What is a process : whenever we want to perform any work , we always follow some steps , that is a process

Starting point -> Destination (Difference is in their approach of doing something)

Analysis : Analysing performance ,we measure , by checking time OR hurdles that they have faced

Who has taken least time :

Algorithm : Logic used to perform some task

Int add(int x, int y)

{

Int result ;

Result = x+y;

Return result;

}

Int add(int x, int y)

{

Return x+y;

}

Factorial :

Non recursive

int fact(int n)

{

int i,j; 1

i=1; 2

for(j=2;j<=n;j++) 3

{

i=i\*j; 4

}

return i; 5

}

Recursive

int(fact int n)

{

if(n==1) return 1; 1

else

return (n\*fact(n-1)); 2

}

Out of these programs , which one is better?

We need some ~~measurement~~, and for that we need some criteria

1. ~~No. of lines of code~~
2. ~~Easy to understand / Easy to read~~
3. ~~Time to execute~~

Is time to execute is related to no. of lines of code?

1. Which takes less space
2. No. of statements executing (No of statements & no of statements executing are they same?)

No. of lines of code

HLL > Preprocessor > Pure HLL > Compiler > Assembly Form > (Machine Lang) > Linker Loader

Preprocessor > Will remove all the comments /blank lines > It only gives single line to compiler to compile

Easy to understand / Easy to read : Easiness is again defined by users

No of lines of code /Easiness could be secondary parameter, can not be primary parameters

Time to execute

A = b + c;

HLL > Preprocessor > Pure HLL > Compiler > Assembly Form > (Machine Lang) > Linker Loader LOAD R0, A

LOAD R1, B

ADD R0,R1

STORE R1

Assembly Lang is dependent on Machine Architecture

If we write same statement A = B + C on different machines, it will different times

AMD : 2 clock cycles , 1 Clock cycles

Intel : 1 clock cycles , 3 Clock cycles

Machine Architecture ???

4. Which takes less space (Space to execute)

During program execution, we need some memory

To execute program, we allocate some memory as a process

Here we store

executable code

Global Variables / Static Variables

Heap and stack

Recursive programs are stored in stack

int fact(int n)

{

int i,j; 1

i=1; 2

for(j=2;j<=n;j++) 3

{

i=i\*j; 4

}

return i; 5

}

Find factorial of 10

Recursive

int(fact int n)

{

if(n==1) return 1; 1

else

return (n\*fact(n-1)); 2

}

Fact (10)

In case of recursive calls, size is dependent on N (space taken in stack is different and is affected by no. of inputs)

In case of non recursive calls, size is independent on N

Fibonacci Series

55

Recursive

int fib(int n)

{

if(n==0) return 0;

if(n==1) return 1;

return (fib((n-1) + f (n-2))

}

0 1 1 2 3 5 8 13

Fib(5)

Fib (5)

Fib(4) + fib(3)

Fib(3) + fib(2) fib(2) + fib(1)

Fib(2) + fib(1) fib(1) + fib(0) fib(1) + fib(0)

Fib(1) fib(0)

Fib(5) > 1

Fib(4) > 1

Fib(3) > 2

Fib(2) > 3

Fib(1) > 5

Fib(0) > 3

Fib(5) > Total Recursive Calls > 15

function calls made by fib(0) => 1

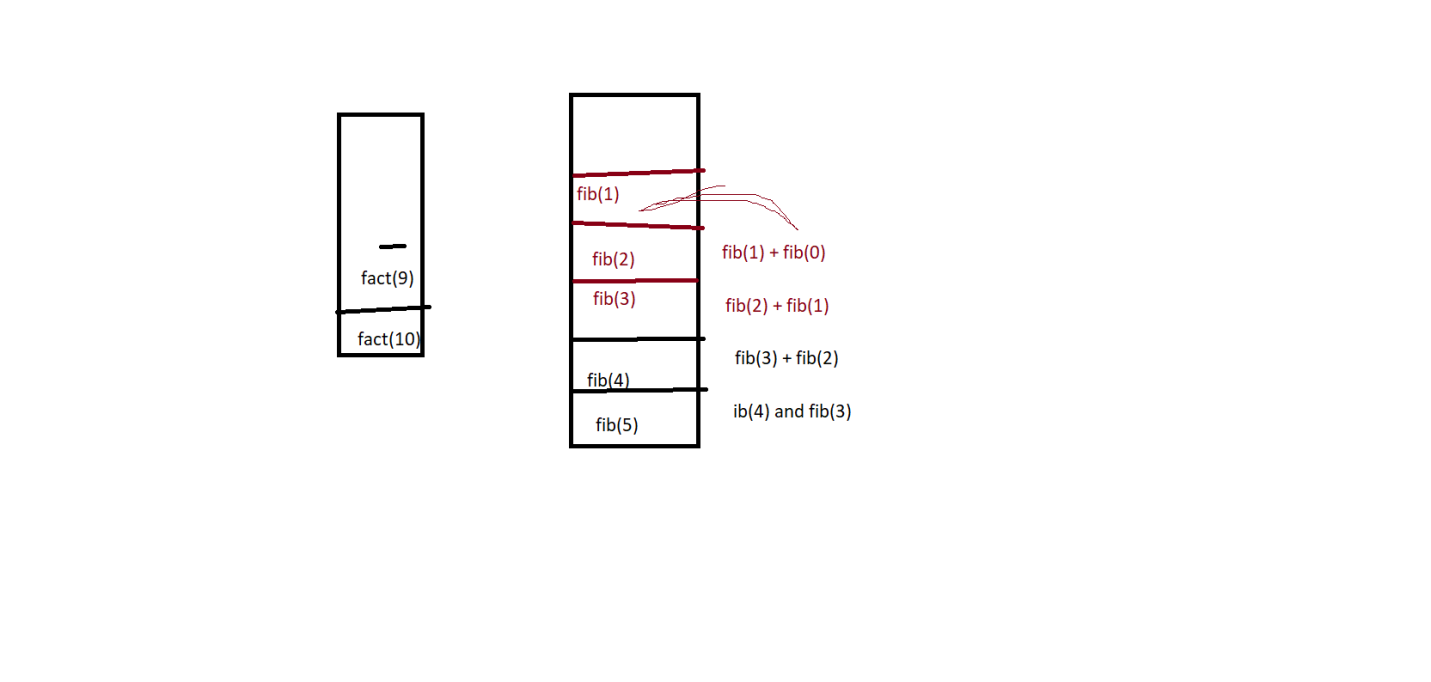
function calls fib(1) => 1

function calls fib(2) => 3

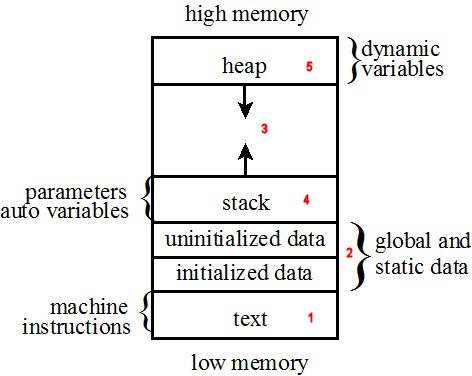
function calls fib(3) => 5

function calls fib(4) => 9

function calls fib(5) => 15



Recursive



1. The text area contains the program's machine instructions (i.e., the executable code).
2. [Global](http://icarus.cs.weber.edu/~dab/cs1410/textbook/1.Basics/definitions.html#extern) variables are defined in global scope outside of any function or object; [static variables](http://icarus.cs.weber.edu/~dab/cs1410/textbook/1.Basics/data.html#auto_static) have the keyword static included as part of their definition. The memory that holds global and static variables is typically allocated at program startup.
3. The space illustrated here was actually allocated but unused on a segmented-memory system, and provided space in to which the stack and heap could grow. On a paged-memory system this space is not actually allocated but signifies that there is space available for stack and heap growth.
4. The stack (sometimes called the runtime stack) contains all of the [automatic](http://icarus.cs.weber.edu/~dab/cs1410/textbook/1.Basics/data.html#auto_static) (i.e., non-static) variables.
5. Memory is allocated from and returned to the heap with with the new and delete operators respectively.

Time Complexity : Time taken /number of steps required to solve entire problem using some algorithm

1GB 1GB

ME Friend

(250 KB) Internet

(6GB) Physically go and bring

Time Taken to get File > (K1 + K2 + K3)

T(p) = C + tp(instance)

T(p) > Time complexity of program P

C > Compile Time

tp(instance) > Run Time

For(int i=0;i<n;++) 11 10 > n times

{

K=1i; 1 10

A=y; 1 k

I=y-1; 1

}

T(p) = (n+1) \* 3

3n + 3

10 \* 3 = 30

Int x = 10; k1

Int y=20;

For(int i=0;i<n;++) > n times

{

K=1i;

A=y; k2\*n

}

For(int j=0;j<n;j++)

{

I=y-1; k3\*n

}

K1 + k2n + k3n

N(k2+k3)

N(k4) = N

Algorithm : Sum(a[],n) 0

{ 0

S=0; 1

For(i=1;i<=n ;i++) n+1

{

S=s+a[i]; n

}

Return s; 1

} 0

Time = (2n+3)

If n = 5 2\*(5) + 3

If n=10 2\*(10) + 3

Time > n

Algorithm : Sum(a[], n m) 0

{ 0

S=0; 1

For(i=1;i<=n;++) n+1

{ for(j=1;j<=m;j++) n(m+1)

{

S= s+ a[i][j]; nm

}

}

Return s; 1

}

Time = 2n + 2nm + 3

Space Complexity

It is defined as the amount of space and memory taken by the algorithm to solve a problem

S(p) = C + sp(instance)

S(p) means Space complexity

C > Fixed Part

Sp(instance) > Variable part

We assume that every variable takes 1 byte

Algorithm : abc(a,b,c)

{

Return a+c+b + (a\*b\*c) + 4.0;

}

Space Complexity : 3

Algo : Sum(a[], n)

{

S=0;

For(i=1 ; i<n;++)

{

S= s+ a[i];

}

Return s;

}

A[] = n bytes  
n = 1

S=1;

I=1

S(p) = n+3

**Stack**, where automatic variables are **stored**, along with information that is saved each time a **function** is called. Each time a **function** is called, the address of where to return to and certain information about the caller's environment, such as some of the machine registers, are saved on the **stack**

All allocation made by malloc(), calloc() or realloc() are stored on the heap, while all **local variables** are stored on the stack. All global and **static variables** are stored in the data segment, while **constants** are stored in the code segment.

Big(O) tells us about the simple analysis of Algorithm efficiency how much it increases/decreases or how when input increases how its performance is

Complexity is termed in terms of input size , N

Machine independent

Follows basic computer steps

Measures worst case complexity

How to use

1. Ignore constants

Example 10 n > Here ignore 10 , Its O(n)

1. Certain terms dominates

O(n) < O(logn) < (n) <O(nlogn) < O(n2) < O(2n) <O(n!)

Ie. Ignore lower order terms as n increases.

O(n) has the smallest run time

Time complexity Analysis

1. Constant Time

X = 10 \* 5 , there is no input , O(1)

Print (x) > O(1)

1. Linear Time

F(x) = 2x+1, 0 – n

For(int x =1 x<=n;x++)

Print (2x+1) > O(1) ------- n \* O(1) = O(n)

Steps used to find Time complexity

1. Divide your program into fragments
2. Drop the non dominant terms
3. Break the code into fragments

Int i=10;

Int j = 10; --------> k1

For(int i=0;i<n;i++)

{

K=i;

K++;

------------🡪 k2 , k2\* n

Z=y+I;

}

Here , total time > K1 + K2 \* n

Drop K1

So we get

K2\*n > O(n)

Int i=10;

Int j = 10; --------> k1

For(int i=0;i<n;i++)

{

K=i;

K++;

------------🡪 k2 , k2\* n

Z=y+I;

}

For(int j=0;j<n;j++)

{

K=i;

K++;

------------🡪 k3 , k3\* n

Z=y+I;

}

Here , total time > K1 + (K2 \* n) + (K3 \* n)

Drop K1

So we get

(K2 + K3) \*n

n\*K4 > O(n)

Int i=10;

Int j = 10; --------> k1

For(int i=0;i<n;i++)

{

K=i;

For(iny j=0;j<10;j++)

{

K++;

------------🡪 k2 , k2\* n

Z=y+I;

}

}

Here , total time > K1 + K2

When I =0 , j = 0 ….n (n)

When I =1 , j = 0 -------------n

When I =2 , j = 0 --------------------n

When I =n , j = 0 ------------------------n

I 0 1 2 3 4 ---- n

J n n n n n ------n

So , we have (n + n + n + n + n-1)

n(1 + --- (n-1))

n \* n

n2

k1 + k2 \* n2

Drop K1 and K2

So we get n2

> O(n2)

// Find the time Complexity of the following program

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Codility\_Day1

{

class Program

{

static void Main(string[] args)

{

int[] arr = new int[] { 3, 5, 66 };

func1(arr, 2);

}

static void func1(int[] arr, int length)

{

int sum = 0; k1

int product = 1;

for(int i=0;i<length;i++)

{ k2 \* n

sum += arr[i];

}

for (int i = 0; i < length; i++)

{ k3 \* n

product \*= arr[i];

}

}

}

}

K1 + k2\*n + k3\*n

Drop K1

K2\* n + K3\* n

(k2+ K3) \* n

K4 n

(O) n

Consider the recursive algorithm where the random (int n) spends one unit of time to return a random integer which is evenly distributed within the range [0,n]. If the average processing time is T(n), what is the value of T(6)?

Int(function(int n)

{

Int I; -> K1=0

If(n<=0)

Return 0;

}

Else

{

I=random(n-1); **> 1**

Print(“\n”);

Return function(i) + function(n-1-i);

}

}

Which of the following are equivalent to O(N)

1. O(N+P) where P<N/9
2. O(pN-k)
3. O(N+8logN)
4. O(N+M2)

Here P is Nondominant Term

O(N)

