**Assignment-2**

1. Write a program for insertion sort and display all the elements in each iteration and also compute the complexity analysis.

#include <stdio.h>

int main()

{

int n,temp,j;

scanf("%d",&n);

int a[n];

for(int i=0;i<n;i++){

scanf("%d",&a[i]);

}

for(int i=1;i<n;i++){

int p=a[i];

printf("\n");

for(int i=0;i<n;i++){

printf("%d ",a[i]);

}

for(j=i-1;j>=0;j--){

if(a[j]>p){

a[j+1]=a[j];

}

else{

break;

}

}

a[j+1]=p;

}

printf("\n");

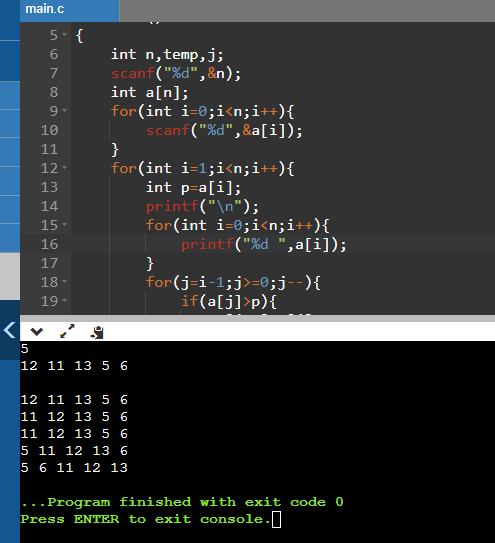
for(int i=0;i<n;i++){

printf("%d ",a[i]);

}

return 0;

}



Complexity Analysis:

Best case: O(N)

Average and Worst case: The no.of iterations are as follows:

For I = 1 2 3 4 5…….. N

J iterations: 1 2 3 4 5……… N

1+2+3+4+5+…..N=N\*(N+1)/2 = N^2 /2 + N/2 =O(N^2)

We will consider the higher powers and neglect the remaining while computing time complexity.

1. Write a program for quick sort and print all the elements in each iteration and also compute the complexity analysis

#include<stdio.h>

int n;

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

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

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

printf("\n");

for(int i=0;i<n;i++){

printf("%d ",arr[i]);

}

quickSort(arr, low, pivot - 1);

quickSort(arr, pivot + 1, high);

printf("\n");

for(int i=0;i<n;i++){

printf("%d ",arr[i]);

}

}

}

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

{

int pivot = arr[high];

int i = (low - 1),temp;

for (int j = low; j <= high- 1; j++)

{

if (arr[j] < pivot)

{

i++;

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

temp=arr[i+1];

arr[i+1]=arr[high];

arr[high]=temp;

return (i + 1);

}

int main()

{

scanf("%d",&n);

int arr[n];

for(int i=0;i<n;i++){

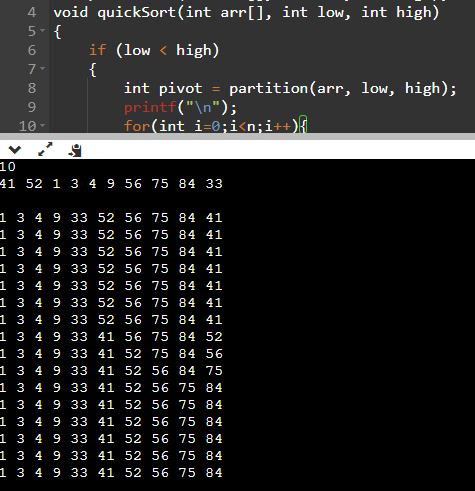
scanf("%d",&arr[i]);

}

quickSort(arr, 0, n - 1);

return 0;

}



Best and average time complexity: O(N\*log N)

Worst time complexity: O(N^2)

When the pivot is first or last element and the array is already sorted then the Worst time complexity is O(N^2), but if the pivot is the middle element then the worst case would be O(N\*log N)

1. Write a program for merge sort and print all the elements in each iteration and also compute the complexity analysis.

#include <stdio.h>

#include <stdlib.h>

int n;

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

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

printf("\n");

for(int p=0;p<n;p++){

printf("%d ",arr[p]);

}

for (i = 0; i < n1; i++)

L[i] = arr[i+1];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

int main()

{

scanf("%d",&n);

int arr[n];

for(int i=0;i<n;i++){

scanf("%d",&arr[i]);

}

mergeSort(arr, 0, n - 1);

printf("\n");

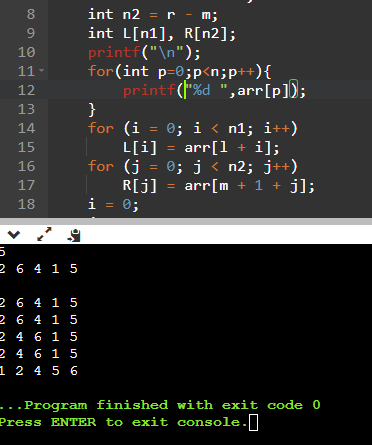
for(int p=0;p<n;p++){

printf("%d ",arr[p]);

}

return 0;

}



Complexity Analysis:

For dividing the unsorted array into two it takes O(N) time complexity.And after diving, merging the sub-problem solutions take O(N) time.

Total Time complexity: O(N\*log2 N)

1. Write a program for shell sort and print all the elements in each iteration and also compute the complexity analysis.

#include<stdio.h>

int shellSort(int arr[], int n)

{

for (int i = n/2; i > 0; i /= 2)

{

for (int j = i; j < n; j += 1)

{

int temp = arr[j];

int k;

for (k = j; k >= i && arr[k - i] > temp; k -= i){

arr[k] = arr[k - i];

}

arr[k] = temp;

printf("\n");

for(int p=0;p<n;p++){

printf("%d ",arr[p]);

}

}

}

return 0;

}

int main()

{

int n;

scanf("%d",&n);

int arr[n];

for(int i=0;i<n;i++){

scanf("%d",&arr[i]);

}

shellSort(arr, n);

printf("\n");

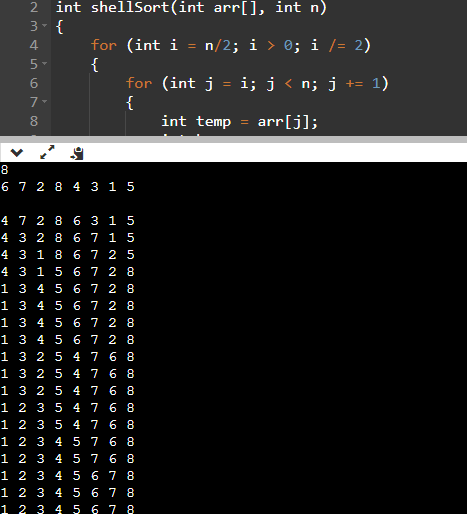
for(int i=0;i<n;i++){

printf("%d ",arr[i]);

}

return 0;

}



Complexity Analysis:

Best and average case:O(N\*Log N)

Worst case: O(N^2)

1. Write a program for radix sort and print all the elements in each iteration and also compute the complexity analysis.

#include<stdio.h>

int largest\_count(int[],int);

void radixsort(int[],int);

void main()

{

int i,n;

scanf("%d",&n);

int a[n];

for(i=0;i<n;i++){

scanf("%d",&a[i]);

}

radixsort(a,n);

for(i=0;i<n;i++){

printf("%d ",a[i]);

}

}

int largest\_count(int a[],int n)

{

int large=a[0],i,nop=0;

for(i=1;i<n;i++)

{

if(a[i]>large)

large=a[i];

}

while(large>0)

{

nop++;

large/=10;

}

return nop;// nop return the no.of digits in the greatest number

}

void radixsort(int a[],int n)

{

int bucket[10][10],bc[10];

int i,j,k,remainder;

int nop,divisor=1,large,pass,count;

nop=largest\_count(a,n);

for(pass=0;pass<nop;pass++)

{

for(i=0;i<10;i++)

bc[i]=0;

for(i=0;i<n;i++)

{

remainder=(a[i]/divisor)%10;

bucket[remainder][bc[remainder]]=a[i];

bc[remainder]+=1;

}

i=0;

for(k=0;k<10;k++)

{

for(j=0;j<bc[k];j++)

{

a[i]=bucket[k][j];

i++;

}

}

divisor\*=10;

printf("\n");

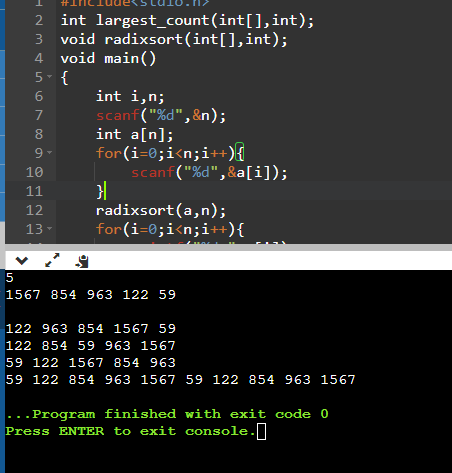
for(int p=0;p<n;p++){

printf("%d ",a[p]);

}

}

}



Complexity:

O(d\*(n+b)),d=no.of digits,n=number of elements,b=base value,here it is 10.

1. Finding minimum and maximum element in BST.

#include<stdio.h>

#include<stdlib.h>

struct node{

int key;

struct node\* left;

struct node\* right;

};

struct node \*insert(struct node \*root, int val)

{

if(root == NULL){

struct node \*temp = (struct node\*)malloc(sizeof(struct node));

temp->key = val;

temp->left = NULL;

temp->right = NULL;

return temp;}

if(root->key < val)

root->right = insert(root->right,val);

else if(root->key > val)

root->left = insert(root->left,val);

return root;

}

void min\_max(struct node \*root){

int max=root->key,min=root->key;

struct node\* ptr=root;

while(ptr->left!=NULL){

ptr=ptr->left;

}

min=ptr->key;

ptr=root;

while(ptr->right!=NULL){

ptr=ptr->right;

}

max=ptr->key;

printf("Minimum: %d\nMaximum: %d",min,max);

}

int main()

{

struct node \*root = NULL;

int n,val;

scanf("%d",&n);

for(int i=0;i<n;i++){

scanf("%d",&val);

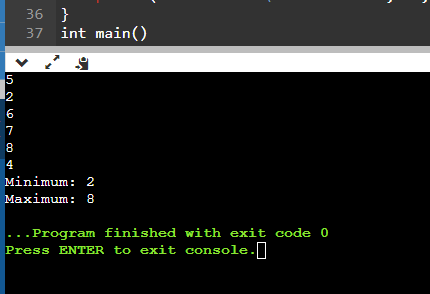
root=insert(root,val);

}

min\_max(root);

return 0;

}



1. Find kth smallest/largest element in BST.

#include<stdio.h>

#include<stdlib.h>

struct node{

int key;

struct node\* left;

struct node\* right;

};

struct node \*insert(struct node \*root, int val)

{

if(root == NULL){

struct node \*temp = (struct node\*)malloc(sizeof(struct node));

temp->key = val;

temp->left = NULL;

temp->right = NULL;

return temp;}

if(root->key < val)

root->right = insert(root->right,val);

else if(root->key > val)

root->left = insert(root->left,val);

return root;

}

int i=0,arr[100];

void inorder(struct node \*root)

{

if(root == NULL)

return;

inorder(root->left);

arr[i++]=root->key;

inorder(root->right);

}

int main()

{

struct node \*root = NULL;

int n,val,k;

scanf("%d",&n);

for(int i=0;i<n;i++){

scanf("%d",&val);

root=insert(root,val);

}

inorder(root);

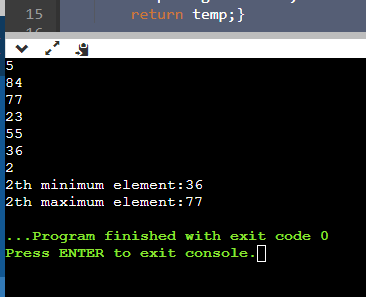
scanf("%d",&k);

printf("%dth minimum element:%d\n",k,arr[k-1]);

printf("%dth maximum element:%d",k,arr[n-k]);

return 0;

}



1. Write a program for converting the array to BST.

#include<stdio.h>

#include<stdlib.h>

struct node{

int key;

struct node\* left;

struct node\* right;

};

struct node \*insert(struct node \*root, int val)

{

if(root == NULL){

struct node \*temp = (struct node\*)malloc(sizeof(struct node));

temp->key = val;

temp->left = NULL;

temp->right = NULL;

return temp;}

if(root->key < val)

root->right = insert(root->right,val);

else if(root->key > val)

root->left = insert(root->left,val);

return root;

}

void inorder(struct node \*root)

{

if(root == NULL)

return;

inorder(root->left);

printf("%d ",root->key);

inorder(root->right);

}

int main()

{

struct node \*root = NULL;

int n;

scanf("%d",&n);

int arr[n];

for(int i=0;i<n;i++){

scanf("%d",&arr[i]);

}

for(int i=0;i<n;i++){

root=insert(root,arr[i]);

}

inorder(root);

return 0;

}

1. Write a program to remove duplicates in an array using BST.

#include<stdio.h>

#include<stdlib.h>

struct node{

int key;

struct node\* left;

struct node\* right;

};

int i=0,b[100];

struct node \*insert(struct node \*root, int val)

{

if(root == NULL){

b[i++]=val;

struct node \*temp = (struct node\*)malloc(sizeof(struct node));

temp->key = val;

temp->left = NULL;

temp->right = NULL;

return temp;}

if(root->key < val){

root->right = insert(root->right,val);

}

else if(root->key > val){

root->left = insert(root->left,val);

}

return root;

}

int main()

{

struct node \*root = NULL;

int n;

scanf("%d",&n);

int arr[n];

for(int i=0;i<n;i++){

scanf("%d",&arr[i]);

}

for(int i=0;i<n;i++){

root=insert(root,arr[i]);

}

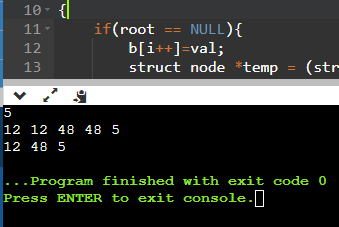
for(int j=0;j<i;j++){

printf("%d ",b[j]);

}

return 0;

}



1. Write a program to build a binary tree for the given elements and give traversal functions(pre-order, post order and in order)

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node\* left;

struct node\* right;

};

void inorder(struct node\* root){

if(root == NULL)

{

return;

}

inorder(root->left);

printf("%d ", root->data);

inorder(root->right);

}

void preorder(struct node\* root){

if(root == NULL)

{

return;

}

printf("%d ", root->data);

preorder(root->left);

preorder(root->right);

}

void postorder(struct node\* root) {

if(root == NULL)

{

return;

}

postorder(root->left);

postorder(root->right);

printf("%d ", root->data);

}

struct node\* createNode(int value)

{

struct node\* newNode = malloc(sizeof(struct node));

newNode->data = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

void main()

{

struct node\* root = createNode(1);

root->left=createNode(12);

root->right=createNode(9);

root->left->left=createNode(10);

root->left->right=createNode(15);

printf("Inorder traversal \n");

inorder(root);

printf("\nPreorder traversal \n");

preorder(root);

printf("\nPostorder traversal \n");

postorder(root);

}

