Arrays (Part 2)

Computer Programming 2
2nd Semester 2022-2023

TOPICS

Searching Arrays

Sorting Arrays

Searching Arrays: Linear Search and Binary Search

- Search an array for a *key value*
- *Binary search*
  - For sorted arrays
- *Linear search*
  - Useful for small and unsorted arrays

Searching Arrays: Linear Search

- Simple
- Compare each element of array with key value
- Useful for small and unsorted arrays

Searching Arrays: Linear Search

```
#include<stdio.h>

int main()
{
    int a[20],i,key,n;
    printf("How many elements? ");
    scanf("%d",&n);

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

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

    for(i=0;i<n;++i)
        if(a[i]==key)
            break;

    if(i<n)
        printf("\nElement found at index %d",i);
    else
        printf("Element not found");

    return 0;
}
```

Output

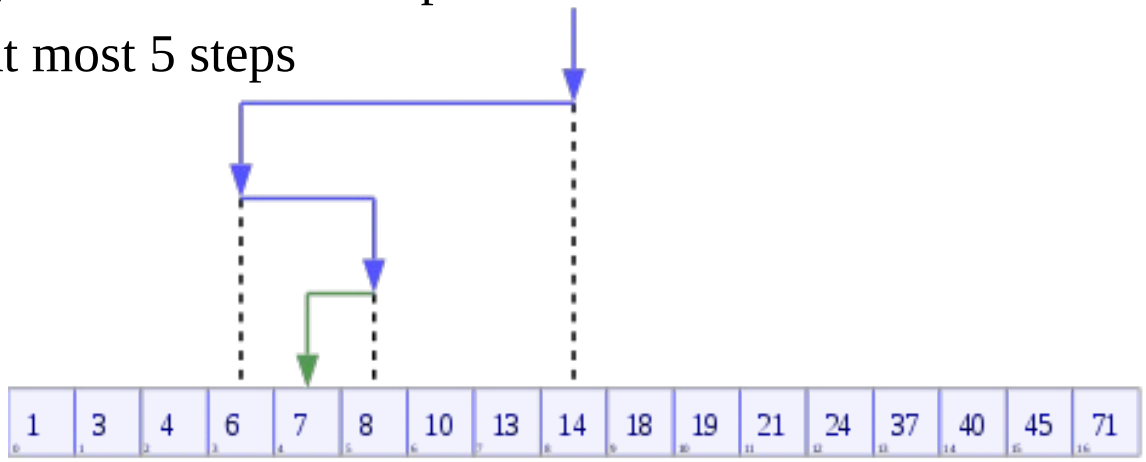
How many elements? 4
Enter array elements:
6 8 9 1

Enter element to search: 9

Element found at index 2

Searching Arrays: Binary Search

- For sorted arrays
- Compares **`middle`** element with **`key`**
  - If equal, match found
  - If **`key < middle`**, looks in first half of array
  - If **`key > middle`**, looks in last half
  - Repeat
- Very fast; at most n steps, where 2^n > number of elements
  - 30 element array takes at most 5 steps
    - 2^5 > 30 so at most 5 steps



Searching Arrays: Binary Search

```
#include<stdio.h>

int main()
{
    int
arr[50],i,n,key,flag=0,first,last,mid
;

    printf("Enter size of array:");
    scanf("%d",&n);
    printf("\nEnter array
element(ascending order)\n");

    for(i=0;i<n;++i)
        scanf("%d",&arr[i]);

    printf("\nEnter the element to
search:");
    scanf("%d",&key);
```

```
    first=0;
    last=n-1;

    while(first<=last)
    {
        mid=(first+last)/2;

        if(key==arr[mid]){
            flag=1;
            break;
        }
        else
            if(key>arr[mid])
                first=mid+1;
            else
                last=mid-1;
    }

    if(flag==1)
        printf("\nElement found at
position %d",mid+1);
    else
        printf("\nElement not found");

    return 0;
}
```

Enter size of array: 6

Enter array element(ascending order)
20 27 40 50 58 99

Enter the element to search: 27

Element found at position 2

Sorting Arrays

- Sorting data can be of utmost of importance. Especially in situations where you need data arranged in a specific order.
- **Sorting** an array is the ordering of the array elements in ***ascending*** (increasing -from min to max) or ***descending*** (decreasing – from max to min) order.

Example:

- {2 1 5 3 2} →{1, 2, 2, 3, 5} ***ascending*** order
- {2 1 5 3 2} →{5, 3, 2, 2, 1} ***descending*** order

Sorting Arrays

SORTING ALGORITHMS

- Selection sort
- Insertion Sort
- Bubble sort
- Merge sort
- Quicksort
- Heapsort

Sorting Arrays:Selection Sort

- This sorting algorithm, iterates through the array and finds the smallest number in the array and swaps it with the first element if it is smaller than the first element. Next, it goes on to the second element and so on until all elements are sorted.

- **Algorithm for Selection Sort:**

  1)Set min to the first location

  2)Search the minimum element in the array

  3)Swap the first location with the minimum value in the array

  4)Assign the second element as min.

  5)Repeat the process until we get a sorted array.

Sorting Arrays:Selection Sort

```
#include <stdio.h>
int main() {
    int a[100], n, i, j, position, swap;

    printf("Enter the size of array: ");
    scanf("%d", &n);
    printf("Enter the array elements: ", n);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

    for(i = 0; i < n-1; i++){
        position=i;
        for(j = i + 1; j < n; j++){
            if(a[position] > a[j])
                position=j;
        }
        if(position != i) {
            swap=a[i];
            a[i]=a[position];
            a[position]=swap;
        }
    }
    printf("Array after sorting: ");
    for(i = 0; i < n; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Enter the size of array: 4
Enter the array elements: 3 7 9 2

Array after sorting: 2 3 7 9

Sorting Arrays: Insertion Sort

- **Insertion Sort** is a sorting algorithm where the array is sorted by taking one element at a time. The principle behind insertion sort is to take one element, iterate through the sorted array & find its correct position in the sorted array.

- Insertion Sort works in a similar manner as we arrange a deck of cards.

Sorting Arrays: Insertion Sort

Algorithm for Insertion Sort:

1)If the element is the first one, it is already sorted.

2)Move to next element

3)Compare the current element with all elements in the sorted array

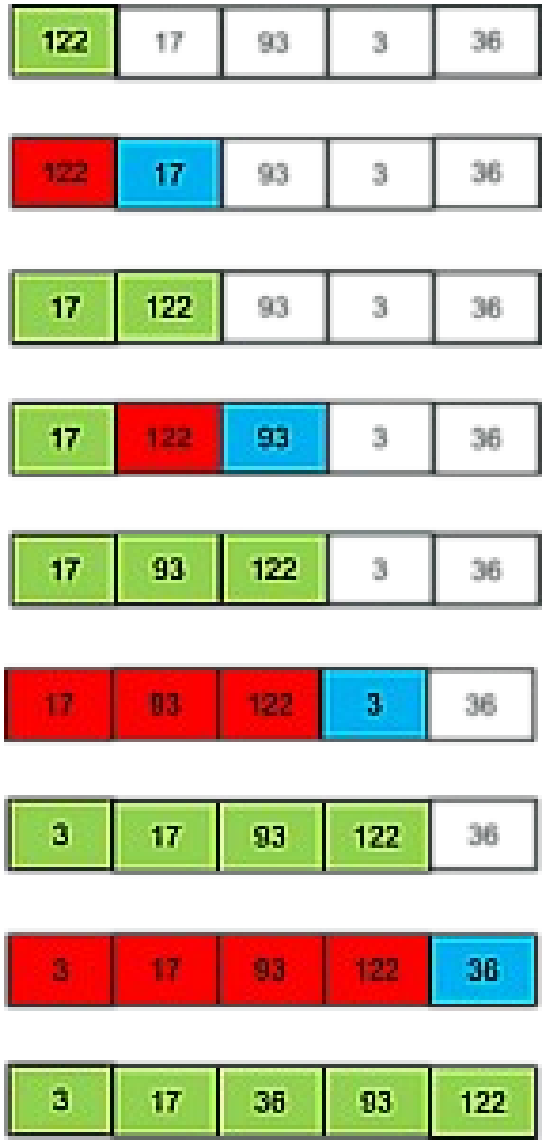
4)If the element in the sorted array is smaller than the current element, iterate to the next element. Otherwise, shift all the greater element in the array by one position towards the right

5)Insert the value at the correct position

6)Repeat until the complete list is sorted

Sorting Arrays: Insertion Sort

Example:



Sorting Arrays: Insertion Sort

```
/*Insertion Sort Function */

void insertionSort(int array[], int n)
{
    int i, key, j;

    for (i = 1; i < n; i++)
    {
        key = array[i];
        j = i - 1;

        while (j >= 0 && array[j] > key)
        {
            array[j + 1] = array[j];
            j = j - 1;
        }

        array[j + 1] = key;
    }
}
```

Sorting Arrays:Bubble Sort

- Smaller values in the list gradually “bubble” their way upward to the top of the array.

- The technique makes several passes through the array. On each pass successive pairs of elements are compared. If the pair is in increasing order (or equal) the pair is unchanged. If a pair is in descending order, their values are swapped in the array

Sorting Arrays:Bubble Sort

Pass = 1	Pass = 2	Pass = 3	Pass=4
2 1 5 3 2	**1 2** 3 2 5	**1 2** 2 3 5	**1 2** 2 3 5
1 **2 5** 3 2	1 **2 3** 2 5	1 **2 2** 3 5	1 **2 2** 3 5
1 2 **5 3** 2	1 2 **3 2** 5	1 2 **2 3** 5	1 2 **2 3** 5
1 2 3 **5 2**	1 2 2 **3 5**	1 2 2 **3 5**	1 2 2 **3 5**
1 2 3 2 5	1 2 2 3 5	1 2 2 3 5	1 2 2 3 5

- Underlined pairs show the comparisons. For each pass there are **size-1** comparisons.
- Total number of comparisons= **(size-1)²**

Sorting Arrays:Bubble Sort

```
#include<stdio.h>

int main()
{
    int a[50],n,i,j,temp;
    printf("Enter the size of array: ");
    scanf("%d",&n);
    printf("Enter the array elements: ");

    for(i=0;i<n;++i)
        scanf("%d",&a[i]);

    for(i=1;i<n;++i)
        for(j=0;j<(n-i);++j)
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }

    printf("\nArray after sorting: ");
    for(i=0;i<n;++i)
        printf("%d ",a[i]);

    return 0;
}
```

Enter the size of array: 4
Enter the array elements: 3 7 9 2

Array after sorting: 2 3 7 9

Sorting Arrays: Merge Sort

- **Merge sort** is one of the most powerful sorting algorithms.
- Merge Sort is one of the best examples of Divide and Conquer algorithm.
- In Merge sort, we divide the array recursively in two halves, until each sub-array contains a single element, and then we merge the sub-array in a way that it results into a sorted array. merge() function merges two sorted sub-arrays into one, wherein it assumes that array[l .. n] and arr[n+1 .. r] are sorted.

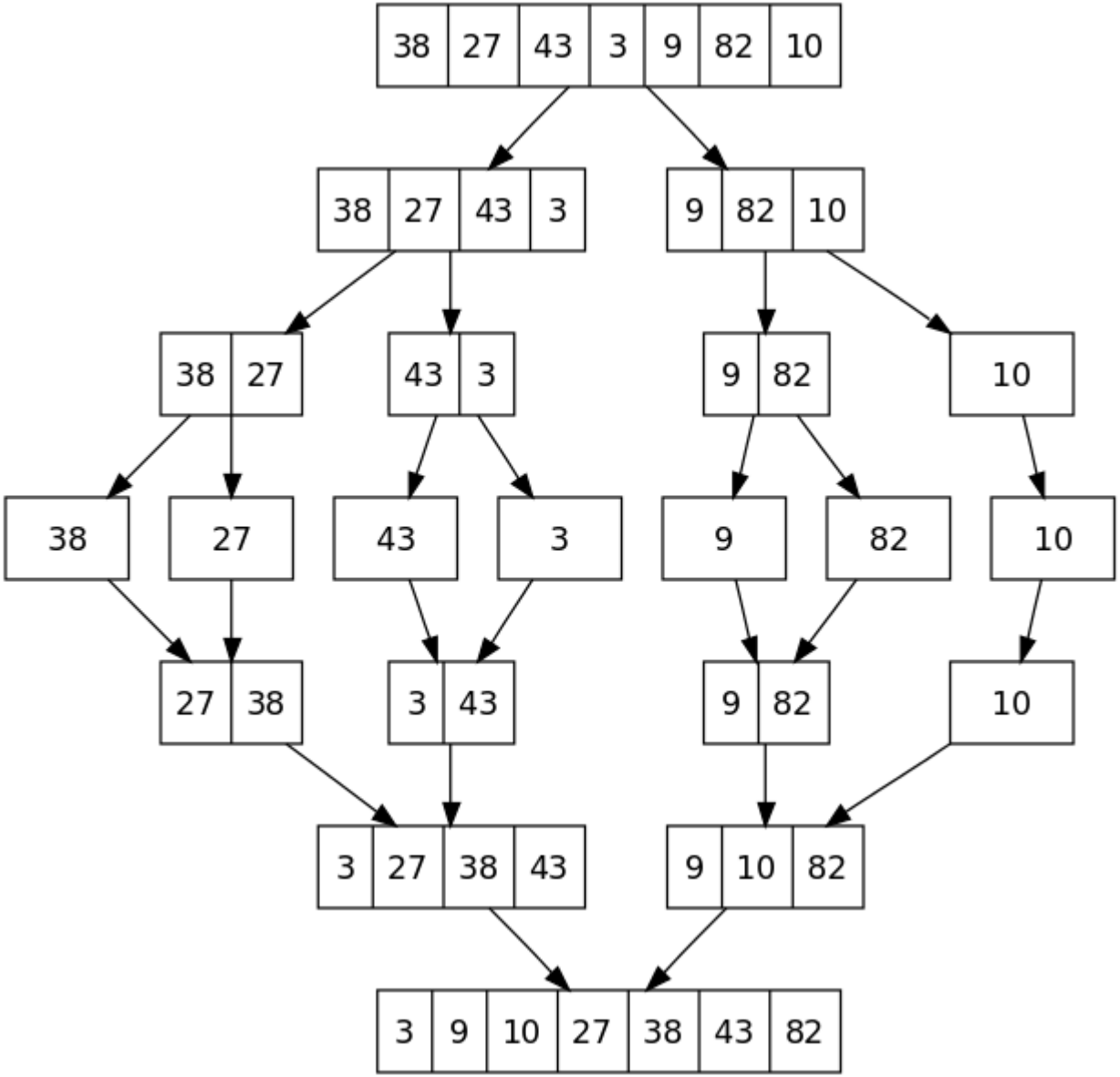
Sorting Arrays: Merge Sort

Algorithm for Merge Sort:

Given an array of length, say n, we perform the following steps to sort the array:

1) Divide the array into 2 parts of lengths n/2 and n-n/2 respectively (if n is odd, round off the value of n/2). Let us call these arrays as left half and right half respectively.
2) Recursively sort the left half array and the right half array.
3) Merge the left half array and right half-array to get the full array sorted.

Sorting Arrays: Merge Sort



Sorting Arrays: Merge Sort

```
#include<stdio.h>

void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);

int main()
{
    int a[30],n,i;
    printf("Enter no of elements:");
    scanf("%d",&n);
    printf("Enter array elements:");

    for(i=0;i<n;i++)
        scanf("%d",&a[i]);

    mergesort(a,0,n-1);

    printf("\nSorted array is :");
    for(i=0;i<n;i++)
        printf("%d ",a[i]);

    return 0;
}
```

Sorting Arrays: Merge Sort

```
void mergesort(int a[],int i,int j)
{
     int mid;

     if(i<j)
     {
          mid=(i+j)/2;
          mergesort(a,i,mid);        //left recursion
          mergesort(a,mid+1,j);     //right recursion
          merge(a,i,mid,mid+1,j);   //merging of two sorted sub-arrays
     }
}
```

Sorting Arrays: Merge Sort

```
void merge(int a[],int i1,int j1,int i2,int j2)
{
     int temp[50];    //array used for merging
     int i,j,k;
     i=i1;    //beginning of the first list
     j=i2;    //beginning of the second list
     k=0;

     while(i<=j1 && j<=j2)//while elements in both lists
     {
          if(a[i]<a[j])
               temp[k++]=a[i++];
          else
               temp[k++]=a[j++];
     }

     while(i<=j1) //copy remaining elements of the first list
          temp[k++]=a[i++];

     while(j<=j2) //copy remaining elements of the second list
          temp[k++]=a[j++];

     //Transfer elements from temp[] back to a[]
     for(i=i1,j=0;i<=j2;i++,j++)
          a[i]=temp[j];
}
```

End of Lecture 3