PROBLEM SOLVING AND PROGRAMMING

Functions – Recursion

Recursion

- Recursion - A function calling itself with a different set of argument values.
- Every recursive function definition must contain statements to check whether to terminate the call or to call the function once again.
- Consider the factorial function:

  n! is defined as n * (n – 1)!.

  If the value of (n – 1)! is known,

  then n! can be computed as n*(n – 1)!

  So 6! = 6 * 5!,

  5! = 5 * 4!, 4! = 4 * 3!, 3! = 3 * 2!

  2! = 2 * 1!, 1! = 1 * 0!, 0! = 1

Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$.

$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$

Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + {}_{fn\ -2}$ for $n \geq 2$.

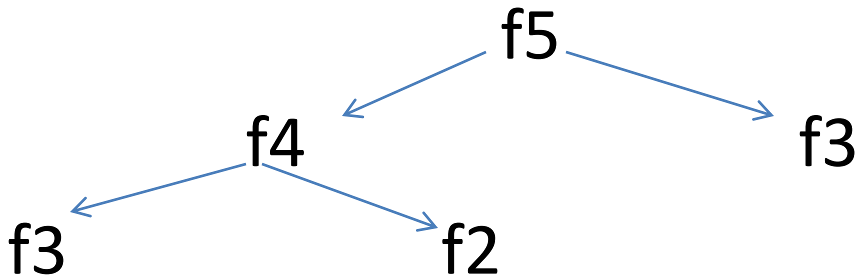
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$

$$
\begin{array}{ccc}
 & f5 & \\
\swarrow & & \searrow \\
f4 & & f3
\end{array}
$$

Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + {}_{fn\ -2}$ for $n \geq 2$.

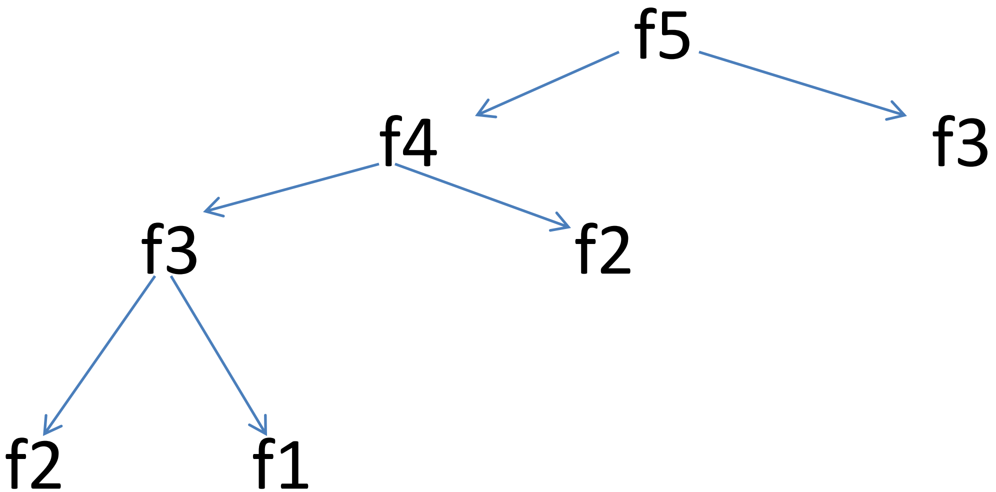
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + {}_{fn\ -2}$ for $n \geq 2$.

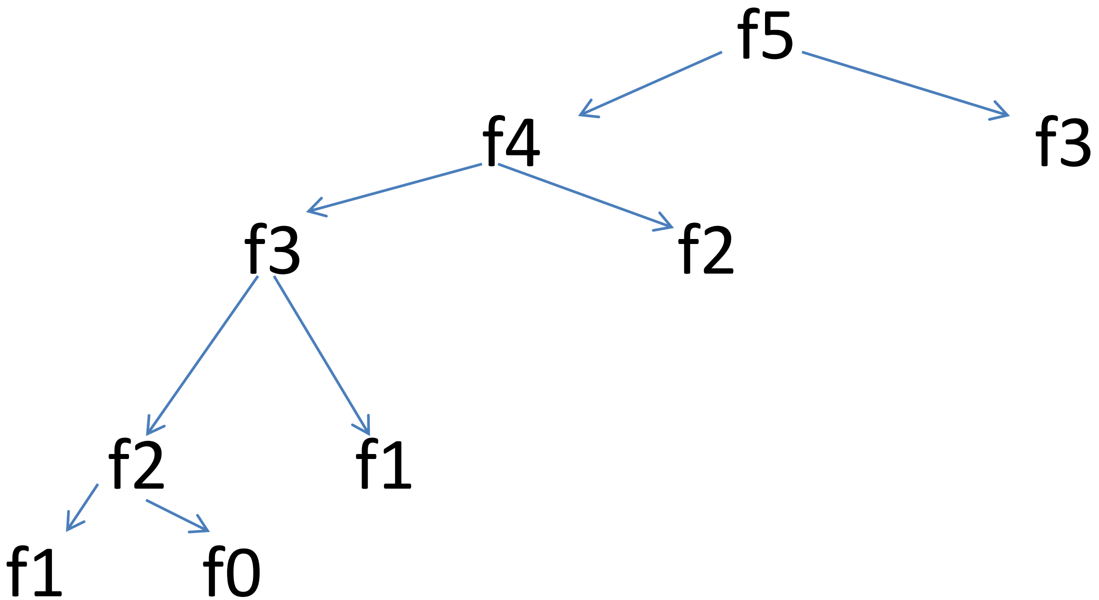
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + {}_{fn\ -2}$ for $n \geq 2$.

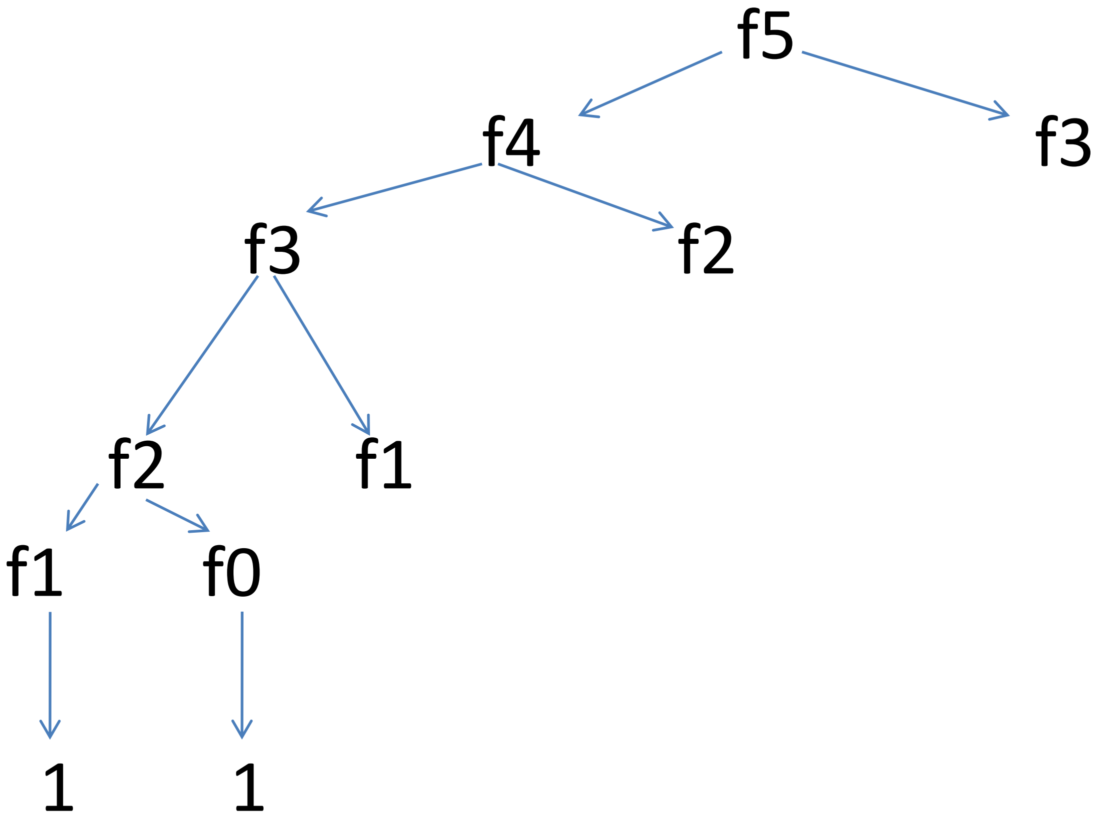
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + {}_{fn\,-2}$ for $n \geq 2$.

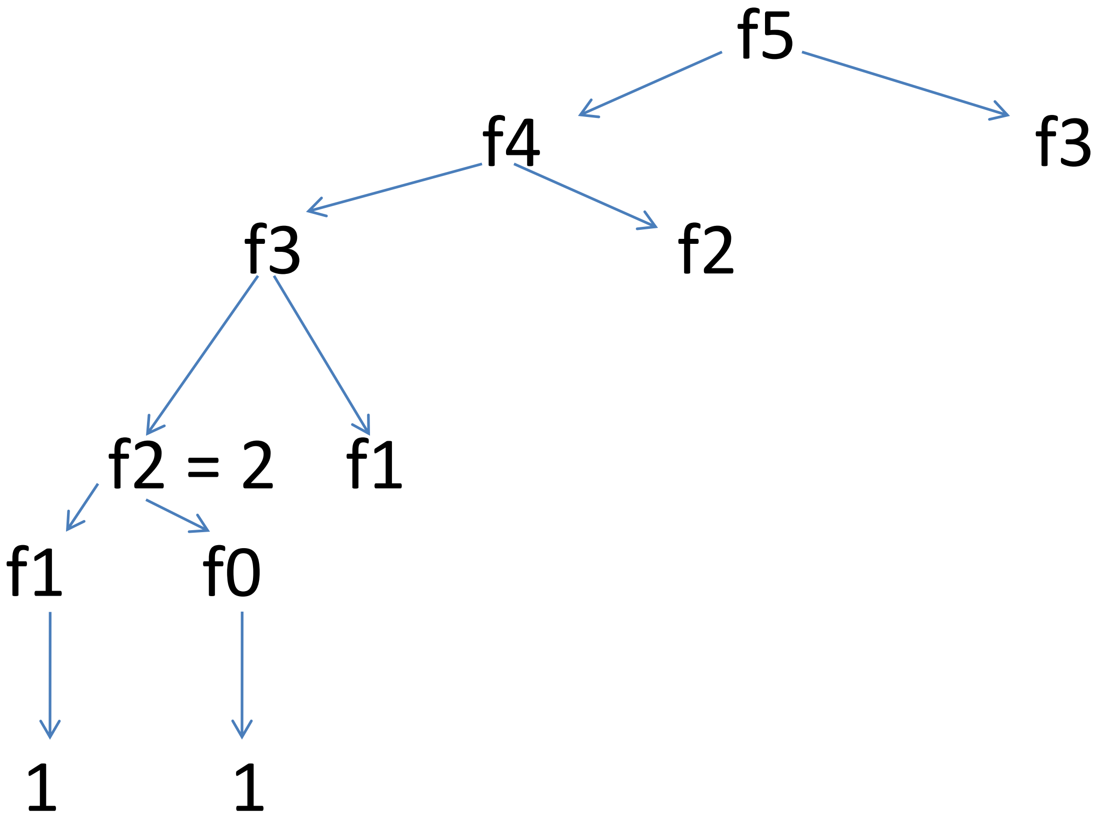
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + {}_{fn\ -2}$ for $n \geq 2$.

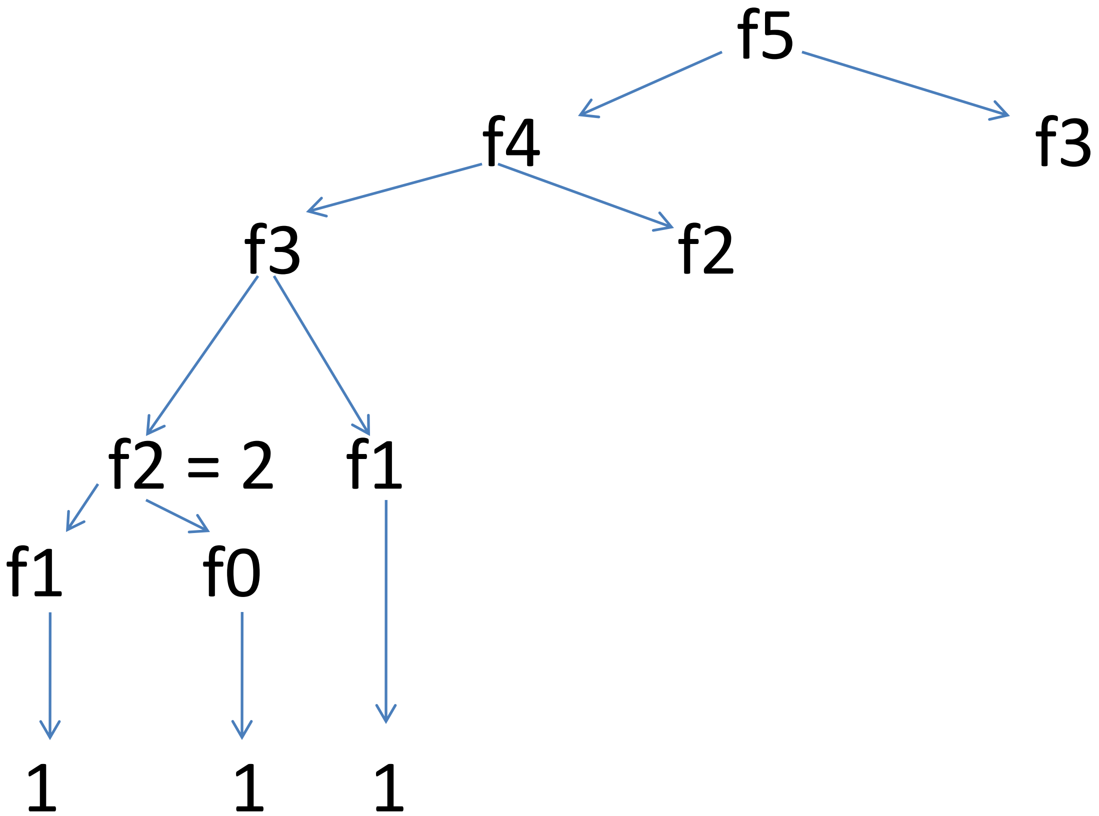
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + {}_{fn\ -2}$ for $n \geq 2$.

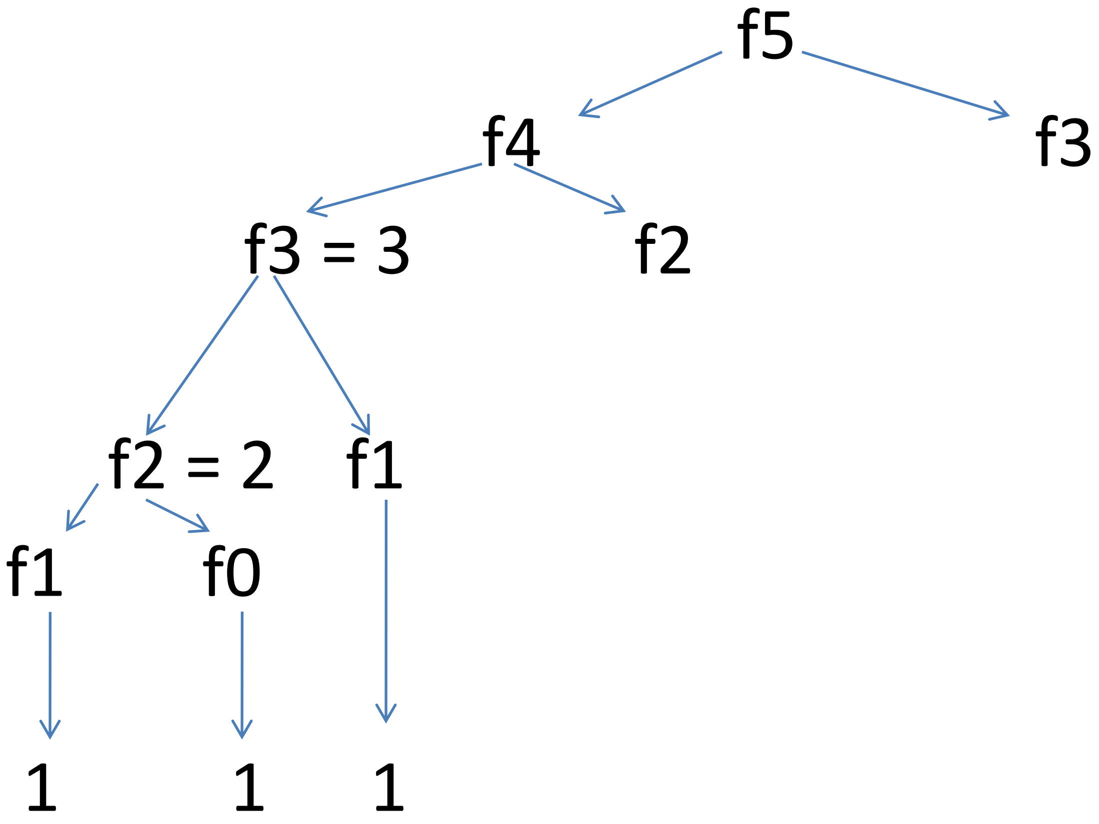
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + {}_{fn\ -2}$ for $n \geq 2$.

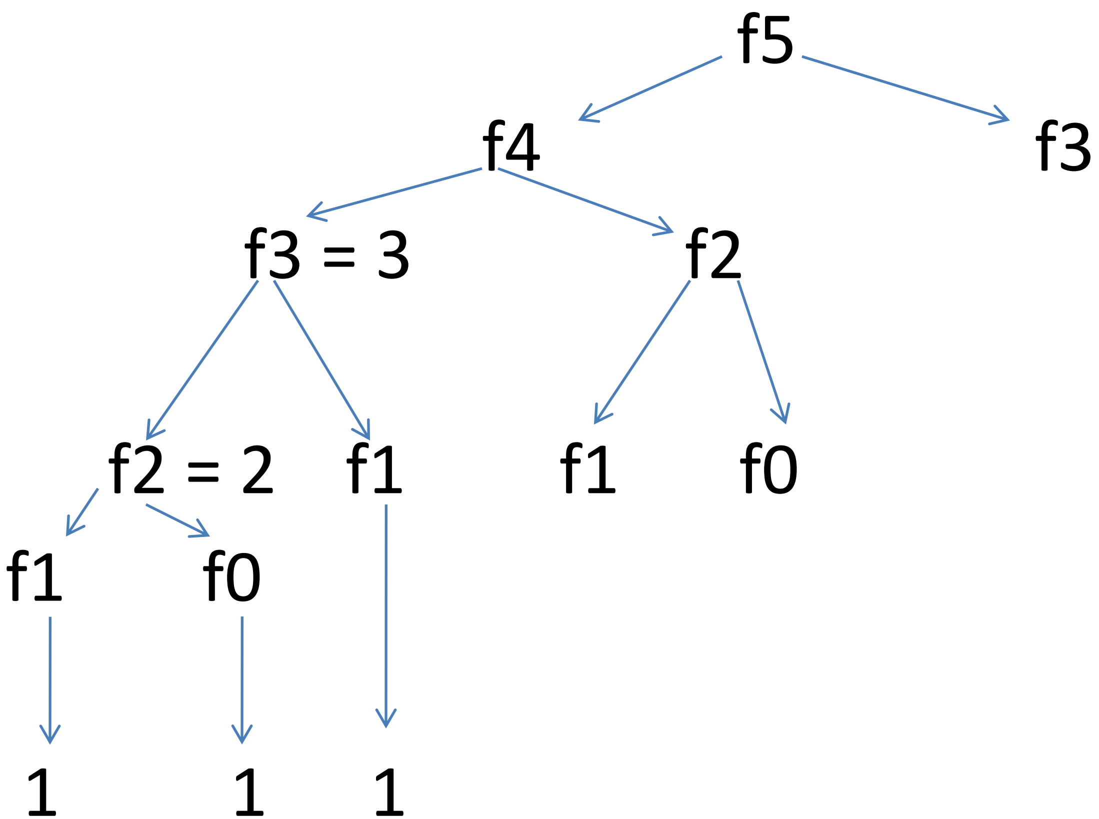
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + {}_{fn\ -2}$ for $n \geq 2$.

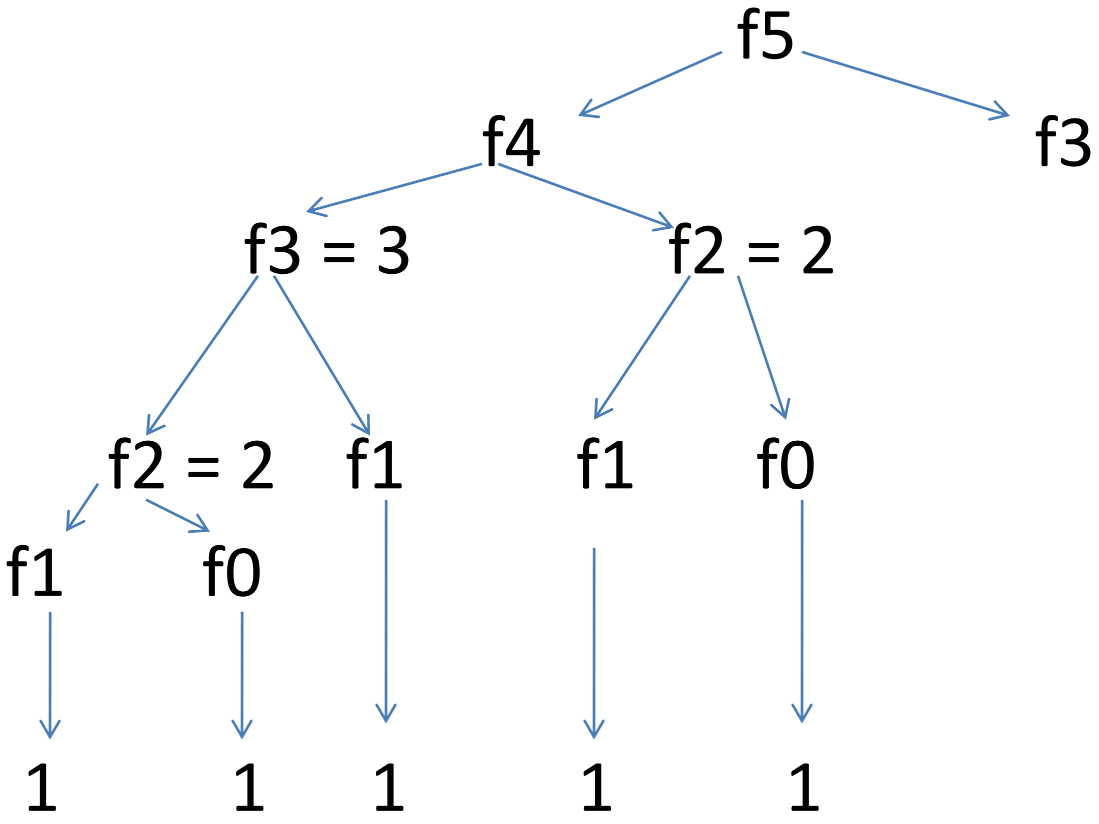
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$.

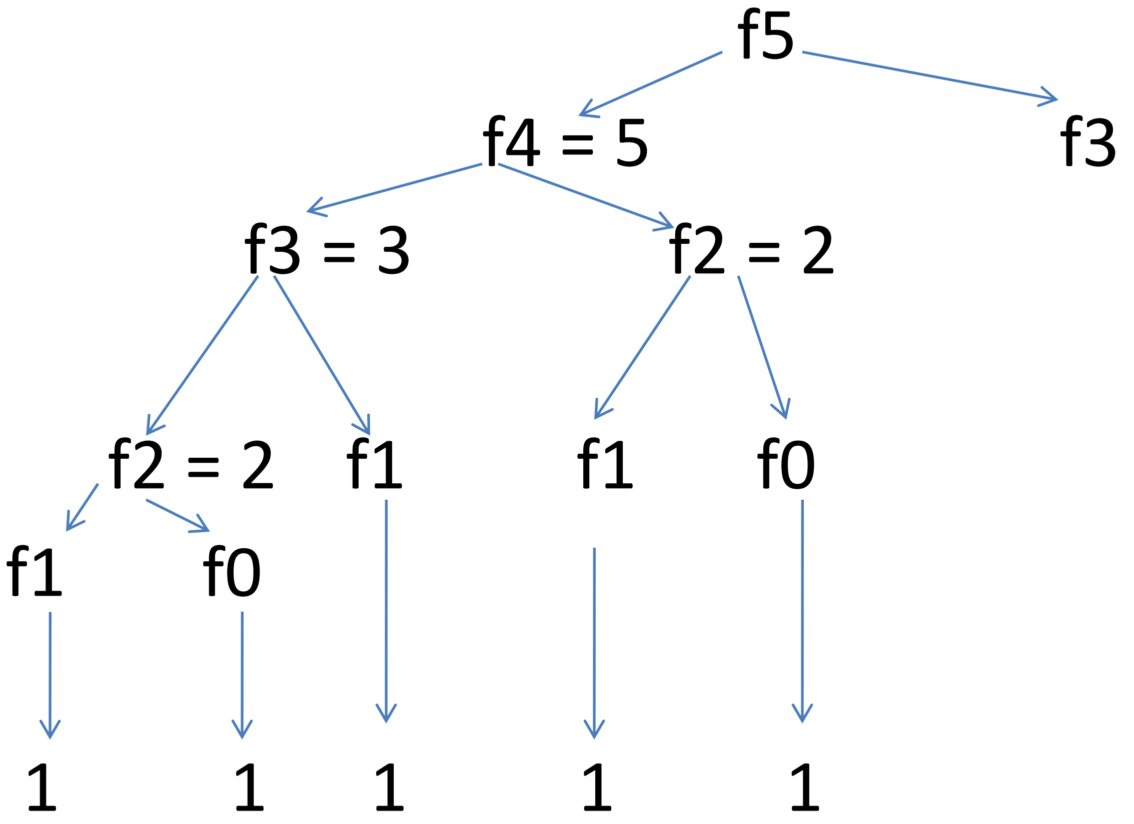
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$.

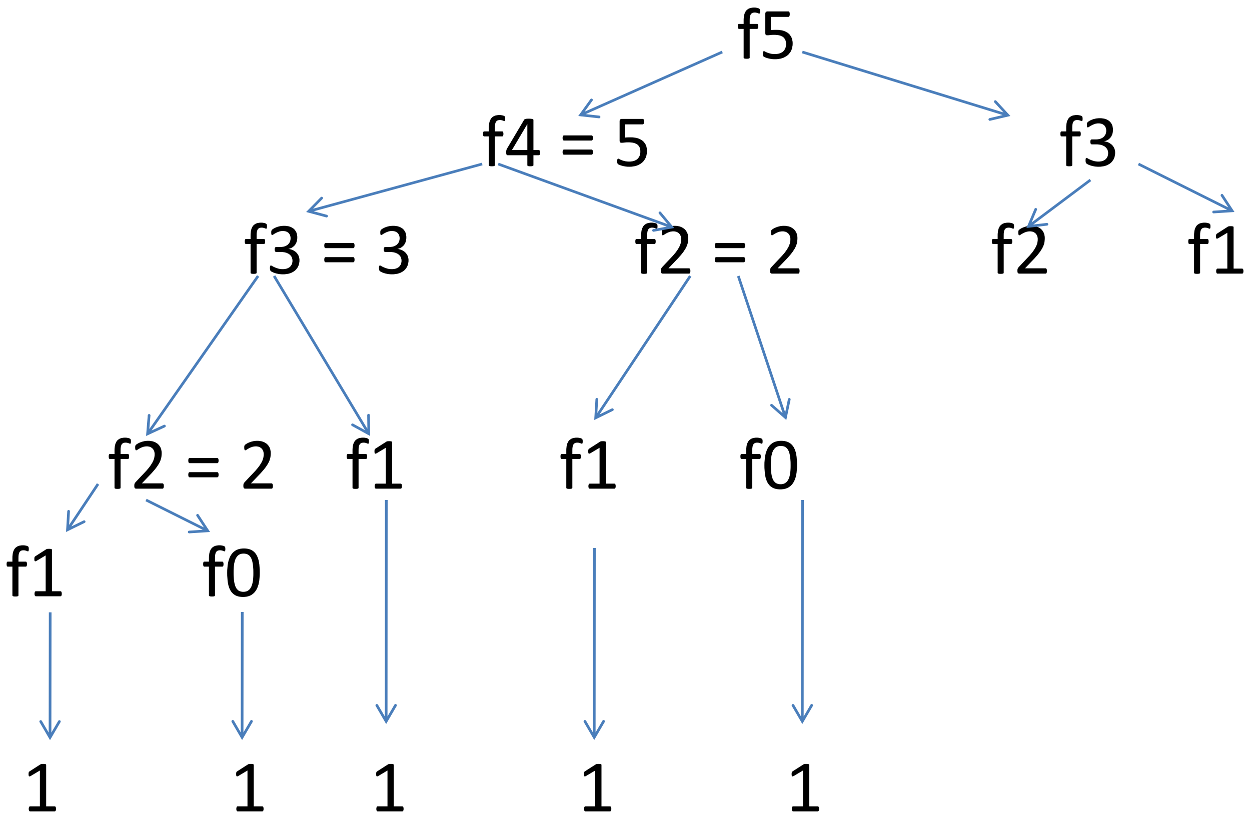
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$.

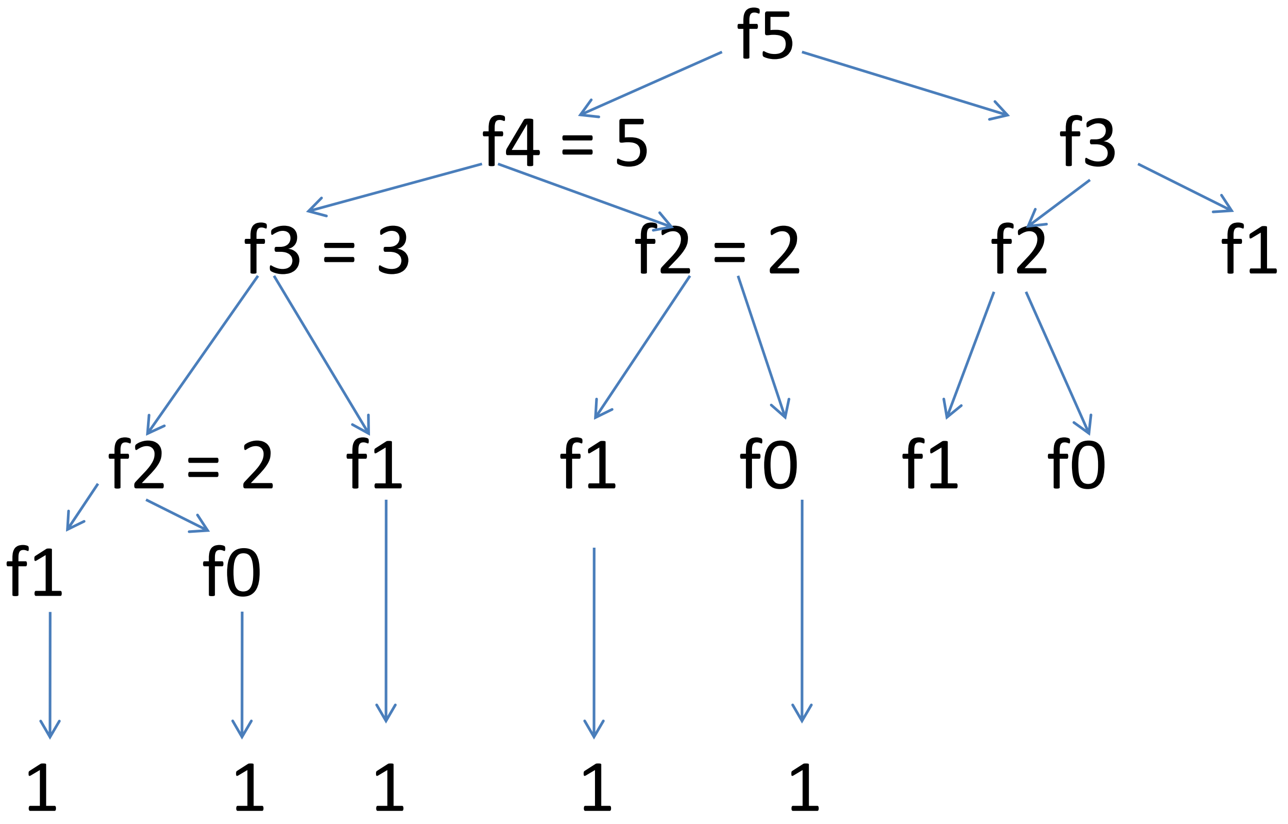
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + {}_{fn\ -2}$ for $n \geq 2$.

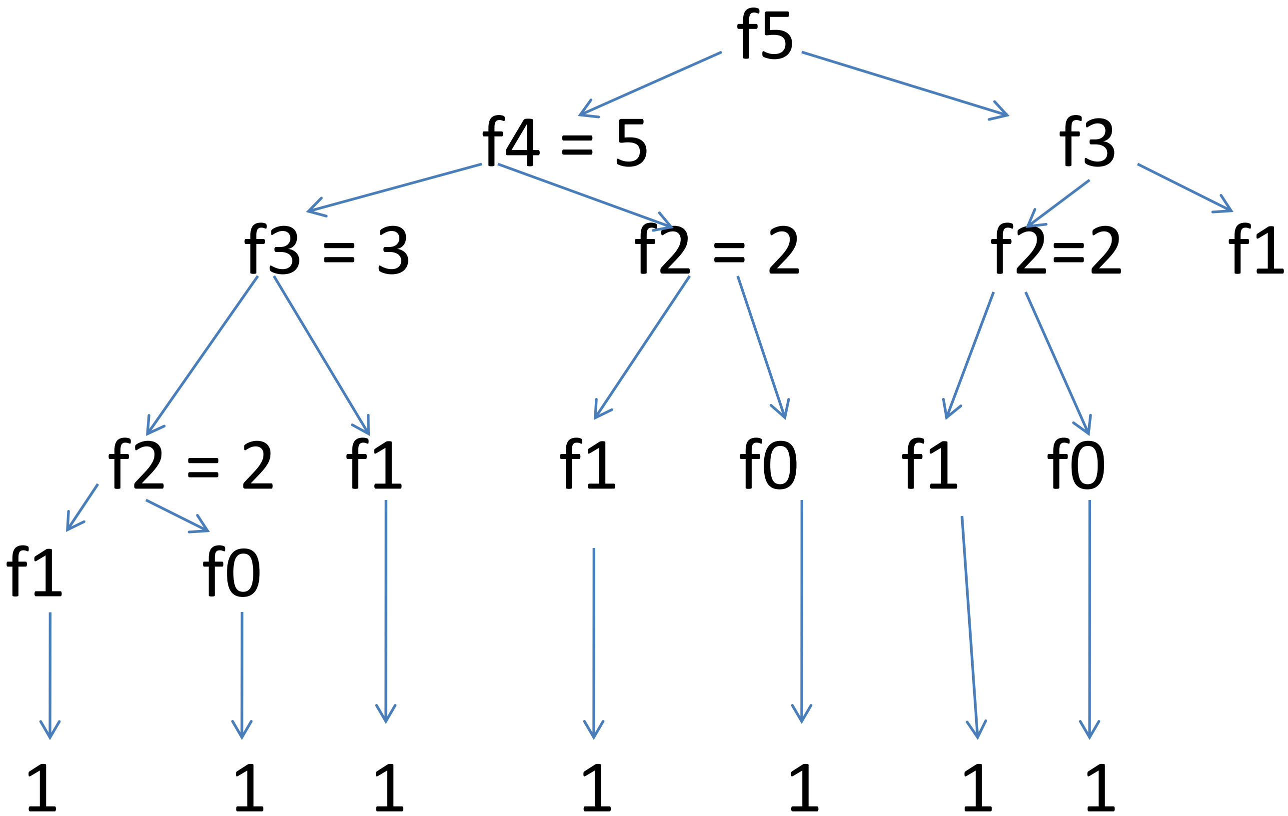
$F_5 = f_4 + f_3,\ f_4 = f_3 + f_2,\ f_3 = f_2 + f_1,\ f_2 = f_1 + f_0\ ,\ f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + {}_{fn\ -2}$ for $n \geq 2$.

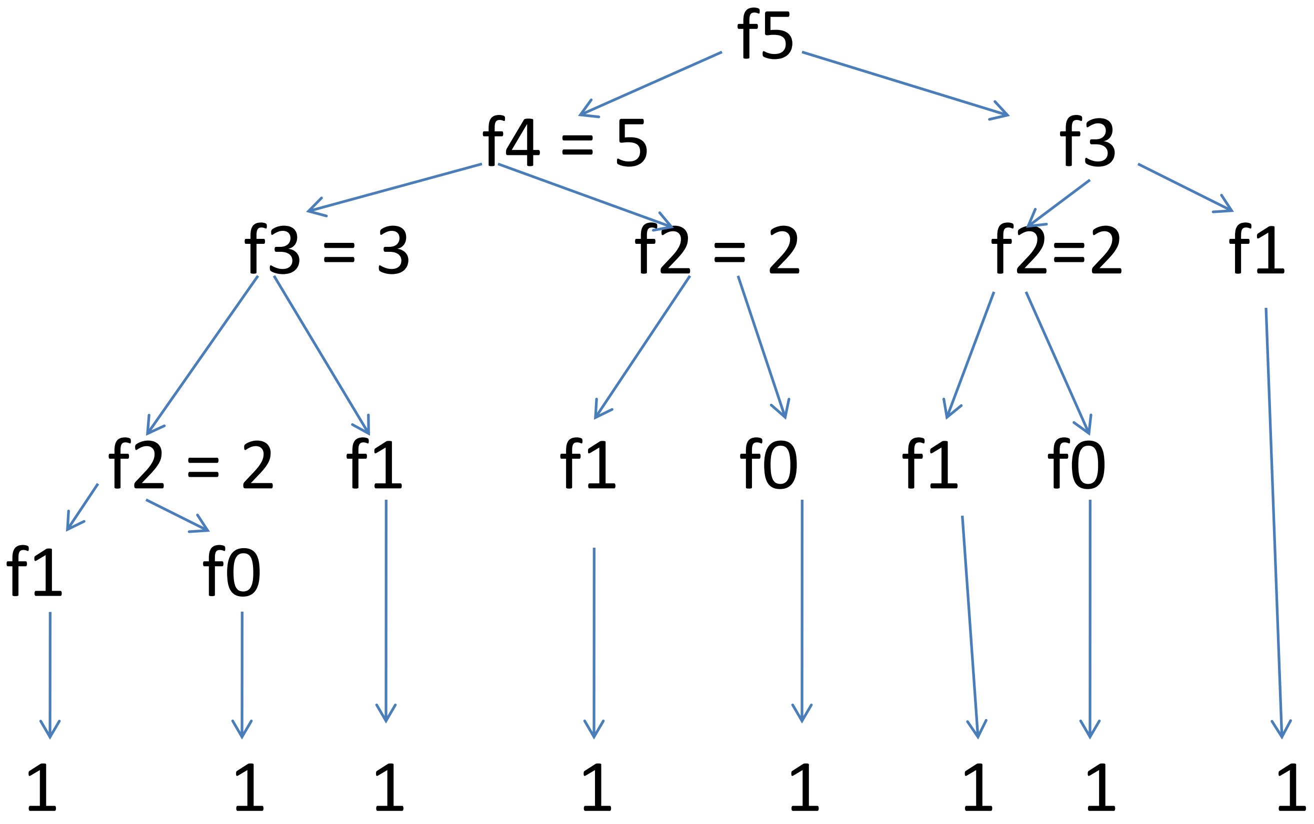
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + {}_{fn\ -2}$ for $n \geq 2$.

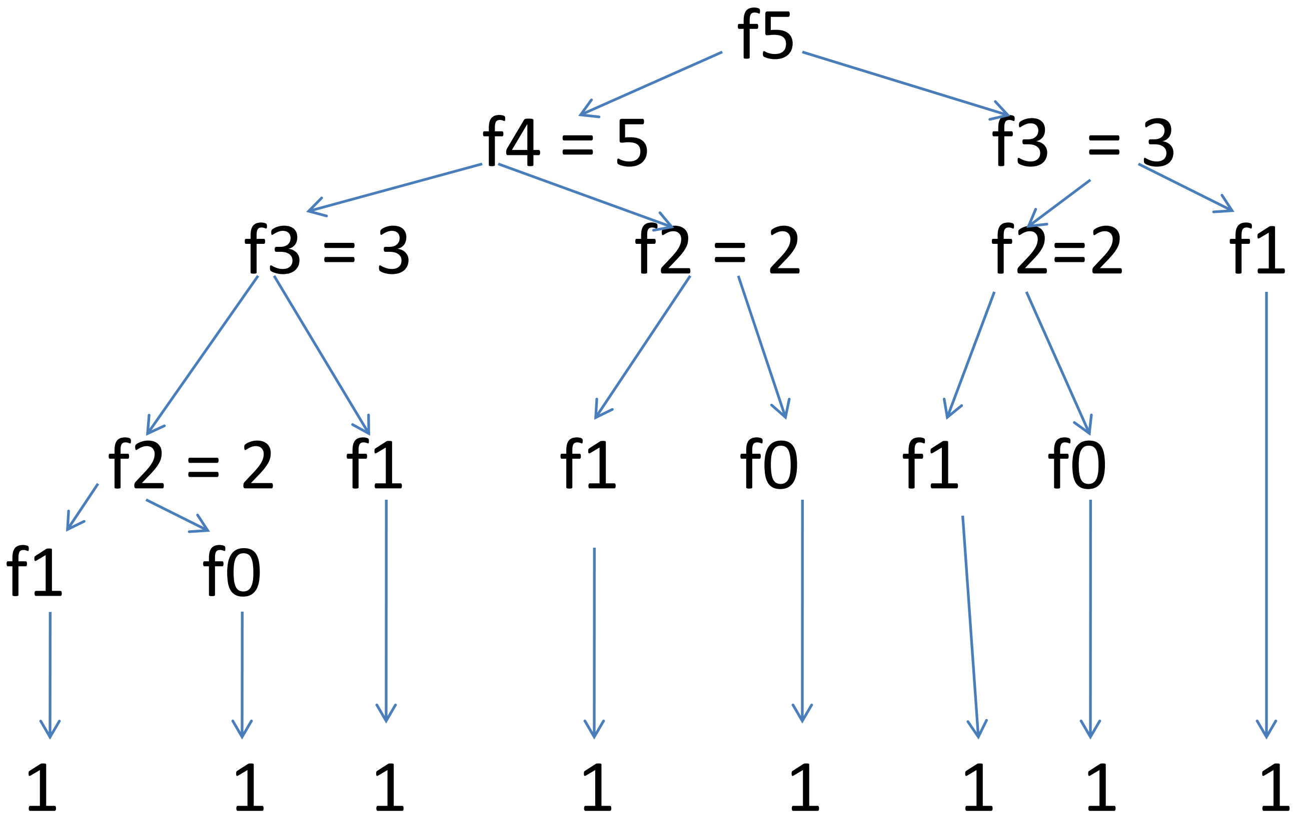
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + {}_{fn\ -2}$ for $n \geq 2$.

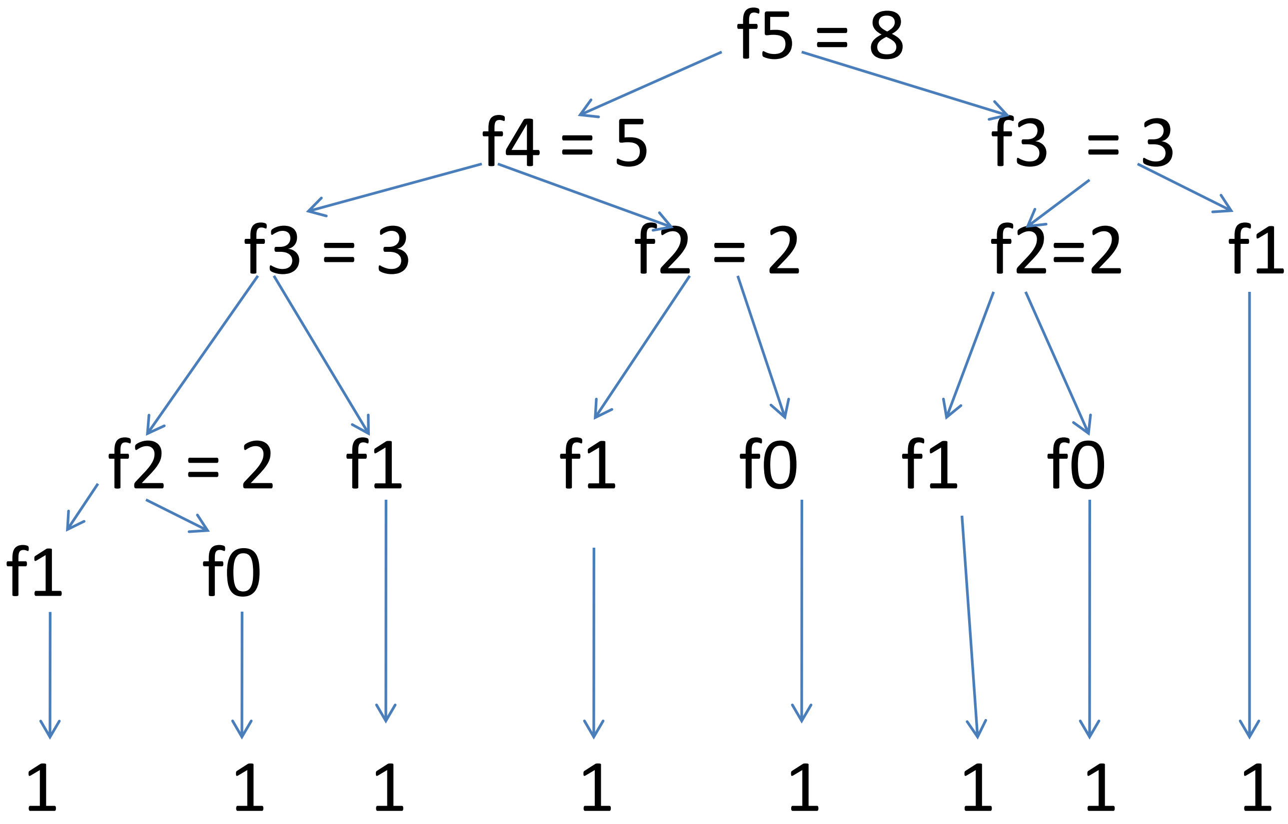
$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



Recursion – nth Fibonacci Number

Fibonacci number $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$.

$F_5 = f_4 + f_3$, $f_4 = f_3 + f_2$, $f_3 = f_2 + f_1$, $f_2 = f_1 + f_0$, $f_1 = f_0 = 1$



In order to compute n^{th} Fibonacci number, need to find f_{n-1} and f_{n-2} and the result is sum of these two.

f5 = f4 + f3

= (f3 + f2) + (f2 + f1)

= ((f2 + f1)) + (f1 + f0)) + ((f1 + f0) + f1)

= (((f1 + f0) + f1)) + (f1 + f0)) + ((f1 + f0) + f1)

= 8

Number of times the function f is to be called is ??

Recursion – nth Fibonacci Number

Recursive function to compute n^{th} Fibonacci number

```
int f(int n)
{
    if (n == 0 || n == 1) return 1; // termination
    else return (f( n – 1) + f(n – 2));  // Recursion
}
```

Number of times the function f is to be called is ??

$f_{n+1} + 1$

Factorial

```
#include <iostream>
using namespace std; //Factorial function
int f(int n){ /* This is called the base condition, it is * very important to
specify the base condition * in recursion, otherwise your program will
throw * stack overflow error. */
if (n <= 1)
    return 1;
 else
    return n*f(n-1);
}

int main()
{
int num;
  cout<<"Enter a number:;
   cin>>num;
   cout<<"Factorial of entered number: "<<f(num); return 0;
}
```

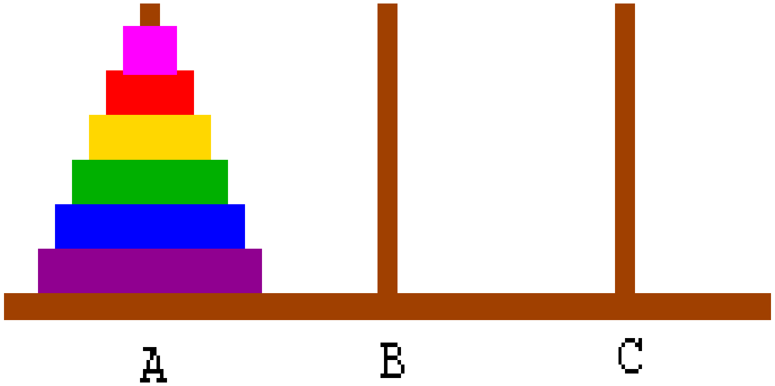
Recursion - Example

```
#include<iostream>
using namespace std;
Int fact(int); // function prototype
Int main()
{ int n, f;
  cout << "\n Enter a number: " ;
  cin >> n;
  f = fact (n) // function call
  cout << " The factorial of " << n << " is " << f;
  return 0;
}

int fact(int k)   // function fact definition
{  if (k == 0 || k == 1)   // base case
        return 1;
   else  return k * fact( k - 1 );  // recursive function call
}
```

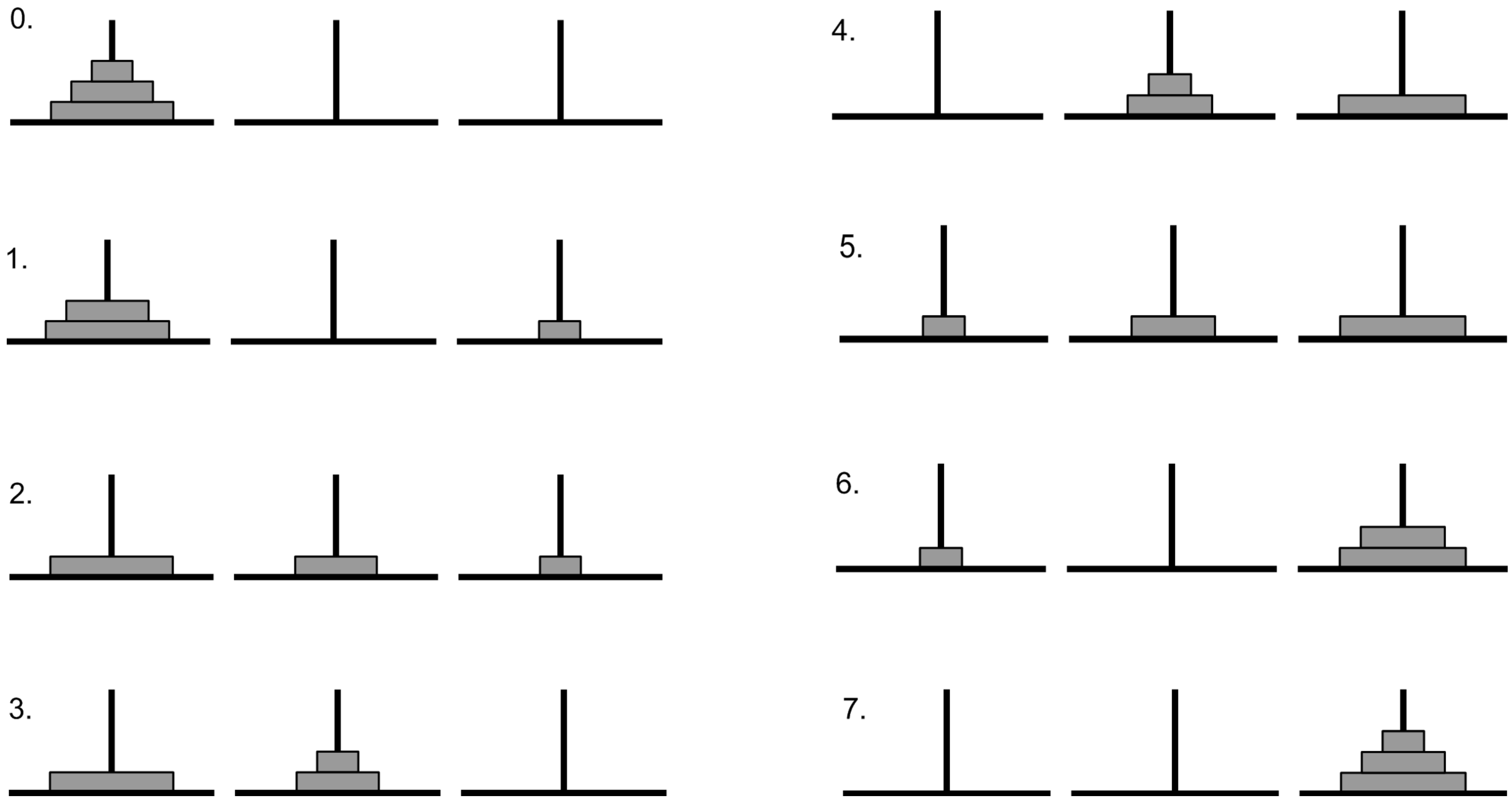
Tower of Hanoi

Problem : is there are 3 towers are given and one tower contains stack of disc in decreasing order from bottom to up. We need to move this disks from tower A to tower C.

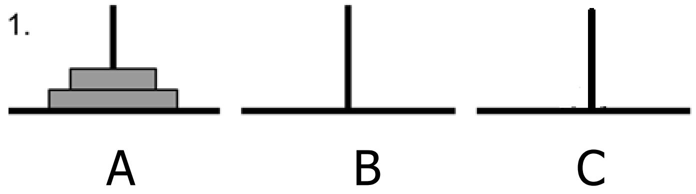


Conditions :

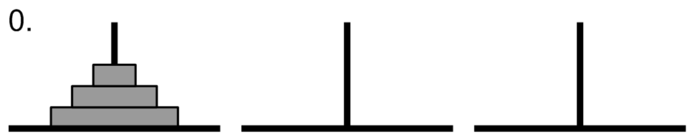
- Only one disc can be transfer at a time.
- Each move consists of taking the upper disc from one of the tower and placing it on the top of another tower i.e. a disc can only be moved if it is the uppermost disc of the tower.
- Never a larger disc is placed on a smaller disc during the transfer.



0.
1.
2.
3.
4.
5.
6.
7.



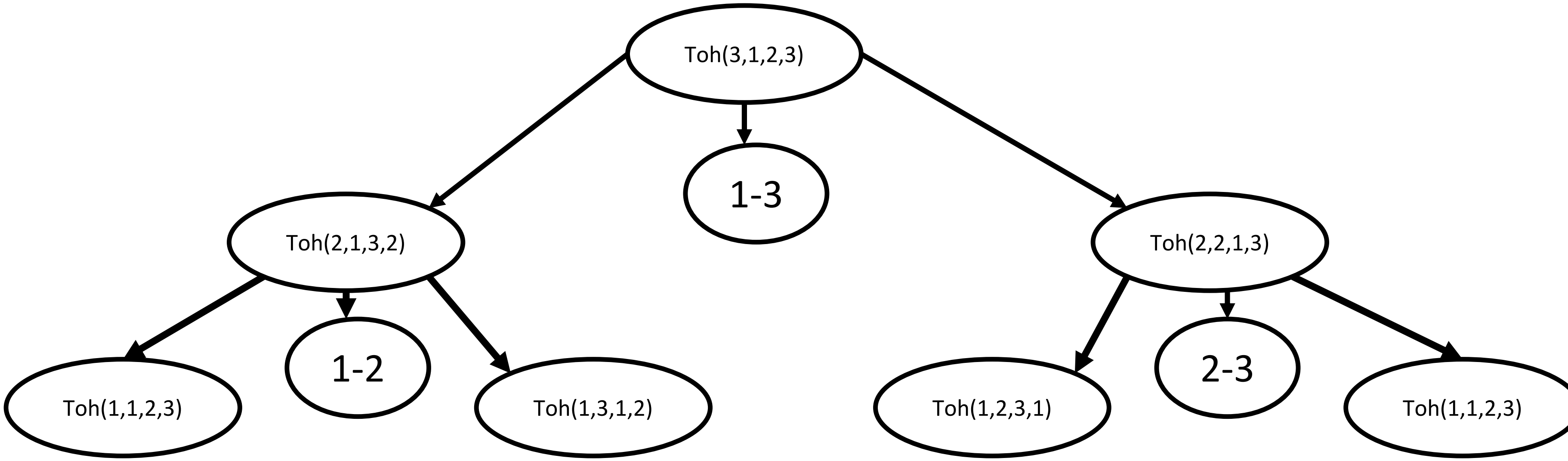
- Move disc from A to B using C
- Move disc from A to C
- Move disc from B to C using A



- Move 2 discs from A to B using C (recursive)
- Move disc from A to C
- Move 2 discs from B to C using A (recursive)

- Move n-1 discs from A to B using C (recursive)
- Move disc from A to C
- Move n-1 discs from B to C using A (recursive)

```
#include <iostream.h>
#include <conio.h>
void tower(int n, int A, int B,int C)
{
if(n==1)
        {
                cout<<"\t\tMove disc 1 from "<<A<<" to "<<C<<"\n";
        return;
        }
 else
        {
                tower(n-1,A,C,B);
                cout<<"\t\tMove disc "<<n<<" n "<<A<<" to "<<C<<"\n";
                tower(n-1,B,A,C);
        }
}
void main()
{
int n;
cout<<"\n\t\t*****Tower of Hanoi*****\n";
cout<<"\t\tEnter number of discs : ";
cin>>n;
tower(n,‘X',‘Y',‘Z'); getch(); }
```



1-3 1-2 3-2 1-3 2-1 2-3 1-3

```
int main() {
    ... ..                  3
    result = sum(number);
    ... ..
}
            3
int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}
            2
int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}
            1
int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}
            0
int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}
```

Displaying the digits of a Positive integer

- Consider a problem to display the digits of a positive integer such that each digits is separated by two spaces. If 4589 is the input, then the display should be 4 5 8 9.
- Algorithm – Assume that a function display_digits(int n) displays the digits of integer n.
- To display the digits of n, somehow display the digits of n/10 and then display the digit n%10.

Displaying the digits of a Positive integer

```
void display_digits(int n)
{
int d;
   if(n > 9)
      {d = n % 10;  display_digits ( n / 10);  cout << setw(4) << d;
    }
   else cout << setw(4) << d;
}
Int main()
{ int n;
   cout << " Input the number to be displayed:"; cin >> n;
   cout << " The number " << n << " is displayed as " << endl;
    display_digits (n);
}
```

Decimal to Binary Conversion

```
#include<iostream>
using namespace std;
void Dec_to_Binary(int); // function prototype
int main()
{ int n;
  cout << “\n Enter a number: “ ;
  cin >> n;
  cout<<"\n The equvalent binary of " << n << " is : ";
  Dec_to_Binary(n); // function call
  return 0;
}
void  Dec_to_Binary(int k)   // function Dec_to_Binary definition
{  if (k == 0 || k == 1)   // base case
         cout << k;
   else
      {    Dec_to_Binary (k/2);  // recursive function call
           cout << k % 2;
      }
}
```

Decimal to Binary Conversion

```
#include<iostream>
using namespace std;
void Dec_to_Binary(int); // function prototype
int main()
{ int n;
  cout << "\n Enter a number: " ;
  cin >> n;
  cout<<"\n The equvalent binary of " << n << " is : ";
  // Assume that the n value entered is 5
  Dec_to_Binary(n); // function call
  return 0;
}
void  Dec_to_Binary(int k)   // function Dec_to_Binary definition
{  if (k == 0 || k == 1)   // base case
        cout << k;
   else
     {    Dec_to_Binary (k/2);  // recursive function call
          cout << k % 2;
     }
}
```

Decimal to Binary Conversion

```
#include<iostream>
using namespace std;
void Dec_to_Binary(int); // function prototype
int main()
{ int n;
  cout << “\n Enter a number: “ ;
  cin >> n;
  cout<<"\n The equvalent binary of " << n << " is : ";
  // Assume that the n value entered is 5
  Dec_to_Binary(n); // function call
  // Dec_to_Binary (5)
  return 0;
}
void  Dec_to_Binary(int k)   // function Dec_to_Binary definition
{  if (k == 0 || k == 1)   // base case
         cout << k;
   else
      {    Dec_to_Binary (k/2);  // recursive function call
           cout << k % 2;
      }
}
```

Decimal to Binary Conversion

```
void  Dec_to_Binary(int k)   // k value is 5
 {  if (k == 0 || k == 1) cout << k;   // base case  fails
     else
      {   Dec_to_Binary (k/2);  //recursive function call – Dec_to_Binary(2)


void  Dec_to_Binary(int k)   // k value is 2
 {  if (k == 0 || k == 1) cout << k;   // base case  fails
     else
      {   Dec_to_Binary (k/2);  //recursive function call – Dec_to_Binary(1)


void  Dec_to_Binary(int k)   // k value is 1
 {
    if (k == 0 || k == 1) cout << k;   // base case is true  it prints 1

}
```

Decimal to Binary Conversion

```
void  Dec_to_Binary(int k)   // k value is 5
 {  if (k == 0 || k == 1) cout << k;   // base case  fails
     else
       {   Dec_to_Binary (k/2);  //recursive function call – Dec_to_Binary(2)


void  Dec_to_Binary(int k)   // k value is 2
 {  if (k == 0 || k == 1) cout << k;   // base case  fails
     else
      {   Dec_to_Binary (k/2);  //recursive function call – Dec_to_Binary(1)


void  Dec_to_Binary(int k)   // k value is 1
 {
    if (k == 0 || k == 1) cout << k;   // base case is true  it prints 1
 }
```

Decimal to Binary Conversion

```
void  Dec_to_Binary(int k)   // k value is 5
 {  if (k == 0 || k == 1) cout << k;   // base case  fails
    else
     {   Dec_to_Binary (k/2);  //recursive function call – Dec_to_Binary(2)


void  Dec_to_Binary(int k)   // k value is 2
 {  if (k == 0 || k == 1) cout << k;   // base case  fails
    else
    {    Dec_to_Binary (k/2);  //recursive function call – Dec_to_Binary(1)
         cout << k % 2;   // It prints 0 , observe that k value in this function call is 2
```

Decimal to Binary Conversion

```
void  Dec_to_Binary(int k)   // k value is 5
 {  if (k == 0 || k == 1) cout << k;   // base case  fails
     else
    {    Dec_to_Binary (k/2);  //recursive function call – Dec_to_Binary(2)


void  Dec_to_Binary(int k)   // k value is 2
 {  if (k == 0 || k == 1) cout << k;   // base case  fails
     else
     {    Dec_to_Binary (k/2);  //recursive function call – Dec_to_Binary(1)
          cout << k % 2;   // It prints 0 , observe that k value in this function call is 2
     }
}
```

Decimal to Binary Conversion

```
void  Dec_to_Binary(int k)   // k value is 5
 {  if (k == 0 || k == 1) cout << k;   // base case  fails
     else
    {     Dec_to_Binary (k/2);  //recursive function call – Dec_to_Binary(2)
          cout << k%2;   // // It prints 1 , observe that k value in this function call is 5
    }
}
```

Note – The order of output is   -   101