Sorting Algorithm

Lecture 2

Outlines

- Introduction
- bubble sort
- Insertion Sort

Introduction

- Sorting data (i.e., placing the data into a particular order such as ascending or descending) is one of the most important computing applications

bubble sort

- Assuming descending sort, In what is called the bubble sort or the sinking sort the smaller values gradually “bubble” their way upward to the top of the array like air bubbles rising in water, while the larger values sink to the bottom of the array

bubble sort

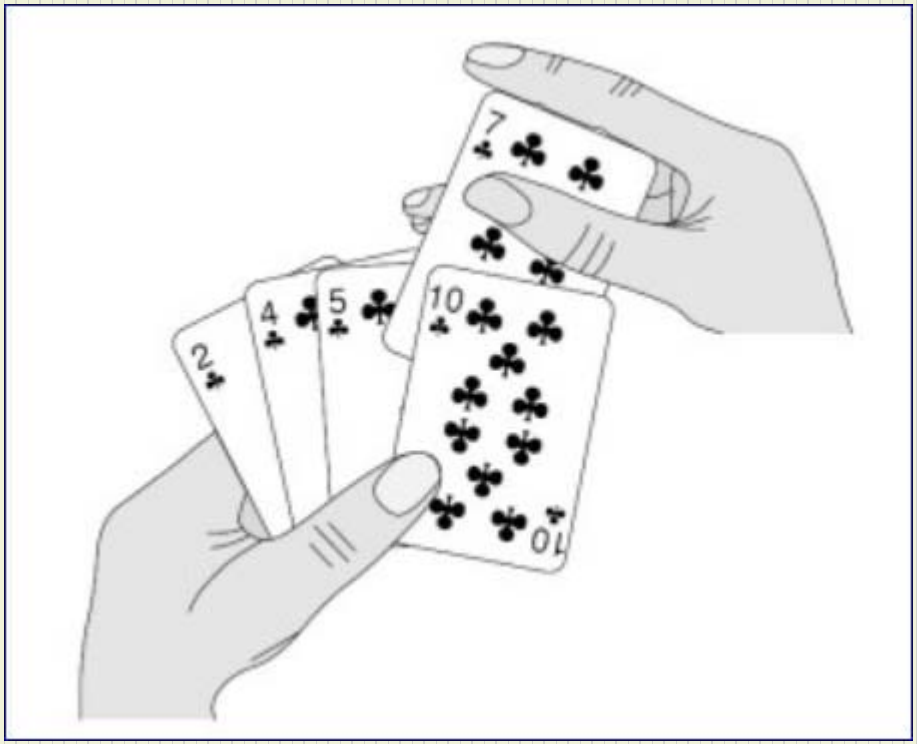
		0	1	2	3	4	5	6	7
i = 0	j								
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i =1	0	3	1	5	8	2	4	7	9
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	8	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	7		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	5			
	1	1	2	3	4				
	2	1	2	3	4				
i = 5	0	1	2	3	4				
	1	1	2	3					
i = 6	0	1	2	3					
		1	2						

```c
// Fig. 6.15: fig06_15.c
// Sorting an array's values into ascending order.
#include <stdio.h>
#define SIZE 10

// function main begins program execution
int main( void )
{
   // initialize a
   int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
   int pass; // passes counter
   size_t i; // comparisons counter
   int hold; // temporary location used to swap array elements

   puts( "Data items in original order" );

   // output original array
   for ( i = 0; i < SIZE; ++i ) {
      printf( "%4d", a[ i ] );
   } // end for

   // bubble sort
   // loop to control number of passes
   for ( pass = 1; pass < SIZE; ++pass ) {

      // loop to control number of comparisons per pass
      for ( i = 0; i < SIZE - 1; ++i ) {

         // compare adjacent elements and swap them if first
         // element is greater than second element
         if ( a[ i ] > a[ i + 1 ] ) {
            hold = a[ i ];
            a[ i ] = a[ i + 1 ];
            a[ i + 1 ] = hold;
         } // end if
      } // end inner for
   } // end outer for
```

```
38
39      puts( "\nData items in ascending order" );
40
41      // output sorted array
42      for ( i = 0; i < SIZE; ++i ) {
43         printf( "%4d", a[ i ] );
44      } // end for
45
46      puts( "" );
47   } // end main
```

```
Data items in original order
   2   6   4   8  10  12  89  68  45  37
Data items in ascending order
   2   4   6   8  10  12  37  45  68  89
```

Insertion Sort

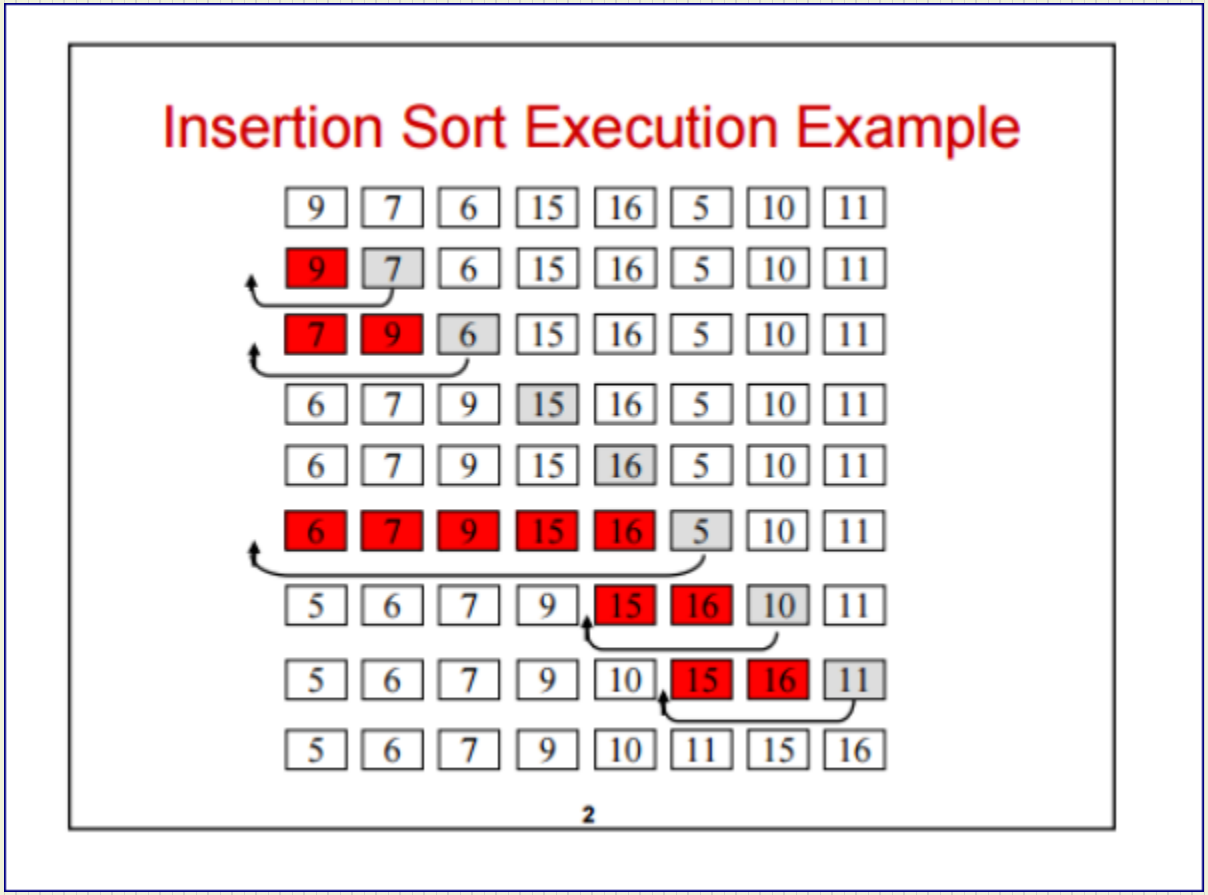


Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands

Insertion Sort

Algorithm
// Sort an arr[] of size n
insertionSort(arr, n)
Loop from i = 1 to n-1.
……a) Pick element arr[i] and insert it into sorted sequence arr[0…i-1]

Insertion Sort



```
// C program for insertion sort
#include <stdio.h>
#include <math.h>

/* Function to sort an array using insertion sort*/
void insertionSort(int arr[], int n)
{
   int i, key, j;
   for (i = 1; i < n; i++)
   {
       key = arr[i];
       j = i-1;

       /* Move elements of arr[0..i-1], that are
          greater than key, to one position ahead
          of their current position */
       while (j >= 0 && arr[j] > key)
       {
           arr[j+1] = arr[j];
           j = j-1;
       }
       arr[j+1] = key;
   }
}
```

```
// A utility function ot print an array of size n
void printArray(int arr[], int n)
{
   int i;
   for (i=0; i < n; i++)
       printf("%d ", arr[i]);
   printf("\n");
}



/* Driver program to test insertion sort */
int main()
{
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr)/sizeof(arr[0]);

    insertionSort(arr, n);
    printArray(arr, n);

    return 0;
}
```

End