Data Structures and Algorithm (Referred from udemy Abdul Bari)

data structure : data structure can be defined as arrangement of collection of of data items so that they can be utilized efficiently, operations on that data can be done efficiently

→ So it's all about the arrangement of data and the operations on the data that are efficient operations on the data.

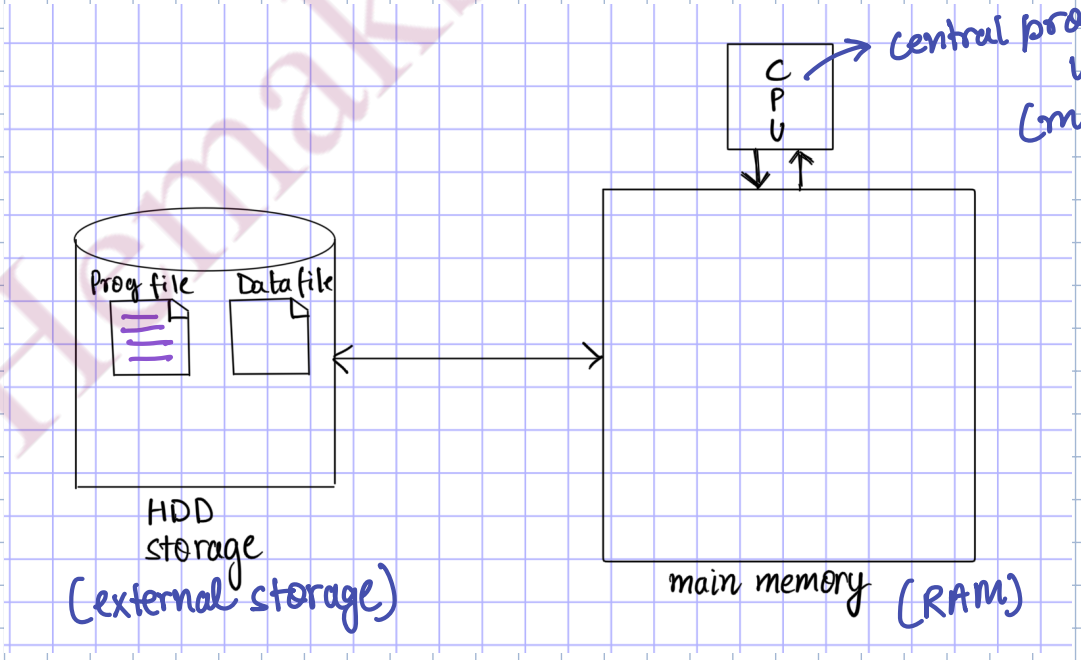
But the question is where? Inside the main memory during the execution of program.

So without data structure there cannot be any application. Every application will have a set of instructions which will perform operation on data, so data is mandatory.

Then where the data is kept? Inside the main memory.
where the program will be? Inside the main memory.

So during the execution of a program, how the program will manage data inside the main memory and perform the operations, that is data structure.

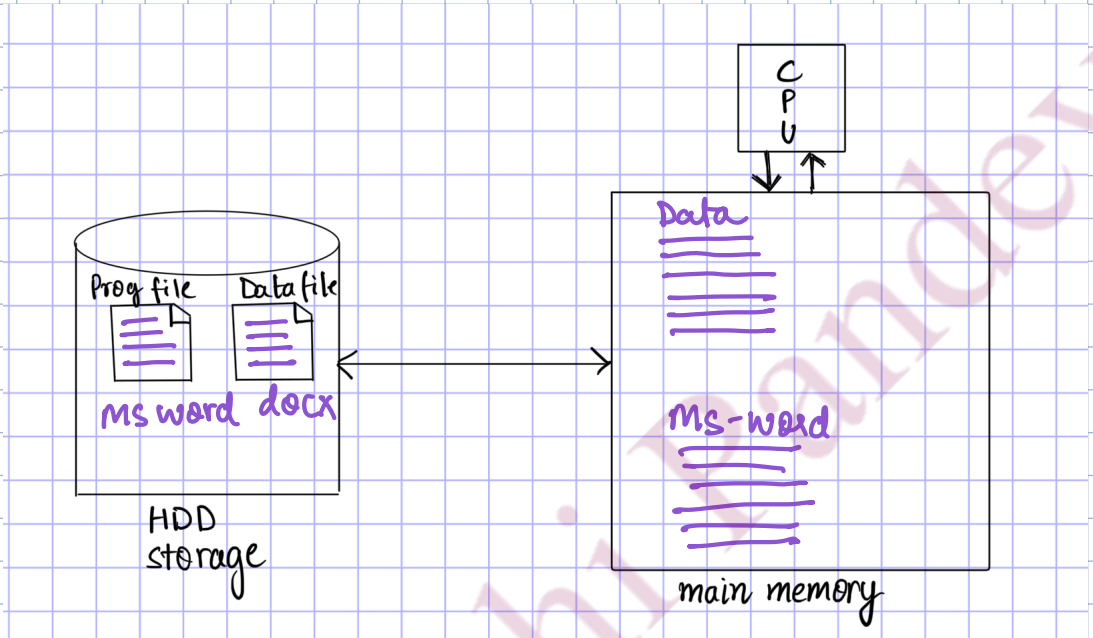
How the program utilizes data and how they put the data inside the main memory.



→ CPU will execute our program, i.e, it is a processor, it will execute the instructions.
→ main memory (RAM) is a temporary memory that is working memory. It is also called as primary memory
→ HDD storage is permanent storage.

where do we keep our programs? When we install any program in our PC or on our mobile phone, a program or an application, will get installed on storage

Then where do we keep our data? That also, we keep it on hard disk. Suppose you have any pictures or photos or videos or any documents that all you keep it on your hard disk or in your mobile phone, you keep it in a storage. So these are data files.

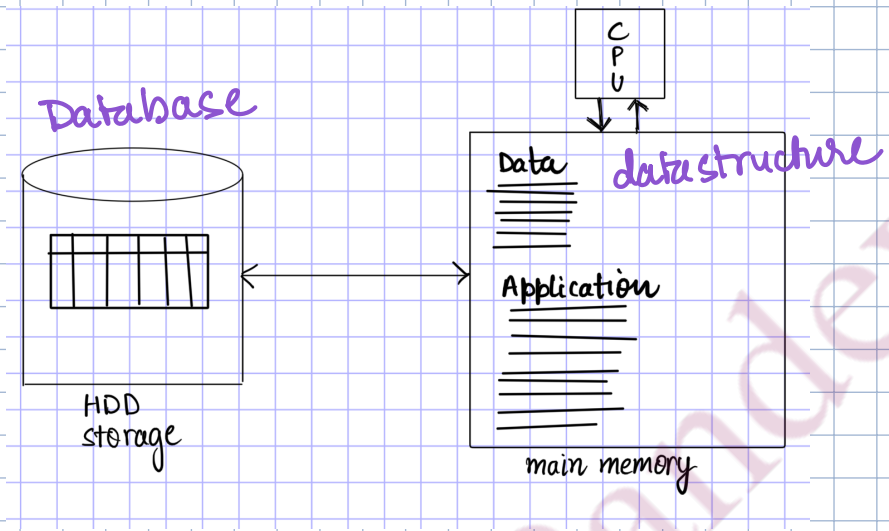


→ Every application deals with some data, whether it is MS-Word or if it is Notepad or chrome and that data has to be inside main memory. How you organize data inside the main memory, so that it can be easily used by this application, efficiently utilized by the application program. So how you organize the data?

→ So the arrangement or organizing of the data inside the main memory for efficient utilization by the application, that arrangement is called as data structure. So data structure are formed in main memory during the execution time of the program

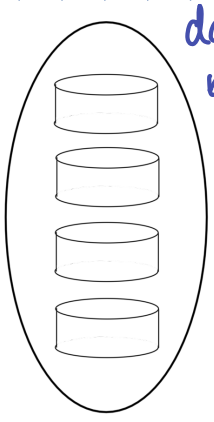
→ When the program runs it needs data. So the question how it will arrange the data in the main memory for performing its operations. So that arrangement is called as data structures So data structure is a part of running program. You may be knowing different data structures like arrays or linked lists or trees or hashtables, whatever the data structure is suitable, application can use that particular data structure here for arranging its data (text data or multimedia data like images or videos)

Database :- when the data is larger in size or commercial data that is used in businesses like banks or retail stores or manufacturing farms, they will have a lot of data and they will have some organized data in the form of database tables or relational data and where they keep that relational data, all that data is stored on the disk.



→ when you are pulling the data from the hard disk or from the storage to the main memory during execution then you need definitely data structure. So arrangement of data here in the main memory is data structure. Then how is this data is organised in form of table on the disk. This is a database. A database means arranging the data in some model like say a relational model in the permanent storage, so that it can be retrived or accessed by applications easily. That arrangment in the hard disk or in the permanent storage it's called as database

→ Commercial data can be categorized into two types :-
① Operational data - is used daily
② legacy data (old data) - can be kept as storage somewhere, if required, we can fetch the data and use it or you can say historical data. (10 years or 50 years of data is kept on array of disk)



datawarehouse → So mostly commercial firm will have their datawarehouses. That datawarehouse is helpful for analyzing the bussiness or making policies or stating a new trend or giving offers to the customers, dealing with the customers, so that previous data or old data will help organisation in taking decisions. So this large sized data is datawarehouse and algorithms used for algorithm written for analyzing data → data mining algorithms

Big Data :- With the start of internet, a huge sized data is accumulated day by day in internet, that data is about things, about people, about places; lot of data is available in internet and by analyzing that data we can take a lot of decisions, that is for management, for governance or for businesses, analysis is very useful about that data. Storing and utilizing that very large sized data that study is Big Data.

Stack Vs Heap memory

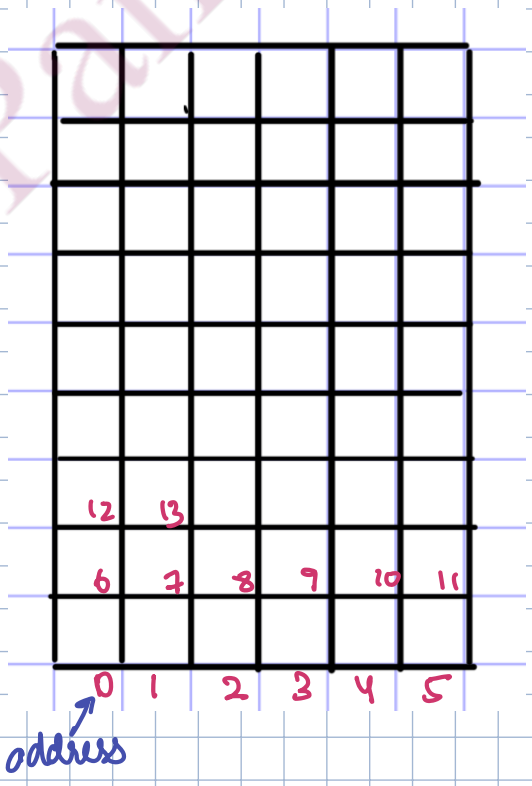
https://www.geeksforgeeks.org/difference-between-static-and-dynamic-memory-allocation-in-c/

Static vs dynamic memory Allocation

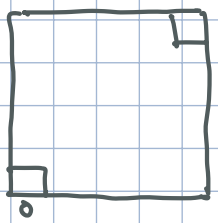
1. About Main Memory
2. How Program use main memory
3. Static Allocation
4. Dynamic Allocation

1. About Main Memory

Suppose the block on right side shows a memory, this is a memory. So memory is divided into smaller addressable unit that is called as byte. So memory is divided into bytes. So those check-boxes let us assume, those are bytes and the entire block is a memory. Every byte is having its address. Addresses will have single value and they are linear. So depends on the size of memory, every byte will have its own address.



If you have larger size of RAM that is 4 GB or 8 GB, that entire memory is not used as a single unit but it is a divided. It is divided into manageable pieces that are called segment. Usually size of segment will be 64 KB



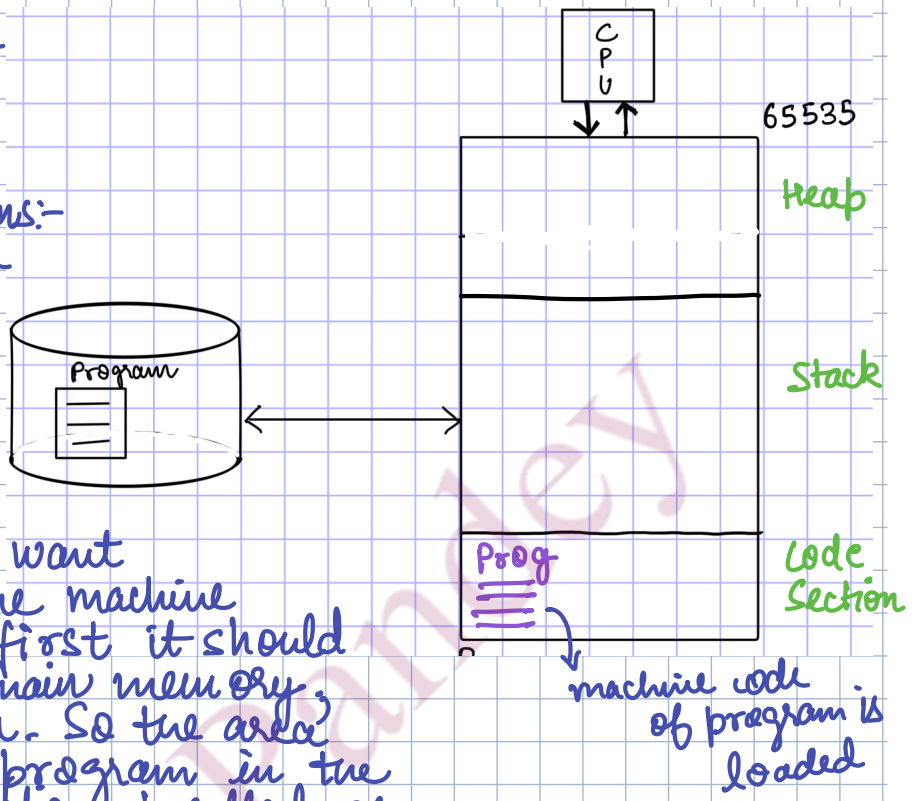
Total

0 - 65535 = 65536

= 64 × 1024 = 64KB

2: How a program uses main memory

→ The entire main memory is divided into three sections and used by a program. So a program uses main memory by dividing into three sections:- code section, the stack and the heap.



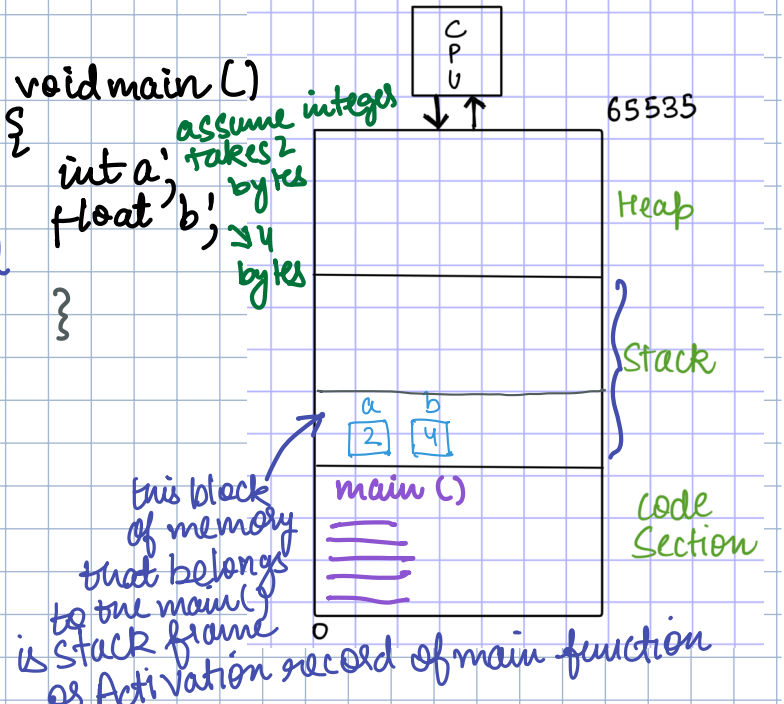
How a program uses these three sections.

→ We have our program file on the harddisk. If we want to run this program, the machine code of the program, first it should be brought inside the main memory, inside the code section. So the area that is occupied by the program in the main memory, that section is called as code section, that may not be fixed, it depends on size of program.

Once the machine code is loaded, the CPU will start executing the program, and this program will utilize the remaining memory as divided into stack and heap.

How this stack and heap work?

We have these two variable 2 bytes and 4 bytes, total 6 bytes of memory. So that 6 bytes of memory is allocated inside the stack.
The size of the memory required by a function was decided at compile time only (by compiler) This is called as static memory allocation. Static means how many bytes of memory is required by this function was decided at compile time. So what is static? Size of the memory is static value [when it was decided? compile time.



Now the next thing we should also know that if there are sequence of function calls. Then how the memory is allocated inside stack?

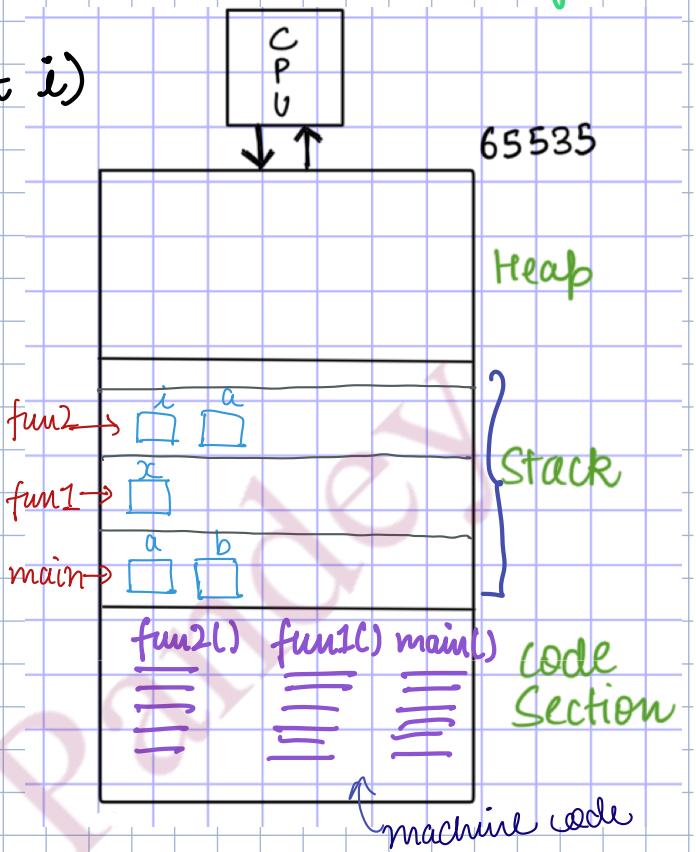
① When we run this program, the machine code of this program will be copied in code section

```
void fun2(int i)
{ int a;
  ——
  ——
}
```

② When the program starts executing it will start from main function, the moment it enters inside main function it requires a variable. So the memory for 'a' and 'b' will be allocated inside stack frame area.

```
void fun1()
{ int x;
  fun2(x);
}

void main()
{ int a;
  float b;
  ——
  ——
  fun1();
}
```



③ main function call function fun1()
Now the control goes to fun1(). The moment control goes here inside fun1(), the first thing is variable is required. Variable declaration is there. So the variable is created inside the stack for this function fun1() i.e x. Right now function fun1() is executing. So this topmost activation record belong to currently executing fun1().

④ Then fun1() will call fun2(). So again the control goes to function fun2(). Then that is having 2 variables one is its parameter and other one is its local variable. So memory is allocated for those variables 'i' and 'a' and this is for fun2(). Now presently fun2() is running and the topmost activation record in stack area is fun2().

⑤ Now, one thing you can observe that we started from main function. It has not yet finished but it has called fun1(). So main function activation record is as it is inside the stack. Then activation record for fun1() is created, means memory for fun1() is allocated, then it is still running but it has called fun2(), so the activation record for fun2() is created and activation record of fun1() is still there in the memory.

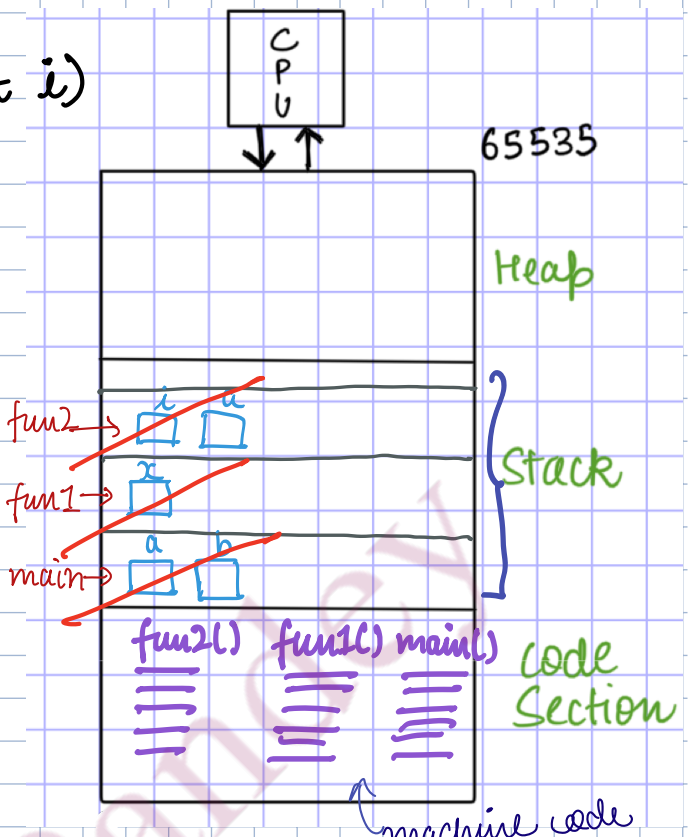
⑥ fun2() is the currently executing function. Now, when fun2() has finished, terminated, then control goes to back to fun1(). What happens to activation record of that function fun2()? This will be deleted.

⑦ Then after this fun1() has finished executing this statement, it will come back to the main function after this call fun1(). Once the function fun1() ends, its activation record is also removed from the main memory, that is, from the stack.

```
void fun2(int i)
{ int a;
  ——
  ——
}

void fun1()
{ int x;
  fun2(x);
}

void main()
{ int a;
  float b;
  ——
  ——
  fun1();
}
```



⑧ Then main function also ends. So it's activation record is also deleted from main memory and the program ends.

→ So now you can see that how the activation records for sequence of function calls were created.

So the way activation records are created or deleted, this mechanism is stack. So that's why this section of memory behaves like a stack during the function call. So that's why it is named as stack

So that's all, that's how the main memory is used or stack memory is used for function calls.

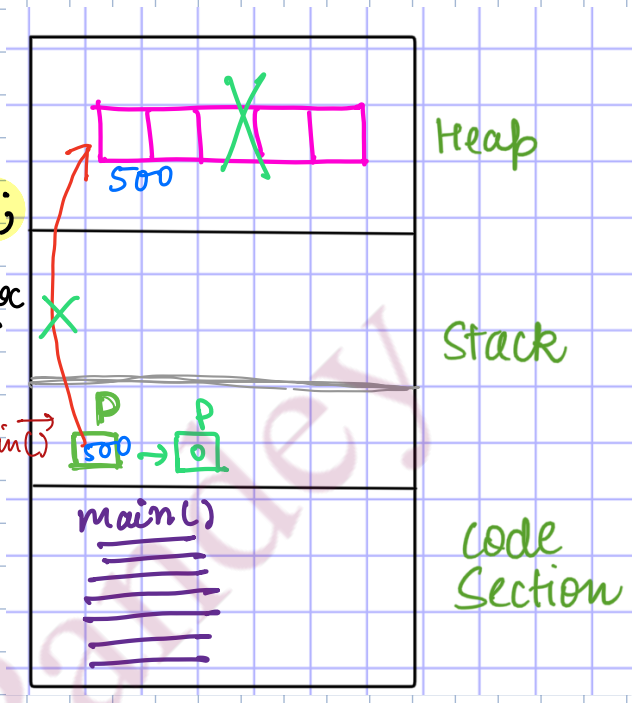
Now one important thing to observe; how much memory is required for a function? → depends on the number of variables and their sizes and this is decided by compiler only.
So this memory is automatically created and automatically destroyed, the programmer doesn't have to do anything for its allocation and destruction, just programmer has to declare the variable. So the conclusion is whatever the variables you declare in program or whatever the parameters your functions are taking, for all of them, memory is allocated inside the stack and it is automatically created and automatically destroyed when the function ends.

→ heap and how memory is dynamically allocated from heap.

→ how heap memory is utilized by a program.

Heap: Heap means what? just piling up. If the things are kept one above the another or just randomly, we use the term Heap. So, heap is used in 2 cases. One if the things are properly organised like a tower like thing, then also it is a heap and if it is not organised and also its looking like a tower then also we call it as a heap. Heap Word or term Heap can be used for organised things as well as unorganised things. So here, Heap is the term used for unorganised memory. It's not organised, stack memory is organized.

```
void main()
{
    int *p;        // 2 bytes
C++ p = new int[5];
C-lang p = (int*) malloc
           (2*5);
    delete[]p;
    p = NULL;
}
```



→ Heap memory should be treated like a resource. Heap memory should be used like a resource, when required you take the memory, when you don't require, you release the memory.

→ Program can't directly access heap memory. It can directly access anything inside code section, anything inside stack. But it will not access heap memory. Then how do they access heap memory? Using pointer

→ 'new' is used for allocating memory in the heap in C++ and Java. Now program cannot directly access heap, it has to access pointer and pointer will give the address of that memory then the program can reach that location and access these integers

→ When you don't need the memory, you should deallocate it, as you have requested for allocation, same way you should request deallocation of the memory. Heap memory should be explicitly requested and explicitly released or disposed. Otherwise if you are not releasing it then the memory will be still belong to your program and that memory cannot be used again so it causes loss of memory and it is also called as memory leak.

Physical vs Logical Data Structures

https://dotnettutorials.net/lesson/physical-vs-logical-data-structure/

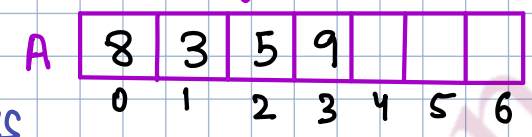
Types of Data structures

1. physical Data structures
2. logical Data structures

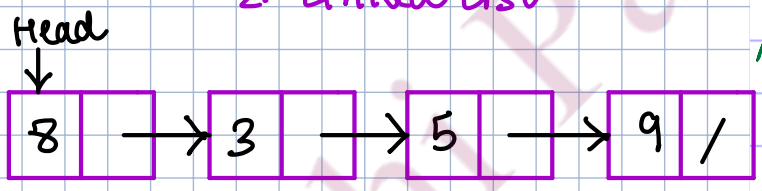
① Physical Data structures

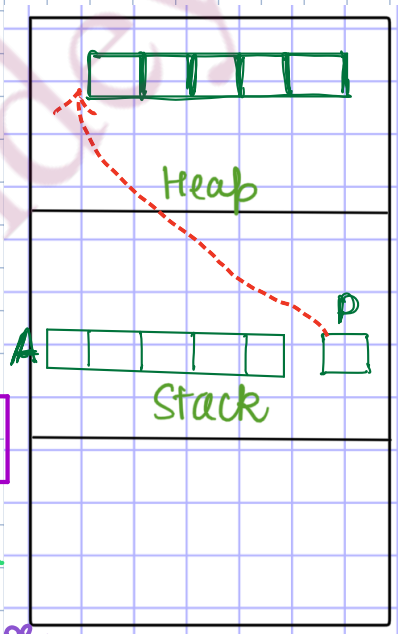
These are the two physical data structure
1. Array
2. Linked List
We can have more physical data structures by taking the combination of array and Linked list, we can have some variations in them.

1. Array



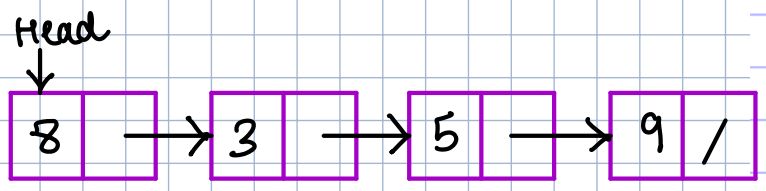
2. Linked List

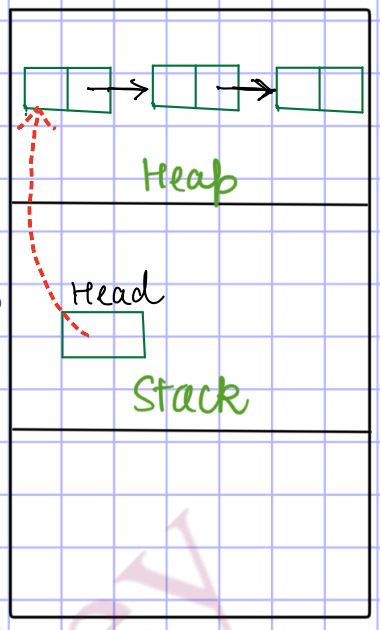




Why do we call them (array and Linked list) as physical data structure? The reason is, these data structure decides or defines how the memory is organized, how the memory is allocated.

array: (directly supported by programming languages like it is there in C language, in C++ and even in Java. This is directly supported. This is collection of contiguous memory locations, all these locations are side by side. If we have an array for seven integers then all these places for seven integer are together, They are at one place. This array will have fixed size, once it is created of some size, then that size cannot be increased or decreased. So it is of fixed size. So the size of an array is static. Where this array can be created? An array can be created either inside stack or it can be created inside heap. When to use this data structure? When you are sure, what is the maximum number of elements that you are going to store, if you know the length of the list then you can go for array.





Linked List: This is a complete dynamic data structure and it is a collection of nodes, where each node contains data and is linked to the next node. The length of this list can grow and reduce dynamically. So, it is having variable length. So as per, your requirement, you can go on add more and more nodes and add more elements or you can reduce the size.

→ The linked list is always created in a heap. Collection of nodes are always created in heap. Head pointer may be inside the stack.

→ We use linked list when the size of the list is not known

So array and linkedlist are physical because they define how the memory should be organized for storing the elements or for storing the data.

② Logical Data Structure

1. Stack] linear → works on discipline, i.e, LIFO (Last in First out)
2. Queues] linear → works on discipline, FIFO (First in First Out)
3. Trees] non linear → organised like a hierarchy
4. Graphs] non linear → collection of nodes and links b/w nodes
5. HashTable] tabular

difference b/w physical and logical data structure

→ Physical data structure are actually meant for storing the data, they will hold the data, they will actually store the data in the memory. Then, when you have a list of values you may be performing operations like, inserting more values, or deleting existing values or searching for the values and many more operations. How you want to utilize those values? How you will be performing insertion and deletion? What is the discipline that you are going to follow? That discipline is defined by these data structure- Stack, Queues, trees, graphs and hash table.

Stack, Queues } linear
Trees, Graph } non-linear
HashTable] tabular

→ So these data structures are actually used in applications and they are actually used in algorithms. And for implementing these data structures, we either use array or linked list. These logical data structures are implemented using any of these physical data structures, either array or linked list or combination of array and linked list.

ADT (Abstract Datatype)

1. What is ADT
2. List ADT

① What is ADT (abstract Datatype)

What is meant by DataType?
So a datype is defined as :-
1. Representation of Data
2. Operation on Data

This is how integer is represented inside memory in 2 bytes. So this is the representation of integer datatype.

→ This is not ADT, this is a primitive datatype

int x; (in C, C++)

one bit is reserved as signed bit to allow both positive as well as negative numbers. Rest of the 15 bits are allowed for storing data. (i.e storing a number)

What are the operations allowed on integer type data in C, C++ languages? arithmetic operation are allowed on them. That is: +, −, *, /, %. Apart from these relational operation <, > and increament, decrement operators are allowed: ++ and −−.

→ Any datatype in a language will have its representations and the set of operations on the data.

What is Abstract? Abstract means hiding internal details. Integer datatype and its operations, for performing those operation, do we really need to know how they are performed in the binary form inside the main memory? → No. We are concerned about declaring a variable and using it by performing these operations. So we need not know internal details, how these operations are performed. Without knowing them, we can use them. So internal details are abstract for us.

What is abstract data type?

This is related to Object Oriented programming languages. When the Object Oriented Programming Languages being started to use in Software Development, then using the classes, we can define our own datatypes, that are Abstract, i.e, without knowing the internal details, we can use them. So this term `ADT` is related to object oriented programming.

Let's take example of a list, that is list of elements or collection of elements.

List → 8, 3, 9, 4, 6, 10, 12
0 1 2 3 4 5 6

How can we represent a list? So what are the things that I have to store for representing a list?

Data : 1. space for storing elements

2. Capacity of a list

3. inside that capacity, how many element already we have in the list i.e length of the list or size of the list. → size

So for representation of this list we have two options:-

1. Array
2. Linked List.

list → 8, 3, 9, 4, 6, 10, 12, 15
0 1 2 3 4 5 6 7

Operation: add(x)
remove()
search(key)
⋮

- add(element) / append(ele)
- add(index, ele) / insert(index, ele)
- remove(index)
- set(index, ele) / Replace(index, ele)
- get(index)
- search(key) / contains(key)
- Sort

Now this list is an abstract data type. It is having representation of data and the operations on the data. So when you have 2 things → data representations and operations on the data, together it becomes a data type. How internally the things are working, I need not worry, that's what abstract means. So, the concept of ADT, define the data and the operations on data together and let it be used as data type, by hiding all the internal details.

Time and Space Complexity

Time complexity :- In daily life, when we do any task, any task, we want to know how much time it takes for that, performing that particular task. Suppose, it is a one hour task or a one day task. So that amount of time required depending on the work that we have to do. So, usually in daily life, we measure the time based on the work that we have to do.

→ So now we are using machines to do our work, that is, computer to do our work. So we want to know how much time the machine takes for doing the same task. So how much time a machines takes is very important for us. So we use computer for problem solving. What type of problem solving? The work that we used to using pen and paper. That same work we want our computer to do that. So computer are used for performing computation task. So computation also needs time. So you want to measure how much time a machine will take.

→ So actually that depends on the process or the procedure for completing task. So the Time Complexity basically depends on the procedure that you are adopting.

n
O(n)
A

2	5	9	6	4	12	15	8	3	7
0	1	2	3	4	5	6	7	8	9

→ This is array of some size and some element are there. There are total 10 elements. The list of elements in an array → we say n elements. 'n' means some no. of element. let say there are n elements in this list Now if we want to add these elements, we have to go through all of them, one by one you have to take the element and go on adding it. So how much time it takes? → depends on the number of elements. So what is time time taken? → n, we say n.

Now we want to search for a particular number in array. Now whether element 12 is there or not, search for it. Yes 12 is there. Now search for element 21. It is not there. So, utmost how much time is it taking? It depends on the number of elements that you have to compare. So how many elements are there? n elements are there. So what is the time taken? n. So the time is n.

So it means, in a list, if you have some elements and you are going through all of them just once, then the time is 'n'. So this n, we represent it as a degree. So we can say O(n) (order of n)

Next important thing, if you want to access all the elements then what the code that you have to write?

```
for(i=0; i<n; i++)
{
   ==
}
```

logic or procedure (adding elements or counting elements or search elements)

n → O(n)

A

2	5	9	6	4	12	15	8	3	7
0	1	2	3	4	5	6	7	8	9

This for loop is taking us through all those elements once. So how many elements? → n. What is the time? → O(n)
For finding the time complexity either you can measure the time based on the work that you are doing, means what is your procedure, if you are clear with your procedure, you can know the time or else from the code, program code also you can find the time complexity. If there is a for loop, going through 0 to last element, then it's n, it's taking O(n) time, whatever it is there inside, it will repeat for n times. So we analyze based on the procedure also, based on the code also. And the most confusing thing is, when the code is given, we get confused, how to analyze this one. So actually what the code is doing, you do that work and based on the work you analyse it.

So if a for loop is used means there is a chance that time is O(n)

Now we move to next situation

In this list, suppose, being on the first element, we are comparing or processing all other elements once

A

2	5	9	6	4	12	15	8	3	7
0	1	2	3	4	5	6	7	8	9

Then being on the second element, again, we are processing all other elements.

O(n²)

A

2	5	9	6	4	12	15	8	3	7
0	1	2	3	4	5	6	7	8	9

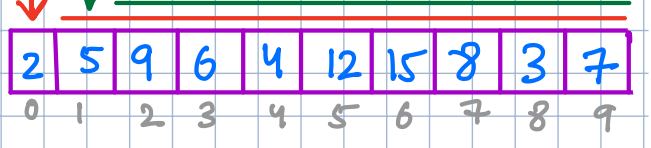
So, n elements are being processed. But, how many times? → For each element n elements are processed. So this will be: $n \times n = n^2$. So we can say $O(n^2)$

how the code should look like?

So when you have nested for loops, its n^2.

```
for(i=0; i<n; i++)
{  for(j=0; j<n; j++)
   {
      ==
   }
}
```

Third situation, for the same array

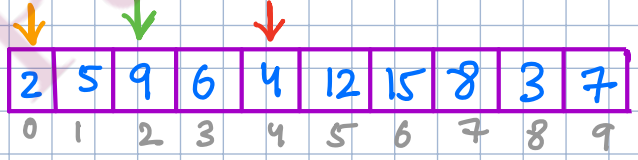


Suppose being on first element, we are processing the rest of the elements. So, n-1 elements we are processing. Then being on the second element, we are processing next rest of the elements after it, i.e, n-2. Then being on the third element, we are processing rest of them. i.e n-3. Like that, it will go on reducing, then finally it will be 4 elements, 3 elements, 2 elements, then 1. This is Sum of first n natural numbers. This will be $\frac{n^2-n}{2}$. So what is the degree of this polynomial

$$n-1+n-2+n-3+\ldots+4+3+2+1 = \frac{n(n-1)}{2} = \frac{n^2-n}{2} = O(n^2)$$

This is n^2. So it's $O(n^2)$. So that's what the time complexity we are writing is the degree of the polynomial.

Fourth situation, for the same array



Now, in the given array, suppose first of all we are processing the middle element. Suppose 4 is the middle element. So 5th element is the middle element. Then, next, either on the left side or on the right side, again we go in the middle. suppose third element is the middle element on this side. Then again the first element is middle element on that side. So in this way we are not processing the entire list, we are processing half of the list. Then again its half. Then again it's half. So that process is alway dividing the list by two.

So when something is successively divided until it reaches one, that is represented as log. And we are dividing it by 2, 1/2 → $\log_2$ So log of n elements → $\log_2 n$

$$\log$$

$$1/2 \rightarrow \log_2$$

$$n \text{ elements} \rightarrow \log_2 n$$

Code

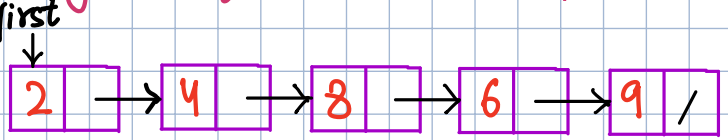
```
for(i=n; i>1; i=i/2)
{
   ——————
   ——————
   ——————
}
```

$\log_2 n$

```
i=n;
while(i>1)
{
   i=i/2;
}
```

So the time complexity is logn, if we are not processing all elements

What all we studied about array, same things apply to linked list.



Next is matrix

	0	1	2	3
0	8	3	5	9
1	7	6	4	2
2	6	5	3	9
3	10	4	2	6

A

next is matrix. Matrix is having how many elements? The dimension is 4x4. So total how many elements. If dimension is nxn, then total n^2 elements. So when you are processing upon a matrix, then it will require n^2 amount of time → $O(n^2)$, if you are processing all the elements.

4 x 4
$n \times n = n^2$
→ $O(n^2)$
[for processing all elements once]

If you process just a row, then a row is having n elements → O(n). If you are processing just a column. So again its n. → O(n). If you are processing all elements, then it is n^2.

So for processing a matrix of element we need two nested for loops.

```
for (i=0; i<n; i++)
{
    for (j=0; j<n; j++)
    {
        ==========
    }
}
```

$O(n^2)$

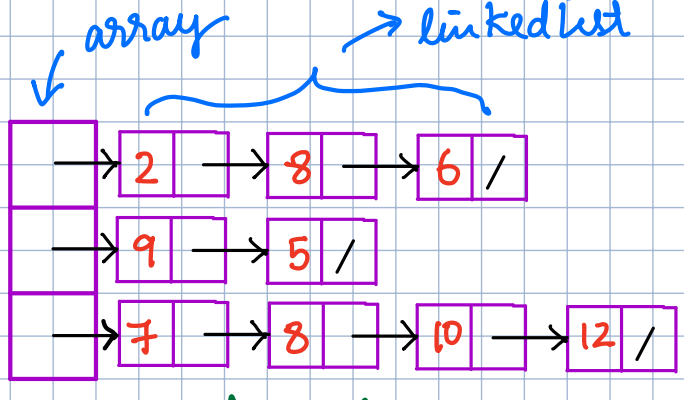
```
for(i=0;i<n; i++)
{ for(j=0;j<n;j++)
  {
    for(k=0;k<n;k++)
    {
      ==
    }
  }
}
```

$O(n^3)$

for three nested for loop, then time will be more → n^3

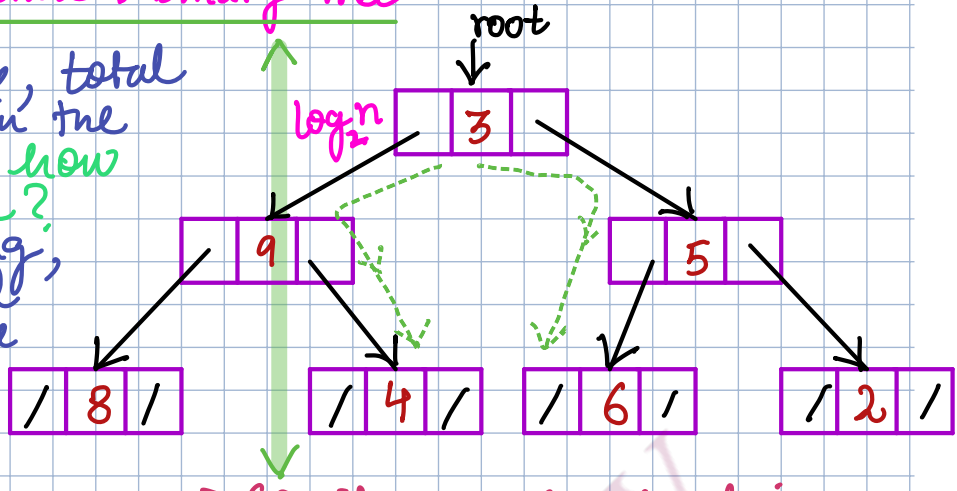
Coming to next structure: Array of Linked List

This structure looks like array of linked list. So total how many elements are there? let say n elements are there and these are of different sizes and array is of size m. So total how much processing is required? We have to process all these n elements as well as using this array. So we can say (m+n) processing is required. If you don't want to consider array then → O(n) i.e consider only number of elements.



$O(m+n)$

Coming to the next structure : binary tree



If you have binary tree, total 7 elements are there in the given binary tree. Then how much processing is done? Suppose, you are processing, searching or something and you are processing upon the elements, only along the path. Suppose you are processing only along the path, then what is the height? It depends on the height. So how many elements you are processing? See there are some elements, divided by 2, then divided by 2 until it reaches one element. So total seven elements are there. As from the bottom, if you see the number of element are divided by 2 and divided by 2. So it looks like $\log n$. $\rightarrow \log_2 n$ ($\log n$ base 2)

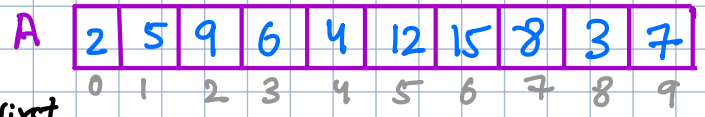
→ So if you are spending time upon a tree along the height of tree then it is $\log n$! $O(\log n)$

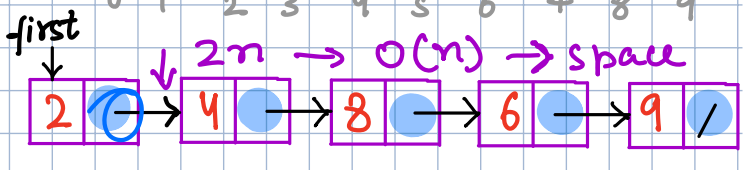
Next situation if you say, I want to process all elements, then how many elements are there? → n elements. So it is $O(n)$.

So the time complexity is solely and solely dependent on the work that you are doing.

Space Complexity :

We want to know how much space is consumed in main memory during the execution of a program. So in array n elements are there, so it's $O(n)$.

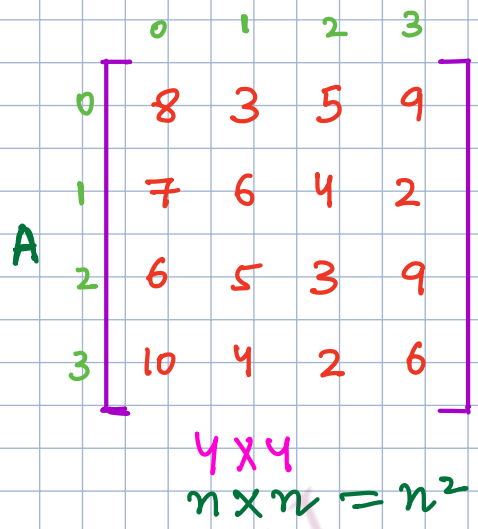




In linked list, n elements are there, so its $O(n)$. Space is n, means n integers or n float or n doubles, whatever it is, n elements we say. We are not concerned about the number of bytes. We want to know, the space is dependent on what? → n. If the value of n is more, the space will be more, that what the idea is. In linked list along with the elements, you need space for links also. So let say its → 2n. Again the degree is what? $O(n)$

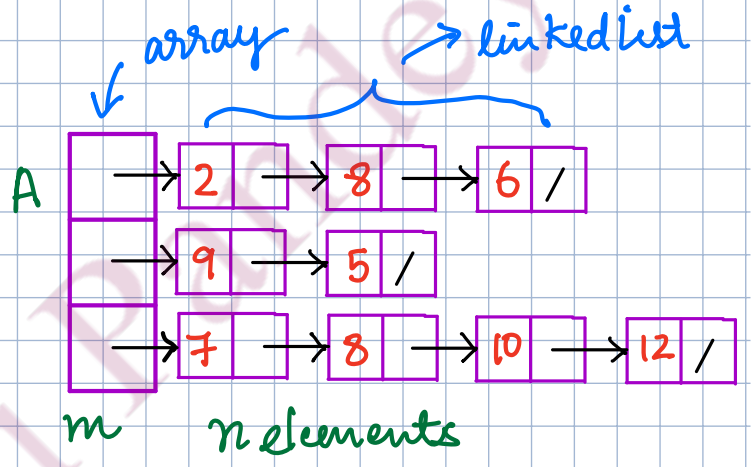
Now, coming to the matrix, the space is n^2.

$\rightarrow O(n^2)$

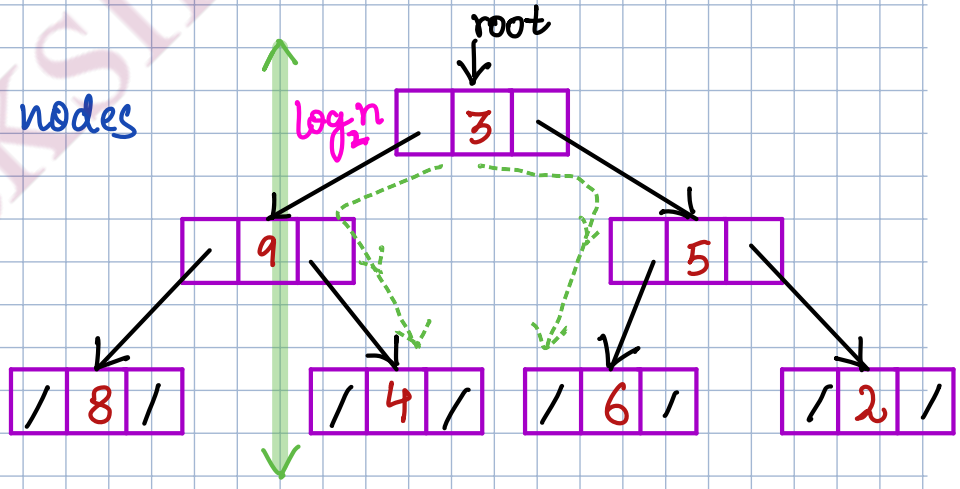


Here the space is $m+n$

$O(m+n)$



The space here is, n nodes are there, so its $O(n)$



Time and Space Complexity from Code

→ how to find out the time complexity from the program code.

So when we analyze the program code, we get a proper function, time function. So how to get a time function? So we have taken three example program codes, i.e, three functions, we have taken. Now how to analyze? we assume that every single statement in function or the program takes one unit of time. What does it mean by simple statement? The statement may be having arithmetic operations, assignment or conditional statement

Example1 This is swap function, taking two parameter and interchanging the value of x and y. What are the statements inside this function? These are the three statements. How much time each statements takes? One unit of time because it is just an assignment, So total time is 3. So $f(n) = 3$. So you got the time function as three. So three (3) is constant. → O(1). Why is it constant? because degree of n is zero → $O(n^0)$

```
void swap (x, y)
{
  int t;
  t = x;      ——— 1
  x = y;      ——— 1
  y = t;      ——— 1
}             f(n) = 3 × n^0
```

→ O(1)

→ $O(n^0) = O(1)$

Example2 Now in the given program the first statement is assignment So → one unit of time.

The next statement is for loop and the statement inside it will execute for n times. The assignment i=0 will execute one time initially and the increment i++ will happen for n times, condition also n times but one time when condition will fail it will stop → So total n+1 times execution. So you can ignore all other and take n+1. let us say time taken is n+1

The statement inside for loop execute for n times

```
int sum (int A[], int n)
{
  int s, i;
  s = 0;                  ——→ 1
  for (i=0; i<n; i++)     i=0: 1;  i<n: n+1;  i++: n;  n+1 → n+1
  {
    s = s + A[i];         ——→ n
  }
  return s;               ——→ 1
}
```

So total how much → f(n) = 2n+3 → O(n)

The last return statement will execute 1 time.

So time taken is → O(n).

Example 3

Now in the given code the first for loop will take n+1 times for execution. The second for loop inside first for loop will execute for n times → But since the loop is inside a loop (nest loop) it will be $n \times (n+1) = n^2 + n$

The statement inside nested loop will execute for $n \times n$.

So the time function is :-

$f(n) = 2n^2 + 2n + 1$

What is the degree ? → 2

So it is $O(n^2)$

```
void Add (int n)
{
  int i, j;
  for (i=0; i<n; i++)        → n+1
  {
    for ( j=0; j<n; j++)     → n x (n+1)
    {
      C[i][j]=A[i][j]+
              B[i][j];
    }                        → n x (n)
  }
}
```

$f(n) = 2n^2 + 2n + 1$

$O(n^2)$

$O(n^2)$

$\Theta(n^2)$

$\Omega(n^2)$

Example 4

Now if we want to find the time complexity, as we say, every statement, we should consider it as one unit of time

How much time a function is taking, you find out that one, don't say the statement is 1.

Hence the time taken by fun1() is $O(n)$

```
fun1()
{
  fun2();        ——— n
}                ——— O(n)

fun2()
{
  for (i=0; i<n; i++)      ——— n
  {
    ———
    ———
  }
}
```

loop

Asymptotic Notations : Big Oh, Omega, theta

$$1 < \log n < n < n\log n < n^2 < n^3 < \ldots < 2^n < 3^n < n^n$$

polynomial — Exponential time complexities

→ So all these time complexities are arranged in the increasing order of their value.

→ So the starting value of n, need not be zero or one, it can be any starting value, but beyond that let us say :-

So from whichever point it is starting from there to infinity, if someone is greater, then we say it is greater.

like for example we have n^3 and 2^n. Now 2^n is much greater than n^3. How?

n^3	2^n	
$2^3 = 8$	$2^2 = 4$	→ so don't take small value of n
$10^3 = 1000$	$2^{10} = 1024$	} So from some value of n, that is correct.
$11^3 = 1221$	$2^{11} = 2048$	}

→ All the algorithm that we analyze, we are getting the time complexity like time function :-

$$f(n) = n \longrightarrow O(n)$$
$$f(n) = n^2 \longrightarrow O(n^2)$$

we did not analyze the code line by line, we were writing the time complexity based on the work done.

Now sometimes when we analyze a function, suppose, we got a time complexity as :-

$$f(n) = \sum_{i=1}^{n} i \longrightarrow O(n^2)$$

Now if you know the mathematical expansion of the sigma then

$$f(n) = \sum_{i=1}^{n} i = 1+2+3+\ldots+n = \frac{n(n+1)}{2}$$
$$= O(n^2)$$

$$f(n) = \sum_{i=1}^{n} i \times 2^i$$ → how to solve this?

→ we can't get exact formula, we have to get the approximate formula. So by solve the above function, we get approximate formula

→ So if you a getting a time function in some form, which cannot be simplified in term of n directly, then you cannot give the time complexity. you cannot mention the time, if you cannot get into form polynomial of n exactly, if you are getting approximate, then if that approximation is lower value or a higher value.

→ So the result of a function in the form of a polynomial is that polynomial, lower value or a higher value, its not exact. So which one you are taking?

So if you are taking lower value, then you can say

omega → $\Omega()$

if you are taking upper value, then you can say

Big Oh → $O()$ (at most)

if you are taking exact value, then we say

theta → $\Theta()$ (exact function)

These notations are lower bound, upper bound and tight bound. But on the safe side most we say Big Oh.

That means the Big Oh of n is not only upper only, it equal or upper bound, Omega is equal or lower bound. When you got theta you can also use Big Oh.

if $f(n) = O(n)$ as well $f(n) = \Omega(n)$ as then; → $f(n) = \Theta(n)$

So that's why Big Oh is commonly used.

Lower bound Ω

Upper bound O

tight bound Θ

These are useful in the situation when you cannot get the exact polynomial for our formula for a function then you can go for O (Big Oh)

Now, if you are not interested in analyzing the below function at most this will be $O(n2^n)$. you can see that is at most $n2^n$ → upper bound, big O notation. For omega $\sum_{i=1}^{n} i2^i \rightarrow \Omega(2^n)$ → (lower bound)

$$f(n) = \sum_{i=1}^{n} i \times 2^i$$

→ if we solve this and we can get the exact value. then we can go for theta

suppose $\Theta(n2^n)$ → exact function

Pepcoding (BPIT Placement Program Notes) ← referred from

Time complexity

Big oh O( ) → worst case time

Omega Ω( ) → Best case time

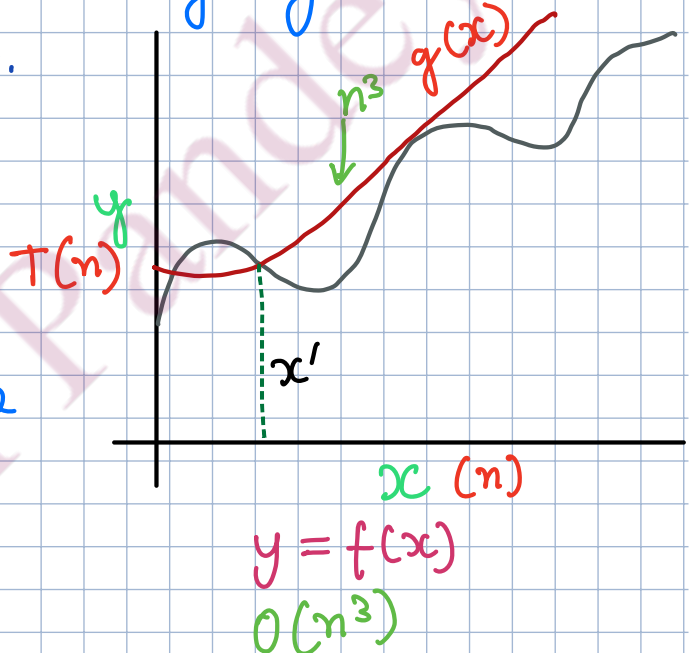
theta Θ( ) → Average case time

Big oh O( ) → Time taken by algoritam to exute in worst case. This is denoted using Big Oh Notation.

T(n) is time for n element.

g(x) graph lies above orignal graph for certain value of x (for large value).

→ Big Oh notation is used to describe asymptotic upper bound.



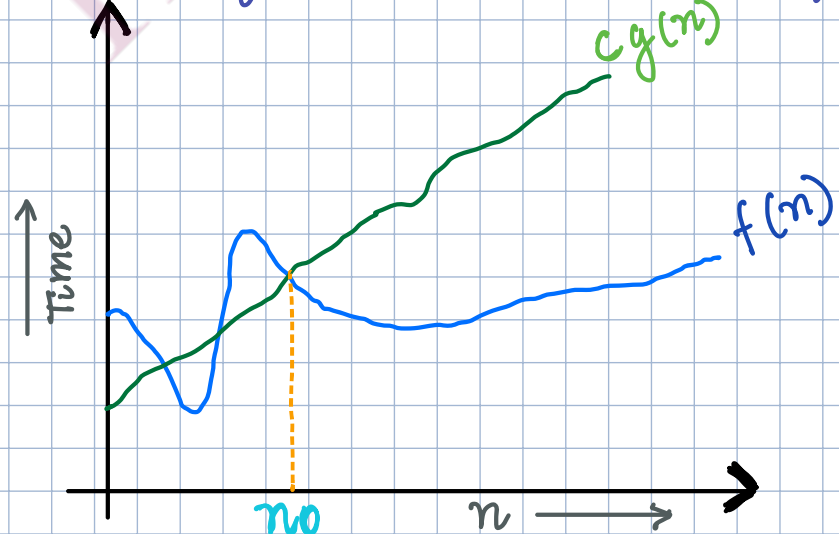
$y = f(x)$

$O(n^3)$

→ Mathematically if f(n) describes the running time of an algorithm; f(n) is O(g(n)), if there exist positive constant C and no such that

$0 <= f(n) <= c\ g(n)$ for all $n >= no$

n = used to give upper bound on a function



Omega $\Omega()$ → Time take by algorithm to execute in best case. Ω notation provides asymtotic lower bound.

let $f(n)$ define running time of an algorithm.

$f(n)$ is said to be $\Omega(g(n))$ if there exist positive constant C and (no) such that :-

$0 <= C\,g(n) <= f(n)$ for all $n >= no$

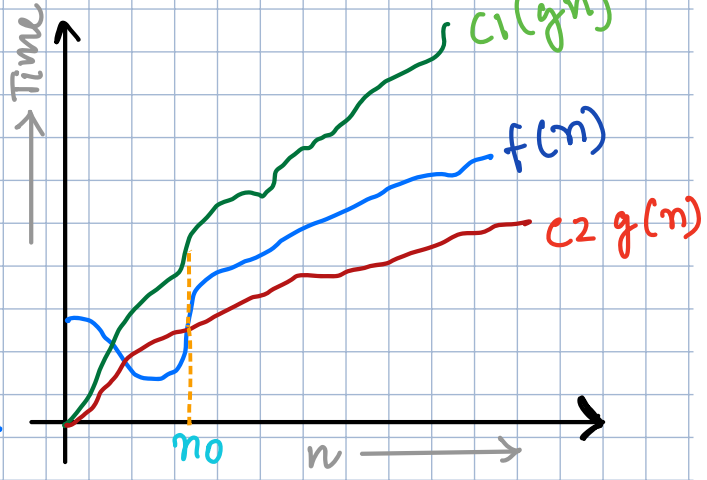
n = used to give lower bound on a function

Big Theta notation (Θ) :→ let $f(n)$ define running time of an algorithm.

$f(n)$ is said to be $\Theta(g(n))$ if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$. Mathematically,

$0 <= f(n) <= c1\,g(n)$ for $n >= no$

$0 <= c2\,g(n) <= f(n)$ for $n >= no$



Merging both the equation, we get:

$0 <= c2\,g(n) <= f(n) <= c1\,g(n)$ for $n >= no$

The equation simply means there exist positive constants c1 and c2 such that $f(n)$ is sandwich between $c2\,g(n)$ and $c1\,g(n)$.

Big Oh	Big Omega	Big theta
1. It is like <= rate of growth of an algorithm is less than or equal to a specific value.	It is like >= rate of growth is greater than or equal to specified value.	It is like == meaning the rate of growth is equal to a specified value.
2. The upper bound of algorithm is represented by Big O notation. Only the above function is bounded by Big O. The asymptotic upper bound is given by Big O notation.	The algorithm's lower bound is represented by Omega notation. The asymptotic lower bound is given by Omega notation.	The bound of function from above and below is represented by theta notation. The exact asymptotic behavior is done by this theta notation.
3. Big oh (O) - Worst case	Big Omega (Ω) - Best case.	Big Theta (Θ) - Average Case.
4. Big-O is a measure of the longest amount of time it could possibly take for the algorithm to complete.	Big-Ω take a small amount of time as compared to Big-O, it could possibly take for the algorithm to complete.	Big-Θ take very short amount of time compare to Big-O and Big-Ω, it could possibly take for the algorithm to complete.
5. Mathematically - Big O is :- $0 <= f(n) <= c\,g(n)$ for all $n >= n_0$	Mathematically - Big Omega is :- $0 <= c\,g(n) <= f(n)$ for all $n >= n_0$	Mathematically, Big theta is :- $0 <= c_2\,g(n) <= f(n) <= c_1\,g(n)$ for all $n >= n_0$

Linear Search

15	-5	-1	91	12	13	25	26	57
0	1	2	3	4	5	6	7	8

Sorted array is not nessary for linear search unlike binary search. d = 15

let say if I have to search for data = 15, then how many comparisons I have to make? I only have to check first element, this is a best time case.

If there exist total n elements in array then time taken to find 'd' in best case

Best case ← $T(n) = K$ (constant) → for comparision

```
if (arr[0] == data)
```

T(n) → time for n elements

if d = 100

Then we have to compare all n elements and we will not find 100 anywhere.

n	15	-5	-1	91	12	13	25	26	57
	0	1	2	3	4	5	6	7	8

n-1

for first element, it took constant time, then for rest n-1 elements it will take T(n-1) time

$$T(n) = K + T(n-1)$$

Now we will solve this equation

$$T(n) = K + T(n-1)$$

T(n-1) can be written as :-

$$T(n-1) = K + T(n-2)$$
$$T(n-2) = K + T(n-3)$$
$$T(n-3) = K + T(n-4)$$
$$\vdots$$
$$T(n-n) = K + T(n-(n-1))$$

+ adding all these equation

$$T(n) = (n+1)K + T(1)$$

$$\rightarrow T(n) = (n+1)K$$

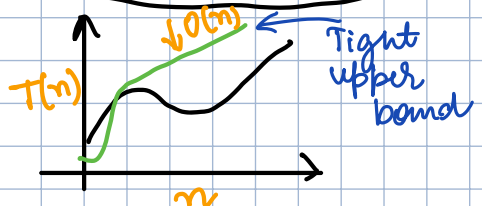
$$\rightarrow T(n) = nK + K$$

$$\rightarrow \boxed{T(n) = n.K}$$

we are removing constant means we are taking upper bound. in worst case

$$T(n) = n.K \Rightarrow O(n)$$

worst case



Average case = (best + worst) / 2

$$\theta(n) = \frac{K + n.K}{2} \simeq \frac{nK}{2}$$

$$\theta(n) = \frac{n}{2}$$

$O(n)$ ← worst case

$\Omega(1)$ ← best case

$\theta(n/2)$ ← Average case

so how will we judge which algorithm is superior?

we will always check worst case

Binary Search

Binary search will always take place in sorted array.

-5	-1	3	9	15	25	59	70	100
0	1	2	3	4	5	6	7	

data = -5

① → find middle & compare
② → discard left or right part

① since -5 < 15 → discard

-5	-1	3	9	15	25	59	70	100
0	1	2	3	4	5	6	7	

l = 0, r = 3

$\frac{0+8}{2} = 4$

finding middle and comparing will take constant time so we can ignore that.

Now T(n) is time complexity for binary search on n elements.

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

k: find middle and compare

T(n/2): Time taken by n/2 elements after discarding

Solving this equation :—

$T(n) = k + T(n/2)$ — $T(\frac{n}{2^0})$

$T(n/2) = k + T(n/4)$ — $T(\frac{n}{2^1})$

$T(n/4) = k + T(n/8)$ — $T(\frac{n}{2^2})$

$T(n/8) = k + T(n/16)$

$T(\frac{n}{2^x}) = k + T(1)$

$T(n/n) = k + T(0)$ → one or no element left → 0

$T(n) = (x+1)\cdot k + T(0)$

0 → x

⇒ x+1 terms

⇒ $\log_2 n + 1$

$\frac{n}{2^x} = 1$

$n = 2^x$

$\log_2 n = \log_2 2^x$

$\log_2 n = x$

$T(n) = (x+1)k$

$T(n) = (\log_2 n + 1)k$

$\Rightarrow O(n) = \log_2 n$

tight upper bound.

Recursion → time complexity

Factorial

Q → given a number n.
→ calculate the factorial of the number.

code :-

```
public static void main (String args) throws Exception {
  Scanner scn = new Scanner(System.in);
  int n = scn.nextInt();
  System.out.println(factorial(n));
}

public static int factorial(int n){
  if(n==0){
    return 1;
  }
  return n * factorial(n-1);
}
```

comparision → constant time (nano seconds)

let T(n) be time taken for calculate the factorial of n.

$T(n) = K + T(n-1) + K$

$T(n) = K' + T(n-1)$

Solving the equation

$T(n) = K' + T(n-1)$

$T(n-1) = K' + T(n-2)$

$T(n-2) = K' + T(n-3)$

⋮ ⋮ ⋮

$T(1) = K' + T(0)$

$T(n) = (n+1).K'$ → $T(n) = nK'$

$O(n) \simeq n$

→ upper bound Big Oh

Power linear (recursion)

→ given a number x.
→ given another number n.
→ calculate x raised to the power n.

code :-

```
public static void main (String[] args) throws Exception {
Scanner scn = new Scanner(System.in);
int x = scn.nextInt();
int n = scn.nextInt();
System.out.println(power(x,n));
}
public static int power(int x, int n){
    if(n==0){
        return 1;
    }
    int xpnm1 = power(x, n-1);
    int xpn = xpnm1 * x;
    return xpn;
}
```

x^n

$T(n) = k + T(n-1) + k$

$T(n) = k' + T(n-1)$

Solving the equation :-

$T(n) = k' + T(n-1)$

$T(n-1) = k' + T(n-2)$

$T(n-2) = k' + T(n-3)$

⋮ ⋮ ⋮

$T(1) = k' + T(0)$ (constant k')

$T(n) = (n+1)k' + k'$

$T(n) = nk' + 2k'$

$T(n) = n.k'$

→ $O(n)$

Power logarithmic (recursion)

x^n

```
public static int power (int x, int n) {
  if (n==0) {
    return 1;
  }
  int xpnb2 = power (x, n/2);
  int xpn = xpnb2 * xpnb2;
  if (n%2 ==1)
    xpn *= x;
  }
  return xpn;
}
```

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

$T(n) = k' + T\left(\frac{n}{2}\right)$

Solving the equation

$T\left(\frac{n}{2^0}\right) = k' + T\left(\frac{n}{2}\right)$

$T\left(\frac{n}{2}\right) = k' + T\left(\frac{n}{2^2}\right)$

$T\left(\frac{n}{2^2}\right) = k' + T\left(\frac{n}{2^3}\right)$

$\vdots \quad \vdots \quad \vdots$

$T(1) = k' + T(0)$

$\frac{n}{2^x} = 1$

$n = 2^x$

$\log_2 n = \log_2 2^x$

$\log_2 n = x$

$T(n) = (x+1)k'$

$T(n) = k' \log_2 n + k'$

$T(n) = k' \cdot \log_2 n$

$\rightarrow O(\log_2 n)$

logarithmic smart

Power logarithmic (recursion)

x^n

```
public static int power (int x, int n) {
  if (n==0) {
    return 1;
  }
  int xpn = power1(x, n/2) * power1(x, n/2);
  if (n%2 ==1)
    xpn *= x;
  }
  return xpn;
}
```

$T(n) = k + T(\frac{n}{2}) + T(\frac{n}{2}) + k$

$T(n) = k' + 2T(\frac{n}{2})$

Solving the equation

fake smart

$T(\frac{n}{2^0}) = k' + 2T(\frac{n}{2})$

$2T(\frac{n}{2^1}) = 2k' + 2 \times 2T(\frac{n}{4})$ → multiplying the whole equation by 2

$4T(\frac{n}{2^2}) = 4k' + 4 \times 2T(\frac{n}{8})$ → multiply the whole equation by 4

$8T(\frac{n}{2^3}) = 8k' + 8 \times 2T(\frac{n}{16})$ → multiply the whole equation by 8

$2^x T(\frac{n}{2^x}) = 2^x \cdot k' + 2 \cdot 2^x \cdot T(0) \to k$

$\frac{n}{2^x} = 1$

$n = 2^x$

$x = \log_2 n$

$T(n) = k'[1 + 2 + 4 + 8 \cdots + 2^x] + 2^{x+1} \cdot k$

G.P $= \frac{a(r^n - 1)}{r - 1}$

$= \frac{1(2^{x+1} - 1)}{2 - 1}$

$T(n) = k'[2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^x] + 2^{x+1} k$

$(x+1)$ term

$T(n) = k'(2^{x+1} - 1) + 2^{x+1} k$

$T(n) = 2^{x+1}(k' + k) - k'$

$O(2^{x+1}) \Rightarrow O(2^x \cdot 2)$

$\Rightarrow O(2^{\log_2 n} \cdot 2) \Rightarrow O(n \cdot 2)$

$\approx O(n)$

Print Maze Paths (recursion → on way up)

→ given a number n and a number m representing no. of rows and columns in a maze.
→ you are standing in the top-left corner and have to reach the bottom-right corner. Only two moves allowed 'h' → horizontal and 'v' (vertical) (1-step)
→ print list of all paths that can be used to move from top-left to bottom right

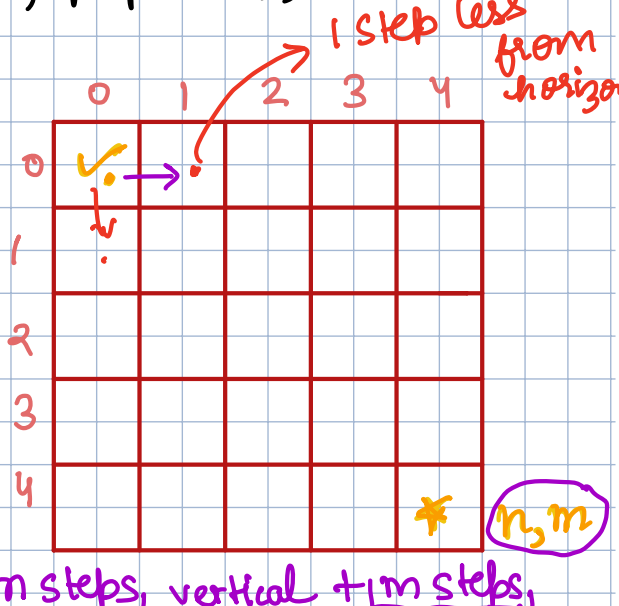
```
public static void main (String[] args) throws Exception {
    Scanner scn = new Scanner (System.in);
    int n = scn.nextInt();
    int m = scn.nextInt();
    printMazePaths (0, 0, n-1, m-1, "");
}
public static void printMazePaths (int sr, int sc, int dr, int dc,
                                   String psf) {
    if (sr==dr && sc==dc) {
        System.out.println (psf);
        return;
    }

    if (sc+1 <=dc) {
    print.MazePaths (sr, sc+1, dr, dc, psf+"h");
    }
    if (sr+1 <=dr) {
    printMazePaths (sr+1, sc, dr, dc, psf+"v");
    }
}
```

$$T(n+m) = T(n+(m-1)) + ((n-1)+m) + k$$

$$T(n+m) = T(n+m-1) + T(n+m-1) + k$$

Solving equation



$$T(n+m) = T(n+m-1) + T(n+m-1) + K$$

$$T(n+m) = 2T(n+m-1) + K$$

let $n+m = x$

$$T(x) = 2^1 T(x-1) + K$$

$$2T(x-1) = 2 \times 2T(x-2) + 2K$$ → multiply the equation by 2

$$4T(x-2) = 4 \times 2T(x-3) + 4K$$ → multiply the equation by 4

$$8T(x-3) = 8 \times 2T(x-4) + 8K$$ → multiply the equation by 8.

$$2^{x-1} T(x-(x-1)) = 2^x \cdot T(0) + 2^{x-1} K$$ → multiply by 2^{x-1}

$$T(x) = 2^x K' + (K + 2K + 4K + \cdots + 2^{x-1} \cdot K)$$

$$T(x) = 2^x K' + K(1 + 2 + 4 + \cdots + 2^{x-1})$$

$$T(x) = 2^x K' + K(1 + 2^1 + 2^2 + 2^3 + \cdots + 2^{x-1})$$

$$T(x) = 2^x K' + 2^x \cdot K + K$$

$$T(x) = 2^x K'' + K$$

$$T(n+m) = 2^{n+m} \cdot K'' + K$$

$$T.C \Rightarrow \boxed{O(2^{n+m})}$$

GP $= \dfrac{a(r^n - 1)}{r - 1}$

$= \dfrac{1 \cdot (2^x - 1)}{(2-1)}$

Iterative case example

Example 1

```
for (int i=1; i<=n; i++) {
    for (int j=1; j<=n; j++) {
        ——— (work)
        print ("hi")
    }
}
```

n times (outer loop)

n times (inner loop)

T.C ⇒ $n \times n$

i = 1 2 3 4 5 ... n

(n) + (n) + (n) + (n) + (n) + ... + (n)

n times

n^2 times

Example 2

```
for (int i=0; i<=n; i+=k) {
    for (int j=1; j<=k; j++) {
    }
}
```

n, k → constant

T.C ⇒ ?

k times

for every value of i, inner loop will execute for k times

x+1 terms

$x \cdot k = n$

i = 0 k 2k 3k 4k ... n

(k) (k) (k) (k) (k) (k)

x+1 times

$x \cdot k = n$

$$x = \frac{n}{k}$$

$\Rightarrow (x+1) \cdot k$

$\approx x \cdot k$

$= \frac{n}{k} \cdot k \Rightarrow O(n)$

Example 3

```
for (int i=1 ; i<=n ; i*=2) {
    print ("hi")
}
```

2^0 2^1 2^2 2^3 2^4 ..... 2^x ($2^x = n$) } $x+1$ terms

1 2 4 8 16 ..... 2^x

$x + 1$ terms

$2^x = n$

$\log_2 2^x = \log_2 n$

$x = \log_2 n$

$\log_2 n + 1$

$\rightarrow O(\log_2 n)$

Example 4

```
for (int i=1 , j=1; j<=n ; i++, j= i+j) {
    print ("hi")
}
```

i	j
1	1
2	3
3	6
4	10
5	15
6	21
7	28
	...
	n

$\sqrt{n}$

Sum of first 5 natural number

i → 1 2 3 4 5 x

j → 1 3 6 10 15 21 28 ... n

$\frac{x(x+1)}{2} = n$

$x^2 \simeq n$

$x \simeq \sqrt{n}$

$O(\sqrt{n})$

Example 5

```
for(int i=n; i>=1; i/=2){

}
```

→ if i/=3 $\log_3 n$

if i/=4 $\log_4 n$

n

$\frac{n}{2}$

$\frac{n}{2^2}$

$\frac{n}{2^3}$

⋮

1

$\Rightarrow \frac{n}{2^x} = 1$

$n = 2^x$

$\log_2 n = x$