CSE101

Array searching and sorting techniques

Outline

- Introduction
- Linear search
- Binary search
- Bubble sort

Introduction

- The process of finding a particular element of an array is called ***SEARCHING***.
- Search an array for a *key* value, called **ITEM**.
- Searching techniques:
  - Linear search
  - Binary search

Linear search

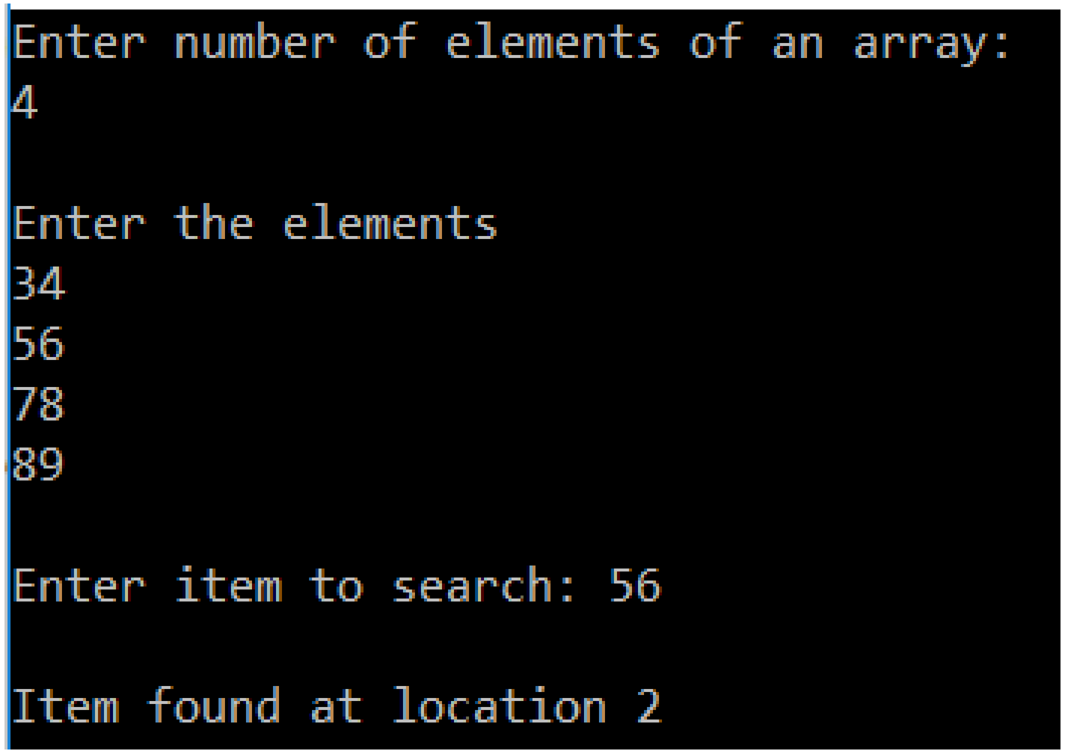
- Linear search
  - Simple
  - Compare each element of array with key value
  - Useful for small and unsorted arrays
- It simply examines each element sequentially, starting with the first element, until it finds the key element or it reaches the end of the array.

  Example: If you were looking for someone on a moving passenger train, you would use a sequential search.

Program of linear search in an array.

```
#include<stdio.h>
int main()
{
   int a[10], i, item, n;
   printf("Enter number of elements of an array:\n");
   scanf("%d",&n);

   printf("\nEnter the elements \n");
   for(i=0; i<n; i++)
              scanf("%d", &a[i]);

   printf("\nEnter item to search: ");
   scanf("%d", &item);

   for(i=0; i<10; i++)
   {
      if(item == a[i])
      {
         printf("\nItem found at location %d", i+1);
         break;
      }
   }
   if(i>9)
      printf("\nItem does not exist.");
                        return 0;
}
```

OUTPUT



```
Enter number of elements of an array:
4

Enter the elements
34
56
78
89

Enter item to search: 56

Item found at location 2
```

Binary Search

- Binary search
  - Applicable for **sorted** arrays
- The algorithm locates the **middle** element of the array and compares it to the key value.
  - Compares `middle` element with the `key`
    - If equal, match found
    - If `key < middle`, looks in left half of `middle`
    - If `key > middle`, looks in right half of `middle`
    - Repeat (the algorithm is repeated on **one-quarter** of the original array.)

Binary search

- It repeatedly divides the sequence in two, each time restricting the search to the half that would contain the element.
- This is a tremendous increase in performance over the linear search that required comparing the search key to an average of half of the array elements.
- You might use the binary search to look up a word in a dictionary

Program of binary search in an array

```
#include<stdio.h>
int main()
{
  int loc=-1, key, beg=0,last=9,mid;
  int a[10]={ 2,4,6,8,10,12,37,45,68,89};//sorted array
  printf("Enter integer value to search in sorted array:");
  scanf( "%d", &key );
   while(beg<=last)
   {
    mid = (beg + last) / 2; // determine middle element
    if(a[mid]==key)
    {
      loc=mid; //save the location of element.
      break;
    }
    else if(a[mid]>key) //key less than middle element
    {
      last=mid-1;
    }
    else if(a[mid]<key) //key greater than middle element
    {
      beg=mid+1;
    } //end of if else
   } //end of while
   if(loc!=-1)
   {
     printf("element found at %d", loc+1);
   }
   else
   {
     printf("element not found");
   }
   return 0;
}
```

output

```
Enter integer to search in sorted array:
4
Element found at 2


Enter integer search in sorted array:
85
Element not found
```

Sorting

- Sorting data
  - Important computing application
  - Virtually every organization must sort some data

Bubble Sort: Idea

- Idea: bubble in water.
  - Bubble in water moves upward. Why?
- How?
  - When a bubble moves upward, the water from above will move downward to fill in the space left by the bubble.

Bubble Sort Example

9, 6, 2, 12, 11, 9, 3, 7

Bubblesort compares the numbers in pairs from left to right exchanging when necessary. Here the first number is compared to the second and as it is larger they are exchanged.

Now the next pair of numbers are compared. Again the 9 is the larger and so this pair is also exchanged.

In the third comparison, the 9 is not larger than the 12 so no exchange is made. We move on to compare the next pair without any change to the list.

The 12 is larger than the 11 so they are exchanged.

The twelve is greater than the 9 so they are exchanged

The end of the list has been reached so this is the end of the first pass. The twelve at the end of the list must be largest number in the list and so is now in the correct position. We now start a new pass from left to right.

The 12 is greater than the 7 so they are exchanged.

Bubble Sort Example

First Pass 6, 2, 9, 11, 9, 3, 7, 12

Second Pass 2, 6, 9, 9, 3, 7, 11, 12

Notice that this time we do not have to compare the last two numbers as we know the 12 is in position. This pass therefore only requires 6 comparisons.

Bubble Sort Example

First Pass

6, 2, 9, 11, 9, 3, 7, 12

Second Pass

2, 6, 9, 9, 3, 7, 11, 12

Third Pass

2, 6, 9, 3, 7, 9, 11, 12

This time the 11 and 12 are in position. This pass therefore only requires 5 comparisons.

Bubble Sort Example

First Pass

6, 2, 9, 11, 9, 3, 7, 12

Second Pass

2, 6, 9, 9, 3, 7, 11, 12

Third Pass

2, 6, 9, 3, 7, 9, 11, 12

Fourth Pass

2, 6, 3, 7, 9, 9, 11, 12

Each pass requires fewer comparisons. This time only 4 are needed.

Bubble Sort Example

First Pass

6, 2, 9, 11, 9, 3, 7, 12

Second Pass

2, 6, 9, 9, 3, 7, 11, 12

Third Pass

2, 6, 9, 3, 7, 9, 11, 12

Fourth Pass

2, 6, 3, 7, 9, 9, 11, 12

Fifth Pass

2, 3, 6, 7, 9, 9, 11, 12

The list is now sorted but the algorithm does not know this until it completes a pass with no exchanges.

Bubble Sort Example

First Pass

6, 2, 9, 11, 9, 3, 7, 12

Second Pass

2, 6, 9, 9, 3, 7, 11, 12

Third Pass

2, 6, 9, 3, 7, 9, 11, 12

Fourth Pass

2, 6, 3, 7, 9, 9, 11, 12

Fifth Pa

This pass no exchanges are made so the algorithm knows the list is sorted. It can therefore save time by not doing the final pass. With other lists this check could save much more work.

Sixth Pass

2, 3, 6, 7, 9, 9, 11, 12

Bubble Sort Example

Quiz Time

1. Which number is definitely in its correct position at the end of the first pass?

Answer: The last number must be the largest.

2. How does the number of comparisons required change as the pass number increases?

Answer: Each pass requires one fewer comparison than the last.

3. How does the algorithm know when the list is sorted?

Answer: When a pass with no exchanges occurs.

4. What is the maximum number of comparisons required for a list of 10 numbers?

Answer: 9 comparisons, then 8, 7, 6, 5, 4, 3, 2, 1 so total 45

Bubble Sort

```
•       #include<stdio.h>
•       #include<conio.h>
•       #include<stdio.h>
•       void bubble_sort(int [],int);
•       int main()
•       {
•          int a[30],n,i;
•          printf("\nEnter no of elements :");
•          scanf("%d",&n);
•          printf("\nEnter array elements :");
•          for(i=0;i<n;i++)
•          scanf("%d",&a[i]);
•          bubble_sort(a,n);
•       }

•       void bubble_sort(int a[],int n)
•       {
•       int i,j,k,temp;
•         printf("\nUnsorted Data:\n");
•          for(k=0;k<n;k++)
•              printf("%5d",a[k]);
•          for(i=1;i< n;i++)
•          {
•              for(j=0;j< n-1;j++)
•                                              {
•                                              if(a[j]>a[j+1])
•                        {
•                        temp=a[j];
•                        a[j]=a[j+1];
•                                               a[j+1]=temp;
•                                                                                    }
•                                              }
•                        }
•                     printf("\nafter interchange\n");
•                      for(k=0;k< n;k++)
•                    printf("%5d",a[k]);
•                     getch();
•       }
```

Sort Set of string alphabetically

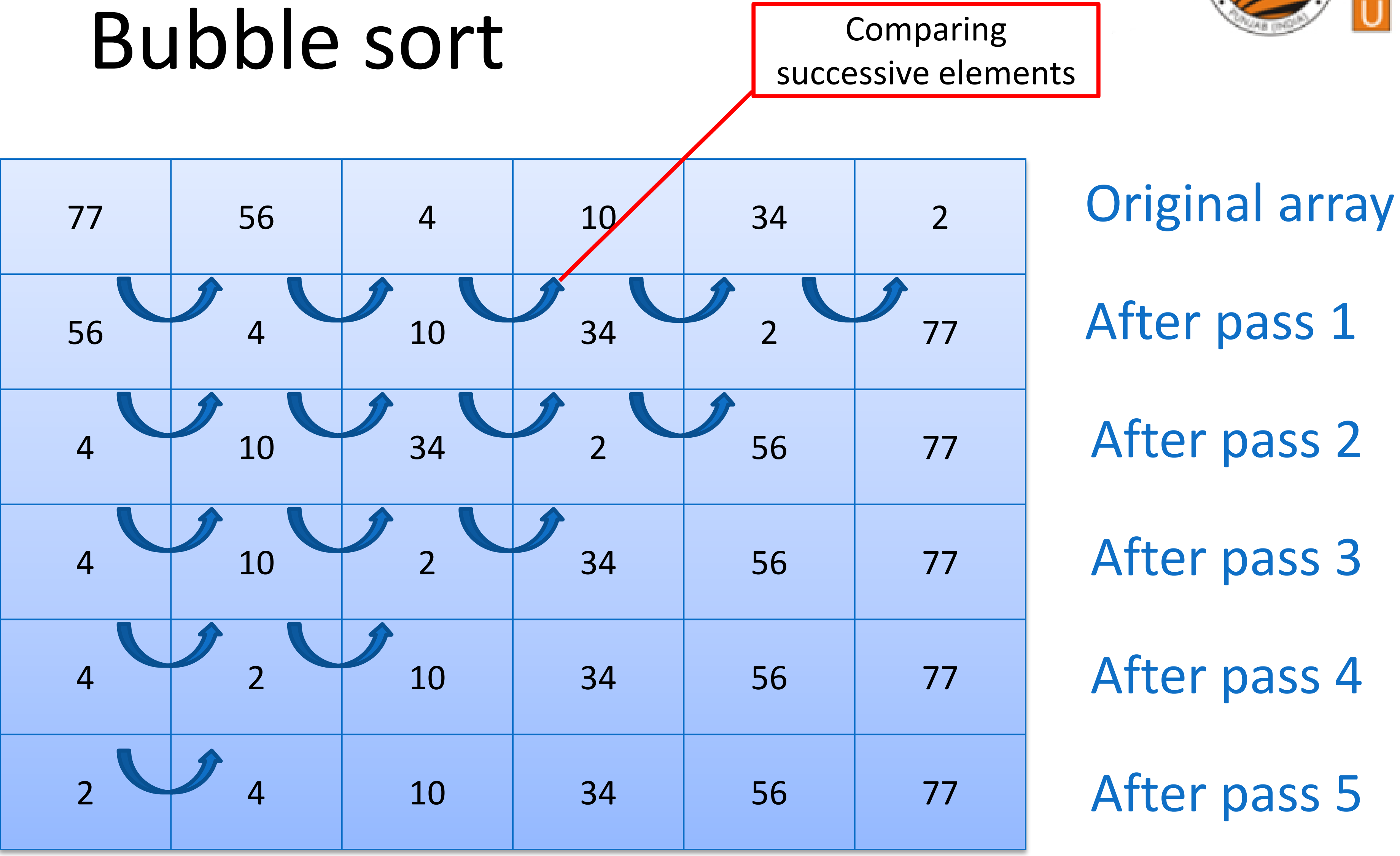
```
#include<stdio.h>
#include<string.h>
#include<conio.h>
int main()
{
char s[5][20],t[20];
int i,j;
//clrscr();
printf("Enter any five strings : \n");
for(i=0;i<5;i++)
   scanf("%s",s[i]);

for(i=1;i<5;i++)
   {
   for(j=1;j<5;j++)
      {
      if(strcmp(s[j-1],s[j])>0)
         {
         strcpy(t,s[j-1]);
         strcpy(s[j-1],s[j]);
         strcpy(s[j],t);
         }
      }
   }

printf("Strings in order are : ");
for(i=0;i<5;i++)
   printf("\n%s",s[i]);
getch();
}
```

Bubble sort

Bubble sort (sinking sort)

- A simple but inefficient sorting technique.
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements exchanged
  - Repeat

Bubble sort



77	56	4	10	34	2	Original array
56	4	10	34	2	77	After pass 1
4	10	34	2	56	77	After pass 2
4	10	2	34	56	77	After pass 3
4	2	10	34	56	77	After pass 4
2	4	10	34	56	77	After pass 5

Total number of pass required for sorting: n-1

Bubble sort

- It’s called bubble sort or sinking sort because smaller values gradually “bubble” their way to the top of the array (i.e., toward the first element) like air bubbles rising in water, while the larger values sink to the bottom (end) of the array.
- The technique uses nested loops to make several passes through the array.
  - Each pass compares successive pairs of elements.
  - If a pair is in increasing order (or the values are equal), the bubble sort leaves the values as they are.
  - If a pair is in decreasing order, the bubble sort swaps their values in the array.

Bubble sort

- The first pass compares the first two elements of the array and swaps their values if necessary. It then compares the second and third elements in the array. The end of this pass compares the last two elements in the array and swaps them if necessary.
- After one pass, the largest element will be in the last index. After two passes, the largest two elements will be in the last two indices.

BUBBLE SORT

```
#include <stdio.h>
int main()
{
  int a[10] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
  int pass, temp; // temporary location used to swap array elements
  printf( "Data items in original order" );
  for ( i = 0; i < 10; ++i ) {
    printf( "%4d", a[ i ] );
  } // end for
  for ( pass = 1; pass < 10; ++pass ) // loop to control number of passes
{
   // loop to control number of comparisons per pass
    for ( i = 0; i < 10 - 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

  printf( "\nData items in ascending order" );
  for ( i = 0; i < 10; ++i ) {
    printf( "%4d", a[ i ] );
  } // end for
} // end main
```

OUTPUT

```
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
```

Next Class: Defining and initializing strings

cse101@lpu.co.in