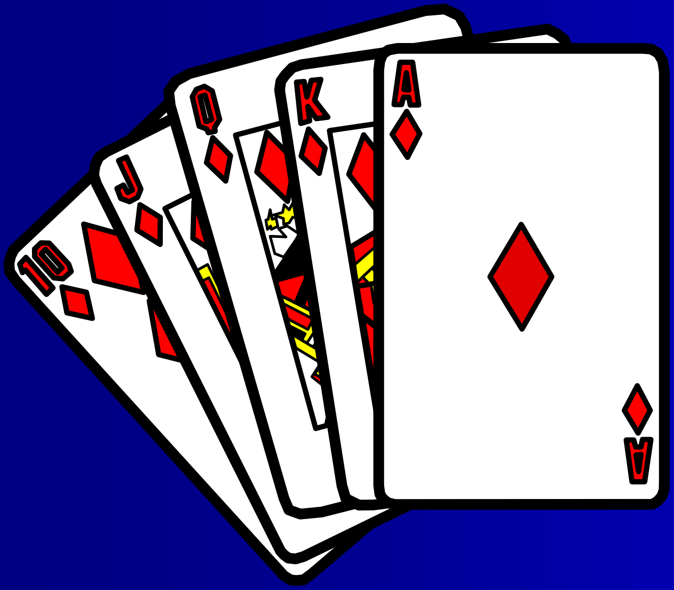
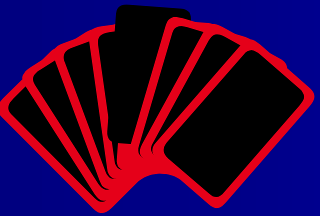


The Insertion Sort

Mr. Dave Clausen

La Cañada High School



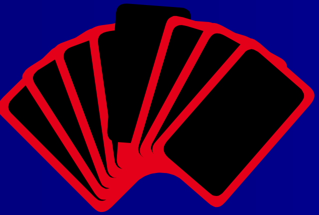
Insertion Sort Description

The *insertion sort* takes advantage an array's partial ordering and is the most efficient sort to use when you know the array is already partially ordered.

On the *k*th pass, the *k*th item should be inserted into its place among the first *k* items in the vector.

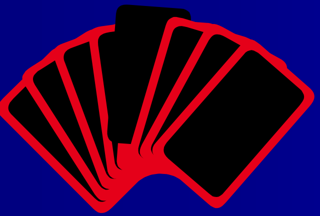
After the *k*th pass (*k* starting at 1), the first *k* items of the vector should be in sorted order.

This is like the way that people pick up playing cards and order them in their hands. When holding the first (*k* - 1) cards in order, a person will pick up the *k*th card and compare it with cards already held until its sorted spot is found.



Insertion Sort Algorithm

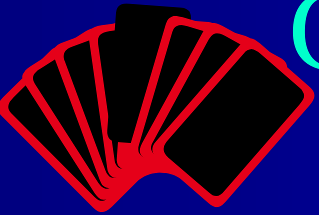
```
For each k from 1 to n - 1 (k is the index of vector element to insert)
  Set item_to_insert to v[k]
  Set j to k - 1
  (j starts at k - 1 and is decremented until insertion position is found)
  While (insertion position not found) and (not beginning of vector)
    If item_to_insert < v[j]
      Move v[j] to index position j + 1
      Decrement j by 1
    Else
      The insertion position has been found
  item_to_insert should be positioned at index j + 1
```



Java Code For Insertion Sort

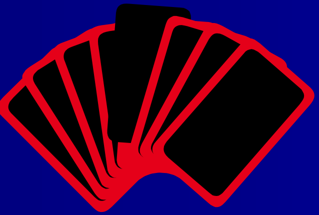
```
public static void insertionSort(int[ ] list){
    int itemToInsert, j; // On the kth pass, insert item k into its correct position among
    boolean stillLooking; // the first k entries in array.

    for (int k = 1; k < list.length; k++){
           // Walk backwards through list, looking for slot to insert list[k]
      itemToInsert = list[k];
      j = k - 1;
      stillLooking = true;
      while ((j >= 0) && stillLooking )
        if (itemToInsert  < list[j]) {
          list[j + 1] = list[j];
           j--;
        }else
          stillLooking = false;
         // Upon leaving loop, j + 1 is the index
        // where itemToInsert  belongs
        list[j + 1] = itemToInsert;
    }
  }
```



C ++ Code For Insertion Sort

```
void Insertion_Sort(apvector<int> &v){
  int item_to_insert, j;     // On the kth pass, insert item k into its correct
  bool still_looking;      // position among the first k entries in vector.
  for (int k = 1; k < v.length(); ++k)
  {     // Walk backwards through list, looking for slot to insert v[k]
    item_to_insert = v[k];
    j = k - 1;
    still_looking = true;
    while ((j >= 0) && still_looking )
      if (item_to_insert  < v[j])
      {
         v[j + 1] = v[j];
         --j;
       }
      else
        still_looking = false;      // Upon leaving loop, j + 1 is the index
      v[j + 1] = item_to_insert;   // where item_to_insert  belongs
   }
 }
```



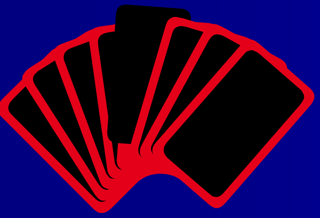
Insertion Sort Example

The Unsorted Vector:

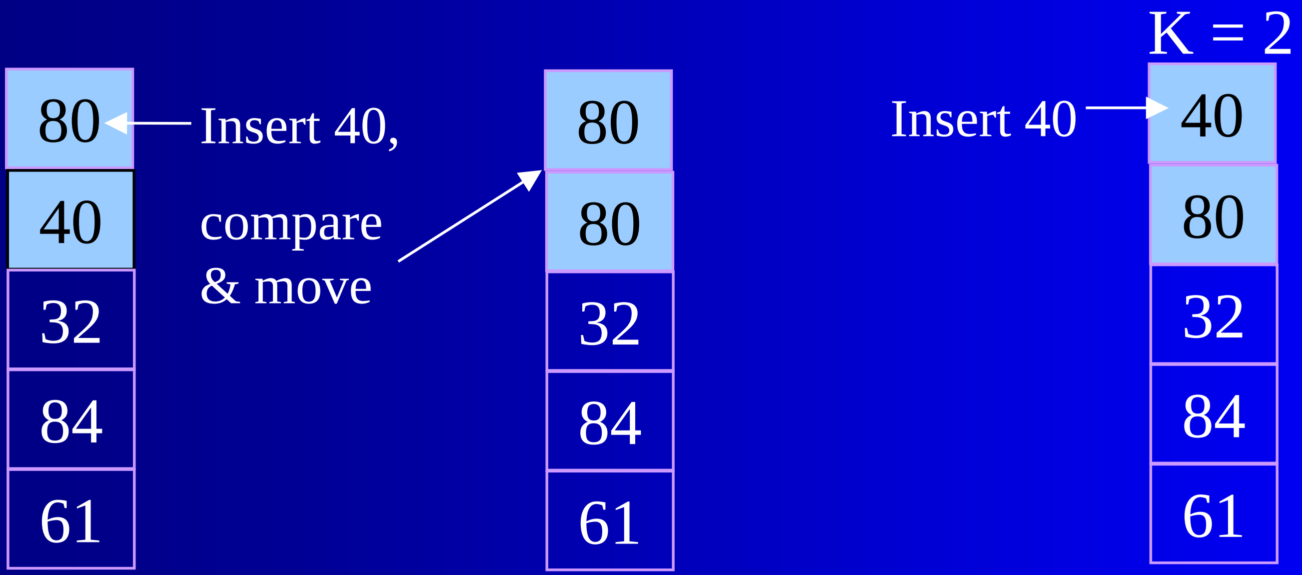
80
40
32
84
61

For each pass, the index *j* begins at the (*k* - 1)st item and moves that item to position *j* + 1 until we find the insertion point for what was originally the *k*th item.

We start with k = 1
and set j = k-1 or 0 (zero)



The First Pass



K = 2

80	80	40
40	80	80
32	32	32
84	84	84
61	61	61

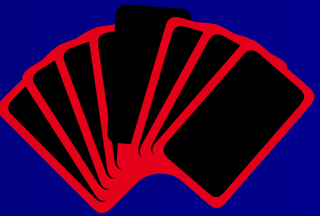
Insert 40,

compare
& move

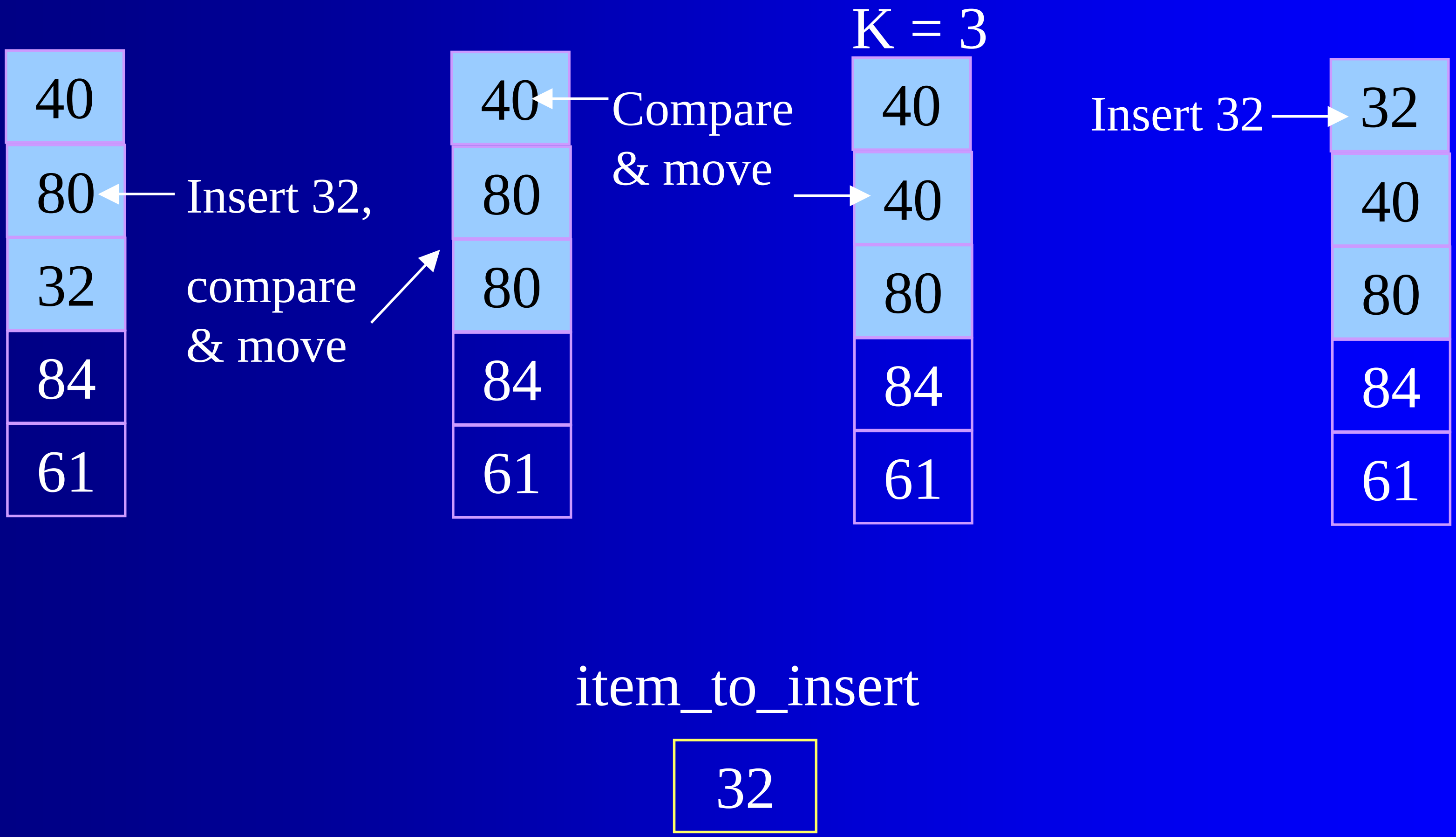
Insert 40

item_to_insert

40



The Second Pass



K = 3

40
80
32
84
61

Insert 32,

compare
& move

40
80
80
84
61

Compare
& move

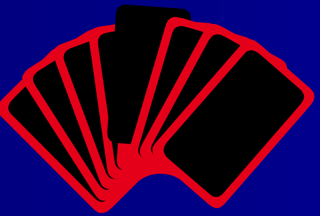
40
40
80
84
61

Insert 32

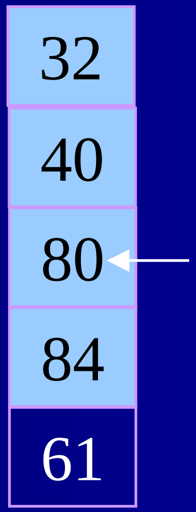
32
40
80
84
61

item_to_insert

32



The Third Pass



K = 4

32

40

80

84

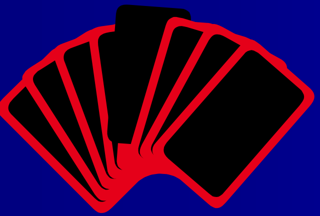
61

Insert 84?

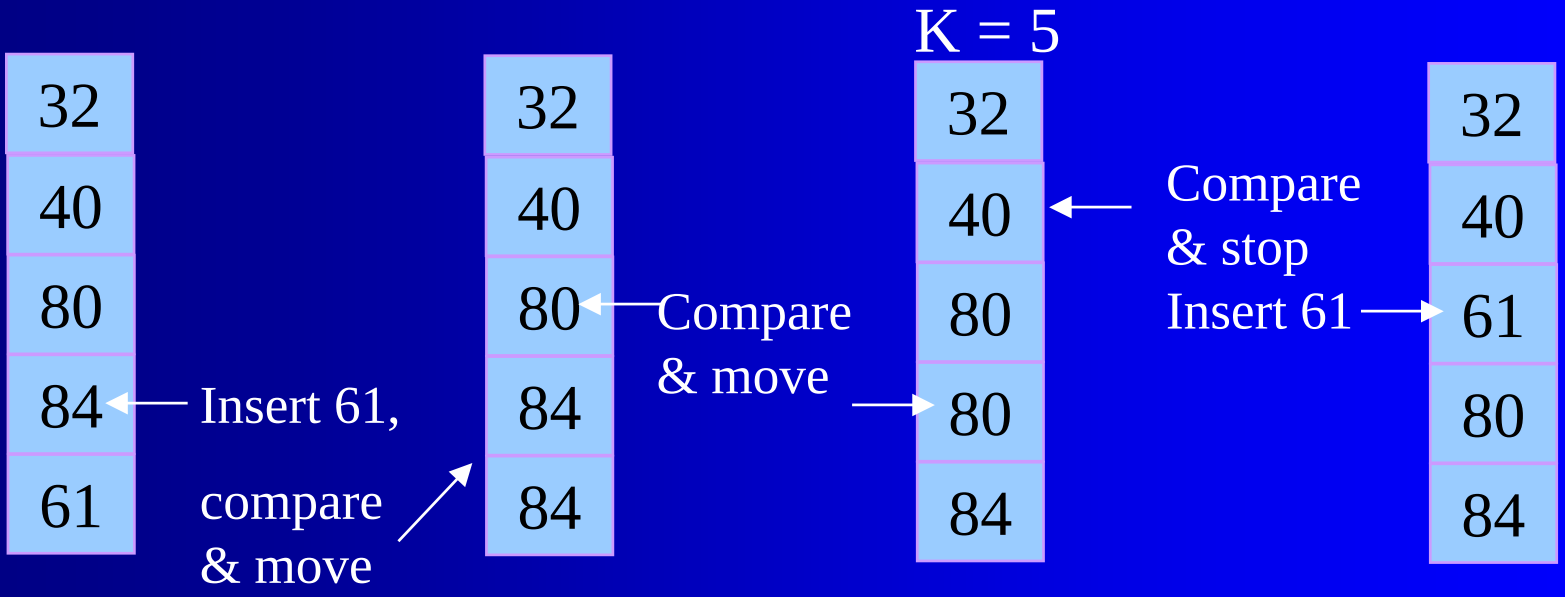
compare
& stop

item_to_insert

84



The Fourth Pass



32
40
80
84
61

Insert 61,

compare
& move

32
40
80
84
84

Compare
& move

K = 5

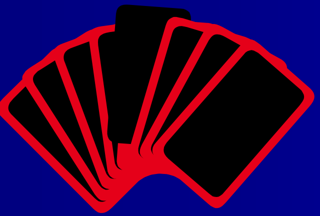
32
40
80
80
84

Compare
& stop
Insert 61

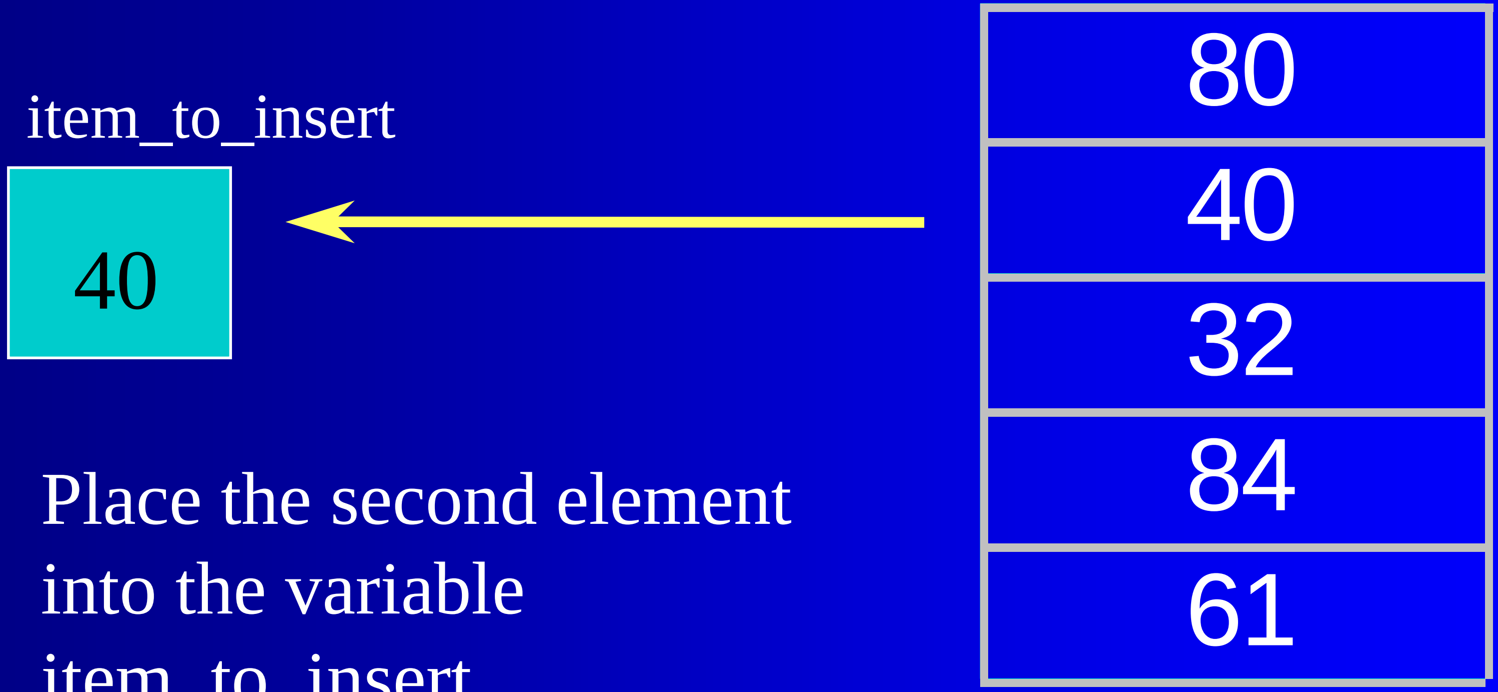
32
40
61
80
84

item_to_insert

61



What “Moving” Means

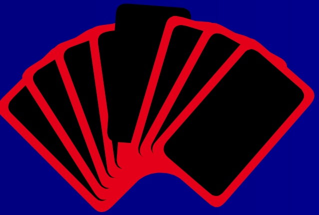


item_to_insert

40

80
40
32
84
61

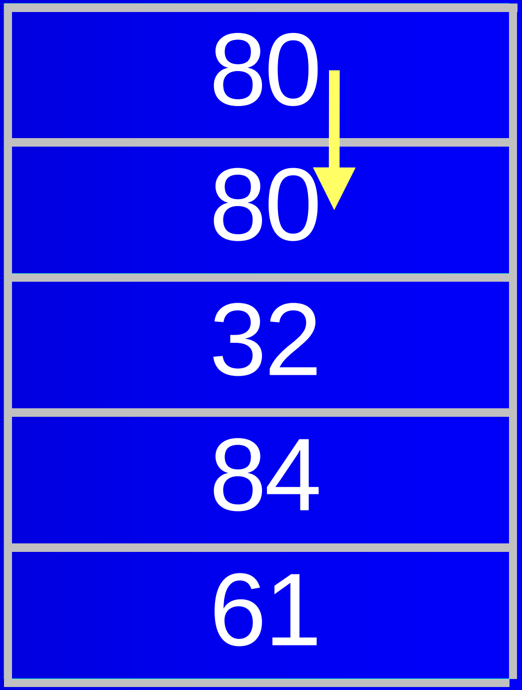
Place the second element into the variable item_to_insert.



What “Moving” Means

item_to_insert

40



80

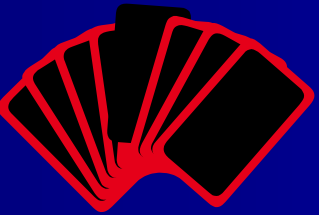
80

32

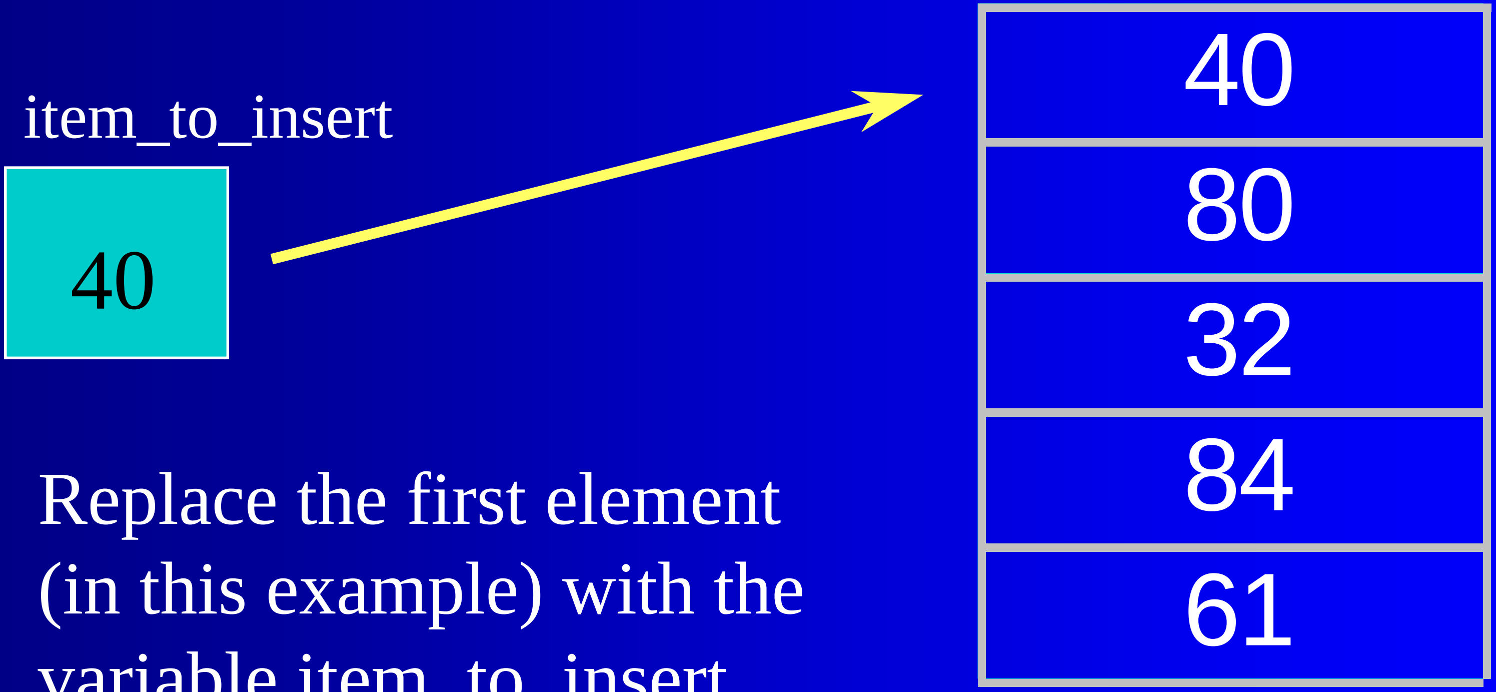
84

61

Replace the second element with the value of the first element.



What “Moving” Means

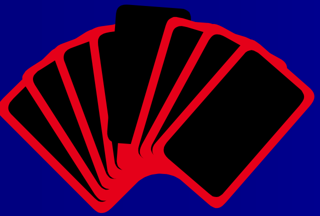


item_to_insert

40

40
80
32
84
61

Replace the first element (in this example) with the variable item_to_insert.



Big - O Notation

Big - O notation is used to describe the efficiency of a search or sort. The actual time necessary to complete the sort varies according to the speed of your system. Big - O notation is an approximate mathematical formula to determine how many operations are necessary to perform the search or sort. The Big - O notation for the Insertion Sort is $O(n^2)$, because it takes approximately n^2 passes to sort the “n” elements.