NAME:VISHAL CHAUHAN

PRN:2020BTECS00090

TOPIC:Searching & Sorting Technique

SUB:DSA

1->Searching tech

1-i linear search

#include<iostream>

using namespace std;

int main()

{

    int arr[]={45,12,35,68,47,95,15,29,89};

    int size=sizeof(arr)/sizeof(arr[0]);

    int search\_num;

    cout<<"Enter the Number to be Search:";

    cin>>search\_num;

    int flag=0;

    for(int i=0;i<size;i++)

    {

        if(arr[i]==search\_num)

        {

            cout<<"element "<<search\_num<<" is found at index:"<<i<<endl;

            flag=1;

            break;

        }

    }

    if(flag==0)

    cout<<"Element "<<search\_num<<" is not found!";

}

OUTPUT:

Enter the NUmber to be Search:56

Element 56 is not found!

1-II Binary Search

// search iteratevely

#include <iostream>

using namespace std;

void display(int arr[], int size)

{

    for (int i = 0; i < size; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

}

int binary\_search\_method1(int arr[], int l, int r, int find\_ele)

{

    int mid = (l + r) / 2;

    while (l <= r && arr[mid] != find\_ele)

    {

        if (arr[mid] > find\_ele)

        {

            r = mid - 1;

        }

        if (arr[mid] < find\_ele)

        {

            l = mid + 1;

        }

        mid = (l + r) / 2;

    }

    if (arr[mid] == find\_ele)

        return mid;

    else

        return -1;

}

int binary\_search\_method2(int arr[], int l, int r, int find\_ele)

{

    while (l <= r)

    {

        int mid = (l + r) / 2;

        if (arr[mid] == find\_ele)

            return mid;

        else if (arr[mid] < find\_ele)

            l = mid + 1;

        else if (arr[mid] > find\_ele)

            r = mid - 1;

    }

    return -1;

}

int main()

{

    int arr[] = {12, 23, 34, 56, 67, 78, 89, 99, 111, 222, 333, 444, 555, 666};

    int size = sizeof(arr) / sizeof(arr[0]);

    display(arr, size);

    int find\_ele = 666;

    // int search\_1\_ele\_pos = binary\_search\_method1(arr, 0, size, find\_ele);

    // if (search\_ele\_pos == -1)

    // {

    //     cout << "OPPS! element " << find\_ele << " not found!";

    // }

    // else

    // {

    //     cout << "position of Element " << find\_ele << " = " << search\_1\_ele\_pos;

    // }

    int search\_2\_ele\_pos = binary\_search\_method2(arr, 0, size, find\_ele);

    if (search\_2\_ele\_pos == -1)

    {

        cout << "OPPS! element " << find\_ele << " not found!";

    }

    else

    {

        cout << "position of Element " << find\_ele << " = " << search\_2\_ele\_pos;

    }

}

OUTPUT:

12 23 34 56 67 78 89 99 111 222 333 444 555 666

position of Element 666 = 13

1-III **Fibonacci Search**

//

#include<iostream>

using namespace std;

/\* Find min of given number \*/

int fibonaccianSearch(int arr[], int x, int n);

int min(int x, int y)

{

    return (x<=y)? x : y;

}

/\* Returns index of x if present,  else returns -1 \*/

int fibonaccianSearch(int arr[], int x, int n)

{

    /\* Initialize fibonacci numbers \*/

    int fbM2 = 0;   // (m-2)'th Fibonacci number

    int fbM1 = 1;   // (m-1)'th Fibonacci number

    int fbM = fbM2 + fbM1;  // m'th Fibonacci

    // Marks the eliminated range from front

    int offset = -1;

    /\* fbM is going to store the smallest Fibonacci

       number greater than or equal to n \*/

    while (fbM < n)

    {

        fbM2 = fbM1;

        fbM1 = fbM;

        fbM  = fbM2 + fbM1;

    }

    /\* while there are elements to be inspected. Note that

       we compare arr[fibM2] with x. When fbM becomes 1,

       fbM2 becomes 0 \*/

    while (fbM > 1)

    {

    // Check if fbM2 is a valid location

    int i = min(offset+fbM2, n-1);

    /\* If x is greater than the value at index fbM2,

       cut the subarray array from offset to i \*/

    if (arr[i] < x)

    {

        fbM  = fbM1;

        fbM1 = fbM2;

        fbM2 = fbM - fbM1;

        offset = i;

    }

    /\* If x is greater than the value at index fbMm2,

       cut the subarray after i+1  \*/

    else if (arr[i] > x)

    {

        fbM  = fbM2;

        fbM1 = fbM1 - fbM2;

        fbM2 = fbM - fbM1;

    }

    /\* element found. return index \*/

    else return i;

    }

    /\* comparing the last element with x \*/

    if(fbM1 && arr[offset+1]==x)

    return offset+1;

    /\*element not found. return -1 \*/

    return -1;

}

/\* main function \*/

int main()

{

    int l;

    cout<<"\nEnter the number of elements in array which should be less than 10:";

    cin>>l;

    int arr[10];

    cout<<"Enter array of elements for array:";

    for(int i=0;i<l;i++)

    {

    cin>>arr[i];

    }

    int n = sizeof(arr)/sizeof(arr[0]);

    int x;

    cout<<"\nEnter element to be searched :" ;

    cin>>x;

    cout<<"Found at index:"<<fibonaccianSearch(arr, x, n);

    return 0;

}

OUTPUT:

Enter the number of elements in array which should be less than 10:8

Enter array of elements for array:12 45 89 330 154 225 70 60

Enter element to be searched :154

Found at index:4

2-SORTING TECH

2-I ->Insertion Sort

#include <iostream>

using namespace std;

void display(int arr[], int size)

{

    for (int i = 0; i < size; i++)

    {

        cout << arr[i] << " ";

    }

    cout << "\n";

}

void swap(int &a, int &b)

{

    int temp = a;

    a = b;

    b = temp;

}

void insertion\_sort(int arr[], int size)

{

    cout << "insertion sort start...\n";

    for (int i = 1; i < size; i++)

    {

        // cout << "outer loop " << i << endl;

        int temp = arr[i];

        int j ;

        for (j = i - 1; j >= 0 && temp < arr[j]; j--)

        {

            // cout << "inner loop " << j << endl;

            arr[j + 1] = arr[j];

        }

        if (temp != arr[i])

        {

            // cout << "j=" << j << endl;

            arr[j + 1] = temp;

        }

    }

}

int main()

{

    int arr[] = {45, 12, 535, 122, 89, 57, 919, 68};

    int size = sizeof(arr) / sizeof(arr[0]);

    display(arr, size);

    insertion\_sort(arr, size);

    display(arr, size);

}

OUTPUT:

45 12 535 122 89 57 919 68

insertion sort start...

12 45 57 68 89 122 535 919

-----------------------------------

2-II->Shell Sort

#include<iostream>

using namespace std;

void display(int arr[],int size)

{

    for(int i=0;i<size;i++)

    {

        cout<<arr[i]<<" ";

    }

    cout<<endl;

}

void swap(int &a,int &b)

{

    int temp=a;

    a=b;

    b=temp;

}

void shell\_sort(int arr[],int size)

{

    int gap;

    for(gap=size/2;gap>=1;gap=gap/2)

    {

        for(int j=gap;j<size;j++)

        {

            for(int i=j-gap;i>=0;i=i-gap)

            {

                if(arr[i+gap]<arr[i])

                {

                    swap(arr[i+gap],arr[i]);

                }

                else

                {

                    break;

                }

            }

        }

    }

}

int main()

{

    int arr[]={12,566,4444,32,1558,67,43,88};

    int size=sizeof(arr)/sizeof(arr[0]);

    display(arr,size);

    shell\_sort(arr,size);

    display(arr,size);

}

2-III->Heap Sort

#include<iostream>

using namespace std;

void swap(int arr[],int i,int j)

{

    int temp=arr[i];

    arr[i]=arr[j];

    arr[j]=temp;

}

void max\_heapify(int arr[],int i,int size) //max\_heapify the i th element of array

{                                       //means heapify ith and ussse down wale node of tree per

    int largest=i;

    int left=i\*2+1;

    int right=i\*2+2;

    if(left<=size && arr[left]>arr[largest]) //checking that left wala exit karta h kya by left<=size

    {

        largest=left;

    }

    if(right<=size && arr[right]>arr[largest]) //checking that right wala exit karta h kya by right<=size

    {

        largest=right;

    }

    if(largest!=i) //check if root is not largest then swap

    {

        swap(arr,largest,i);

        max\_heapify(arr,largest,size);

    }

    else

    return ;

}

void  Build\_max\_heap(int arr[],int size)

{

    // as the leaf node is already heapify so don't need to heapify again them

    // since node without child is start from size/2-1 th index of array so we start from there as they are not heapify

    for(int i=size/2-1;i>=0;i--)

    {

        max\_heapify(arr,i,size);

    }

}

void display(int arr[],int size)

{

    for(int i=0;i<size;i++)

    {

        cout<<arr[i]<<" ";

    }

    cout<<endl;

}

void heap\_sort(int arr[],int size)

{

    max\_heapify(arr,0,size);

    for(int k=size;k>0;k--) //decreasing the heap size

    {

        swap(arr,0,k);

        size--; //swaping max\_\_heap\_val with last element of heap

        max\_heapify(arr,0,size);

    }

}

int main()

{

    int arr[]={23,56,45,78,89,12,68,49};

    int size=sizeof(arr)/sizeof(arr[0]);

    display(arr,size);

    Build\_max\_heap(arr,size-1);

    cout<<"displayin max Heap    :";

    display(arr,size);

    cout<<"displayin sorted array:";

    heap\_sort(arr,size-1);

    display(arr,size);

}

OUTPUT:

23 56 45 78 89 12 68 49

displayin max Heap :89 78 68 49 56 12 45 23

displayin sorted array:12 23 45 49 56 68 78 89

2-IV->Quick Sort

#include <iostream>

using namespace std;

void display(int arr[], int size)

{

    for (int i = 0; i < size; i++)

    {

        cout << arr[i] << " ";

    }

    cout << "\n";

}

void swap(int &a, int &b)

{

    int temp = a;

    a = b;

    b = temp;

}

int partition(int arr[], int low, int high);

void quick\_sort(int arr[], int low, int high)

{

    if (low < high)

    {

        int pi = partition(arr, low, high);

        quick\_sort(arr, low, pi - 1);

        quick\_sort(arr, pi + 1, high);

    }

}

int partition(int arr[], int low, int high)

{

    int pivot=arr[low]; //taking pivot elemetn as mid element

    int i=low;

    int j=high;

    while(i<j)

    {

        while(arr[i]<=pivot)i++;

        while(arr[j]>pivot)j--;

        if(i<j)

        swap(arr[i],arr[j]);

    }

    swap(arr[j],arr[low]);

    return (j);

}

int main()

{

    int arr[] = {45, 12, 535, 122, 89, 55557, 919, 68};

    int size = sizeof(arr) / sizeof(arr[0]);

    display(arr, size);

    quick\_sort(arr, 0, size - 1);

    display(arr, size);

}

OUTPUT:

45 12 535 122 89 55557 919 68

12 45 68 89 122 535 919 55557

2-V->Merge Sort

#include<iostream>

using namespace std;

void display(int arr[],int size)

{

    for(int i=0;i<size;i++)

    {

        cout<<arr[i]<<" ";

    }

    cout<<endl;

}

int b[15]={0};

void merge(int arr[],int l,int mid,int r)

{

    int i=l,k=l,j=mid+1;

    while(i<=mid && j<=r)

    {

        if(arr[i]>arr[j])

        {

            b[k++]=arr[j++];

        }

        else

        {

            b[k++]=arr[i++];

        }

    }

    if(i>mid)

    {

        while(j<=r)

        b[k++]=arr[j++];

    }

    if(j>r)

    {

        while (i<=mid)

        {

            b[k++]=arr[i++];

        }

    }

    for(int x=l;x<k;x++)

    {

        arr[x]=b[x];

    }

}

void merge\_sort(int \*arr,int l,int r)

{

    if(l<r)

    {

        int mid=(l+r)/2;

        merge\_sort(arr,l,mid);

        merge\_sort(arr,mid+1,r);

        merge(arr,l,mid,r);

    }

}

int main()

{

    int arr[]={35,26,12,34,18,377,98,65,59,38};

    int size=sizeof(arr)/sizeof(arr[0]);

    display(arr,size);

    merge\_sort(arr,0,size-1);

    display(arr,size);

}

OUTPUT:

35 26 12 34 18 377 98 65 59 38

12 18 26 34 35 38 59 65 98 377

2-VI->RADIX SORT

#include<iostream>

#include<list>

#include<cmath>

using namespace std;

void display(int \*array, int size) {

   for(int i = 0; i<size; i++)

      cout << array[i] << " ";

   cout << endl;

}

void radixSort(int \*arr, int n, int max) {

   int i, j, m, p = 1, index, temp, count = 0;

   list<int> pocket[10];      //radix of decimal number is 10

   for(i = 0; i< max; i++) {

      m = pow(10, i+1);

      p = pow(10, i);

      for(j = 0; j<n; j++) {

         temp = arr[j]%m;

         index = temp/p;      //find index for pocket array

         pocket[index].push\_back(arr[j]);

      }

      count = 0;

      for(j = 0; j<10; j++) {

         //delete from linked lists and store to array

         while(!pocket[j].empty()) {

            arr[count] = \*(pocket[j].begin());

            pocket[j].erase(pocket[j].begin());

            count++;

         }

      }

   }

}

int main() {

   int n, max;

   cout << "Enter the number of elements: ";

   cin >> n;

   cout << "Enter the maximum digit of elements: ";

   cin >> max;

   int arr[n]; //create an array with given number of elements

   cout << "Enter elements:" << endl;

   for(int i = 0; i<n; i++) {

      cin >> arr[i];

   }

   cout << "Data before Sorting: ";

   display(arr, n);

   radixSort(arr, n, max);

   cout << "Data after Sorting: ";

   display(arr, n);

}

OUTPUT:

Enter the number of elements: 7

Enter the maximum digit of elements: 555

Enter elements:

230

152

45

26

90

425

333

Data before Sorting: 230 152 45 26 90 425 333

Data after Sorting: 26 45 90 152 230 333 425

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*