Sorting and Searching

● **Problem:**
  Read in a parameter value n, then read in a set of n numbers.
  Print the numbers in their original order. Sort the numbers into ascending order. Finally, print the numbers in sorted order.

  e.g.,

<table>
<thead>
<tr>
<th>Original Order</th>
<th>Sorted Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>125</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
</tr>
<tr>
<td>21</td>
<td>125</td>
</tr>
</tbody>
</table>
The Linear or Selection Sort

- **Algorithm for the Linear or Selection Sort:**
  
  for each position in the array (except the last)  
  for each candidate for that position  
  compare the candidate to the element currently in that position  
  if the candidate is smaller  
  swap them

- **Trace of the Linear Sort for the Above Example:**

- **A Function to Implement a Linear Sort:**

  ```c
  /* Function linearsort() */  
  /* Input: */  
  /* numb - array to be sorted */  
  /* n - number of elements to sort */  
  /* Process: */  
  /* linear sort */  
  /* Output: */  
  /* numb sorted into ascending order */  
  */

```c
void linearsort(int numb[], int n)
{
    int pass, cand;
    int temp;

    for (pass = 0; pass < (n-1); pass++)
        for (cand = (pass+1); cand < n; cand++)
            if (numb[pass] > numb[cand]) {
                temp = numb[pass];
                numb[pass] = numb[cand];
                numb[cand] = temp;
            }
    return;
}
```
Main Program:
/* program to illustrate linear sort */
#include <stdio.h>
define MAXSIZE 100

/* Function Prototypes */
void linesort(int [], int);
void printarray(int [], int);

void main()
{
    int numb[MAXSIZE];
    int i,n;

    printf("Enter the number of elements in the array: ");
    scanf("%d",&n);
    for (i = 0; i < n; i ++ ) {
        printf("Enter a number into the array: ");
        scanf("%d",&numb[i]);
    }
    printf("\nOriginal Data\n");
    printarray(numb,n);
    linesort(numb,n);
    printf("\nSorted Data\n");
    printarray(numb,n);
}

/* Function to print an array */
void printarray(int numb[], int n)
{
    int i;

    for (i = 0; i < n; i ++ )
        printf("%d\n",numb[i]);
    return;
}
The Bubble Sort

- **Algorithm for the Bubble Sort:**
  do the following as long as there has been a swap on the last pass  
  for each element of the array  
  compare the element to its neighbor  
  if they are out of order swap them

- **A Function to Implement a Bubble Sort:**
  /* Function bubblesort()  
   * Input:  
   * numb - array to be sorted  
   * n - number of elements to sort  
   * Process:  
   * bubble sort  
   * Output:  
   * numb sorted into ascending order  
   */  
  #define TRUE 1  
  #define FALSE 0  
  void bubblesort(int numb[], int n)  
  {  
    int pos, swapped;  
    int temp;  
    do {  
      swapped = FALSE; // initialize swapped to FALSE  
      for (pos = 0; pos < (n-1); pos++)  
        if (numb[pos] > numb[pos+1])  
          {  
            temp = numb[pos];  
            numb[pos] = numb[pos+1];  
            numb[pos+1] = temp;  
            swapped = TRUE; // indicate swap has occurred  
          }  
    } while (swapped);  
    return;  
  }
### Function to Sort Strings:

```c
void linearSort(char name[][25], int n) {
    int pass, cand;
    char temp[25];

    for (pass = 0; pass < (n-1); pass++)
        for (cand = (pass+1); cand < n; cand++)
            if (strcmp(name[pass], name[cand]) > 0) {
                strcpy(temp, name[pass]);
                strcpy(name[pass], name[cand]);
                strcpy(name[cand], temp);
            }
    return;
}
```

### Notes on Sorting Strings:
- "a" < "b"
- "Z" < "a"
- "Bob" < "bob"
- "Ccc" < "cc" < "ccc"
- "an" < "an" < "an"
Searching

- **Problem:**
  We have an array of names:
  - Johnson
  - Martin
  - Barnes
  - Brock

  Search for:
  - Barnes
  - Smith

  and return the position within the array (-1 if not found).
Linear Search

- **Pseudocode for Linear Search:**
  for each position in the array
  compare the element in that position to the search value
  if they are equal
    return the position in the array
  if no element is equal to the search value
    return a failure signal

- **Function to Implement a Linear Search:**
  /* Function linearsearch() */
  * Input:
  * numb - array to be searched
  * n - number of elements in the array
  * searchvalue - the value to search for
  * Process:
  * linear search
  * Output:
  * returns the position of searchvalue in the array
  * returns -1 if searchvalue not found
  */
  int linearsearch(int numb[], int n, int searchnumber)
  {
    int position;

    for (position = 0; position < n; position++)
      if (numb[position] = = searchnumber)
        return (position); // search value found
    return (-1); // search value not found
  }
Binary Search of Sorted Array

- Pseudocode for Binary Search:
  
  ```
  low = 0;
  high = n-1;
  while (low <= high)
    look at the element halfway between low and high
    if the element equals the search value
      return its position
    else if the element is larger than the search value
      high = the tested position - 1
    else
      low = tested position + 1
  ```

  e.g.,
  Abel
  Barnes
  Bishop
  Charles
  Chester
  Davis
  Fisher
Function to Implement a Binary Search:

/* Function binarysearch() */
* Input:
*    numb - array to be searched
*    n - number of elements in the array
*    searchvalue - the value to search for
* Process:
*    binary search
* Output:
*    returns the position of searchvalue in the array
*    returns -1 if searchvalue not found
* /

int binarysearch(int numb[], int n, int searchnumber)
{
    int low, high, test;

    low = 0;
    high = n - 1;
    while (low <= high) {
        test = (low + high) / 2;
        if (numb[test] == searchnumber)
            return (test); // search value found
        else if (numb[test] > searchnumber)
            high = test - 1;
        else
            low = test + 1;
    }
    return (-1); // search value not found
}