CS101 Introduction to computing

Recursive Function and Problem solving using function

A. Sahu and P. Mitra

Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Outline

- Recursive function
- Mutual recursion
- Problem solving using recursions
- Recursion : using Array

Fib (N): Number of Recursive Call

- Multiple Recursive Calls

  Fib(N) =Fib(n-1) + Fib(n-2)

- Number of recursive call for N
  - Claim: Number of recursive call for Fib(n-1) is higher than number of recursive call for Fib(n-2)
  - Denote number of recursive call for Fib(n) = f_n
  - $f_{n-1} > f_{n-2}$

Fib (N): Number of Recursive Call

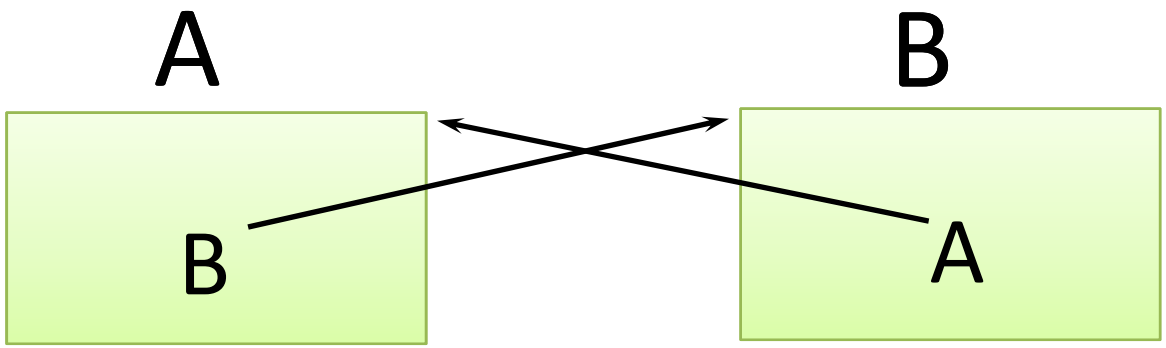
- Can I Say : $f_n = f_{n-1}+f_{n-2} > f_{n-2}+f_{n-2} = 2.\ f_{n-2}$
- Then $f_n > 2.f_{n-2} > 2.2.f_{n-4} > 2.2.2.f_{n-6}$

  $= \ldots = 2^{n/2} f_1$ **So $f_n > 2^{n/2}$**
- Number of recursive call require to compute Fib(n) is $> 2^{n/2}$
- Can you calculate for 200 Fibonacci using recursive program
  - **Will take at least 2^{100} recursive call : huge time and space**

Mutual Recursion

Recursion doesn't always occur because a routine calls itself...

Mutual Recursion occurs when two routines call each other.



Mutual Recursion Example EvenOdd

- Problem: Determine whether a number, N, is odd or even.
  - If N is equal to 0, then n is even
  - N is odd if N-1 is even

```
int  Even(int N){
 if(n==0) return 1;
  return Odd(n-1);
}
```

```
int  Odd(int N){
 if(n==0) return 0;
 return Even(n-1);
}
```

```
int  main(){
 printf("E(20)=%d O(101)=%d",
            Even(20), Odd(101));
}
```

Recursion Example: Problem Solving

```
int Reverse(int n) {
int RevNum=0, Rem;
RevNum=0;
while(n != 0)  {
   Rem = n%10;
   RevNum=RevNum*10+Rem;
   n=n/10;
   }
  return RevNum;
}
```

```
int Reverse(int n) {
static int  Rev=0;
if(n ==0) return 0;
 Rev = Rev *10;
 Rev = Rev + N%10;
 Reverse(N/10);
 return Rev;
}
```

GCD

- By definition from Euclid's algorithm
- Recursive
  - GCD(a,b) if a > b GCD(a%b, b)

    if b > a GCD(a, b%a)
- Code looks simpler

Recursive GCD

- Code looks simpler

```
int GCD(int a, int b) {
    if(a==0) return b;
    return gcd(b%a, a);
}
```

- Trace 35, 10 == > 10, 35 == > 5, 10 == > 0 , 5
- Trace 10, 15 == > 5, 10 == > 0, 5
- Trace 31, 2 == > 2, 31 ==> 1, 2 == >0, 1

Modular C Code : Binary Search

```
int BinSrch
 (int Rmin,
  int Rmax, int X){
while (Rmin<Rmax){

mid=(Rmin+Rmax)/2;
  if(X==mid)
    return mid;
  if (X>mid)
      Rmin=mid+1;
  else Rmax=mid;
  }
  return -1;
}
```

```
int BinSrch(int Rmin,
     int Rmax, int X){
int mid;
mid=(Rmin+Rmax)/2;
if(X==mid)return mid;
if (X>mid)
return
  BinSrch(mid+1,Rmax,X);
else
return BinSrch(Rmin,mid,X);
 }
  return -1;
}
```

Binary Search Analysis

- BinSrch(Range,X)
  - mid=Range/2;
  - BinSrch(Range/2,X);
- $B(R) = B(R/2) + c$

  $= B(R/4) + 2c$

  ..

  $= B(1) + \log_2 R.c$

  $= c.ceil(\log_2 R)$

Problem Solving
Array, Function, Recursion

Problem Solving : Examples

- Max of an Array
  - –Iterative and recursive procedure
- Sieve of Eratos-thenes
- Array Reversal
- Sorting an array
  - –Bubble sort

Maximum of an Array

Max of an Array

- Iterative Approach

```
int MaxOfAnArray(int A[], int n){
   int i, L=A[0];
   for(i=1; i<n; i++)
      if(A[i]>L)
            L=A[i];
   return L;
}
```

- Number of steps required
  - N : linear code

Max of an Array: Recursive Approach

Max(2, 5, 12, 8, 16, 23, 1, 5)

= Max (Max of 1st Half, Max of 2nd Half)

= Max (Max(2,5,12,8), Max(16,23,1,5))

```
int RMax(int A[], int n1, int n2){
  int L1, L2;
   if(n1==n2) return A[n1];
  L1=Rmax(A, n1, (n1+n2)/2);
  L2=Rmax(A,(n1+n2)/2, n2);
   if (L1>L2) return L1;
   else return L2;
}
```

Max of an Array: Recursive Approach

- Number of steps required
  - R(N) = R(N/2) + R(N/2) + C
    = 2.R(N/2)+ C
    = 4.R(N/4)+2C = 8.R(N/8)+4C = N.R(1)+N/2.C
    = N +N/2.C = linear number of steps

```
int RMax(int A[], int n1, int n2){
int L1, L2; if(n1==n2) return A[n1];
L1=Rmax(A, n1, (n1+n2)/2);
L2=Rmax(A,(n1+n2)/2, n2);
if (L1>L2) return L1;  else return L2;
}
```

Sieve of Eratosthenes

What is the sieve of Eratosthenes?

- Used to find prime number between 2 and N
- It works by gradually eliminating multiple of smallest unmark prime (x) in the given interval [2-N]
  - Till $x^2 > N$
- Let us see with an example

Sieve of Eratosthenes: 1 to 20

- Current Prime 2

[2,3,4,5,6,7,8,9,10,11,12,13, 14,15,16,17,18,19,20]

- Current Prime 2

[2,3,~~4~~,5, ~~6~~,7,~~8~~,9,~~10~~,11,~~12~~,13,~~14~~,15,~~16~~,17,~~18~~,19,~~20~~]

- Current Prime 3

List : [2, 3, 5, 7, 9, 11, 13, 15, 17, 19]

- Current Prime 3

List : [2, 3, 5, 7, ~~9~~, 11,13,~~15~~, 17,19]

- Current Prime 5 and 5< ceil(sqrt(20))

List : [2, 3, 5, 7, 11, 13, 17,19] //All are primes

Sieve of Eratosthenes

```
#define STRIKED 0  //Happened to be composite
#define NONSTRIKED 1   //Assumed Prime
void SeiveOfEratosthenes(int prime[MaxNum],
                         int N){
 int i,j, CP, SqrtM=sqrt(MaxNum)+1;
 for(i=0;i<MaxNum;i++) prime[i]= NONSTRIKED;
 for(i=2;i<SqrtM;i++){
   if (prime[i]== STRIKED) continue ;
   CP=i; //current Prime
   for(j=2*CP;j<MaxNum; j=j+CP) //do striking
         prime[j]= STRIKED;
   }
}
```

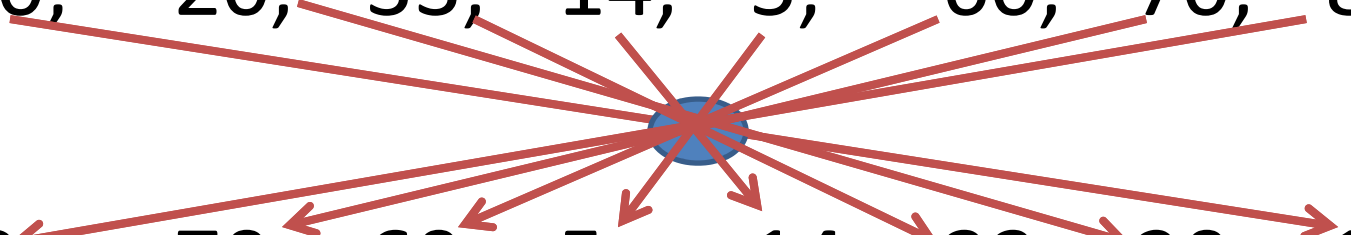
Reversing an Array

Reversing an Array

- Approach
  - Input: 10, 20, 33, 14, 5, 60, 70, 8
  - Output: 8, 70, 60, 5, 14, 33, 20, 10
  - Mirror Image

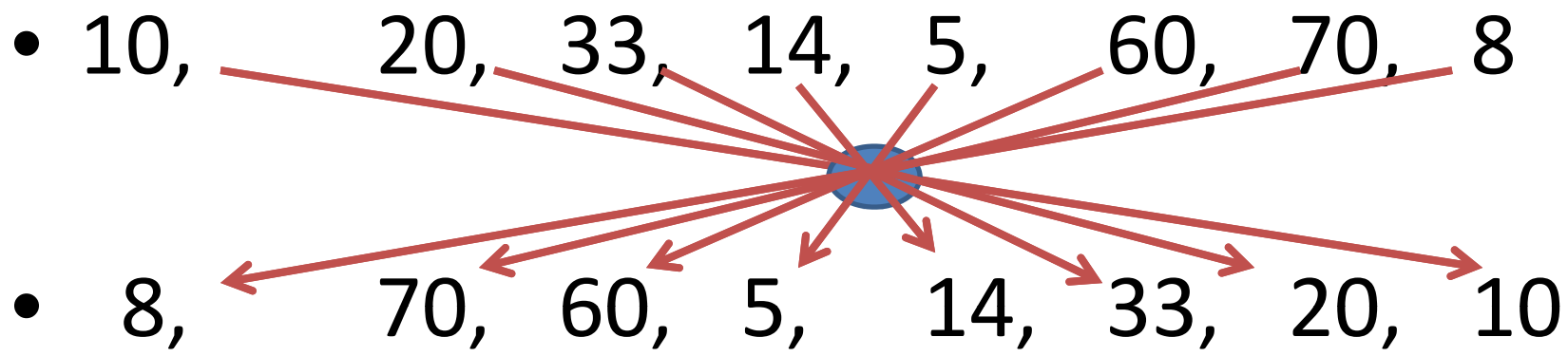
  10, 20, 33, 14, 5, 60, 70, 8 | 8, 70, 60, 5, 14, 33, 20, 10

- 10, 20, 33, 14, 5, 60, 70, 8



- 8, 70, 60, 5, 14, 33, 20, 10

Reversing an Array



- Start from both ends: Exchange element
- Repeat for next elements form both sides till they meet each other

Reversing an Array : Iterative Code

```
void Reverse(int  A[], int N,){
     int tmp,i=0,j=N-1;
     while(i<j){
          tmp=A[i]; [i]=A[j];A[j]=tmp;
          i=i+1,
          j=j-1;
     }
}
```

Reversing an Array : Recursive Code

```
void Reverse(int  A[],
             int i, int j){
int tmp;
if(i<j){
     tmp=A[i]; A[i]=A[j];A[j]=tmp;
     Reverse(A, i+1, j-1);
     }
}
void ReverseArray(int  A[], int N){
     Reverse(A,0, N-1);
}
```

Sorting an Array

Sorting an Array

- Sorting : Arrange the element in some specific order
- Simpler case
  - non-decreasing : smallest element at beginning and biggest element at end
  - non-increasing order: biggest element at beginning and smallest element at end
- Example Non-sorted

  30, 4, 7, 10, 12, 8, 2, 8
- Sorted : non-decreasing

  2, 4, 7, 8, 8, 10, 12, 30

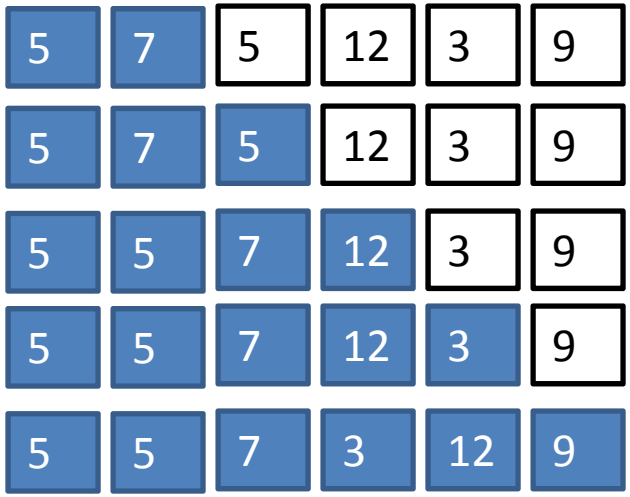
Bubble Sort

1. Start at the beginning of the data set.
2. Compare the first two elements, and if the first is greater than the second, swaps them.
3. Continue doing this for each pair of adjacent elements to the end of the data set.
4. Start again with the first two elements, repeating until no swaps have occurred on the last pass.

Bubble Sort

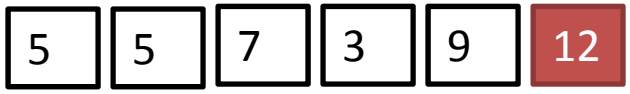
Invariant
Last PassNumber elements will be in sorted order

Pass 1



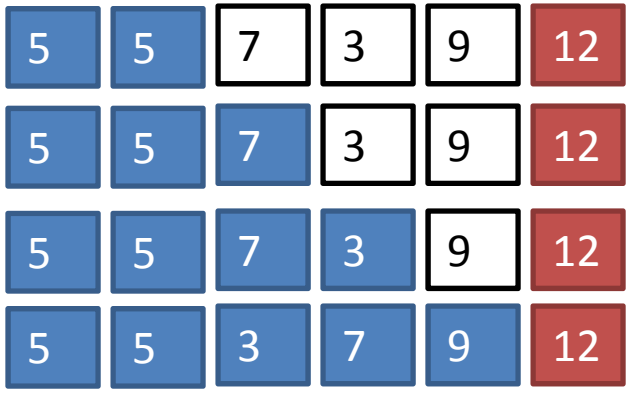
Last 0 elements are in sorted order

Bubble Sort



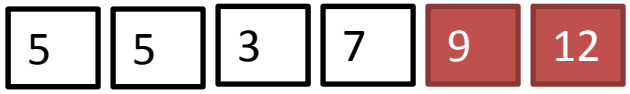
Invariant
Last PassNumber elements will be in sorted order

Pass 2



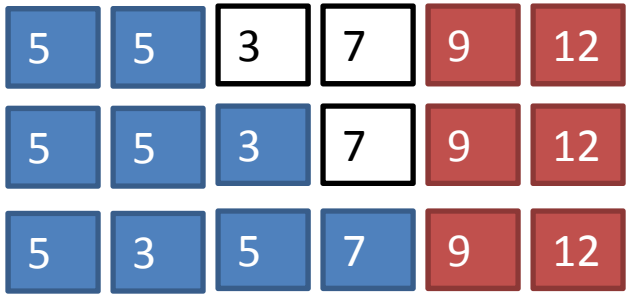
Last 1 element is in sorted order
Show in RED Box

Bubble Sort



Invariant
Last PassNumber elements will be in sorted order

Pass 3

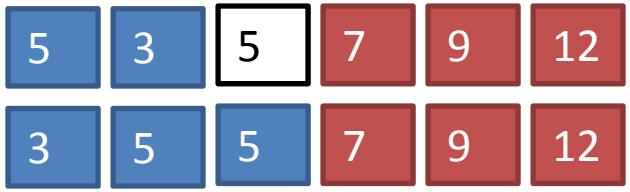


Last 2 elements are in sorted order
Show in RED Box

Bubble Sort

Invariant
Last PassNumber elements will be in sorted order

Pass 4

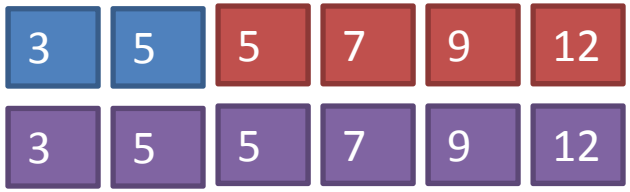


Last 3 elements are in sorted order
Show in RED Box

Bubble Sort

Invariant
Last PassNumber elements will be in sorted order

Pass 5



Last 4 elements are in sorted order
Show in RED Box

Sorted !!!

Bubble Sort

Invariant
Last PassNumber elements will be in sorted order

After Pass 5

All 5 elements are in sorted order
Show in RED Box

Sorted !!!

Bubble Sort: C Code

```
void BubbleSort(int A[], int N){
    int Pass, j, tmp;
    for(Pass=0; Pass<N; Pass++){
      for(j=0; j<(N-Pass-1); j++){
        if(A[j] > A[j+1]){
              tmp=A[j];
              A[j]=A[j+1];
              A[j+1]=tmp;
         }
       }
    }//end Pass
} //end BubbleSort
```

Thanks

Good luck for your MidSem Exam