IC150
Computational Engineering

Lecture 12 – Fibonacci Numbers

Timothy A. Gonsalves

Tree Recursion

When the recursive call is made more than once inside the function

- Eg. 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

Factorial (n) – recursive program

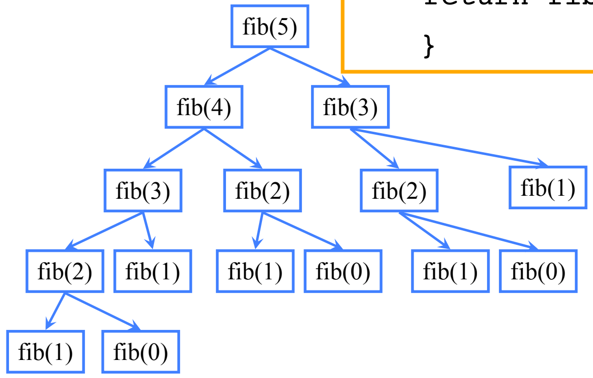
n! = n×n!

```
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!

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);
  }
```



Finonacci 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

1, 1, 2, 3, 5

$$\begin{tikzcd}
\text{fib(5)} \arrow[d] \\
\text{fib\_aux(5, 1, 0)} \arrow[d] \\
\text{fib\_aux(4, 1, 1)} \arrow[d] \\
\text{fib\_aux(3, 2, 1)} \arrow[d] \\
\text{fib\_aux(2, 3, 2)} \arrow[d] \\
\text{fib\_aux(1, 5, 3)} \arrow[d] \\
\text{fib\_aux(0, 8, 5)}
\end{tikzcd}$$

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, as will be demonstrated."*
  - Part 1 of his book *Liber abaci*
- Best known for a simple series of numbers, introduced in *Liber abaci* and later named the *Fibonacci numbers* in his honour.

Fibonacci numbers

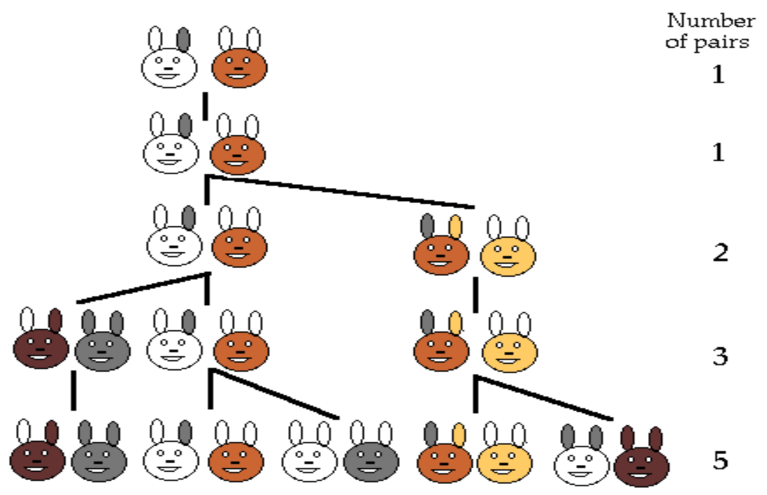
- The series begins with 0 and 1. After that, use the simple rule:
  - **Add the last two numbers to get the next**
  - 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,...
- *Suppose a newly-born pair of rabbits, one male, one female, are put in a field. Rabbits are able to mate at the age 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

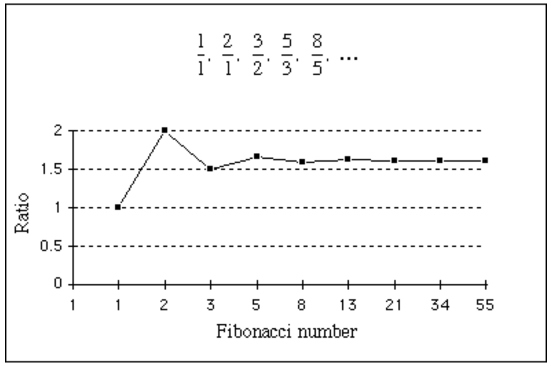
- *How many pairs will there be in one year?*
  1. At the end of the first month, they mate, but there is still one only 1 pair.
  2. At the end of the second month the female produces a new pair, so now there are 2 pairs of rabbits in the field.
  3. At the end of the third month, the original female 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.
- $x_{n+1} = x_n + x_{n-1}$

Dr Ron Knott's http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/fibnat.html



The Golden Ratio

- *golden section, golden ratio or golden mean*
- obtained by taking the ratio of successive terms in the Fibonacci series

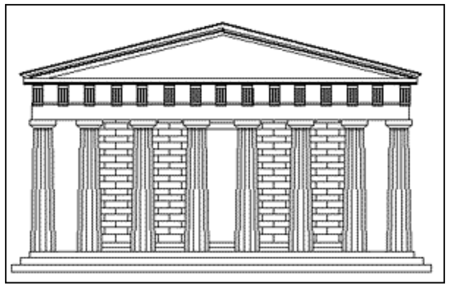


Φ

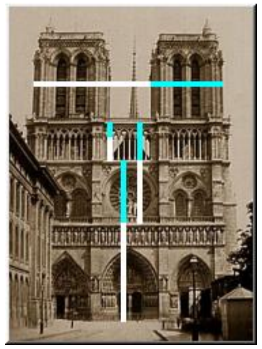
- This limit is actually the positive root of a quadratic equation

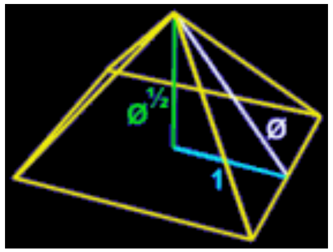
  $\Phi^2 - \Phi - 1 = 0$ $\qquad \Phi = (1 + \sqrt{5}) / 2 \approx 1.618$

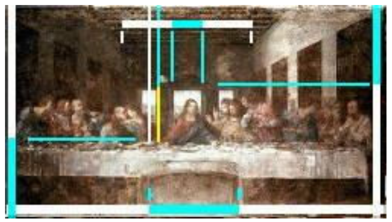
- Greek architects used the ratio 1:*phi* as an integral part of their designs, the most famous of which is the Parthenon in Athens



Art and architecture – "divine ratio"

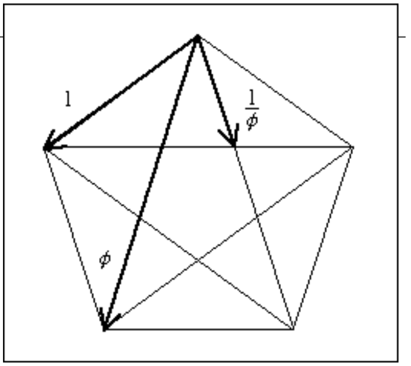




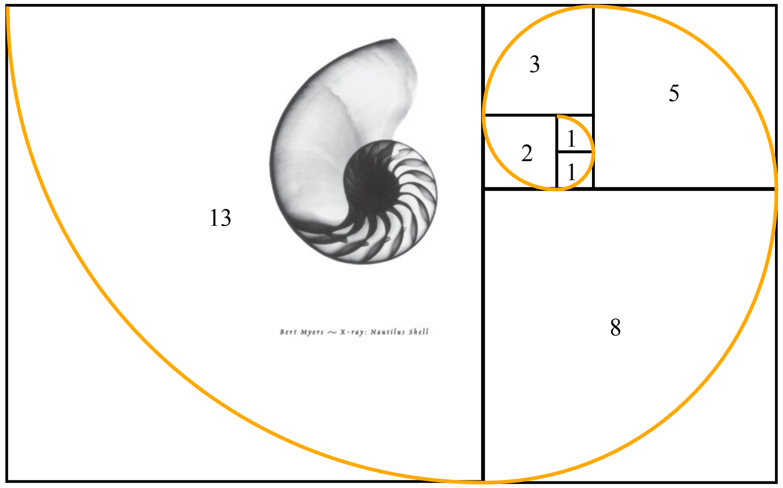


Phi and geometry

- *Phi* also occurs surprisingly often in geometry
- It is the ratio of the side of a regular pentagon to its diagonal
- If we draw in all the diagonals then they each cut each other with the golden ratio too
- The resulting pentagram describes a star which forms part of many of the flags of the world



Fibonacci Rectangles (sides - fib(n) and fib(n+1)

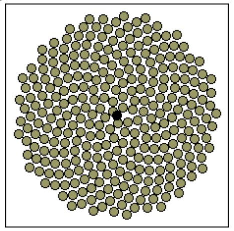


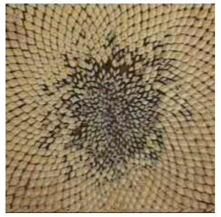
Fibonacci in nature - Petals on flowers

- On many plants, the number of petals is a Fibonacci number:
  - lilies and iris have 3 petals;
  - buttercups and pinks have 5 petals;
  - some delphiniums have 8;
  - corn marigolds have 13 petals;
  - some asters have 21
  - whereas daisies can be found with 34, 55 or even 89 petals

Fibonacci in nature - Arrangement of seeds

- Next time you look at a sunflower, observe the arrangement of the seeds. They appear to be spiraling outwards both to the left and the right. There are a Fibonacci number of spirals (at each distance)! It seems that this arrangement keeps the seeds uniformly packed no matter how large the seed head grows.





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

IC150

Sorting an Array of Numbers

Lecture 13

Timothy A. Gonsalves

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; b = a;}
```

X

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

Where is the largest number?

- Given an array of `size` elements, a starting index `start`, find the index of the largest element from `start` to the end

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

IC150 Computation for Engineers

Selection Sort & Insertion Sort

Lecture 14

Timothy A. Gonsalves

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[2] to marks[99]
  - exchange marks[2] and marks[i]
  - . . .   do this till marks[98]

Selection Sort

N=10

	7	1	12	33	6	95	76	77	8	10
i=0	**95**	1	12	33	6	**7**	76	77	8	10
i=1	95	**77**	12	33	6	7	76	**1**	8	10
i=2	95	77	**76**	33	6	7	**12**	1	8	10
i=3	95	77	76	**33**	6	7	12	1	8	10
i=4	95	77	76	33	**12**	7	**6**	1	8	10
i=5	95	77	76	33	12	**10**	6	1	8	**7**
i=6	95	77	76	33	12	10	**8**	1	**6**	7
i=7	95	77	76	33	12	10	8	**7**	6	**1**
i=8	95	77	76	33	12	10	8	7	**6**	1

Selection Sort

The *last* element need not be tested

```
for (i=0, i < n-1, i++)
   {
     maxIndex = MaxIndex(marks[], i, N)
     if (maxIndex != i) SwapArray(marks[], i, maxIndex);
   }
```

`SwapArray()` takes an array and two indices of elements to be swapped

Name of array, `marks[]`, is hard-coded
=> cannot reuse code

Selection Sort Function

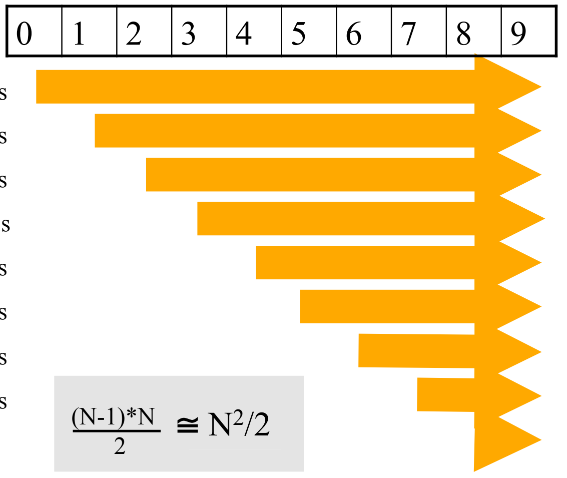
```
void SelectSort(int array[], int size)
{
   int maxIndex, i;
   for (i = 0; i <= size - 2; i++)
    {
     maxIndex = MaxIndex(array[], i, size)
     if (maxIndex != i)
          SwapArray(array[], i, maxIndex);
   }
}
```

Reuseable code

Complexity of Selection Sort

N=10

i=0	N-1=9 comparisions
i=1	N-2=8 comparisions
i=2	N-3=7 comparisions
i=3	N-4=6 comparisions
i=4	N-5=5 comparisions
i=5	N-6=4 comparisions
i=6	N-7=3 comparisions
i=7	N-8=2 comparisions
i=8	N-9=1 comparision



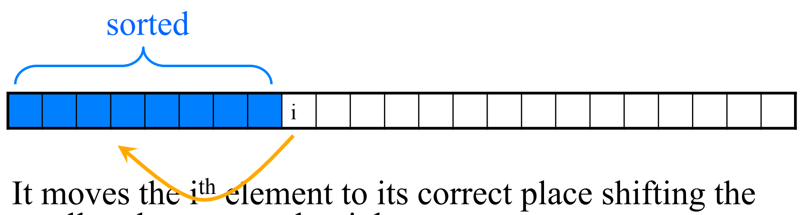
$$\frac{(N-1)*N}{2} \cong N^2/2$$

Complexity of selection sort

- In each iteration `MaxIndex` finds the maximum element
  - complexity of `MaxIndex` is order $n \rightarrow O(n)$
  - 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 $(n-1)\times n/2 \rightarrow O(n^2/2) \rightarrow O(n^2)$

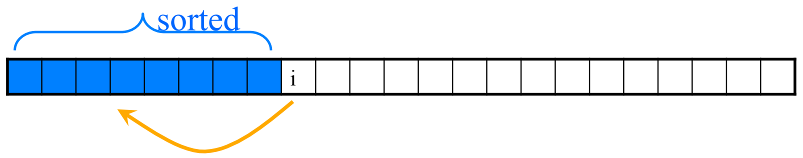
Insertion Sort

- Insertion sort also scans the array left to right.
- When it looks at ith element it has elements up till (i-1) sorted



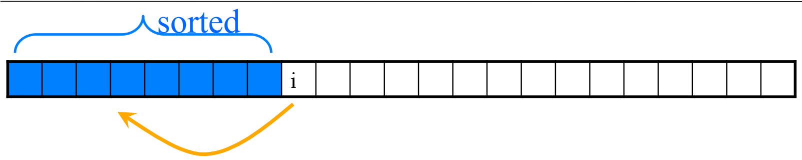
- It moves the ith element to its correct place shifting the smaller elements to the right

Function InsertMax



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

Complexity of InsertMax



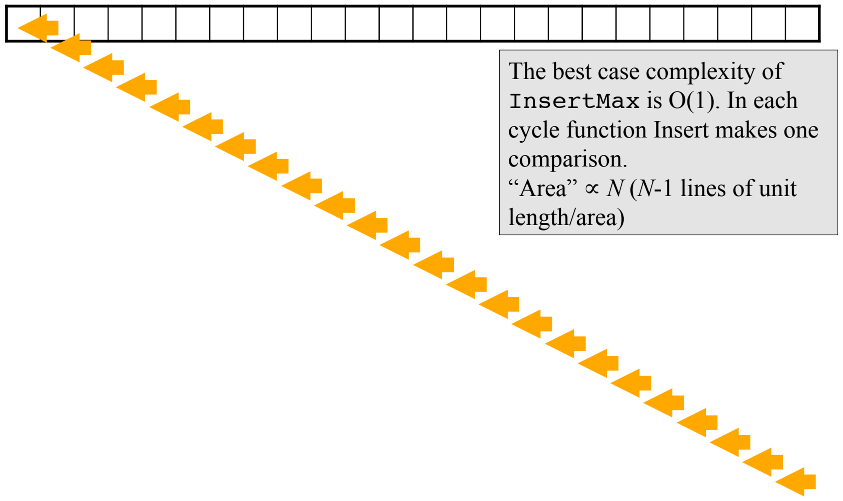
- If the ith element is in sorted order (smaller than the sorted set) no shift is done.
- The maximum number of element shifted is (i-1)
- Complexity
  - worst case O(i)
  - best case O(1) – constant time

Insertion Sort Function

```
void InsertionSort(int array[], int size)
{
  int i;
  for (i = 0; i <= size - 1; i++)
     InsertMax(array[], i);
}
```

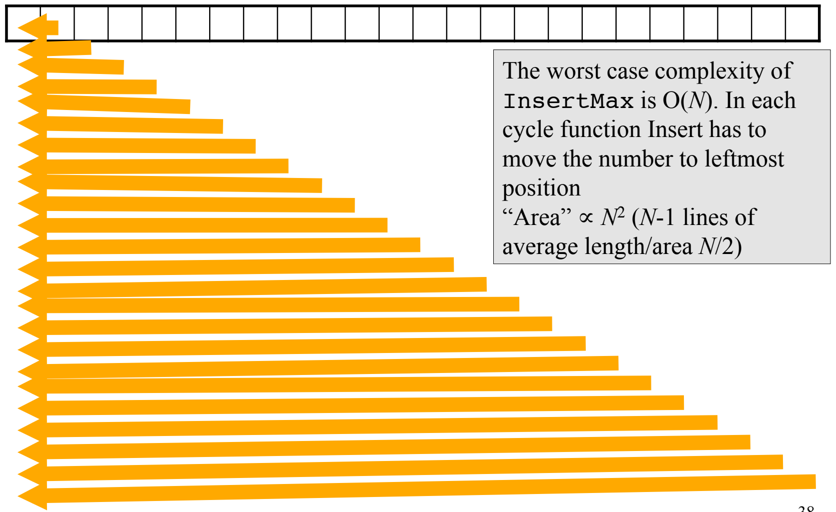
- Complexity
  - best case $O(N)$
  - worst case $O(N^2/2) = O(N^2)$

Insertion Sort – sorted input



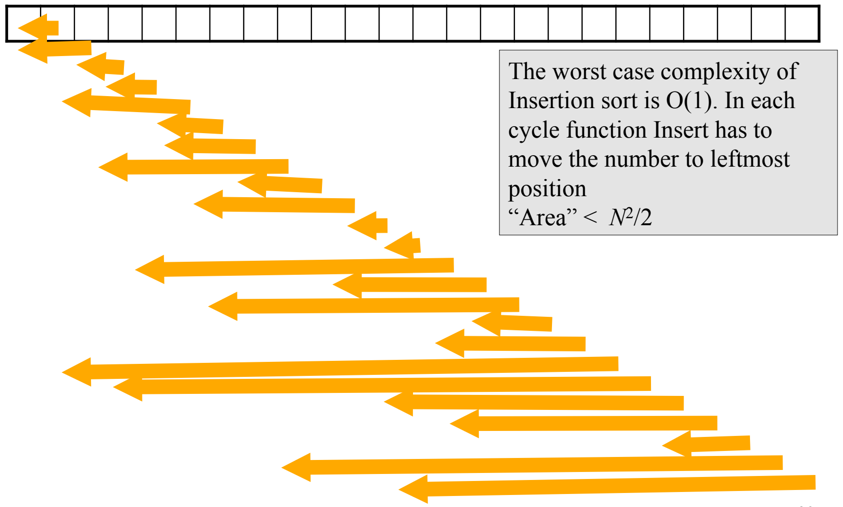
The best case complexity of `InsertMax` is O(1). In each cycle function Insert makes one comparison.
“Area” $\propto N$ (N-1 lines of unit length/area)

Insertion Sort – reverse sorted input



The worst case complexity of `InsertMax` is O(N). In each cycle function Insert has to move the number to leftmost position
“Area” $\propto N^2$ (N-1 lines of average length/area N/2)

Insertion Sort – random input

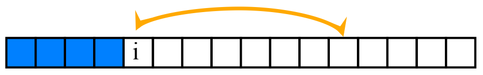


The worst case complexity of Insertion sort is O(1). In each cycle function Insert has to move the number to leftmost position
“Area” $< N^2/2$

Selection vs. Insertion sort

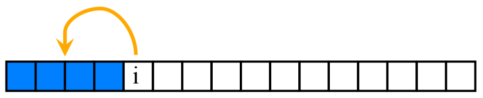
Scanning from left to right . . .

selection sort



swaps the ith element with the largest unsorted element

insertion sort



insert the ith element into its proper place

Insertion vs. Selection

- 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 the right place, no elements are shifted – 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 that Selection sort
- Selection sort is *unstable*, Insertion sort is *stable*

Exercise for this week

- Given an array of strings, `name` and an array of marks, `marks` such that `marks[i]` contains the marks of `name[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, without using any functions from `string.h`
- *Aliter*: declare an array of `StudentType` and sort this

IC150

Sorting Strings

Lecture 15

Timothy A. Gonsalves

Lexicographic (dictionary) ordering

- Badri < Devendra
- Janak < Janaki
- Shivanandan < Shivendra
- Seeta < Sita

- Based on the ordering of characters

  A < B … < Y < Z < a < b < c < . . . < y < z

upper case before lower case

Lexicographic ordering

- What about blanks?
  - “Bill Clinton” < “Bill Gates”
  - “Ram Subramanian” < “Ram Subramanium”
  - “Ram Subramanian” < “Rama Awasthi”
- In ASCII the blank (code = 32) comes before all other characters. The above cases are taken care of automatically.
- Exercise: Look up the ASCII codes

Lexicographic ordering

- What if two names are identical?
- There is a danger that the character arrays may contain some unknown values beyond ‘\0’
- Solutions
  1. One could begin by initializing the arrays to blanks before we begin.
  2. One could explicitly look for the null character ‘\0’
  3. When the two names are equal it may not matter if either one is reported before the other. Though in *stable sorting* there is a requirement that equal elements should remain in the original order.

Comparing strings (char arrays)

Given two strings A[i][] and A[j][] of length n return the index of the string that comes earlier in the lexicographic order

```
int StrCompare(A[][n], i, j, n)
  {
      int k=0;
      while ((A[i][k] == A[j][k]) && k<n) k++;
      if (A[i][k] == ‘\0’) return i;
      if (A[j][k] == ‘\0’) return j;
      if (A[i][k] < A[j][k]) return i;
        else return j;
}
```

Skip common characters if any

If one string is *prefix* of the other return that

Built in string comparison

Pointers - address of char array – we will look at them later

```
#include <string.h>
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2,
    size_t n);
```

Compares first n characters only

`int strcmp(char*, char*)` – compares two strings (char arrays). The return values are:

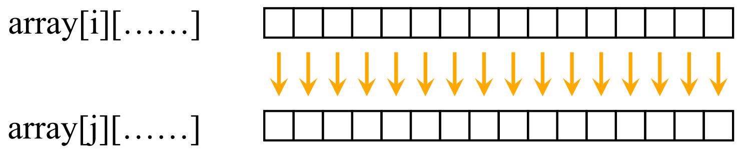
0	– If both strings are equal.
1	– If first string is lexicographically greater than second.
-1	– If second string is lexicographically greater than first.

Other built in string functions

- ***char* strcat(char* dest, char* src)*** - combines two strings and returns a pointer to the destination string. In order for this function to work (and not seg fault), you **must** have enough room in the destination for both strings.
- ***char* strcpy(char* dest, char* src)*** - copies one string to another. The destination must be large enough to accept the contents of the source string.
- ***int strlen(const char* s)*** - returns the length of a string, excluding the terminating NULL character.

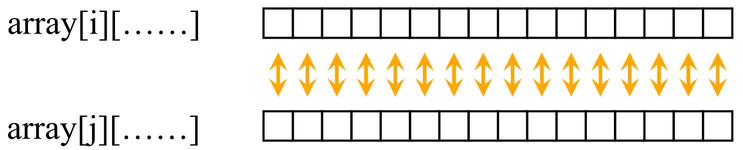
ArrayCopy

Copies content of i^{th} row of array into the j^{th} row



```
void ArrayCopy(char a[][MAX_SIZE],i,j, MAX_SIZE)
  {
    for (k=1; k<MAX_SIZE; k++)
       a[j][k] = a[i][k];
  }
```

ArraySwap



```
void ArraySwap(char a[][MAX_SIZE], i, j, MAX_SIZE)
  {
    for (k=1; k<MAX_SIZE; k++)
        Swap(a[i][k], a[j][k];
  }
```

Note: We exchange the entire arrays. If we knew the length of the longer string we could have a different end condition.

Sorting string arrays

- Modify Insertion Sort to sort array `names[]` of names
- Use `StrCompare` to compare names
- Use `ArrayCopy` to move names
- In the exercise where `names[]` and `marks[]` have to be sorted in concert, modify the sorting algorithm to
  - compare in one array
    - `names[]` for alphabetic order
    - `marks[]` for decreasing marks order
  - move elements of *both*

Compact structures to hold both to be explored later

Printing a reversed string

```
main()
  {
    int i=4;
    char c;
    do {
      c = "hello"[i];
      printf("%c",c);
      i--;
    }
    while(i >= 0);
    printf("\n");
  }
```

Palindromes

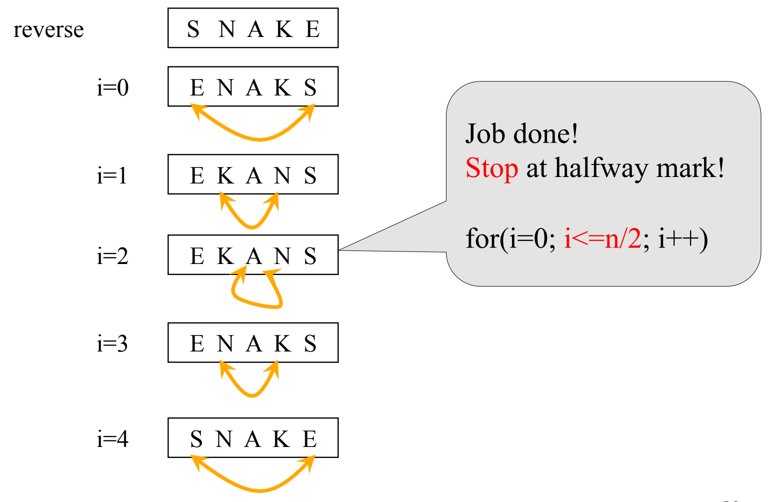
- Strings/sequences that read the same left to right or right to left
- string == reversed string
  1. malayalam
  2. god live evil dog
  3. able was I ere I saw elba
  4. don’t nod
  5. never odd or even
  6. madam i’m adam
- We ignore blanks (4, 5, 6) and other characters (4, 6)
  – preprocess the string to remove them

Reversing an array

- Swap the first element with last
  – a(0) with a(n-1)
- second with second last
  – a(1) with a(n-2)
- … a(i) with a(n-(1+i))
- How about the following code?

```
for (i=0; i<n; i++)
  Swap(a(i), a(n-(i+1));
```

Limits for iteration



Exercise

- Compute the transpose of a matrix
- Compute *in place transpose* of a square matrix

T	A	B	L	E
A	C	E	D	O
S	T	E	E	P

T	A	S
A	C	T
B	E	E
L	D	E
E	O	P

Palindrome Squares

- Write a program to check if a square matrix contains a palindrome table. Two examples are given below

from http://www.fun-with-words.com/palin_example.html

S	T	E	P
T	I	M	E
E	M	I	T
P	E	T	S

R	A	T	S
A	B	U	T
T	U	B	A
S	T	A	R

Concatenating two strings

Method to concatenate s2 to the end of s1

1. Check that s1 is large enough
   a. Find lengths of s1 and s2
   b. Can s1 hold both?
2. If yes, copy s2 to the end of s1

Concatenating two strings

```
#include <stdio.h>
void main() {
  char s1[40] = "String in C";
  char s2[40] = "Another string in C";
  int count1  = 0;       // Length of first string
  int count2  = 0;       // Length of second string

 // Find the length of the first string
  while (s1[count1] != '\0')
    count1++;
```

… Array operations on strings

```
// find the length of the second string
   while (s2[count2] != '\0')
      count2++;
// Check that we have enough space for both strings
   if (sizeof(s1) >= count1 + count2 + 1)
   {            /* Copy 2nd string to first */
      count2 = 0;
      while ((s1[count1++] =
                 s2[count2++]) != '\0');

      printf(”%s\n", s1);    // Output combined string
   }
   else
      printf(“Not enough space for both\n");
}
```