**RECURSION**

1. **Binary search**

#include <iostream>

using namespace std;

void print(int \*arr,int s,int e)

{

for(int i=s;i<=e;i++)

{

cout<<arr[i]<<" ";

}

cout<<endl;

}

bool isBinarySearch(int \*arr,int s,int e,int k)

{

print(arr,s,e);

//base case

if(s>e) //element not found

{

return false;

}

int mid = s + (e-s)/2;

cout<<"Value of arr mid is: "<<arr[mid]<<endl;

if(arr[mid] == k) //element found

{

return true;

}

if(arr[mid] < k)

{

return isBinarySearch(arr,mid+1,e,k);

}

else

{

return isBinarySearch(arr,s,mid-1,k);

}

}

int main()

{

int arr[6] = {2,4,6,10,14,16};

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

int key = 10;

bool ans = isBinarySearch(arr,0,size-1,key);

if(ans)

{

cout<<"Present"<<endl;

}

else

{

cout<<"Not Present"<<endl;

}

}

1. **Linear Search**

#include <iostream>

using namespace std;

void print(int arr[],int size)

{

cout<<"Size of an array is: "<<size<<endl;

for(int i=0;i<size;i++)

{

cout<<arr[i]<<" ";

}

cout<<endl;

}

bool linearSearch(int arr[],int size,int key)

{

print(arr,size);

if(size==0)

{

return false;

}

if(arr[0]==key)

{

return true;

}

else

{

bool remainingPart = linearSearch(arr+1,size-1,key);

return remainingPart;

}

}

int main()

{

int arr[5] = {3,5,1,2,6};

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

int key = 5;

bool ans = linearSearch(arr,size,key);

if(ans)

{

cout<<"Present"<<endl;

}

else

{

cout<<"Not Present"<<endl;

}

}

1. **Get Sum of an array**

#include <iostream>

using namespace std;

int getSum(int arr[],int size)

{

//base case

if(size==0)

{

return 0;

}

if(size==1)

{

return arr[0];

}

int remainingPart = getSum(arr+1,size-1);

int sum = arr[0] + remainingPart;

return sum;

}

int main()

{

int arr[5] = {2,1,9,9,9};

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

int sum = getSum(arr,size);

cout<<"Sum is: "<<sum<<endl;

}

1. **Check if an array is sorted or not**

#include <iostream>

using namespace std;

bool isSorted(int \*arr,int size)

{

if(size==0 || size==1)

{

return true;

}

if(arr[0] > arr[1])

{

return false;

}

else

{

bool remainingPart = isSorted(arr+1,size-1);

return remainingPart;

}

}

int main()

{

int arr[5] = {2,4,6,8,9};

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

bool ans = isSorted(arr,size);

if(ans)

{

cout<<"Array is sorted"<<endl;

}

else

{

cout<<"Array is not sorted"<<endl;

}

}

1. **Say Digit**

**/\***

**Ex: Input->243**

**Output-> three four two**

**\*/**

#include <iostream>

using namespace std;

void sayDigit(int n,string arr[])

{

//base case

if(n==0)

{

return;

}

//processing

int digit = n%10;

n = n/10;

cout<<arr[digit]<<" ";

//recursive call

sayDigit(n,arr);

}

int main()

{

string arr[10] = {"zero","one","two","three","four","five","six","seven","eight","nine"};

int n;

cin>>n;

cout<<endl;

sayDigit(n,arr);

}

1. **Reverse a string**

#include <iostream>

using namespace std;

string reverse(int i,int j,string &str)

{

cout<<"Call receiving for the string: "<<str<<endl;

//base case

if(i>j)

{

return "";

}

swap(str[i],str[j]);

i++;

j--;

return reverse(i,j,str);

}

int main()

{

string str = "abcde";

reverse(0,str.length()-1,str);

cout<<str;

}

1. **Power of a number**

#include <iostream>

using namespace std;

int power(int n)

{

//base case

if(n==0)

{

return 1;

}

//recursive relation

int smallerProb = power(n-1);

int biggerProb = 2 \* smallerProb;

return biggerProb;

}

int main()

{

int n;

cin>>n;

int ans=power(n);

cout<<ans<<endl;

}

1. **Print Count**

#include <iostream>

using namespace std;

void printCounting(int n)

{

if(n==0)

{

return;

}

cout<<n<<" ";

printCounting(n-1);

}

int main()

{

int n;

cin>>n;

// int ans=printCounting(n);

// cout<<ans<<endl;

printCounting(n);

}

1. **Source Destination**

#include <iostream>

using namespace std;

void reachHome(int src,int dest){

cout<<"Source: "<<src<<" Destination: "<<dest<<endl;

if(dest==src)

{

cout<<"Destination reached"<<endl;

return;

}

//processing

src++;

//recursive call

reachHome(src,dest);

}

int main()

{

int dest=10;

int src=1;

cout<<endl;

reachHome(src,dest);

}

1. **Quick Sort**

#include <iostream>

using namespace std;

int partition(int arr[],int s,int e)

{

int pivot = arr[s];

int cnt = 0;

for(int i=s+1;i<=e;i++)

{

if(arr[i] <= pivot)

{

cnt++;

}

}

//place pivot at right place

int pivotIndex = s+cnt;

swap(arr[pivotIndex],arr[s]);

//left and right part

int i=s,j=e;

while(i<pivotIndex && j>pivotIndex)

{

while(arr[i] <= pivot)

{

i++;

}

while(arr[j] > pivot)

{

j--;

}

if(i<pivotIndex && j>pivotIndex)

{

swap(arr[i++],arr[j--]);

}

}

}

void quickSort(int arr[],int s,int e)

{

if(s>=e) //base case

{

return;

}

//partition

int p = partition(arr,s,e);

//left part

quickSort(arr,s,p-1);

//right part

quickSort(arr,p+1,e);

}

int main()

{

int arr[5] = {2,4,1,6,9};

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

quickSort(arr,0,size-1);

for(int i=0;i<size;i++)

{

cout<<arr[i]<<" ";

}

}

1. **Quick Sort (Method-2)**

#include <iostream>

#include <algorithm>

using namespace std;

int partition(int \*A,int start,int end)

{

int pivot = A[end];

int partitionIndex = start;

for(int i=start;i<end;i++)

{

if(A[i] <= pivot)

{

swap(A[i],A[partitionIndex]);

partitionIndex++;

}

}

swap(A[partitionIndex],A[end]);

return partitionIndex;

}

void quickSort(int \*A,int start,int end)

{

if(start>=end)

{

return;

}

if(start<end)

{

int partitionIndex = partition(A,start,end); //calling partition

quickSort(A,start,partitionIndex-1);

quickSort(A,partitionIndex+1,end);

}

}

int main()

{

int A[] = {7,6,5,4,3,2,1,0};

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

quickSort(A,0,size-1);

for(int i=0;i<size;i++)

{

cout<<A[i]<<" ";

}

}

1. **Merge Sort**

#include <iostream>

#include <vector>

using namespace std;

void merge(int arr[],int s,int e)

{

int mid = s+(e-s)/2;

int len1 = (mid-s)+1;

int len2 = e-mid;

int \*first = new int[len1]; //length from 0 to mid

int \*second = new int[len2]; //length from mid+1 to end

//copy values

int mainArrayIndex = s;

for(int i=0;i<len1;i++)

{

first[i] = arr[mainArrayIndex];

mainArrayIndex++;

}

mainArrayIndex = mid+1;

for(int i=0;i<len2;i++)

{

second[i] = arr[mainArrayIndex];

mainArrayIndex++;

}

//merge two sorted arrays

int index1 = 0, index2 = 0;

int mergeIndex = s;

while(index1 < len1 && index2 < len2)

{

if(first[index1] < second[index2])

{

arr[mergeIndex++] = first[index1++];

}

else

{

arr[mergeIndex++] = second[index2++];

}

}

while(index1 < len1)

{

arr[mergeIndex++] = first[index1++];

}

while(index2 < len2)

{

arr[mergeIndex++] = second[index2++];

}

delete []first ;

delete []second;

}

void solve(int arr[],int s,int e)

{

if(s>=e)

{

return;

}

int mid = s + (e - s)/2;

//we have to sort left part

solve(arr,s,mid);

//we have to sort right part

solve(arr,mid+1,e);

//merge the arrays

merge(arr,s,e);

}

int main()

{

int arr[5] = {2,5,3,6,1};

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

solve(arr,0,size-1);

for(int i=0;i<size;i++)

{

cout<<arr[i]<<" ";

}

}

1. **Sort an array using Recursion**

#include <iostream>

using namespace std;

void sortArray(int \*arr,int size)

{

//base case

if(size==0 || size==1)

{

return;

}

for(int i=0;i<size;i++)

{

if(arr[i] > arr[i+1])

{

swap(arr[i],arr[i+1]);

}

}

sortArray(arr,size-1);

}

int main()

{

int arr[5] = {2,5,3,6,1};

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

sortArray(arr,size);

for(int i=0;i<size;i++)

{

cout<<arr[i]<<" ";

}

}

1. **Check if the given string is palindrome or not**

**#include <iostream>**

**using namespace std;**

**bool checkPalindrome(string str,int i,int j)**

**{**

**if(i>j)**

**{**

**return true;**

**}**

**if(str[i]!=str[j])**

**{**

**return false;**

**}**

**else**

**{**

**return checkPalindrome(str,i+1,j-1);**

**}**

**}**

**int main()**

**{**

**string str = "abbccbbe";**

**cout<<endl;**

**bool ans = checkPalindrome(str,0,str.length()-1);**

**if(ans)**

**{**

**cout<<"String is palindrome"<<endl;**

**}**

**else**

**{**

**cout<<"Its not a palindrome"<<endl;**

**}**

**}**

1. **Power of a number**

**#include <iostream>**

**using namespace std;**

**int power(int a,int b)**

**{**

**//base case**

**if(b==0)**

**{**

**return 1;**

**}**

**if(b==1)**

**{**

**return a;**

**}**

**int ans = power(a,b/2);**

**if(b%2==0) //if b is even**

**{**

**return ans\*ans;**

**}**

**else**

**{**

**return a\*ans\*ans;**

**}**

**}**

**int main()**

**{**

**int a,b;**

**cin>>a>>b;**

**int ans = power(a,b);**

**cout<<"Answer is: "<<ans<<endl;**

**}**