15B17CI371 – Data Structures Lab

ODD 2024

Week 5-LAB B

9923103023 – F1

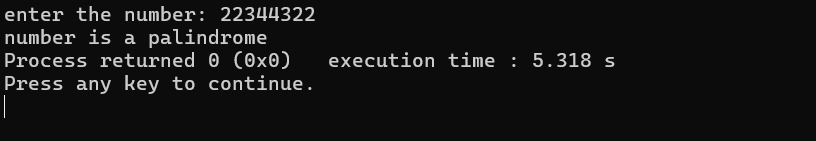
Practice Lab

1. Write a program to find whether the number is Palindrome or not. A number

is a Palindrome if it remains the same when its digits are reversed.

Assumption: N is a positive integer.

Ans :



#include<iostream>

#include<string>

using namespace std;

bool chk\_pal(int num,int temp,int rev)

{

if(temp==0)

{

if(num==rev) return true;

return false;

}

rev\*=10;

rev += temp%10;

temp /= 10;

chk\_pal(num,temp,rev);

}

int main()

{

int number;

cout<<"enter the number: ";

cin>>number;

if(chk\_pal(number,number,0))

{

cout<<"number is a palindrome ";

}

else

{

cout<<"number is not a palindrome ";

}

}

Time complexity: O (log n)

Space complexity: O (log n)

2. Write a program to implement a recursive function to calculate the sum of

digits of a given number.

ANS:

#include<iostream>

using namespace std;

int sum\_recur(int number)

{

int sum=0;

while(number>0)

{

int dig = number%10;

sum += dig;

return sum + sum\_recur(number/10);

}

}

int main()

{

int number;

cout<<"enter the number : ";

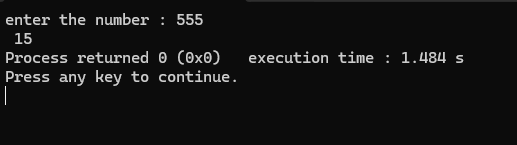
cin>>number;

cout<<" "<<sum\_recur(number);

}

Time complexity: O (log n)

Space complexity: O (log n)



3. Write a program to implement a recursive function to find the maximum and

minimum elements in a given array.

Ans :

#include<iostream>

using namespace std;

void find\_max(int \*arr,int s,int e, int m,int small)

{

if(s>e)

{

cout<<"greatest element : "<<m<<" smallest element : "<<small;

return ;

}

if(arr[s]>m)

{

m = arr[s];

}

else if(arr[s]<small)

{

small = arr[s];

}

if(s<=e)

{

find\_max(arr,s+1,e,m,small);

}

}

int main()

{

int n;

cout<<"enter the size: ";

cin>>n;

int \*arr = new int[n];

cout<<"enter the array : ";

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

{

cin>>arr[i];

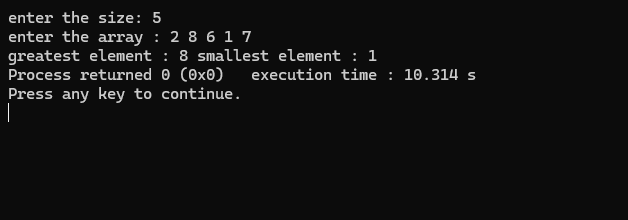
}

find\_max(arr,1,n-1,arr[0],arr[0]);

}

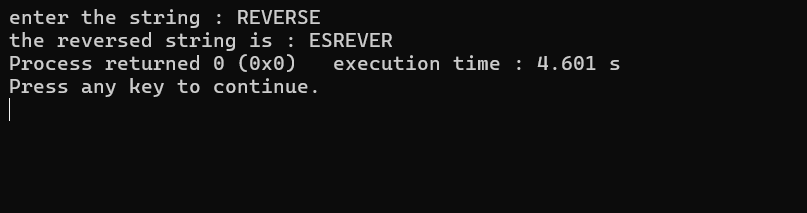
Time complexity: O (n)

Space complexity: O (n)



4. Write a program to reverse a string using recursion.

Ans:



#include<iostream>

#include<string>

using namespace std;

void rev\_str(string str,int s,int n)

{

if(s == n/2)

{

cout<<str;

return;

}

swap(str.at(s),str.at(n-s-1));

rev\_str(str,s+1,n);

}

int main()

{

string s;

cout<<"enter the string : ";

cin>>s;

int e;

e = s.length();

cout<<"the reversed string is : ";

rev\_str(s,0,e);

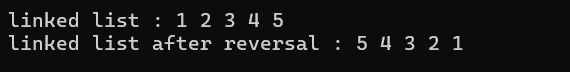
}

Time complexity: O (n)

Space complexity: O (n)

5. Write a program to implement a recursive function to reverse a linked list.

Ans:



#include<iostream>

#include<string>

using namespace std;

template<class T>

class LL;

template<class T>

class node

{

public:

T data;

node<T> \* next=NULL;

node(T val)

{

data = val;

}

friend class LL<T>;

};

template<class T>

class LL

{

public:

node<T> \* head = NULL;

node<T> \* tail =NULL;

void inattail(T value)

{

node<T>\* temp = new node<T>(value);

if(!head)

{

head = temp;

tail = temp;

return;

}

node<T> \* t = head;

while(t->next!=NULL)

{

t = t->next;

}

t->next = temp;

}

void print\_list()

{

node<T>\* temp = head;

while(temp!= NULL)

{

cout<<temp->data<<" ";

temp = temp->next;

}

cout<<endl;

}

};

node<int>\* reverseList(node<int>\*& head)

{

if(head == nullptr || head->next ==nullptr)

{

return head;

}

node<int>\* newhead = reverseList(head->next);

head->next->next = head;

head->next = nullptr;

return newhead;

}

int main()

{

LL<int> \* list1 = new LL<int>;

list1->inattail(1);

list1->inattail(2);

list1->inattail(3);

list1->inattail(4);

list1->inattail(5);

cout<<"linked list : ";

list1->print\_list();

list1->head = reverseList(list1->head);

cout<<"linked list after reversal : ";

list1->print\_list();

}

Time complexity: O(n)

Space complexity: O (1)

6. Write a program to implement a recursive function to find the greatest

common divisor and Least Common Multiple.

Ans:

#include<iostream>

#include<string>

using namespace std;

void gcd(int num1,int num2)

{

if(num1==1 || num2==1)

{

cout<<"GCD is 1 "<<endl;

return;

}

if(num1==0)

{

cout<<"GCD is :"<<num2<<endl;

return;

}

if(num2==0)

{

cout<<"GCD is :"<<num1<<endl;

return;

}

if(num2>num1)

{

if(!(num2%num1))

{

cout<<"the GCD is : "<<num1<<endl;

return;

}

else

{

gcd(num1,num2%num1);

}

}

else

{

if(!(num1%num2))

{

cout<<"the GCD is : "<<num2<<endl;

return;

}

else

{

gcd(num1%num2,num2);

}

}

}

void lcm(int num1,int num2,int times,int temp)

{

if(num1==0 || num2 == 0)

{

cout<<"LCM is : 0 "<<endl;

return;

}

if(num1 == 1)

{

cout<<"LCM IS : "<<num2<<endl;

return;

}

if(num2 == 1)

{

cout<<"LCM IS : "<<num1<<endl;

return;

}

if(num2>num1)

{

if(temp%num1 == 0)

{

cout<<"the LCM is : "<<temp<<endl;

return;

}

else

{

times++;

lcm(num1,num2,times,num2\*times);

}

}

else if(num1>num2)

{

if(!(temp%num2))

{

cout<<"the LCM is : "<<temp<<endl;

return;

}

else

{

times++;

lcm(num1,num2,times,num1\*times);

}

}

}

int main()

{

int num1,num2;

cout<<"enter the numbers : ";

cin>>num1>>num2;

if(num1>num2)

{

lcm(num1,num2,1,num1);

}

else

{

lcm(num1,num2,1,num2);

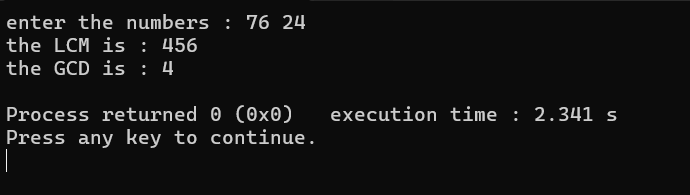
}

gcd(num1,num2);

}

Time complexity: O(log(min(num1, num2)))

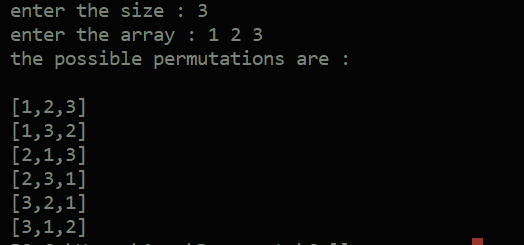
Space complexity: O (1)



7. Write a program to implement a recursive function to generate all

permutations of a given set of numbers.

Ans :



#include <iostream>

using namespace std;

void print\_arr(int \**arr*, int *size*)

{

    cout << "[";

    for (int i = 0; i < *size*; i++)

    {

        if (i == (*size* - 1))

        {

            cout << *arr*[i];

            break;

        }

        cout << *arr*[i] << ",";

    }

    cout << "]";

    cout << endl;

}

void permutations(int \**arr*, int *index*, int *size*)

{

    if (*index* >= *size*)

    {

        print\_arr(*arr*, *size*);

        return;

    }

    for (int i = *index*; i < *size*; i++)

    {

        swap(*arr*[*index*], *arr*[i]);

        permutations(*arr*, *index* + 1, *size*);

        swap(*arr*[*index*], *arr*[i]);

    }

}

int main()

{

    int n;

    cout << "enter the size : ";

    cin >> n;

    int \*arr = new int[n];

    cout << "enter the array : ";

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    cout << "the possible permutations are :\n \n";

    permutations(arr, 0, n);

    return 0;

}

Time complexity: O (n!)

Space complexity: O (n)

8. Assume that you are given a string. You can now form continuous substrings

from the given string. Write a program to count the number of UNIQUE

continuous sets of substrings that have the same starting and ending

characters. (You can use a mix of recursive and non-recursive functions).

Eg:

Input string: “andisan”

Possible substrings:

“a”, “an”, “and”, “andi”, “andis”, “andisa”, “andisan”,

“n”, “nd”, “ndi”, “ndis”, “ndisa”, “ndisan”,

“d”, “di”, “dis”, “disa”, “disan”,

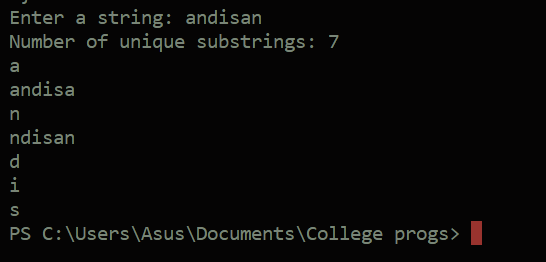
“i”, “is”, “isa”, “isan”,

“s”, “sa”, “san”,

“a”, “an”, “n”, (already considered earlier)

Output: 7 (“a”, “andisa”, “n”, “ndisan”, “d”, “i”, “s”)

Ans :



#include <iostream>

#include <string>

using namespace std;

const int MAX\_SUBSTRINGS = 1000;

bool isUnique(const string *substrings*[], int *count*, const string &*newSubstring*)

{

    for (int i = 0; i < *count*; ++i)

    {

        if (*substrings*[i] == *newSubstring*)

        {

            return false;

        }

    }

    return true;

}

void findSubstrings(const string &*str*, int *start*, int *end*, string *substrings*[], int &*count*)

{

    if (*end* >= *str*.length())

    {

        return;

    }

    if (*str*[*start*] == *str*[*end*])

    {

        string substring = *str*.substr(*start*, *end* - *start* + 1);

        if (isUnique(*substrings*, *count*, substring))

        {

*substrings*[*count*] = substring;

*count*++;

        }

    }

    findSubstrings(*str*, *start*, *end* + 1, *substrings*, *count*);

}

void countUniqueSubstrings(const string &*str*)

{

    string substrings[MAX\_SUBSTRINGS];

    int count = 0;

    for (int i = 0; i < *str*.length(); ++i)

    {

        findSubstrings(*str*, i, i, substrings, count);

    }

    cout << "Number of unique substrings: " << count << endl;

    for (int i = 0; i < count; ++i)

    {

        cout << substrings[i] << endl;

    }

}

int main()

{

    string input;

    cout << "Enter a string: ";

    cin >> input;

    countUniqueSubstrings(input);

    return 0;

}

Time complexity: O (n2)

Space complexity: O (n2)