Ada Lab

1.linear serch

#include <stdio.h>

#include<stdlib.h>

void main(){

int a[1000],i,key,n,t=0,flag=0;

printf("enter the size of the array\n");

scanf("%d",&n);

printf("enter the array elements\n");

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

a[i]=rand()%1000;

}

printf("array elements are\n");

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

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

}

printf("enter the key to search\n");

scanf("%d",&key);

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

t+=3;

if(a[i]==key){

t+=1;

flag=1;

t+=1;

break;

}

}

if(flag==1){

printf("element found\n");

}

else{

printf("element not found");

}

printf("time taken is %d:",t);

}

2 Binary Search Without Recursive

#include <stdio.h>

#include<stdlib.h>

void main(){

int a[1000],i,key,n,l=0,h,mid,t=0,flag=0,temp,j;

printf("enter the size of the array\n");

scanf("%d",&n);

h=n-1;

printf("enter the array elements\n");

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

a[i]=rand()%1000;

}

printf("array elements are\n");

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

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

}

for (i = 0; i < n-1; i++) {

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

if (a[i] > a[j]) {

int temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

printf("array elements are\n");

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

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

}

printf("enter the key to search\n");

scanf("%d",&key);

while (l <= h) {

t += 1;

mid = (l + h) / 2;

t += 3;

if (a[mid] == key) {

t += 1;

flag = 1;

t += 1;

break;

} else if (a[mid] > key) {

t += 1;

h = mid - 1;

t += 2;

} else {

l = mid + 1;

t += 2;

}

}

if(flag==1){

printf("element found at %d:",mid);

}

else{

printf("element not found");

}

printf("time taken is %d:",t);

}

3 Binary Search With Recursive

#include <stdio.h>

#include <stdlib.h>

int t=0;

int bs(int a[], int l, int h, int key) {

t+=1;

if (l <= h) {

int mid = (l + h) / 2;

t+=3;

if (a[mid] == key) {

t+=1;

return mid;

} else if (a[mid] > key) {

t+=1;

return bs(a, l, mid - 1, key);

} else {

t+=1;

return bs(a, mid + 1, h, key);

}

}

return -1;

}

void main() {

int a[1000], i, key, n, l = 0, h, temp, j, result;

printf("Enter the size of the array:\n");

scanf("%d", &n);

h = n - 1;

printf("Generating random array elements...\n");

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

a[i] = rand() % 1000;

}

for (i = 0; i < n - 1; i++) {

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

if (a[i] > a[j]) {

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

printf("Sorted array elements are:\n");

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

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

}

printf("\nEnter the key to search:\n");

scanf("%d", &key);

result = bs(a, l, h, key);

if (result != -1) {

printf("\nElement found at index %d\n", result);

} else {

printf("\nElement not found\n");

}

printf("time taken is %d:",t);

}

4. Fibbonocci with recursive

#include<stdio.h>

#include<stdlib.h>

int t=0;

int fibb(int n){

t+=1;

if(n<=1){

t+=1;

return 1;

}

t+=2;

return fibb(n-1)+fibb(n-2);

}

void main(){

int n,i,res;

printf("enter the term\n");

scanf("%d",&n);

res=fibb(n);

printf("the fibbonocci of %d is %d:\n",n,res);

printf("time taken is %d:",t);

}

6.factorial

#include <stdio.h>

#include <stdlib.h>

void main() {

int n, i, fact = 1, t = 0;

printf("Enter the number:\n");

scanf("%d", &n);

if (n == 0 || n == 1) {

t += 1;

printf("Factorial of %d is 1\n", n);

printf("Time taken is %d\n", t);

return;

}

for (i = 2; i <= n; i++) {

t += 3;

fact = fact \* i;

t += 2;

}

printf("Factorial of the given number %d is %d\n", n, fact);

printf("Time taken is %d\n", t);

}

Factorial

#include<stdio.h>

#include<stdlib.h>

int t=0;

int factorial(int n){

t+=1;

if(n==0 || n==1){

t+=1;

return 1;

}

t+=2;

return n \* factorial(n-1);

}

void main(){

int n,res,i;

printf("enter the number\n");

scanf("%d",&n);

res=factorial(n);

printf("factorial of the give number %d is %d\n",n,res);

printf("time taken is %d:",t);

}

Merge sort

#include <stdio.h>

#include <stdlib.h>

int t = 0;

void merge(int a[], int l, int mid, int h) {

int i = l, j = mid + 1, k = 0;

t += 3;

int temp[h - l + 1];

while (i <= mid && j <= h) {

t += 3;

if (a[i] <= a[j]) {

temp[k++] = a[i++];

} else {

temp[k++] = a[j++];

}

t += 2;

}

while (i <= mid) {

t += 2;

temp[k++] = a[i++];

}

while (j <= h) {

t += 2;

temp[k++] = a[j++];

}

for (i = l, k = 0; i <= h; i++, k++) {

t += 2;

a[i] = temp[k];

}

}

// Merge Sort Function

void mergeSort(int a[], int l, int h) {

t += 1;

if (l < h) {

int mid = (l + h) / 2;

t += 2;

mergeSort(a, l, mid);

mergeSort(a, mid + 1, h);

merge(a, l, mid, h);

}

}

int main() {

int a[1000], n, i;

printf("Enter the size of the array:\n");

scanf("%d", &n);

printf("enter the array elements...\n");

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

a[i] = rand() % 1000;

}

printf("Original array:\n");

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

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

}

mergeSort(a, 0, n - 1);

printf("\nSorted array:\n");

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

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

}

printf("\nTime taken is %d\n", t);

return 0;

}

Quicksort

#include <stdio.h>

#include <stdlib.h>

int t = 0;

int partition(int a[], int low, int high) {

t += 1;

int i = low ;

int pivot = a[low];

int j=high;

t += 3;

while(i<j){

t+=1;

while(a[i]<=pivot){

t+=2;

i++;

t+=2;

}

while(a[j]> pivot){

t+=2;

j--;

t+=2;

}

if(i<j){

t+=1;

int temp=a[i];

a[i]=a[j];

a[j]=temp;

t+=3;

}

}

int temp=a[j];

a[j]=a[low];

a[low]=temp;

t+=3;

return j;

}

void quickSort(int a[], int low, int high) {

if (low < high) {

t += 1;

int q= partition(a, low, high);

t += 2;

quickSort(a, low, q - 1);

quickSort(a, q + 1, high);

}

}

void main() {

int a[1000], n, i;

printf("Enter the size of the array: ");

scanf("%d", &n);

printf("Enter elements into the array\n");

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

a[i] = rand() % 1000;

}

printf("Original array:\n");

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

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

printf("\n");

quickSort(a, 0, n - 1);

printf("Sorted array:\n");

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

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

printf("\nTime taken: %d\n", t);

}

Heapsort

#include <stdio.h>

#include <stdlib.h>

int t = 0;

void heapify(int a[], int n, int i) {

t += 1;

int largest = i;

int left = 2 \* i ;

int right = 2 \* i + 1;

t += 5;

if (left < n && a[left] > a[largest]) {

t += 3;

largest = left;

t += 1;

}

if (right < n && a[right] > a[largest]) {

t += 1;

largest = right;

t += 1;

}

if (largest != i) {

t += 1;

int temp = a[i];

a[i] = a[largest];

a[largest] = temp;

t += 3;

heapify(a, n, largest);

}

}

void heapSort(int a[], int n) {

t += 1;

for (int i = n / 2 - 1; i >= 0; i--) {

t += 5;

heapify(a, n, i);

}

for (int i = n - 1; i > 0; i--) {

t += 4;

int temp = a[0];

a[0] = a[i];

a[i] = temp;

t += 3;

heapify(a, i, 0);

}

}

void main() {

int a[1000], n, i;

printf("Enter the size of the array: ");

scanf("%d", &n);

printf("enter elements to array\n");

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

a[i] = rand() % 1000;

}

printf("Original array:\n ");

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

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

printf("\n");

heapSort(a, n);

printf("Sorted array:\n ");

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

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

printf("\nTime taken: %d\n", t);

}

Binary search tree

#include <stdio.h>

#include <stdlib.h>

int t = 0;

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* createNode(int value) {

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

newNode->data = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

struct Node\* insert(struct Node\* root, int value) {

if (root == NULL) {

return createNode(value);

}

if (value < root->data) {

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

} else {

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

}

return root;

}

int search(struct Node\* root, int key) {

t += 1;

if (root == NULL) {

return -1;

}

if (root->data == key) {

t += 1;

return 1;

} else if (key < root->data) {

t += 1;

return search(root->left, key);

} else {

t += 1;

return search(root->right, key);

}

}

void inorder(struct Node\* root) {

if (root != NULL) {

inorder(root->left);

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

inorder(root->right);

}

}

void main() {

struct Node\* root = NULL;

int n, i, key, value, result;

printf("Enter the number of elements to insert in BST:\n");

scanf("%d", &n);

printf("Generating random elements and inserting into BST...\n");

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

value = rand() % 1000;

root = insert(root, value);

}

printf("In order traversal (Sorted elements in BST):\n");

inorder(root);

printf("\nEnter the key to search:\n");

scanf("%d", &key);

result = search(root, key);

if (result == 1) {

printf("\nElement found in BST\n");

} else {

printf("\nElement not found in BST\n");

}

printf("Time taken (number of steps): %d\n", t);

}