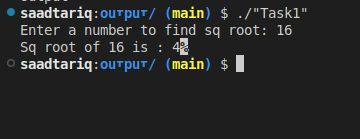
**Saad Tariq**

**22f8785**

**Assignment 1**

**Algorithms**

**Task 1**

****

#include<iostream>

using namespace std;

int sqRoot(int num);

int main()

{

int num=0;

cout<<"Enter a number to find sq root: ";

cin>>num;

cout<<"Sq root of "<<num<<" is : "<<sqRoot(num);

return 0;

}

int sqRoot(int num)

{

int low=0;

int high=num;

int mid;

while (low <= high)

{

mid = (low + high) / 2;

if (mid \* mid == num)

return mid;

if (mid \* mid > num)

high = mid - 1;

else

{

low = mid + 1;

mid = low - 1;

}

}

return mid;

}

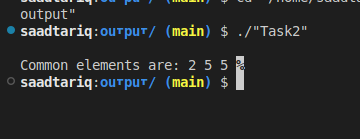
**Task 2**

#include <iostream>

using namespace std;

int\* findCommonElements(int\* list1,int sz1 , int\* list2, int sz2, int&count)

{

count=0;

int i = 0, j = 0;

int sz3=sz1<sz2?sz1:sz2;

int\* result=new int[sz3];

while (i < sz1 && j < sz2)

{

if (list1[i] == list2[j])

{

result[count]=list1[i];

count++;

i++;

j++;

} else if (list1[i] < list2[j])

{

i++;

} else

{

j++;

}

}

return result;

}

int main() {

int list1[] = {2, 5, 5, 5};

int list2[] = {2, 2, 3, 5, 5, 7};

int sz1 = sizeof(list1)/sizeof(list1[0]);

int sz2 = sizeof(list2)/sizeof(list2[0]);

int resSz=0;

int\* res = findCommonElements(list1,sz1, list2,sz2,resSz);

cout << "\nCommon elements are: ";

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

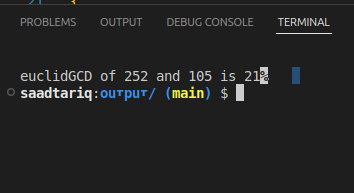
{

cout << res[i] << " ";

}

return 0;

}

**Task 3**

#include<iostream>

using namespace std;

int EuclidGcd(int num1, int num2);

int main()

{

system("clear");

// int a=14142;

// int b=31415;

int a=252;

int b=105;

int euclidGCD=EuclidGcd(a,b);

cout<<"\neuclidGCD of "<< a <<" and "<< b <<" is "<<euclidGCD;

return 0;

}

int EuclidGcd(int num1, int num2)

{

int a = num1>num2?num1:num2;

int b = num1>num2?num2:num1;

int r=-1;

while(r!=0)

{

r=a%b;

a=b;

b=r;

}

return a;

}

**Comparison:**

* Euclid’s algorithm required 10 steps.
* The consecutive integer checking algorithm could require up to 14142 steps.
* The ratio of steps is approximately 14142 / 10 = 1414.2.

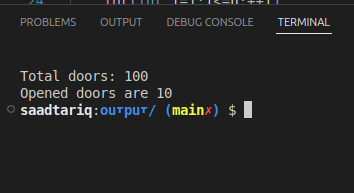
**Euclid's algorithm is approximately 1414 times faster than the consecutive integer checking algorithm in this case.**

**Task 4**

#include<iostream>

using namespace std;

void toggleLockers(bool\* doors, int n);

int main()

{

system("clear");

int n =100;

bool \* doors = new bool[n+1];

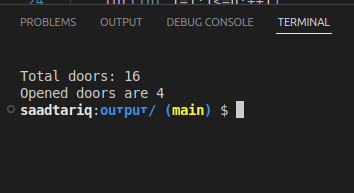
for(int i=1;i<=n;i++)

{

doors[i]=0;

}

toggleLockers(doors, n);



delete[] doors;

return 0;

}

void toggleLockers(bool\* doors, int n)

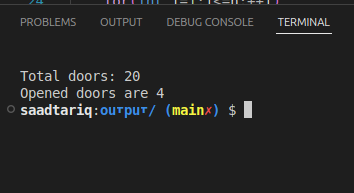
{

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

{

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

{

doors[j]=!doors[j];

}

}

string door="";

int openDoors=0;

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

{

if(doors[i]) openDoors++;

}

cout<<"\nTotal doors: "<<n;

cout<<"\nOpened doors are "<<openDoors<<"\n";

}

# Task 5

The peasant can safely transport all by following steps:

* he first takes the goat across the river. The wolf can’t eat the cabbage.
* He then takes the wolf across and brings the goat back. Peasant is present so goat not eat cabbage.
* Then he takes the cabbage across. The wolf can’t eat the cabbage.
* Finally he takes the goat across .

All are safely transported.

# Task 6

* **P1 and P2 cross the bridge together**:

They take 2 minutes

* **P1 returns with the flashlight**:

This takes 1 minute.

* **P3 and P4 cross the bridge together**:

They take 10 minutes

* **P2 returns with the flashlight**

This takes 2 minutes.

* **P1 and P2 cross the bridge together again**

They take 2 minutes.

(the slower person's pace determines the crossing time).

**Total time:**

2+1+10+2+2=17 minutes

# Task 7

To solve equation ax^2+bx+c=0 we can use the quadric formula

x=(−b ± sqrt(b^2−4ac))​​/ 2a

Algorithm FindRealRoots(a, b, c)

If a is 0 then

If b is 0 then

If c is 0 then

Print "Infinitely many solutions"

Else

Print "No solution"

End If

Else

// If 'a' is 0 but 'b' is not 0, it's a linear equation bx + c = 0

root = -c / b

Print "One real root: ", root

End If

Return

End If

discriminant = b^2 - 4 \* a \* c

If discriminant is greater than 0 then

root1 = (-b + sqrt(discriminant)) / (2 \* a)

root2 = (-b - sqrt(discriminant)) / (2 \* a)

Print "Two real roots: ", root1, " and ", root2

Else If discriminant is equal to 0 then

root = -b / (2 \* a)

Print "One real root: ", root

Else

Print "No real roots" End If

End Algorithm

# Task 8

There are two riverbanks (upper and lower), two land masses (larger and smaller), and seven bridges connecting them. The connections are as follows:

* The larger land mass is connected to the upper bank by 2 bridges, to the lower bank by 2 bridges, and to the smaller land mass by 1 bridge.
* The smaller land mass is connected to the upper bank by 1 bridge, to the lower bank by 1 bridge, and to the larger land mass by 1 bridge (the same one mentioned previously).

If we consider the bridges as edges and the land masses and riverbanks as vertices, each bridge can be seen as contributing to the degree of the vertices it connects.

* The larger land mass has a degree of 5.
* The smaller land mass has a degree of 3.
* Each riverbank has a degree of 3.

Traversing every edge exactly once and returning to the starting vertex is known as an Eulerian circuit. For a graph to be Eulerian, every vertex must have an even degree.

**a)** In the original Konigsberg bridge problem, every vertex has an odd degree, making it impossible to traverse every edge exactly once.

**b)** To solve this problem, you can:

* Add a bridge between the larger and smaller land masses.
* Add another bridge between the upper and lower banks.

These additions will make all vertices have an even degree, thereby allowing a solution to be possible.

# Task 9

Consider the Figure with 20 vertices arranged in three layers:

* 5 vertices on the outermost layer,
* 10 vertices on the middle layer,
* 5 vertices on the innermost layer.

Here's how you can navigate through these vertices:

1. Begin at any vertex on the outermost layer, say vertex **v1**.

2. From **v1**, move to the middle layer, starting at a vertex, let's say **v6**.

3. From **v6**, proceed to an adjacent vertex, **v7**.

4. From **v7**, move to the innermost layer and reach a vertex, say **v16.**

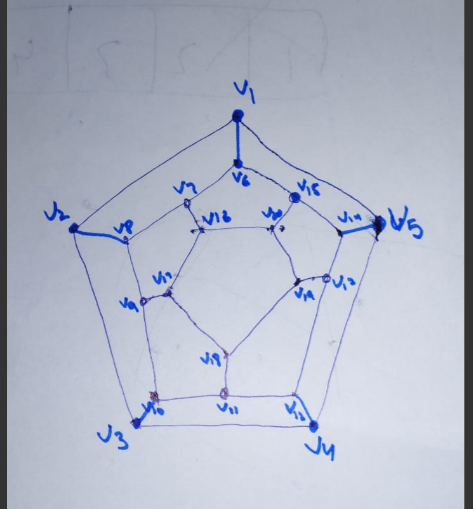
5. From **v16** move to adjacent vertices (**v17**, **v18**, **v19**, **v20**) in an anti-clockwise direction.

6. From **v20**, return to the middle layer at vertex **v15**.

7. From **v15**, move through adjacent vertices **(v14, v13, v12, v11, v10, v9, v8)** in a clockwise direction.

8. From **v8**, return to the outermost layer and reach vertex **v2**.

9. From **v2**, move through the adjacent vertices **(v3, v4, v5)** in an anti-clockwise direction.



# Task 10

First find

Radius = the radius of the given circumference

then

For every point P(x,y) in set

distance = compute the distance of P from origin

//where origin is the center of the given circumference

if Radius is equal to distance

Print point lies on the circumference

Else

Print point does not lie on the circumference

# Task 12

1. Simply replace the last element of the array with the ith element to delete it.

If n is length of array then

Arr[i]=Arr[n-1]

1. Not possible to delete element from sorted array and also keep the array sorted in Constant time.

# Task 13

1. In a sorted array we can use Binary Search to find the element. This has the advantage that the Time Complexity of Binary Search algorithm is **O(log(n)).**
2. In a sorted linked list we have the advantage that we don’t have to search the list until end because we can stop when element is found.

# Task 14

a. Show the stack after each operation of the following sequence that starts with the empty stack:

**push(a), push(b), pop, push(c), push(d), pop**

* a
* ba
* a
* ca
* dca
* ca

b. Show the queue after each operation of the following sequence that starts with the empty queue:

**enqueue(a), enqueue(b), dequeue, enqueue(c), enqueue(d), dequeue**

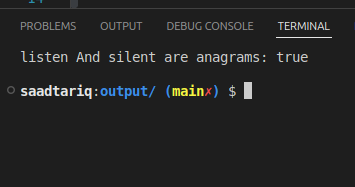
* a
* ab
* a
* ac
* acd
* ac

# Task 17

#include<iostream>

using namespace std;

bool IsAnagram(string word1, string word2);

int main()

{

system("clear");

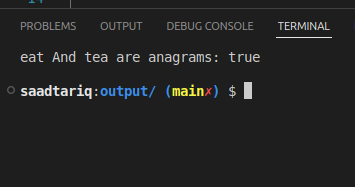
string w1="listen";

string w2="silent";

bool anagram = IsAnagram(w1,w2);

string res= anagram?"true":"false";

cout<<w1<<" And "<<w2<<" are anagrams: "<<res<<endl<<endl;

return 0;

}

bool IsAnagram(string word1, string word2)

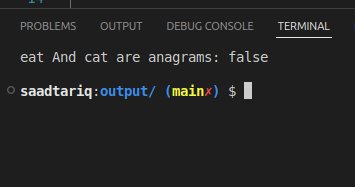
{

if(word1.length()!=word2.length())

{

return false;

}

int count=0;

int len=word1.length();

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

{

for (int j = 0; j < len; j++)

{

if(word1[i]==word2[j])

{

count++;

}

}

}

if(count!=len)

return false;

return true;

}