Zaman University

Data Structures and Algorithms

Chapter 1

Introducing Data Structures and Algorithms

DR. SRUN SOVILA

Outline

- Arrays
- Ordered Arrays
- The Bubble Sort
- The Insertion Sort

Outline

- Arrays
- Ordered Arrays
- The Bubble Sort
- The Insertion Sort

Arrays

- An array is a number of data items of the same type arranged contiguously in memory;
- The array is the most commonly used data storage structure;
- it’s built into most programming languages.

- How do you create an array in C?

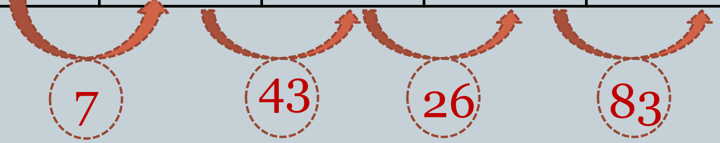
Arrays: Insertion

- Suppose, we have following array:

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Value	12	10	7	43	26	83				

- Insert value 99 to position (index) 2, insertion process:

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Value	12	10	99	7	43	26	83			



7

43

26

83

- How you do it in C?

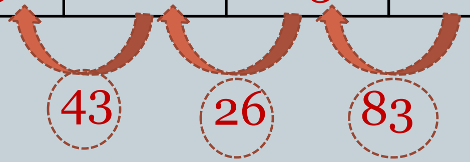
Arrays: Deletion

- For the following array,

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Value	12	10	99	7	43	26	83			

- Delete an element of position (index) 3, deletion process:

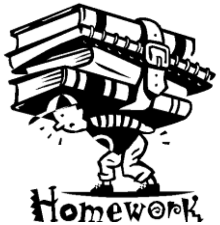
Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Value	12	10	99	43	26	83				



- How you do it in C?

Arrays: Search (Non-duplicate and Duplicate)

- Non-duplicate search by input value in array
  - Check an input value with each value of elements in array, in case an input value is equal to the value of any element of array, the procedure search is finish (break)
- Duplicate search by input value in array
  - Check an input value with every value (till the end element) of elements in array, to find, how many elements of array are equal to input value?



1. Create an array to store data any type, you want (such as: int, char, float,…);
2. Create a function to show elements of array;
3. Create a function to insert an element to array to the position, which is input by user;
4. Create a function to delete an element (by position, which is input by user) from array;
5. Create a function to delete all elements with value, which is input by user, from array;
6. Create a function to search (non-duplicate and duplicate) element(s) in array by value, which is input by user.

Read book of **Robert Lafore**, page: 51 – 73 for next lecture

Outline

- Arrays
- Ordered Arrays
- The Bubble Sort
- The Insertion Sort

Linear Search

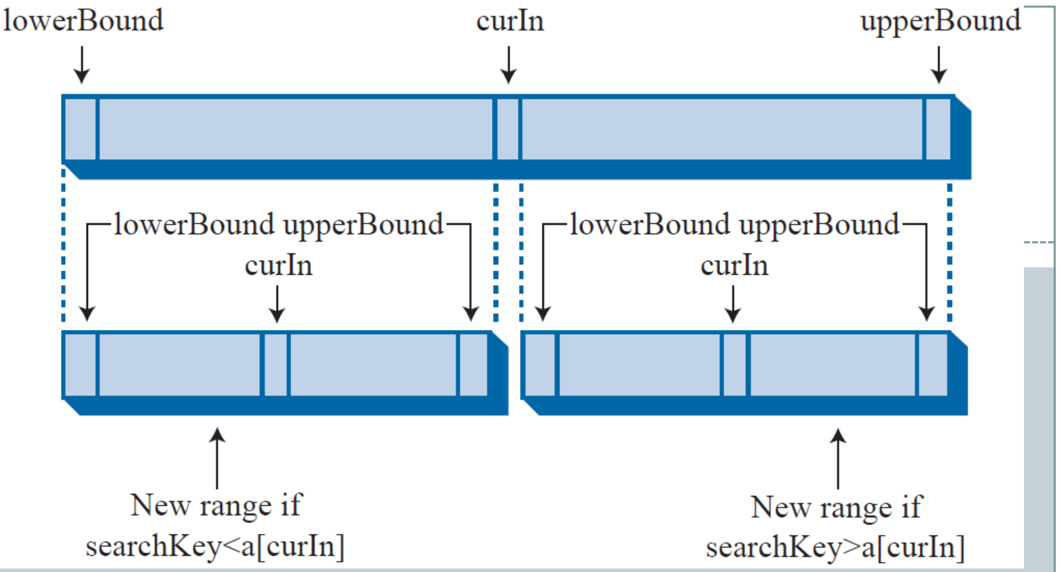
- Linear Search is used with unordered array
- Linear Search is checked value, we want to search with value of the 1st element, 2nd element, 3rd element, and so on
- Thus, on the average it would check about ½ of number of array

Ordered Arrays: Binary Search

- The Guess-a-Number Game
  - A friend asks you to guess a number she's thinking of between 1 and 100
  - When you guess a number (**recommend middle number**), she'll tell you one of three things: your guess is larger or smaller than the number she's thinking of
  - If she says it's too high, you deduce the number is between 1 and 49, so your next guess should be 25.

Step Number	Number Guessed	Result	Rang of Possible Value
0			1 – 100
1	50	Too High	1 – 49
2	25	Too Low	26 – 49
3	37	Too High	26 – 36
4	31	Too Low	32 – 36
5	34	Too High	32 – 33
6	32	Too Low	33 – 33
7	33	Correct	

Ordered Arrays: Binary Search



```
int find(int searchKey){
  int lowerBound = 0;
  int upperBound = nElement-1;
  int curIn;
  while(true){
   curIn = (lowerBound + upperBound ) / 2;
   if(v[curIn]==searchKey){
     return curIn; //found it
   }
   else if(lowerBound > upperBound)
     return nElement; //can’t found it
   else{                //divide range
     if(v[curIn] < searchKey)
       lowerBound = curIn + 1; //it’s in upper half
     else upperBound = curIn - 1; //it’s in lower half
   } //end else divide range
  } //end while
} //end find()
```

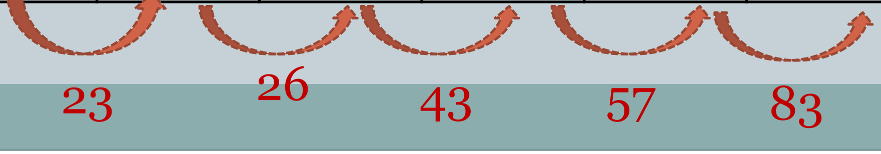
Ordered Array: Insertion

- Insert an element to Ordered Array, first of all, find where (position) by value to put it
- Insert element to the found position and move all bigger ones up (right)

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Value	12	15	23	26	43	57	83			

- Insert an element with value 21 into Array

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Value	12	15	21	23	26	43	57	83		



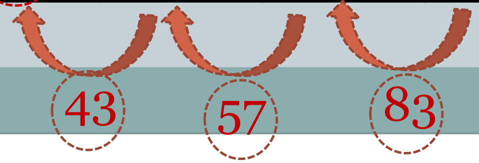
Ordered Arrays: Deletion

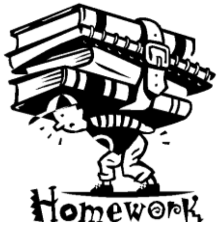
- Delete an element from array, first of all, find the deleted element in Ordered Arrays
- In case, the deleted element found, move bigger ones down (left)

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Value	12	15	23	26	43	57	83			

- Delete an element with value 26 from Array
  - ○ Using Binary Search to find element with value 26

Index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Value	12	15	23	43	57	83				





1. Create an ordered array to store data any type, you want (such as: int, char, float,…);
2. Create a function to show elements of the ordered array;
3. Create a function to insert an element to the ordered array;
4. Create a function to delete an element (deleted value, which is input by user) from the ordered array;
5. Create a function of **binary search** in the ordered array by value, which is input by user.

Read book of **Robert Lafore**, page: 51 – 73 for next lecture

The Running Time of a Program

- There are many algorithms to solve a problem, thus how to select a good algorithm?
- There are two different goals of algorithm selection:
  - An algorithm, which is easy to understand, code, and debug;
  - An algorithm, which makes efficient use of the computer's resources, especially one that run as fast as possible.
- So, how to select solution for problems?

Measuring Running Time of a Program

- The running time of a program depends on factors such as:
  - the input to the program,
  - the quality of code generated by the compiler used to create the object program,
  - the nature and speed of the instructions on the machine used to execute the program, and
  - the time complexity of the algorithm underlying the program.
- The input tells us that the running time of a program, thus time should be defined as a function of the input.

Measuring Running Time of a Program (cont.)

- Example: Linear Search, in searching process we input key and list of items to be searched by the key.
- Given a list of ***n*** item: 12, 35, 25, 67, 9, ...and key to be searched it the list: 88
- The searching process is to check, is there 88 in the item list?
- The process is to check one by one, from first to the end element
- Generally, ***T(n)*** – running time of a program on input of size ***n***.
- $T_{worst}(n)$ – *worst case* of running time, means that *the maximum*, over all inputs of size n, of the running time on the input.
- $T_{avg}(n)$ – *the average*, over all inputs of size n, of the running time on the input.
- $T_{best}(n)$ – *the best*, over all inputs of size n, of the running time on the input.

Measuring Running Time of Linear Search

```
                 1
searchKey = 66;                          //find item with key 66
for(j = 0; j < nElems; j++)                  //for each element,
       1      n+1          n
    if(arr[j] == searchKey)              //found item?
                 n (possible)
             break;                      //yes, exit before end
                 1
if(j == nElems)                          //at the end?
        1
    cout << “Can’t find “ << searchKey << endl; //yes
                                        1 (possible)
else
1 (possible)
    cout << “Found “ << searchKey << endl;      //no
                                   1 (possible)
```

Running time of Linear Search for this source code: **3n + ~~7~~8**

Big O Notation

- In Cambodia, automobiles are divided by engine power into several categories: 1.4, 1.6, 1.8, and so on.
- Similarly, efficient a computer algorithm is shortly called by **Big O** and kind of mathematical functions (constant, logarithmic, linear, quadric, polynomial or algebraic, exponential,…)

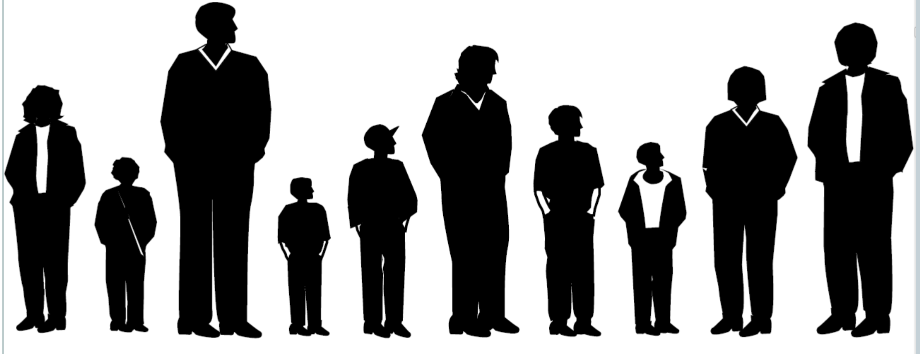
Running time of specific programs	**Big O or O**
$3n + 7$	$T(n) = \text{Big } O(n)$
$2n^2 + 4n + 5$	$T(n) = \text{Big } O(n^2)$
$4n^3 + 2n^2 - 3n + 20$	$T(n) = \text{Big } O(n^3)$
$3log_2n+4$	$T(n) = \text{Big } O(\log n)$
…	…

Outline

- Arrays
- Ordered Arrays
- The Bubble Sort
- The Insertion Sort

Bubble Sort[1]

- For what sorting?
  - Arrange names in alphabetical order, students by grade, customers by zip code,…
  - Sorting data is a preliminary step to search; E.g. applied Binary search to sorted data, is much faster then linear search
- For example of Unordered and Order data



The Unordered (by Height) Baseball Team

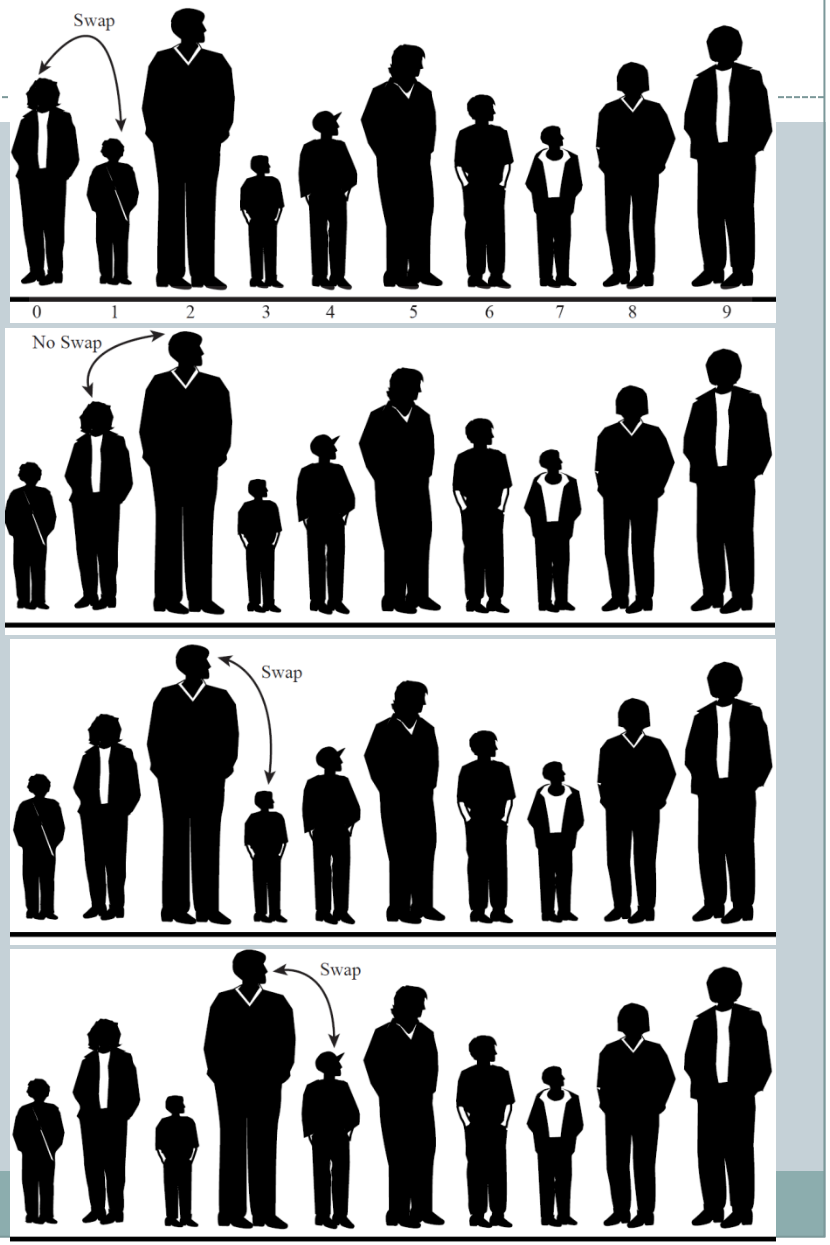
The Ordered (by Height) Baseball Team

Bubble Sort: How Does It Work?

1. Start from left end of line, and compare the two players in position *0* and 1;

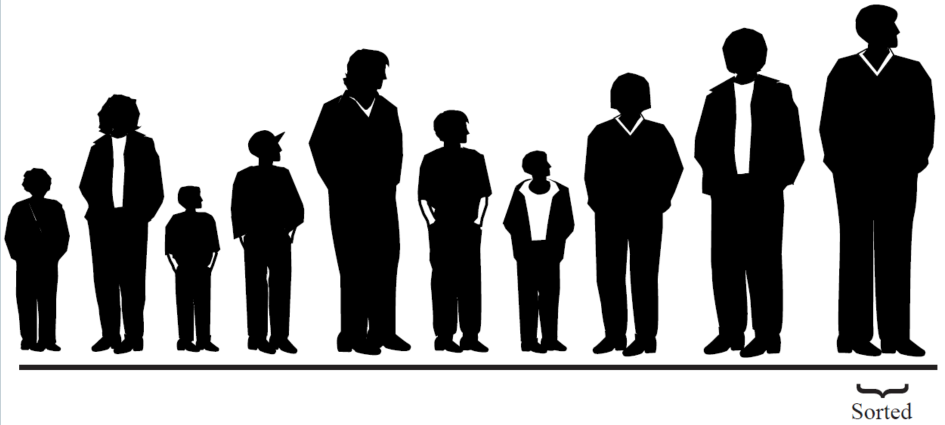
2. If the player on the left (in position 0) is taller, swap them. Otherwise, do nothing;

3. Then, move over one position and compare the players in positions 1 and 2. Again, if the player left is taller, swap them.



Bubble Sort: End of First Pass

- The tallest player will be swapped every time, compare two kids, until eventually he (or she) will reach the right end of the line
- After this first pass, it was *(n-1)* comparisons (where *n* – number of players), and somewhere *0* and *(n-1)* swaps
- The item at the end of the array is sorted and won't be moved again

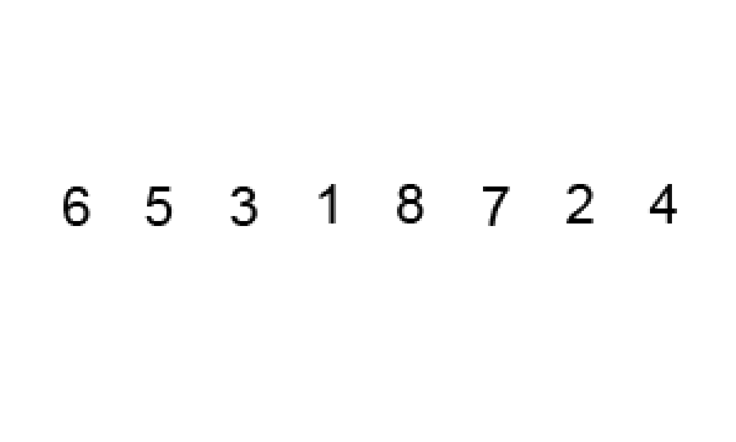


End of First Pass

- Then, go back to the left end of the line, compare player 0 and 1 again, and so on to the right. At this time (second pass), will stop at position *(n-2)*, because the last position (n-1) is the tallest. Continue this process until all the players are in order.

Bubble Sort: Code in C/C++

```
for( i=0; i<nElems-1; i++ ){          //number of sorted elements or ith pass
   for( j=0; j<nElems-i-1; j++ ){ //circle to take the tallest to the right end
      if( arr[j] > arr[j+1] ){     //if the left is taller?
         tmp = arr[ j ];               /*swap arr[ j ] and arr[ j+1 ] */
         arr[j] = arr[ j+1 ];
         arr[ j+1 ] = tmp ~~arr[ j ]~~;
      } //End if
   } //End j loop
}  //End i loop
```



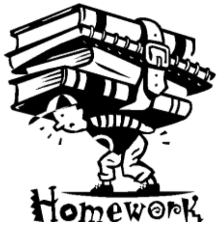
Please try with Bubble sort animation, here: http://cathyatseneca.github.io/DSAnim/web/bubble.html

Bubble Sort: Efficiency

- If n – number of items in the array
- So for the 1st pass there are *(n-1)* comparisons, 2nd pass there are *(n-2)* comparisons, and so on.
- Thus, total of comparisons is:

$$(n-1) + (n-2) + (n-3) + \ldots + 1 = n*(n-1)/2$$

- Because constants do NOT count in Big-O notation, we can say that the bubble sort runs in $O(n^2)$ time.



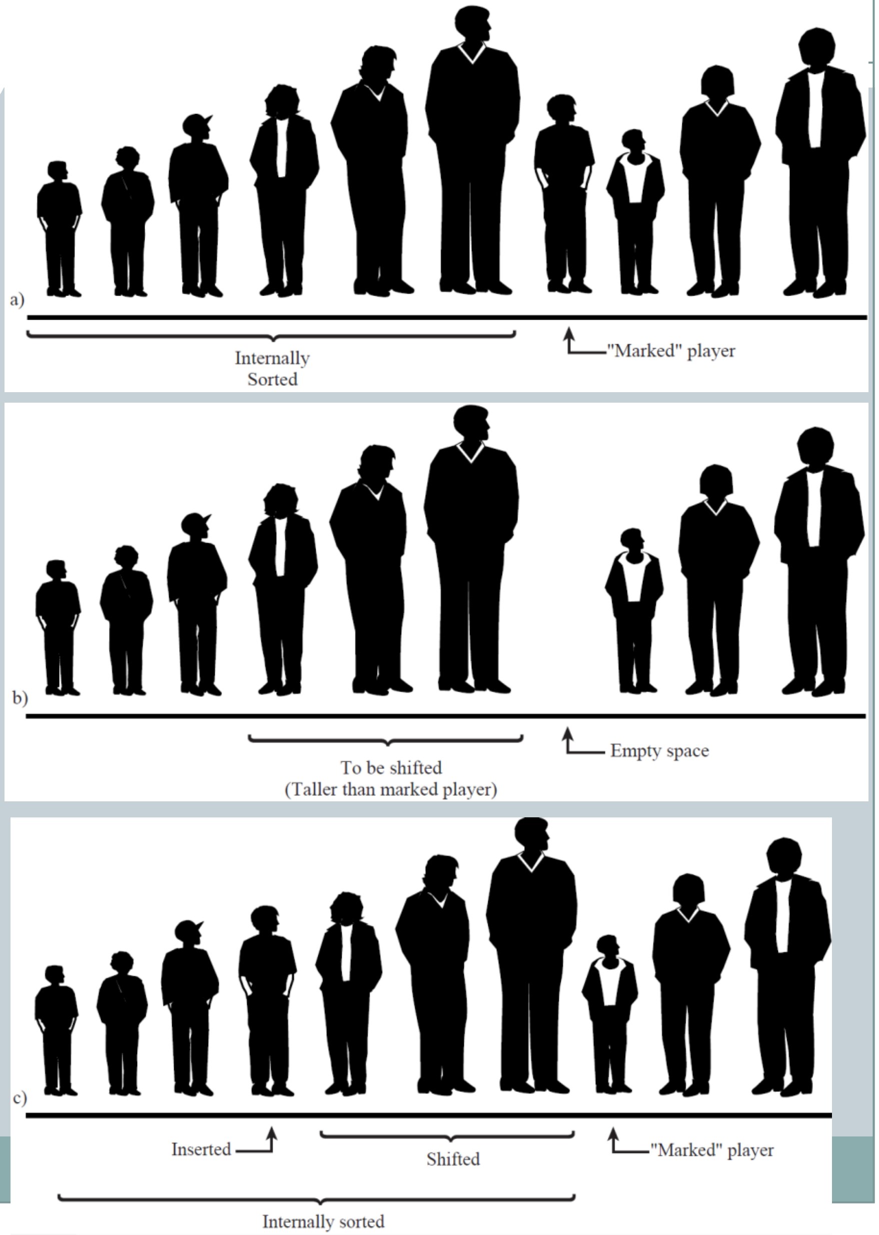
- Write a function, which will sort data of array by ascending or descending with Bubble sort algorithm (Option ascending or descending will be input by user);
- Explain your codes by comments;
- Draw flow chart of your source codes.

Outline

- Arrays
- Ordered Arrays
- The Bubble Sort
- The Insertion Sort

Insertion Sort: How Does It Work?

1. Marked player and take it out;
2. Find position for the marked player (by move the taller to right);
3. Insert the marked player to the found position;
4. Go to 1st point, until the end of right (last player).



Insertion Sort: Code in C/C++

```
for( i=1; i<nElems; i++){                      //marked ith element or divided by i
   tmp = arr[ i ];                             //remove marked item
   j = i;                                      //Start shifts at ith
   while( j>0 && arr[ j-1 ]>=tmp ){//Until the one is smaller
      arr[ j ] = arr[ j-1 ];                   //Otherwise, shift item to right
      --j;                                     //Go left one position
   }
   arr[ j ] = tmp;                             //insert marked item
}
```

Insertion Animations:

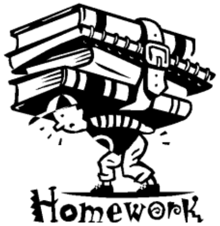
http://cathyatseneca.github.io/DSAnim/web/insertion.html
http://www.ee.ryerson.ca/~courses/coe428/sorting/insertionsort.html

Insertion Sort: Efficiency

- If n – number of items in the array
- How many comparisons does it require?
- For the 1st pass there is *1* comparison, 2nd pass there are *2* comparisons (maximum), and so on up to maximum *(n-1)*
- Thus, total of comparisons is:

$$1 + 2 + 3 + \ldots + (n-1) = n*(n-1)/2$$

- So, insertion sort runs in Big-O(n^2) time for random data.
- In case, the data is almost sorted, what is the run time of Bubble? And Insertion?



- Write a function, which will sort data of array by ascending or descending with Insertion sort algorithm (Option ascending or descending will be input by user);
- Explain your codes by comments;
- Draw flow chart of your source codes.