CS1100
Introduction to Programming

Recursion and Sorting

Madhu Mutyam
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Factorial (n) – Recursive Program

fact(n) = n*fact(n-1)

```
int fact (int n){
if (n == 1) return 1;
return n*fact(n-1);
```

- Shorter, simpler to understand
- Uses fewer variables
- Machine has to do more work running this one!

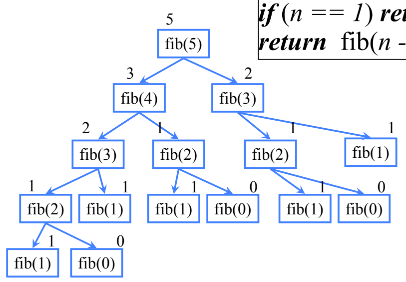
Tree Recursion

- When the recursive call is made more than once inside the function. For example,
- Fibonacci numbers
  - fib(n) = fib(n-1) + fib(n-2) if $n > 1$
    = n if n is 0 or 1
- Ackerman's function
  - One of the fastest growing functions

A(m, n) = n +1 if m = 0
= A(m-1, 1) if n = 0
= A(m-1, A(m, n-1)) otherwise

Fibonacci Numbers

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



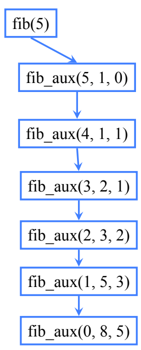
Fibonacci Numbers – Linear Recursion

```
int fib(int n)
{ return fib_aux( n, 1, 0) }
```

```
int fib_aux(int n, int next, int result) {
if (n == 0) return result;
return fib_aux(n - 1, next + result, next); }
```

Computation being done in the recursive call

f: 0,1, 1, 2, 3, 5, 8
n: 0,1, 2, 3, 4, 5, 6



Who Was Fibonacci?

- The "greatest European mathematician of the middle ages"
- His full name was Leonardo of Pisa
- Born in Pisa (Italy), about 1175 AD
- Was one of the first people to introduce the Hindu-Arabic number system into Europe
  - "These are the nine figures of the Indians: 9 8 7 6 5 4 3 2 1. With these nine figures, and with this sign 0 which in Arabic is called zephirum, any number can be written."
  - Part 1 of his book Liber abaci
- Best known for a simple series of numbers, later named the *Fibonacci numbers* in his honour.

Fibonacci Numbers

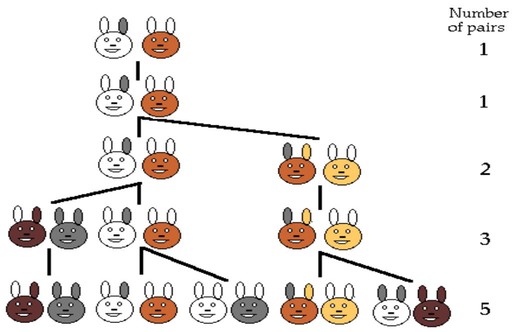
- The series begins with 0 and 1. After that, use the simple rule:
- **Add the last two numbers to get the next**
  - 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,...
- *Suppose a newly-born pair of rabbits, one male, one female, are put in a field. Rabbits are able to mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits. Suppose that our rabbits* ***never die*** *and that the female* ***always*** *produces one new pair (one male, one female)* ***every month*** *from the second month on. The puzzle that Fibonacci posed was...*

  How many pairs will there be in one year?

Rabbit Pairs

- *How many pairs will there be in one year?*
  1. At the end of the first month, they mate, but there is still one only 1 pair.
  2. At the end of the second month the female produces a new pair, so now there are 2 pairs of rabbits in the field.
  3. At the end of the third month, the original female produces a second pair, making 3 pairs in all in the field.
  4. At the end of the fourth month, the original female has produced yet another new pair, the female born two months ago produces her first pair also, making 5 pairs.
- In general, imagine that there are x_n *pairs* of rabbits after n months. The number of pairs in month $n+1$ will be x_n (in this problem, rabbits never die) plus the number of new pairs born. But new pairs are only born to pairs at least 1 month old, so there will be x_{n-1} new pairs.
- $x_{n+1} = x_n + x_{n-1}$

Rabbit Pairs



An Erratic Sequence

- In *Godel, Escher, Bach: An Eternal Golden Braid*, D. R. Hofstadter introduces several recurrences which give rise to particularly intriguing integer sequences.
- Hofstadter's Q sequence (also known as Meta-Fibonacci sequence)
- Q(1) = Q(2) = 1
- Q(n) = Q(n - Q(n - 1)) + Q(n - Q(n - 2)) for n > 2
- Each term of the sequence is the sum of two preceding terms, but (in contrast to the Fibonacci sequence) not necessarily the two last terms.
- The sequence Q(n) shows an erratic behaviour
- 1, 1, 2, 3, 4, 5, 5, 6, 6, 6, 8, 8, 8, 10, 9, 10, …
  - gets more and more erratic

Which is the Biggest?

- Given three numbers *a*, *b,* and *c,* find the biggest amongst them. Define a variable max to store the value.

```
if (a > b && a > c)
        max = a;
else if (b > c)
        max = b;
else
        max = c;
```

- Other similar code also works
- Method works for array elements as well A(1), A(2), A(3)
- But what if the array is large? This approach is not feasible

Highest Marks

- Given an array *marks*[100], find the highest marks in the class.

```
max = marks[0];    /* for the time being */
for (i=1; i<100; i++)
        if (marks[i] > max)
                max = marks[i];   /* update if bigger */
```

More Statistics

- Given an array *marks*[100], find the highest, lowest and average marks in the class.

```
max = marks[0];    /* for the time being */
min = marks[0];
sum = marks[0];
for (i=1; i<100; i++){
        if (marks[i] > max) max = marks[i];
        if (marks[i] < min)  min = marks[i];
        sum += marks[i];
}
average = sum/100;        /*assuming floating point*/
```


Exchanging Values

- Exchange the values of variables (*a* and *b*)

```
{ a = b;        INCORRECT
  b = a;
}
```

Value of *a* is lost!

Need to use a temporary variable

```
{ temp = a; /* save the value of a */
  a = b;
  b = temp;
}
```


Exchanging Values without *temp*

- What about the following method that does not use an extra variable?

```
{
  a = a+b;
  b = a - b;
  a = a - b;
}
```

- Exercise: Does it work? What are the limitations? Do you need to be careful about something?

Swap Array Elements

```
void swap (int array[ ], int i, int j)
{
        int temp;
        temp = array[i];
        array[i] = array[j];
        array[j] = temp;
}
```


Where is the Highest Number?

- Given an array of *n* elements, a starting index *i*, find out where the largest element lies beyond and including *i*.

```
int MaxIndex (int array[ ], int start, int arraySize){
  int i = start; int index = start;
  int max = array[i];
  for ( ; i < arraySize;  i++)    /* observe null statement */
       if (array[i] > max){
                max = array[i];
                index = i;
       }
  return index;
}
```


Sorting an Array of Numbers

- Problem: Arrange the marks in decreasing order starting with the maximum.
- One approach
  - Find the maximum value in *marks*[0] … *marks*[99]
  - Remember the index *i* where it occurred
  - Exchange (values of) *marks*[0] and *marks*[*i*]
  - Find the maximum value in *marks*[1] to *marks*[99]
  - exchange *marks*[1] and *marks*[*i*]
  - . . . do this till marks[98]

Selection Sort

swap is an function that passes array by reference.

```
for (i=0, i <= n - 2, i++) {
swap (marks, i, MaxIndex(marks, i, n));
}
```

or more legibly

The *last* element need not be tested

```
for (i=0, i <= n - 2, i++){
int maxIndex = MaxIndex(marks, i, n);
if (maxIndex != i) swap(marks, i, maxIndex);
}
```


Selection Sort as a Function

```
void selectSort(int array[ ], int size)
{
  int maxIndex, i;
  for (i = 0; i <= size – 2; i++)
  {
     maxIndex = MaxIndex(array, i, size)
     if (maxIndex != i) swap(array, i, maxIndex);
  }
}
```


An Example

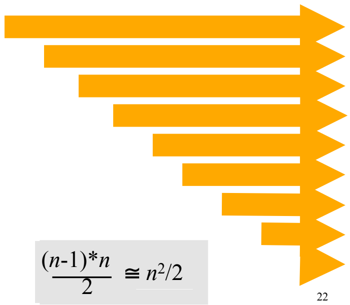
0	1	2	3	4	5	6	7	*i*	maxIndex
2	1	7	5	8	3	6	4	0	4
8	1	7	5	2	3	6	4	1	2
8	7	1	5	2	3	6	4	2	6
8	7	6	5	2	3	1	4	3	3
8	7	6	5	2	3	1	4	4	7
8	7	6	5	4	3	1	2	5	5

Complexity of Selection Sort

0	1	2	3	4	5	6	7	8	9

n=10

i=0 n-1=9 comparisons
i=1 n-2=8 comparisons
i=2 n-3=7 comparisons
i=3 n-4=6 comparisons
i=4 n-5=5 comparisons
i=5 n-6=4 comparisons
i=6 n-7=3 comparisons
i=7 n-8=2 comparisons
i=8 n-9=1 comparison



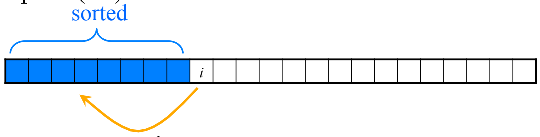
$$\frac{(n-1)*n}{2} \cong n^2/2$$

Complexity of Selection Sort

- In each iteration, MaxIndex finds the maximum element
  - Complexity of MaxIndex is order n → O(n)
  - Can we do this faster? Yes, by arranging the numbers in a data structure called a MaxHeap
  - MaxHeap can extract max element in O(log(n))
  - Algorithm Heapsort – complexity O(n log(n))
- Selection sort does (n-1) passes of reducing length (average length n/2)
  - Complexity (n-1)*n/2 → O(n^2/2) → O(n^2)

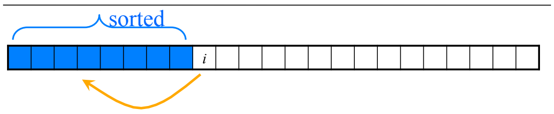
Insertion Sort

- Insertion sort also scans the array from left to right
- When it looks at the i^{th} element, it has elements up till (i-1) sorted



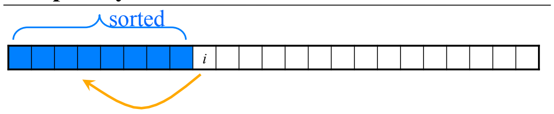
- It moves the i^{th} element to its correct place by shifting the smaller elements to the right

InsertMax Function




```
void InsertMax (int array[ ], int index){
    int i = index;
    int valueAtIndex = array[index];
    while(i > 0 && array[i-1] < valueAtIndex) {
        array[i] = array[i-1];    /*shift right*/
        i--;
    }
    array[i] = valueAtIndex;
}
```


Complexity of InsertMax



- If the i^{th} element is in sorted order (smaller than the sorted set), no shift is done
- The maximum number of shifts is (i-1)
- Complexity
  - worst case O(i)
  - best case O(1) – constant time

Insertion Sort Function

```
void InsertionSort(int array[ ], int size){
    int i;
    for(i = 1; i <= size – 1; i++)
        InsertMax(array[ ], i);
}
```

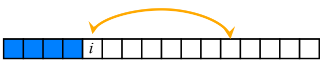
- Complexity
  - best case O(n)
  - worst case O(n^2/2) = O(n^2)

An Example

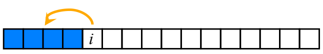
0	1	2	3	4	5	6	7	i	# of comp
2	1	7	5	8	3	6	4	1	1
2	1	7	5	8	3	6	4	2	2
7	2	1	5	8	3	6	4	3	3
7	5	2	1	8	3	6	4	4	4
8	7	5	2	1	3	6	4	5	3
8	7	5	3	2	1	6	4	6	5

Selection Vs Insertion Sort

- Scanning from left to right
- Selection sort



  - Swaps the i^{th} element with the largest unsorted element
- Insertion sort

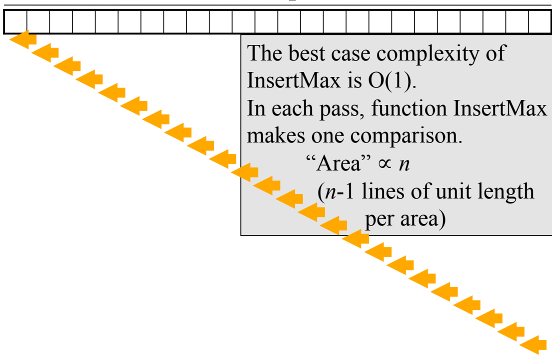


  - Inserts the ith element into its proper place

Selection Vs Insertion

- Selection sort always does the same number of computations irrespective of the input array
- Insertion sort does less work if the elements are partially sorted
  - when the i^{th} element is in place, it does not have to shift any elements – constant time
- If the input is already sorted, Insertion sort merely scans the array left to right – confirming that it is sorted
- On the average, Insertion sort performs better than Selection sort

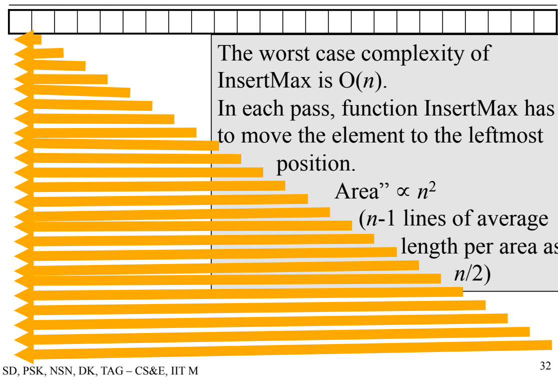
Insertion Sort – Sorted Input



The best case complexity of InsertMax is O(1).
In each pass, function InsertMax makes one comparison.
"Area" $\propto n$
(n-1 lines of unit length per area)

SD, PSK, NSN, DK, TAG – CS&E, IIT M 31

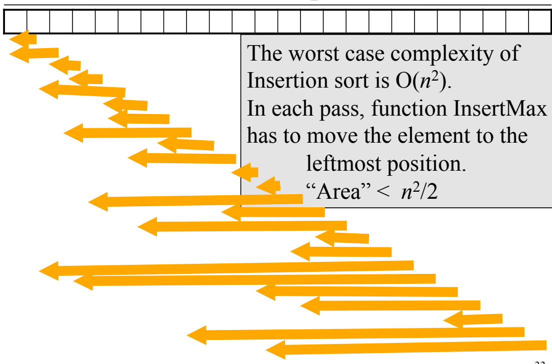
Insertion Sort – Reverse Sorted Input



The worst case complexity of InsertMax is O(n).
In each pass, function InsertMax has to move the element to the leftmost position.
Area" $\propto n^2$
(n-1 lines of average length per area as n/2)

SD, PSK, NSN, DK, TAG – CS&E, IIT M 32

Insertion Sort – Random Input



The worst case complexity of Insertion sort is O(n^2).
In each pass, function InsertMax has to move the element to the leftmost position.
"Area" < $n^2/2$

SD, PSK, NSN, DK, TAG – CS&E, IIT M 33

Exercise for this Week

- Given an array of strings, called *names*, and an array of marks, called *marks*, such that *marks*[*i*] contains the marks of *names*[*i*]
  - sort the two lists in decreasing order of marks
  - sort the two lists in alphabetic order of names
    - figure out how to compare two names to decide which comes first.

SD, PSK, NSN, DK, TAG – CS&E, IIT M 34