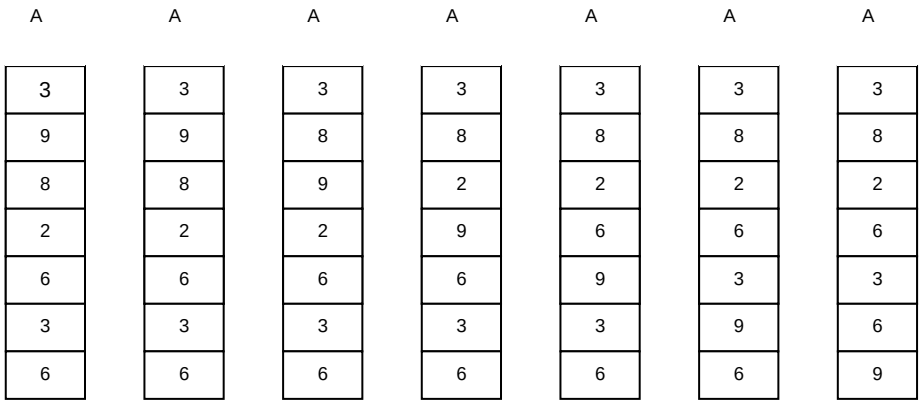
Sorting

- *Sorting* means arranging a group of data in an order, for ex., ordering some words alphabetically, or some numbers from smallest to largest, or from largest to smallest.

- Arranging numeric data in increasing order is named as sorting in *ascending* order, where arranging them in decreasing order is named as sorting in *descending* order. Arranging strings in alphabetical order is also named as sorting in *ascending* order.

Bubble Sort

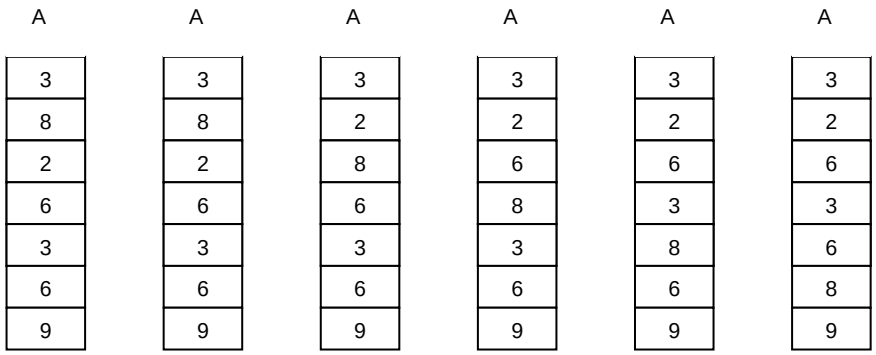
- Compare the first two elements, and exchange them if they are out of order, thus move the smallest one to top. Then compare the new second element with the third one, and exchange them if they are out of order. Go on this operation, thus compare two adjacent elements and order them, until you reach to the end of the array.



First Pass (6)

- When the first pass is completed, notice that the largest element moves to the bottom of the array.

- Now, repeat the same operations, but decrease the number of comparisons, because you don't need to compare the last element with the element before it, since you are sure that it is the largest element.



Second Pass (5)

- When the second pass is completed, notice that the second largest element moves to the bottom of the array. Thus, the number of necessary comparisons in the third pass will decrease to 4.

- Therefore, we can say that for an array of **n** elements, the number of pairs of elements compared in a particular pass is **n - pass** where **pass** is the number of current pass, starting with 1 for the first pass.

- Therefore, we can summarize the steps of Bubble Sort algorithm as follows:

  1. Pass over the whole array comparing two adjacent elements
  2. If the second one is less than the first, exchange these two elements
  3. Repeat the same operation **n - 1** times, making **n – pass** comparisons in each pass

- Let's define a separate function to exchange two elements of an array and use it in our function.

```
void swap(  int ar[],        // (input/output) given array
            int k, int j)    // indices of the elms to be exchanged
{
       int temp; // temporary variable
       temp = arr[k];
       arr[k] = arr[j];
       arr[j] = temp;
}
…
```

Remember that we declared another swap function in our lecture, which was exchanging values of two integer variables using pointers. Can we use it here? Yes, but when we are calling that function, we have to send the addresses of the array elements to be exchanged:

```
void swap(int *x, int *y)
{
       int temp;
       temp = *x;
       *x = *y;
       *y = temp;
}
...
```

- Let's define a function for Bubble Sort:

```
void bubble_sort(int list[], int n)
{
       int pass, k;
       for (pass = 1; pass <= n – 1; pass++)
            for (k = 0; k < n – pass; k++)
                 if (list[k] > list[k+1])
                       swap(&list[k], &list[k+1]);
}
```

- Notice that, this algorithm makes **n - 1** passes, and makes one less comparison in each pass. Thus, it makes (6+5+4+3+2+1=) 21 comparisons to sort our A array, which is exactly the same number of comparisons the Selection Sort algorithm did.

- We can modify the Bubble Sort algorithm so that it works faster than before. Notice that our array is sorted at the end of third pass, and no swap operations are done in the following passes. In fact, if no swap operation is done during a pass, it means that all elements are in the correct place, thus the array is sorted, and we can stop the operation in such a case.

```
void bubble_sort(int list[], int n)
{
	int  k,
	     pass,    // number of current pass
	     sorted;  /* flag to indicate whether sorting is
	                 finished or not */
	pass = 1;
	do
	{
	     sorted = 1; // assumes array is sorted
	     for (k = 0; k < n - pass; k++)
	          if (list[k] > list[k + 1])
	          {
	               swap(&list[k], &list[k + 1]);
	               sorted = 0;
	          }
	     pass++;
	} while (!sorted);
}
```

- Notice that, although the first Bubble Sort algorithm made 21 comparisons to sort our array, this algorithm makes only (6+5+4+3=) 18 comparisons, because it will stop at the end of the fourth pass.

String Sorting

- Let's modify our bubble_sort function so that it sorts an array of strings.
- First of all, we need to rewrite the swap function to swap two strings of maximum STR_LEN characters.

```
void swapStr(char *str1, char *str2) {
       char temp[STR_LEN];
       strcpy(temp, str1);
       strcpy(str1, str2);
       strcpy(str2, temp);
}
```

- Now, we can rewrite the buble_sort function to sort a list of n strings of maximum STR_LEN characters. Thus, we can represent it as a two-dimensional array with n rows and STR_LEN columns.

```
void bubbleSortStr(char list[][STR_LEN], int n)
{
            int  k,
            pass,   // number of current pass
            sorted; // flag to indicate whether sorting is
                    // finished or not
      pass = 1;
      do
      {
            sorted = 1; // assumes array is sorted
            for (k = 0; k < n-pass; k++)
                 if (strcmp(list[k], list[k + 1]) > 0)
                 {
                      swapStr(list[k], list[k + 1]);
                      sorted = 0;
                 }
            pass++;
      } while (!sorted);
}
```

Home Exercise:

Write a main that reads 5 words (each with maximum 20 characters) and displays them in alphabetically sorted order.

Example: Write a function to insert a value to the correct position of the sorted array.

Assume that, the list has already been sorted in ascending order. Your program gets some of numbers from the user until a negative value is entered. After each insert, the list must be in sorted form. Also, write a function to shift the values down to open a space to the correct position.

```
void shiftDown(int ar[], int size, int pos) {
	int k;
	for (k = size - 1; k >= pos; k--)
		ar[k + 1] = ar[k];
}

void insertSortedArr(int A[], int *sizeA, int num )
{
	int k = 0;
	/* until the end of A or B is reached */
	while (k < *sizeA && A[k] < num)
		k++;
	shiftDown (A, *sizeA, k);
	A[k] = num;
	(*sizeA)++;
}



int main(void) {
	int k, value, size = 5;
	int list[10] = { 26, 45, 67, 89, 91 };
	printf ("\n Enter a number:");
	scanf ("%d", &value);
	//insert the given number to the correct position
	while (value > 0) {
		insertSortedArr (list, &size, value);
		printf ("\n Enter a number:");
		scanf ("%d", &value);
	}

	// Display the sorted list
	for (k = 0; k < size; k++)
		printf ("%d\n", list[k]);
	return (0);

}
```