

More Recursion Examples

The well known Fibonacci function

$$\begin{aligned} &= 0 && \text{if } n = 0, \\ f(n) &= 1 && \text{if } n = 1, \\ &= f(n-1) + f(n-2) && \text{if } n \geq 2 \end{aligned}$$

- Similarly, many other well known functions can be defined *recursively*, i.e., in terms of itself

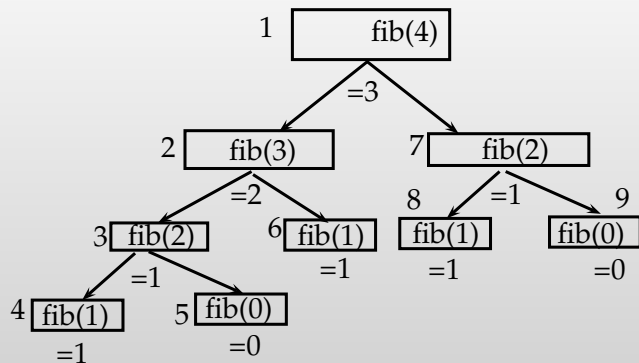
```
int fib ( int n )
{
    if (n == 0) return (0);
    if (n == 1) return (1);
    return (fib(n-1)+fib(n-2));
}
```

Why does this work

```
int fib ( int n )
{
    if (n == 0) return (0);
    if (n == 1) return (1);
    return (fib(n-1)+fib(n-2));
}
```

- Each call instance of fib works with its own copy of n
- The computer “remembers” every previous state of the problem. This information is “held” by the computer on the “activation stack” (i.e., inside of each function’s workspace).

The recursive function call sequence



- The call sequence: 1,2,3,...
- The return sequence: 4,5,3,6,2,8,9,7,1

How about permutations?

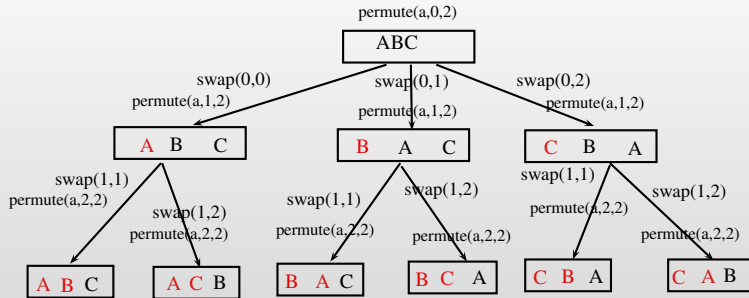
Note that, $\text{permutation}(abc) \rightarrow$

- $a + \text{permutation}(bc) \rightarrow abc, acb$
- $b + \text{permutation}(ac) \rightarrow bac, bca$
- $c + \text{permutation}(ab) \rightarrow cab, cba$

Recursive function for printing all permutations of a given string

```
void permute(char a[], int i, int n)
{ // i=current start index
    int j;
    if (i == n) printf("%s\n", a);
    else{
        for (j = i; j <= n; j++){
            swap(a[i], a[j]);
            permute(a, i+1, n);
            swap(a[i], a[j]); //backtrack
        }
    }
}
```

The recursive function call sequence

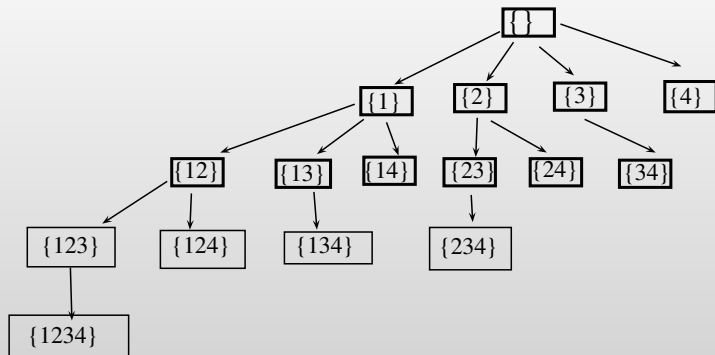


- Before each function call, position of a letter is fixed (marked in red) after swapping
- After any call returns, swapping is again performed to restore state
- Printing is done at leaf level

Another Problem

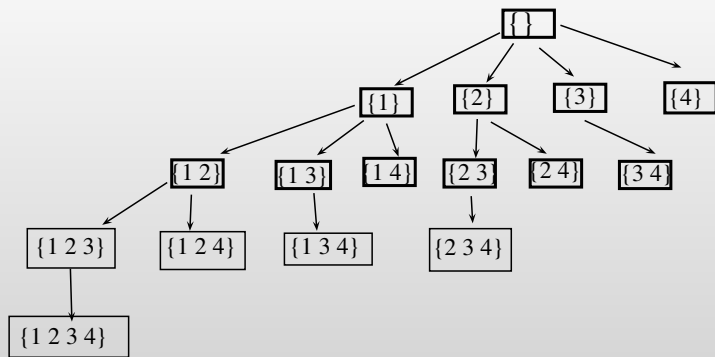
Write a recursive function which takes as argument an integer n and prints all possible subsets of the set $\{1, 2, 3, \dots, n\}$.

Assignment 10: recursion tree, n=4



- With each call, a possible subset is printed
- The printing is done w.r.t. natural ordering of 1,2,3, etc
- 1, 12, 123, 1234, 124, 13, 14, 2, 23,

What exactly is happening



- At recursion depth k , you print all subsets of size k
- Let us say, you printed $\{2\}$ at depth 1. Next make possible calls for printing subsets of size 2 which *occur after* $\{2\}$ i.e., $\{2,3\}$ and $\{2,4\}$.

A sample solution

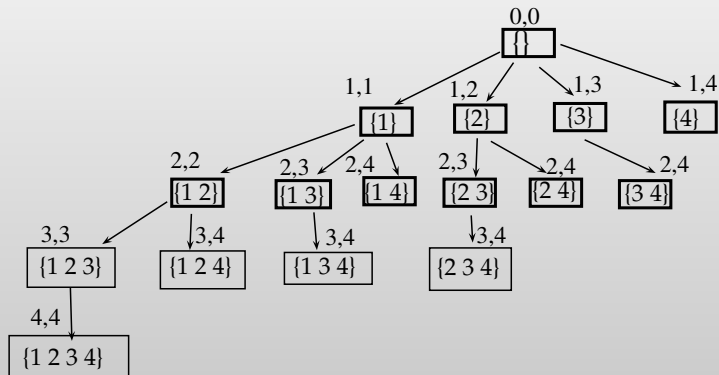
```
int main ()
{
    int A[MAX_SIZE];
    int n;
    printf("Enter n:");
    scanf("%d", &n);
    subset(A,0,0,n); // the recursive call
    return 0;
}
```

- prints the empty string and initiates subsequent calls

```
void subset (int A[],int k,int j,int n)
{ //k is recursion dept
  //j is last value written by previous call
  int i,l;
  printf("{");
  for(i=0;i<k;i++)
    printf("%d ",A[i]);
  printf("}\n");
  for (i=j+1;i<=n;++i)
  {
    A[k]=i;
    subset(A, k+1, i, n);
  }
}
```

- print current content
- populate array before next call
- make call with parameters: depth, last value written

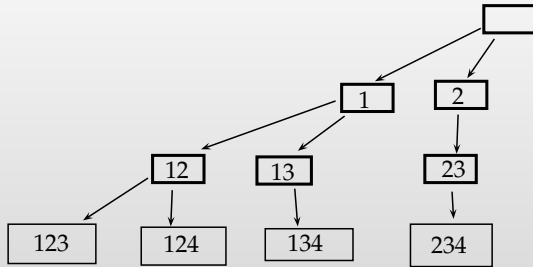
Recursion tree labeled with function call parameters k, j



A variant

Write a recursive function which takes as argument integers n and $k \leq n$ and prints all possible subsets of the set $\{1, 2, 3, \dots, n\}$ of size k (i.e. ${}^n C_k$).

Recursion tree: $n=4, k=3$



- In this case, printing is done only at the leaf nodes.
- Write the recursive function without a loop?? Yes, you can do this by adding another variable in the recursive call apart from those remembering the depth and last value written.