Advanced C & DS

Ch. 02 Sorting And Searching

Chapter : 3 [Syllabus]

- Sorting and Searching
  - Bubble sorting
  - Insertion sorting
  - Quick sorting
  - Bucket sorting
  - Merge sorting
  - Selection sorting
  - Shell sorting
  - Basic searching technique
  - Index searching
  - Sequential searching
  - Binary searching

Introduction of sorting

- Sorting is an algorithm that put elements of list in a certain order.
- Efficient sorting is important to improve the use of other algorithms that required sorted list to work correctly.

Types of Sorting Algorithm

- We can use various sorting algorithm as given.
  - Bubble Sort
  - Insertion Sort
  - Selection Sort
  - Quick Sort
  - Merge Sort
  - Bucket Sort
  - Shell Sort

Bubble Sort

- This sorting method is based on comparing and exchanging pairs of adjacent elements in array.
- One round of scanning through the set of elements of any array is called as passing elements.
- The bubble sort method derives it name from the fact that smallest data item bubbles up to the top of an array of **n** elements.

Bubble Sort

- The algorithm begins by comparing the bottom most elements with its adjacent element and exchanging of the two elements .
- If bottom most element is smaller than adjacent element after n-1 comparisons, smallest among the total number of n elements ascending to the top of the array.
- After the putting the smallest value at the top of the array ,leaving the first element, the remaining n-1 elements are scanned to find the next smallest elements in the array.

Bubble Sort : Algorithm

```
Step 1      :   [initialize]
                    i=0
                    n=no of elements in list
Step 2      :   [loop for pass]
                repeat through Step-5 while(i<n)
Step 3      :   [initialize count for internally]
                j=0
Step 4      :   [loop for pass]
                repeat through Step -5 while(j<n-i)
Step 5      :   [check for exchange]
                if(list[j]<list[j-1])
                    temp=list[j]
                    list[j+1=list[j]
                    list[j]=temp
Step 6      :   exit
```

Insertion Sort

- Insertion sort is a simple algorithm that is relative efficient for small lists and most – sorted lists often is used of more sophisticated algorithms.
- It works by taking elements from the list one by one and inserting them in their correct position into a new sorted list.
- Insertion is expensive requiring shifting all following over by one.

Insertion : Algorithm

Step 1 : [initialize]
a[0]=0
n=no of elements in list

Step 2 : repeat through Step-3 to Step-5
for i=0,1,…. N-1

Step 3 : temp=a[i]
pointer=i-1

Step 4 : while(temp<a[pointer])
a[pointer+1]=a[pointer]
pointer=pointer-1

Step 5 : a[pointer]=temp

Step 6 : exit

Insertion : Example

(3,30,6,16,2) = (3,30,6,16,2)

(3,30,6,16,2) = (3,30,6,16,2)

(3,6,30,16,2) = (3,6,30,16,2)

(3,6,16,30,2) = (3,6,16,30,2)

(2,3,6,16,30)

Selection Sort

- Selection sort is a simple sorting algorithm that improves on the performance of bubble sort.
- It works find smallest or largest elements using a linear scan and swapping it into the first position in the list.
- Then finding second largest elements by scanning the remaining elements and so on.

Selection Sort : Algorithm

Step 1 : [initialize]
Current=0
Size=no of elements in list

Step 2 : repeat through Step-7
While(Current<Size)

Step 3 : x=current+1

Step 4 : repeat through Step-6 while(x<Size)

Step 5 : if(a[current]>a[x])
temp=a[current]
a[current]=a[x]
a[x]=temp

Step 6 : x=x+1

Step 7 : current=current+1

Step 8 : Exit

Quick Sort :

- Quicksort is, on average, the fastest sorting algorithm for sorting collections with a large number of elements.
- ***QuickSort is recursive*** and also uses a "Divide and Conquer (વશ કરવું)" approach to sorting.

Quick Sort : Algorithm

```
Step 1  :       [initialize]
                        low=first
                        high=last
                        pivot=array[(low+high)/2]
Step 2  :       repeat through Step-7
                        While(low<=high)
Step 3  :       repeat step-4 while(array(low)<pivot)
Step 4  :       low=low+1
Step 5  :       repeat step-6 while(array(high)>pivot)
Step 6  :       high=high-1
Step 7  :       if(low<=high)
                                temp=array[low]
                                array[low]=array[high]
                                array[high]=temp
                                low=low+1
                                high=high-1
Step 8  :       if(first<high)
                        Quick_Sort(array,first,high)
Step 9  :       if(low<last)
                        Quick_Sort(array,low,last)
Step 10:        Exit
```

Quick Sort :

- EXAMPLE :
- (5,2,6,1,3,4)

Merge Sort :

- Merge sort takes advantage of the easy of merging already sorted lists into a new sorted list.
- It starts by comparing every two elements and swapping them if the first should come after the second.
- It then merges each of the resulting lists of into lists of fours, then merges those lists of four and so on. Until At last are merged into final sorted list.

Merge Sort : Algorithm

```
Step 1    :    [initialize]
                    i=0, j=0, k=0
Step 2    :    repeat through Step-3
                    While(i<n && j<m)
Step 3    :    if(list_a[i]<list_b[j])
                         result_list[k]=list_a[i]
                         i=i+1
                         k=k+1
               else if(list_a[i]>list_b[j])
                         result_list[k]=list_b[j]
                         j=j+1
                         k=k+1
               else
                         result_list[k]=list_a[i]
                         i=i+1
                         j=j+1
                         k=k+1
```

Merge Sort : Algorithm

Step 4 : [size of list_a is larger than the list_b]

if(i<n)

Step 5 : repeat through step-5 for i=1,i+1,i+2,.. m-1

result_list[k]=list[j]

Step 6 : [size of list_b is larger then list_a]

elseif(j<m)

Step 7 : repeat through step-7 for j=1,j+1,j+2,..m-1

result_list[k]=list[j]

j=j+1

k=k+1

Step 8 : return

Bucket Sort :

- Bucket sort, or bin sort is a sorting algorithm that works by partitioning an array into a number or buckets.
- Each bucket is then sorted individually, either using different sorting algorithm, or by recursively applying the bucket sorting algorithm.

Bucket Sort : Algorithm

Step 1 : Start

Step 2 : Initialize Length
n<-length[A]

Step 3 : Initialize I=1

Step4 : For i=1 to n do
Insert A[i] into list B[nA[i]]

Step 5 : For i=0 to n-1 do
Sort list B with Insertion sort

Step 6 : Concatenate the lists B[0],B[1]
… B[n-1] together in order

Step 7 : Stop

Bucket Sort : Algorithm

Step 1 : Start

Step 2 : Initialize Length
n<-length[A]

Step 3 : Initialize I=1

Step4 : For i=1 to n do
Insert A[i] into list B[nA[i]]

Step 5 : For i=0 to n-1 do
Sort list B with Insertion sort

Step 6 : Concatenate the lists B[0],B[1]
… B[n-1] together in order

Step 7 : Stop

Bucket Sort

(22,15,12,8,10,6,72,81,33,18,50,14)

Shell Sort :

- Shellsort is an in-place comparison sort.
- It generalizes an exchanging sort, such as insertion or bubble sort, such as insertion or bubble sort, by starting the comparison and exchange of elements with elements that are far apart before finishing with neighboring elements.
- Donald Shell published the first version of SHELL SORT in 1959.

Shell Sort : Algorithm

```
Step 1   :   Shell(item [], N)
Step 2   :   Initialize gap by N/2;
Step 3   :   While(gap)
             For(i=gap;i<n;i++)
Step4    :   Initialize X by item[i]
Step 5   :   For(j=i-gap;j>=0
                  && item[j]>x;j=j-gap)
Step 6   :   Update item[j+gap] by x;
Step 7   :   Update gap by gap/2;
Step 8   :   Stop
```

Searching Techniques

- There are two types of searching techniques.
  1. Linear search or Sequential Search
  2. Binary search

Linear search:

- This is the simplest technique to find out an element from an unsorted list.
- It simply traverses from top to bottom in the array and finds for the key value from the list and displays output as well.
- The value of the key is compared with the first element of the list.
- If match is found then an appropriate message is displayed and searching is done on remaining array elements.
- Otherwise next element is fetched and compared with key element and this process is continued till the key is found or array is completely traverse.

Linear search: Algorithm

```
Step 1   :   [Initialize]
             k=0
             fleg=1
Step 2   :   repeat step 3 for k=0,1,2…n-1
Step 3   :   if a[k]=element
             fleg=0
             output "Element Found ", k+1
Step4    :   if fleg=0
             output "Element Not Found"
Step 5   :   exit
```

Binary search:

- Binary Search is the most efficient technique to find the key from an array.
- Binary search only works on sorted lists and it becomes easy to find any information very fast.
- It is used to find location of a given element or record in a list.

Binary search : Algorithm

```
Step 1    :   [Initialize]
                  low=0
                  high=n-1
                  fleg=1
Step 2    :   repeat step 4 while(low<=high)
Step 3    :   mid=(low+high)/2
Step 4    :   if(element<l[mid]) then
                  high=mid-1
              elseif(element>l[mid]) then
                  low=mid+1
              elseif(element=l[mid])
                  output "Found",mid+1
                  flag=0
                  return
Step 5    :   if(flag) then
                  output "NotFound"
Step 6    :   exit
```