there dependably exists a real number y with the end goal that x < y < z. The arrangement of reals is "thick" in an indistinguishable sense from the arrangement of irrationals. Both sets are nondenumerable. There are more real numbers than is conceivable to list, even by suggestion. **[9]**

**v) Empty Set:**

In Empty sets, the invalid set, additionally called the unfilled set, is the set that does not contain anything. In arithmetic, and all the more particularly set hypothesis, the unfilled set is the novel set having no components; its size or cardinality (include of components a set) is zero. Some aphoristic set hypotheses guarantee that the unfilled set exists by including a maxim of purge set; in different speculations, its reality can be reasoned. Numerous conceivable properties of sets are vacuously valid for the vacant set. **[10]**

Empty set was at one time a typical equivalent word for "discharge set",

however, is presently a specialized term in measure hypothesis. The unfilled set may likewise be known as the void set.

We can look at sets in light of their elements. The principal correlations are genuine when one set contains every one of the individuals from another set. In the accompanying definitions, A and B, and C are sets. **[11]**

**vi) Subset:**

A set **A** is a subset of another set **B** if all elements of the set **A** are elements of the set **B**. In other words, the set **A** is contained inside the set **B**.

Mathematically defined, A ⊆ B, where A is a subset of B.

As example,

{9, 14, 28} ⊆ {9, 14, 28}

**viii) Superset:**

**B** is a superset of **A** if set **B** contains all the elements containing set **A** which is indicated by,

A ⊇ B, where A is a superset of B.

As example,

{9, 14, 28} ⊇ {9, 14, 28}

ix) **Proper Subset:**

An appropriate subset of a set A will be a subset of A that is not equivalent to A. In other words, if B is a legitimate subset of A, then all components of B are in A, however A contains no less than one component that is not in B.

Mathematically defined, A ⊂ B where A is a proper subset of B.

As example,

{9, 14} ⊂ {9, 14, 28}

x) **Proper Superset:**

If all elements of A are in B but B contains at least one element that is not in A, then B is a proper superset of A.

Written as A ⊃ B.

For example,

{9, 14, 28} ⊃ {9, 14}

**xi) Union:**

The union of two sets is a set containing all elements that are in A or in B (possibly both).

For example,

 {1,2}∪{2,3}={1,2,3}{1,2}∪{2,3}={1,2,3}.

Thus, we can write x∈(A∪B)x∈(A∪B) if and only if (x∈A)or (x∈B). Note that

A∪B=B∪A. In Figure, the union of sets A and B is shown by the shaded area in the Venn diagram.

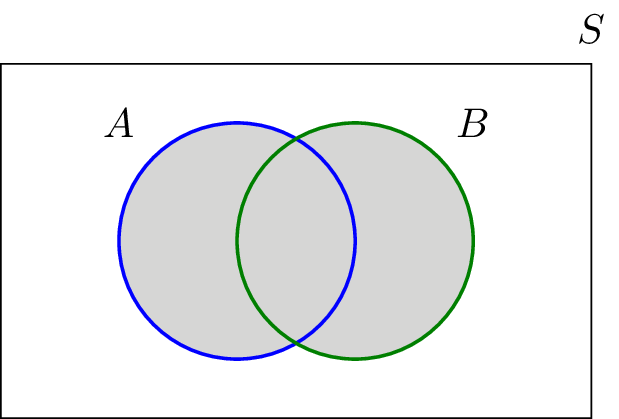


Fig: Union Set

Similarly, we can define the union of three or more sets. In particular, if A1,A2,A3,⋯,An are n sets, their union A1∪A2∪A3 ⋯∪An is a set containing all elements that are in at least one of the sets. We can write this union more compactly by ⋃i=1nAi.⋃i=1nAi.

**xii) Intersection:**

The intersection of two sets AA and BB, denoted by A∩B, consists of all elements that are both in A and B. For example,

{1,2}∩{2,3}={2}{1,2}∩{2,3}={2}. The intersection of sets A and B is shown by the shaded area using a Venn diagram.

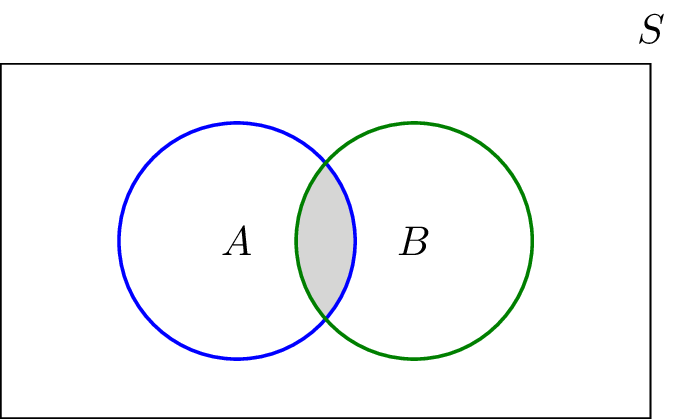


Fig: Intersection Set

**xiii) Set Difference:**

The difference (subtraction) is defined as follows. The set A−B consists of elements that are in A but not in B.

For example if A={1,2,3}A={1,2,3} and B={3,5}B={3,5}, then A−B={1,2}A−B={1,2}. A−B is shown by the shaded area using a Venn diagram. Note that A−B=A∩Bc.

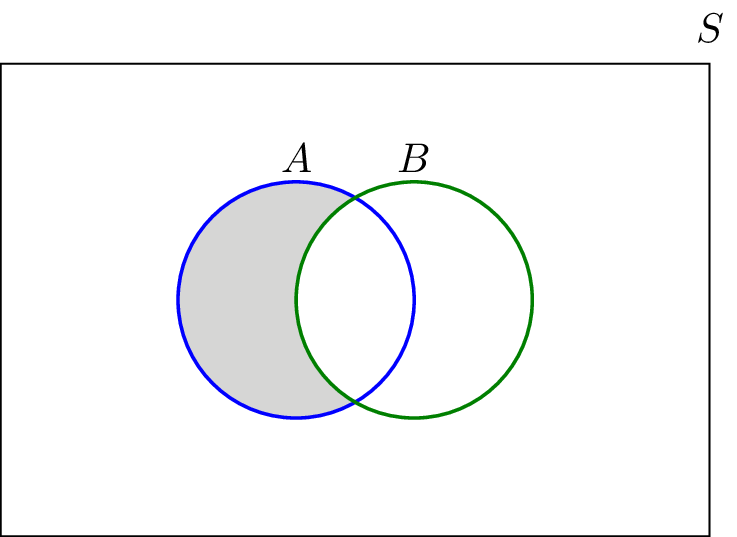


Fig: Set Difference

**xiv) Complement:**

The complement of a set AA, denoted by Ac or A¯, is the set of all elements that are in the universal set SS but are not in AA. In Figure, A¯A¯ is shown by the shaded

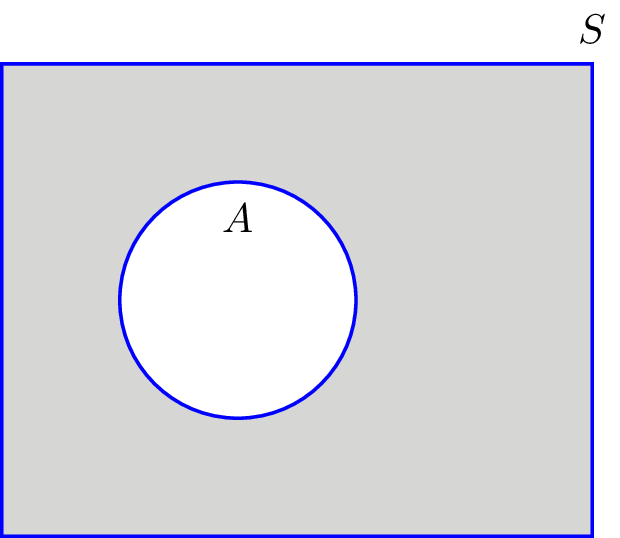
area using a Venn diagram. ****

Fig: Complement set

**xv) Power Set:**

"The set of all the subsets of a set"  
  
Example: For the set {a, b, c}:

• These are subsets: {a}, {b} and {c}  
• And these are subsets: {a, b}, {a, c} and {b, c}  
• And {a, b, c} is also a subset of {a, b, c}   
• And the empty set {} is also a subset of

{a, b, c}  
  
And all the subsets together make the Power Set:

P(S) = { {}, {a}, {b}, {c}, {a, b}, {a, c},

{b, c}, {a, b, c} }

**xvi) Cartesian Product:**

The collection of all ordered pairs of two

given sets such that the first elements of the

pairs are chosen from one set and the secondelements from the other set: this procedure

generalizes to an infinite number of sets.

For example, if  and , then



|  |
| --- |
|  |

**Ordered n-tuple:**

A tuple is a limited requested rundown of components. In arithmetic, a n-tuple is a succession (or requested rundown) of n components, where n is a non-negative whole number. There is just a single 0-tuple, an unfilled succession. A n-tuple is characterized inductively utilizing the development of a requested match. Tuples are normally composed by posting the components inside brackets " ( )" and isolated by commas; for instance, (2, 7, 4, 1, 7) indicates a 5-tuple. Here and there different images are utilized to encompass the components, for example, square sections "[ ]" or point sections "< >".

**Relation:**

A relation on *A* is said to be reflex if for each tex2html_wrap_inline124 *a* is related to *a*. If we let R denote the relation then we have *a*R*a* for each tex2html_wrap_inline124 . An example of a non reflexive relation is the relation "is the father of" on a set of people. As no person is the father of them-self the relation is not reflexive. As another example consider the relation tex2html_wrap_inline188 on tex2html_wrap_inline190 defined by tex2html_wrap_inline192 if tex2html_wrap_inline194 is odd. Then 1 tex2html_wrap_inline188 1 and 3 tex2html_wrap_inline188 3 but 0 tex2html_wrap_inline200 0 and so the relation is not reflexive.

A relation on *A* is said to be irreflexive if for each tex2html_wrap_inline124 *a* is not related to *a*. This is not the negation of the definition of reflexive. The relation "is the father of " is irreflexive.

A relation R on *A* is symmetric if given tex2html_wrap_inline208 then tex2html_wrap_inline210 . The relation "is the sister of" is not symmetric on a set that contains a brother and sister but would be symmetric on a set of females. The empty relation on a set is an example of a symmetric relation since the statement "if *a*R*b*" is always false. **[12]**

A relation R on *A* is antisymmetric if given tex2html_wrap_inline208 then tex2html_wrap_inline218 . Again, it is possible to have relations that are neither symmetric nor antisymmetric.

A relation R on *A* is transitive if given *a*R*b* and *b*R*c* then *a*R*c*. The relation "is an ancestor of" on a set of people is transitive as is the empty relation on a set.

A relation on *A* is said to be *reflexive* if for each tex2html_wrap_inline124 *a* is related to *a*. If we let R denote the relation then we have *a*R*a* for each tex2html_wrap_inline124 .

A relation on *A* is said to be *irreflexive* if for each tex2html_wrap_inline124 *a* is not related to *a*. This is not the negation of the definition of reflexive. The relation "is the father of " is irreflexive.

A relation R on *A* is *symmetric* if given tex2html_wrap_inline208 then tex2html_wrap_inline210 . The relation "is the sister of" is not symmetric on a set that contains a brother and sister but would be symmetric on a set of females. The empty relation on a set is an example of a symmetric relation since the statement "if *a*R*b*" is always false.

A relation R on *A* is *antisymmetric* if given tex2html_wrap_inline208 then tex2html_wrap_inline218 . Again, it is possible to have relations that are neither symmetric nor antisymmetric.

A relation R on *A* is *transitive* if given *a*R*b* and *b*R*c* then *a*R*c*. The relation "is an ancestor of" on a set of people is transitive as is the empty relation on a set **[20].**

**Sequence:**

A **sequence**, in mathematics, is a string of objects, like numbers, that follow a particular pattern **[13].** The individual elements in a sequence are called **terms**. Some of the simplest sequences can be found in multiplication tables:

* 3, 6, 9, 12, 15, 18, 21, …  
  Pattern: “add 3 to the previous number to get the next number”
* 0, 12, 24, 36, 48, 60, 72, …  
  Pattern: “add 12 to the previous number to get the next number”

Of course, we can come up with much more complicated sequences:

* 10.–2 8,×2 16,–2 14,×2 28,–2 26,×2 52, …  
  Pattern: “alternatingly subtract 2 and multiply by 2 to get the next number”
* 0,+2 2,+4 6,+6 12,+8 20,+10 30,+12 42, …  
  Pattern: “add increasing even numbers to get the next number”

**Subsequence:**

A Subsequence is a grouping that can be derived from another succession by erasing a few elements without changing the request of the rest of the components. For instance, the grouping (A, B, D) is the subsequence of (A, B, C, D, E, F) obtained from removal of C, E, F.

The relation of one sequence being the subsequence of another is a preorder. **[14]**

The subsequence should not be confused with substring (A, B, C, D) which can be derived from the above string (A, B, C, D, E, F) by deleting substring (E, F). The substring is a refinement of the subsequence.

**Empty Sequence:**

An empty sequence is a (finite) sequence containing no terms. It is the subsequence of every sequence.

An empty sequence is also known as a null sequence.

**Median:**

The median is also the number that is halfway into the set. To find the median, the data should be arranged in order from least to greatest. It separates the highest half of the values from the lowest half. If there is an even number of items in the data set, then the median is found by taking the mean (average) of the two middlemost numbers.

**Closed Form:**

A Closed-form expression is a mathematical expression that can be evaluated in a finite number of operations. It may contain constant, variables certain "well-known" operations (e.g., + − × ÷), and functions

(e.g., nth~root, exponent, logarithm, trigonometric functions, and inverse hyperbolic functions), but usually no limit. The set of operations and functions admitted in a closed-form expression may vary with author and context.

Problems are said to be tractable if they can be solved in terms of a closed-form expression.

**Asymptotic Notation:**

Some algorithms are more efficient than others. We would prefer to choose an efficient algorithm, so it would be nice to have metrics for comparing algorithm efficiency. **[15]**  
The complexity of an algorithm is a function describing the efficiency of the algorithm in terms of the amount of data the algorithm must process. Usually there are natural units for the domain and range of this function. There are two main complexity measures of the efficiency of an algorithm:  
  
\* **\* Time complexity** is a function describing the amount of time an algorithm takes in terms of the amount of input to the algorithm. "Time" can mean the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed, or some other natural unit related to the amount of real time the algorithm will take. We try to keep this idea of time separate from "wall clock" time, since many factors unrelated to the algorithm itself can affect the real time (like the language used, type of computing hardware, proficiency of the programmer, optimization in the compiler, etc.). It turns out that, if we chose the units wisely, all of the other stuff doesn't matter and we can get an independent measure of the efficiency of the algorithm.  
  
\*\* **Space complexity** is a function describing the amount of memory (space) an algorithm takes in terms of the amount of input to the algorithm. We often speak of "extra" memory needed, not counting the memory needed to store the input itself. Again, we use natural (but fixed-length) units to measure this. We can use bytes, but it's easier to use, say, number of integers used, number of fixed-sized structures, etc. In the end, the function we come up with will be independent of the actual number of bytes needed to represent the unit. Space complexity is sometimes ignored because the space used is minimal and/or obvious, but sometimes it becomes as important an issue as time.  
For example, we might say "this algorithm takes n2 time," where n is the number of items in the input. Or we might say "this algorithm takes constant extra space," because the amount of extra memory needed doesn't vary with the number of items processed.  
For both time and space, we are interested in the asymptotic complexity of the algorithm: When n (the number of items of input) goes to infinity, what happens to the performance of the algorithm?  
  
Asymptotic notations are mathematical tools to represent time complexity of algorithms for asymptotic analysis. The following 3 asymptotic notations are mostly used to represent time complexity of algorithms.

**Θ Notation:**

The theta notation bounds a function from above and below, so it defines exact asymptotic behavior.

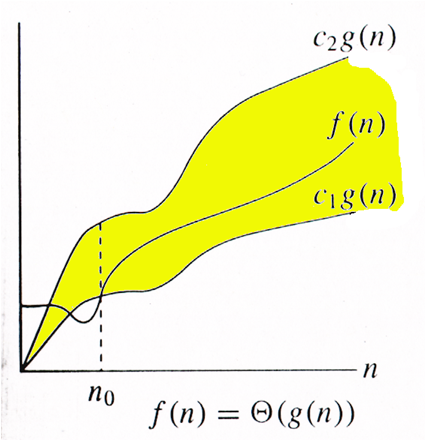
A simple way to get Theta notation of an expression is to drop low order terms and ignore leading constants. For example, consider the following expression.

3n3 + 6n2 + 6000 = Θ(n3)

Dropping lower order terms is always fine because there will always be a n0 after which Θ(n3) has higher values than Θ( n2) irrespective of the constants involved.  
For a given function g(n), we denote Θ(g(n)) is following set of functions.

Θ(g(n)) = {f(n): there exist positive constants c1, c2 and n0 such that

0 <= c1\*g(n) <= f(n) <= c2\*g(n) for all n >= n0}

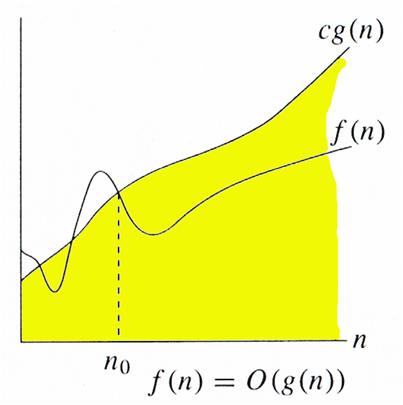
The above definition means, if f(n) is theta of g(n), then the value f(n) is always between c1\*g(n) and c2\*g(n) for large values of n (n >= n0). The definition of theta also requires that f(n) must be non-negative for values of n greater than n0. 

**Big O Notation:**

The Big O notation defines an upper bound of an algorithm, it bounds a function only from above. For example, consider the case of Insertion Sort. It takes linear time in best case and quadratic time in worst case. We can safely say that the time complexity of Insertion sort is O(n^2). Note that O(n^2) also covers linear time.

If we use Θ notation to represent time complexity of Insertion sort, we have to use two statements for best and worst cases:

**1)** The worst-case time complexity of Insertion Sort is Θ(n^2).

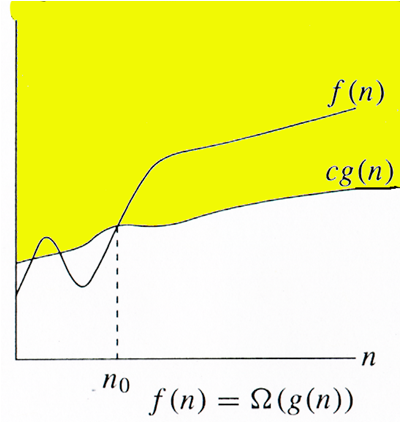
**2)** The best-case time complexity of Insertion Sort is Θ(n).  
The Big O notation is useful when we only have upper bound on time complexity of an algorithm. Many times we easily find an upper bound by simply looking at the algorithm.  
O(g(n)) = { f(n): there exist positive constants c and n0 such that 0 <= f(n) <= cg(n) for   
all n >= n0}

**[17]**

**Big Ω Notation:**

Just as Big O notation provides an asymptotic upper bound on a function, Ω notation provides an asymptotic lower bound.  
Ω Notation < can be useful when we have lower bound on time complexity of an algorithm. As discussed in the previous post, the best-case performance of an algorithm is generally not useful, the Omega notation is the least used notation among all three.  
  
For a given function g(n), we denote by Ω(g(n)) the set of functions.  
Ω (g(n)) = {f(n): there exist positive constants c and

n0 such that 0 <= cg(n) <= f(n) for all n >= n0}.

Let us consider the same Insertion sort example here. The time complexity of Insertion Sort can be written as Ω(n), but it is not a very useful information about insertion sort, as we are generally interested in worst case and sometimes in average case. 

**[18]**

**Algorithm Analysis:**

Analysis of algorithms typically focuses on the asymptotic performance, particularly at the elementary level, but in practical applications constant factors are important, and real-world data is in practice always limited in size. The limit is typically the size of addressable memory, so on 32-bit machines 232 = 4 GiB (greater if segmented memory is used) and on 64-bit machines 264 = 16 EiB. Thus, given a limited size, an order of growth (time or space) can be replaced by a constant factor, and in this sense all practical algorithms are O (1) for a large enough constant, or for small enough data.

This interpretation is primarily useful for functions that grow extremely slowly: (binary) iterated logarithm (log\*) is less than 5 for all practical data (265536 bits); (binary) log-log (log log *n*) is less than 6 for virtually all practical data (264 bits); and binary log (log *n*) is less than 64 for virtually all practical data (264 bits). An algorithm with non-constant complexity may nonetheless be more efficient than an algorithm with constant complexity on practical data if the overhead of the constant time algorithm results in a larger constant factor.

{\displaystyle n<2^{2^{6}}=2^{64}}

For large data, linear or quadratic factors cannot be ignored, but for small data an asymptotically inefficient algorithm may be more efficient. This is particularly used in hybrid algorithms, like Tim sort which use an asymptotically efficient algorithm (here merge sort with time complexity{\displaystyle n\log n}), but switch to an asymptotically inefficient algorithm (here insertion sort, with time complexity{\displaystyle n^{2}}) for small data, as the simpler algorithm is faster on small data.

**Substitution Method**:

We make a guess for the solution and then we use mathematical induction to prove the guess is correct or incorrect.

For example,

consider the recurrence T(n) = 2T(n/2) + n

We guess the solution as T(n) = O(nLogn). Now we use induction to prove our guess.

We need to prove that T(n) <= cnLogn. We can assume that it is true for values, smaller than n.

T(n) = 2T(n/2) + n

<= cn/2Log(n/2) + n

= cnLogn - cnLog2 + n

= cnLogn - cn + n

<= cnLogn

**Tree Method:**

In this method, we draw a recurrence tree and calculate the time taken by every level of tree. Finally, we sum the work done at all levels. To draw the recurrence tree, we start from the given recurrence and keep drawing till we find a pattern among levels. The pattern is typically a arithmetic or geometric series.

For example, consider the recurrence relation

T(n) = T(n/4) + T(n/2) + cn2

cn2

/ \

T(n/4) T(n/2)

If we further break down the expression T(n/4) and T(n/2),

we get following recursion tree.

cn2

/ \

c(n2)/16 c(n2)/4

/ \ / \

T(n/16) T(n/8) T(n/8) T(n/4)

Breaking down further gives us following

cn2

/ \

c(n2)/16 c(n2)/4

/ \ / \

c(n2)/256 c(n2)/64 c(n2)/64 c(n2)/16

To know the value of T(n), we need to calculate sum of tree

nodes level by level. If we sum the above tree level by level,

we get the following series

T(n) = c(n^2 + 5(n^2)/16 + 25(n^2)/256) + ....

The above series is geometrical progression with ratio 5/16.

To get an upper bound, we can sum the infinite series.

We get the sum as (n2)/(1 - 5/16) which is O(n2)

**Master Method:**

Master Method is a direct way to get the solution. The master method works only for following type of recurrences or for recurrences that can be transformed to following type.

T(n) = aT(n/b) + f(n) where a >= 1 and b > 1

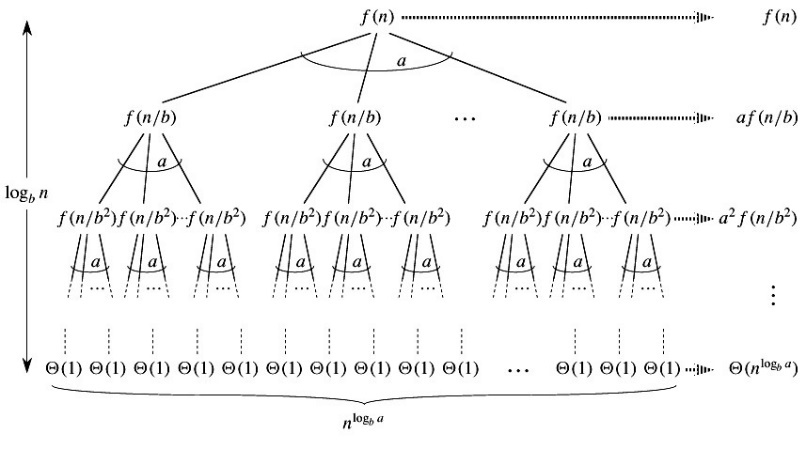
There are following three cases:  
**1.** If f(n) = Θ(nc) where c < Logba then T(n) = Θ (nLogba)

**2.** If f(n) = Θ(nc) where c = Logba then T(n) = Θ (ncLog n)

**3.**If f(n) = Θ(nc) where c > Logba then T(n) = Θ (f(n))

**How does this work?**

Master method is mainly derived from recurrence tree method. If we draw recurrence tree of T(n) = aT(n/b) + f(n), we can see that the work done at root is f(n) and work done at all leaves is Θ(nc) where c is Logba. And the height of recurrence tree is Logbn

[](http://www.geeksforgeeks.org/wp-content/uploads/Master-Theorem.jpg)  
 **[21]**

In recurrence tree method, we calculate total work done. If the work done at leaves is polynomials more, then leaves are the dominant part, and our result becomes the work done at leaves (Case 1). If work done at leaves and root is asymptotically same, then our result becomes height multiplied by work done at any level (Case 2). If work done at root is asymptotically more, then our result becomes work done at root (Case 3). **[19]**

Examples of some standard algorithms whose time complexity can be evaluated using Master Method   
Merge Sort: T(n) = 2T(n/2) + Θ(n). It falls in case 2 as c is 1 and Logba] is also 1. So, the solution is Θ (n Logn)

Binary Search: T(n) = T(n/2) + Θ(1). It also falls in case 2 as c is 0 and Logba is also 0. So the solution is Θ(Logn)

**Conclusion:**

In the existing paper, we learn about mathematical expressions, sets, relations,

asymptotic notation, sequence, subsequence, time complexity, space complexity, algorithm analysis and many more, which is needed for further programming sense.

**Reference:**

**[1]**https://en.wikipedia.org/wiki/Set\_(mathematics)

**[2]**<http://whatis.techtarget.com/definition/natural-number>

**[3]**<https://en.wikipedia.org/wiki/Natural_number>

**[4]**<http://whatis.techtarget.com/definition/integer>

**[5]** <https://en.wikipedia.org/wiki/Integer>

**[6]**<http://whatis.techtarget.com/definition/rational-number>

**[7]**<https://www.factmonster.com/math/numbers/rational-and-irrational-numbers>

**[8]**<http://whatis.techtarget.com/definition/real-number>

**[9]**<https://en.wikipedia.org/wiki/Real_number>

**[10]**<http://whatis.techtarget.com/search/query?q=empty+set>

**[11]**<https://en.wikipedia.org/wiki/Empty_set>

**[12]**<http://www.math.csusb.edu/notes/rel/node2.html>

**[13]** <http://world.mathigon.org/Sequences>

**[14]**https://en.wikipedia.org/wiki/Subsequence

**[15]** <http://www.geeksforgeeks.org/analysis-of-algorithms-set-3asymptotic-notations/>

**[16]** <http://www.geeksforgeeks.org/wp-content/uploads/thetanotation.png>

**[17]** <http://www.geeksforgeeks.org/wp-content/uploads/BigO.png>

**[18]** <http://www.geeksforgeeks.org/wp-content/uploads/BigOmega.png>

**[19]**<http://www.geeksforgeeks.org/analysis-algorithm-set-4-master-method-solving-recurrences/>

**[20]**<http://www.math.csusb.edu/notes/rel/node2.html>

**[21]** http://www.geeksforgeeks.org/wp-content/uploads/Master-Theorem.jpg