Joshua Nguyen

Programming Fundamentals III COSC 2336

February 27 2017

Project 1 (200 Points)

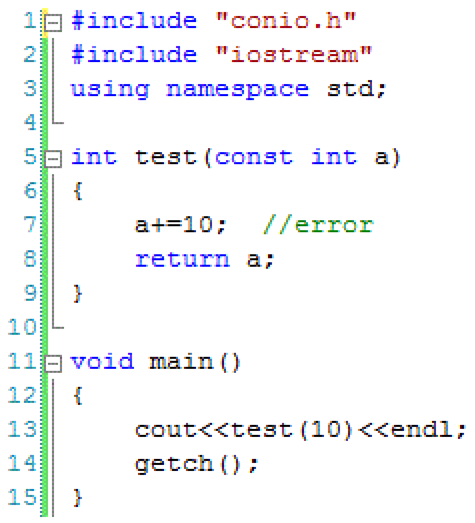
Part I: (100 Points)

Describe in your own words and in detail with example what are the following 7 functions are in Analysis Tools:

The Constant Function:

A **Constant Function** is a function that when a keyword, *const* is used in a function's declaration. It is however cannot be altered by the program while coding in C++. While a function is declared as a constant or in keyword *const*, it can be called on any data type of an object. Non-constant functions act only if called by non-constant objects. It provides a linear, yet basic element of the array. For example, we can have the following program here to return as a constant.

Example:



```
#include "conio.h"
#include "iostream"
using namespace std;

int test(const int a)
{
    a+=10;  //error
    return a;
}

void main()
{
    cout<<test(10)<<endl;
    getch();
}
```

The Logarithm Function:

When we have a **natural logarithm function** in C++, we usually use a keyword *log* calculated the value of the logarithm. In function, we use base-e logarithm as a natural log for the computing algorithm. To note, if a value of x is negative, it will cause a domain error and if x is zero, it will cause a pole error from having an invalid output.

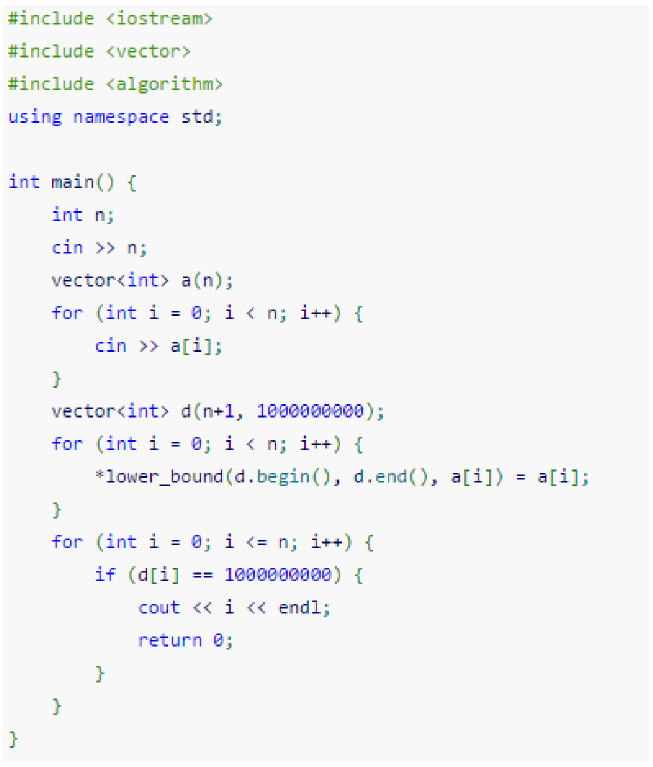
Example:

```
#include <stdio.h>
#include <math.h>

int main ()
{
  double param, result;
  param = 5.5;
  result = log (param);
  printf ("log(%f) = %f\n", param, result );
  return 0;
}
```

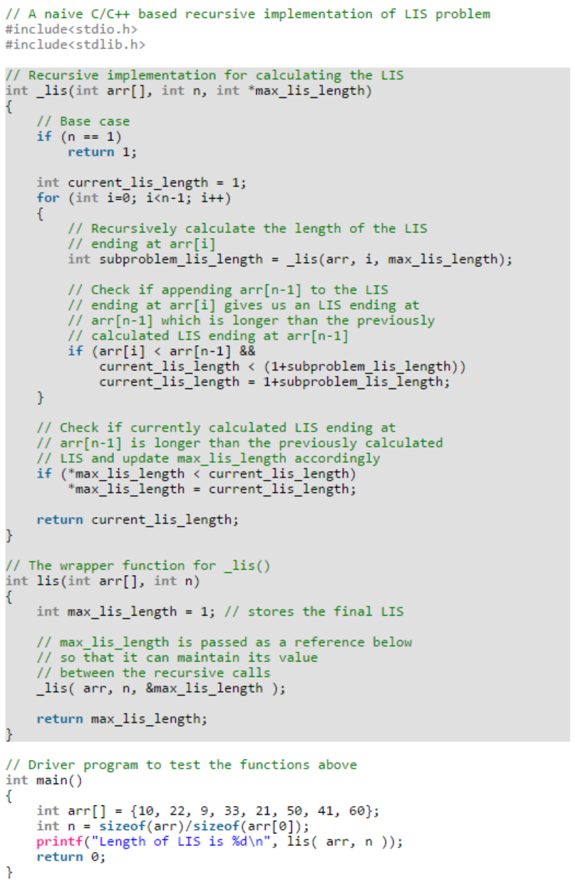
The N-Log-N Function:

The **N-Log N Function** usually associated itself with longest increasing subsequence (LIS) in the array to implement a logarithm value in the algorithm. Longest increasing subsequence is the given sequence problem that the function elements are in sorted order from the lowest to highest value when we analysis an algorithm. The function In these examples, we trying to find the length of the longest subsequence of the given problem of all the sorted elements in the subsequence.

Example:



```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    vector<int> d(n+1, 1000000000);
    for (int i = 0; i < n; i++) {
        *lower_bound(d.begin(), d.end(), a[i]) = a[i];
    }
    for (int i = 0; i <= n; i++) {
        if (d[i] == 1000000000) {
            cout << i << endl;
            return 0;
        }
    }
}
```

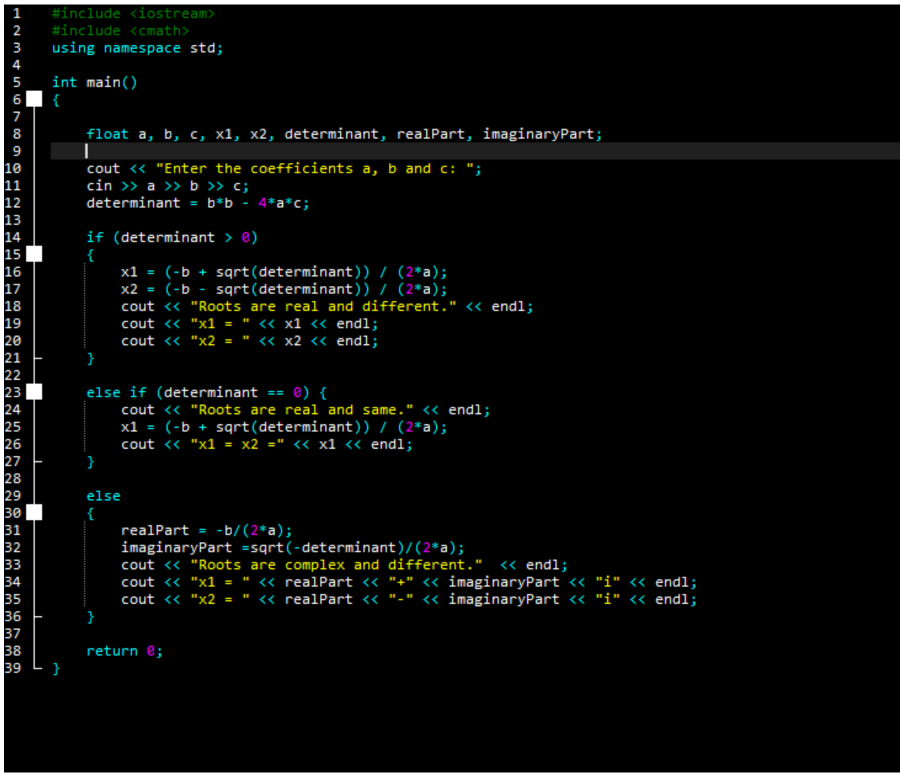


```c
// A naive C/C++ based recursive implementation of LIS problem
#include<stdio.h>
#include<stdlib.h>

// Recursive implementation for calculating the LIS
int _lis(int arr[], int n, int *max_lis_length)
{
    // Base case
    if (n == 1)
        return 1;

    int current_lis_length = 1;
    for (int i=0; i<n-1; i++)
    {
        // Recursively calculate the length of the LIS
        // ending at arr[i]
        int subproblem_lis_length = _lis(arr, i, max_lis_length);

        // Check if appending arr[n-1] to the LIS
        // ending at arr[i] gives us an LIS ending at
        // arr[n-1] which is longer than the previously
        // calculated LIS ending at arr[n-1]
        if (arr[i] < arr[n-1] &&
            current_lis_length < (1+subproblem_lis_length))
            current_lis_length = 1+subproblem_lis_length;
    }

    // Check if currently calculated LIS ending at
    // arr[n-1] is longer than the previously calculated
    // LIS and update max_lis_length accordingly
    if (*max_lis_length < current_lis_length)
        *max_lis_length = current_lis_length;

    return current_lis_length;
}

// The wrapper function for _lis()
int lis(int arr[], int n)
{
    int max_lis_length = 1; // stores the final LIS

    // max_lis_length is passed as a reference below
    // so that it can maintain its value
    // between the recursive calls
    _lis( arr, n, &max_lis_length );

    return max_lis_length;
}

// Driver program to test the functions above
int main()
{
    int arr[] = {10, 22, 9, 33, 21, 50, 41, 60};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Length of LIS is %d\n", lis( arr, n ));
    return 0;
}
```

The Quadratic Function:

For a quadratic equation, we using associated ourselves with the typical formula ax2+bx+c = 0 and -b + sqrt(b^2 -4ac)/2a to get our roots when y is zero. Function like f(n) = n^2 are used in C++ like n*n = n^2 in many operations. In programming, we use keyword *sqrt(determinant)* to get our roots when determinant is greater, equal or less than zero. In this example code, we enter the coefficient of determinant to find the roots from if statements greater, lesser or equal to zero.

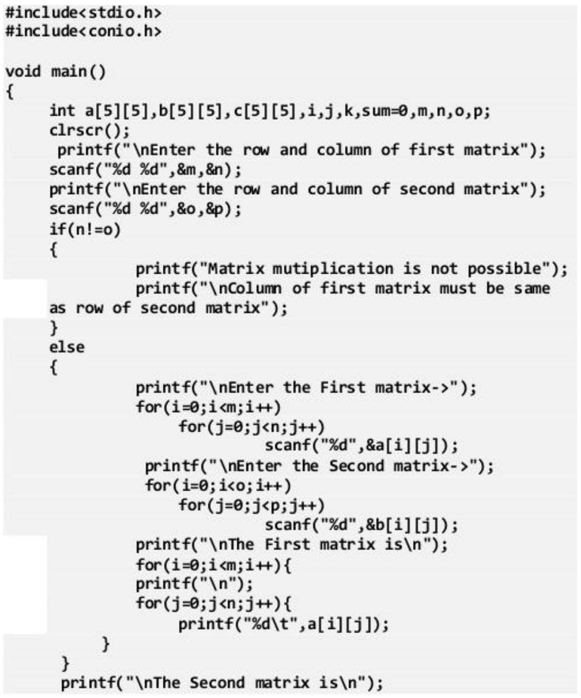
Example:



```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{

    float a, b, c, x1, x2, determinant, realPart, imaginaryPart;

    cout << "Enter the coefficients a, b and c: ";
    cin >> a >> b >> c;
    determinant = b*b - 4*a*c;

    if (determinant > 0)
    {
        x1 = (-b + sqrt(determinant)) / (2*a);
        x2 = (-b - sqrt(determinant)) / (2*a);
        cout << "Roots are real and different." << endl;
        cout << "x1 = " << x1 << endl;
        cout << "x2 = " << x2 << endl;
    }

    else if (determinant == 0) {
        cout << "Roots are real and same." << endl;
        x1 = (-b + sqrt(determinant)) / (2*a);
        cout << "x1 = x2 =" << x1 << endl;
    }

    else
    {
        realPart = -b/(2*a);
        imaginaryPart =sqrt(-determinant)/(2*a);
        cout << "Roots are complex and different."  << endl;
        cout << "x1 = " << realPart << "+" << imaginaryPart << "i" << endl;
        cout << "x2 = " << realPart << "-" << imaginaryPart << "i" << endl;
    }

    return 0;
}
```

The Cubic Function and Other Polynomials:

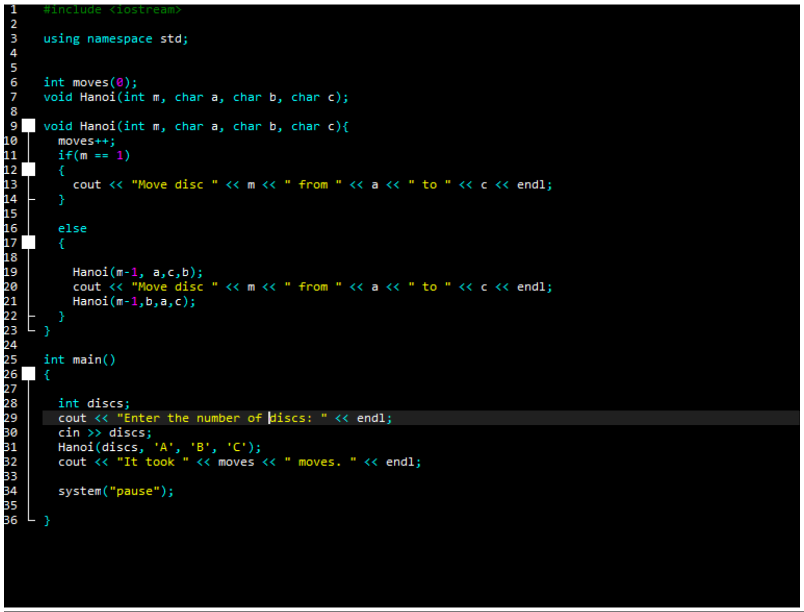
The cubic functions appear less often in the algorithm compared to linear and quadratic functions. When we have n doubles, our running time increases fondly. We use f(n) = n^3 as usual in a cubic problem. For example, we can used n by n matrix multiplication to create a larger class of functions.



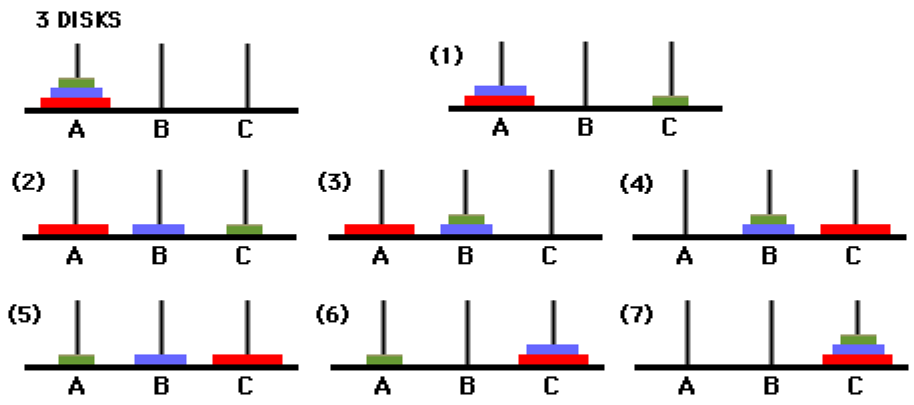
```
#include<stdio.h>
#include<conio.h>

void main()
{
    int a[5][5],b[5][5],c[5][5],i,j,k,sum=0,m,n,o,p;
    clrscr();
     printf("\nEnter the row and column of first matrix");
    scanf("%d %d",&m,&n);
    printf("\nEnter the row and column of second matrix");
    scanf("%d %d",&o,&p);
    if(n!=o)
    {
            printf("Matrix mutiplication is not possible");
            printf("\nColumn of first matrix must be same
    as row of second matrix");
    }
    else
    {
            printf("\nEnter the First matrix->");
            for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&a[i][j]);
             printf("\nEnter the Second matrix->");
             for(i=0;i<o;i++)
                for(j=0;j<p;j++)
                        scanf("%d",&b[i][j]);
            printf("\nThe First matrix is\n");
            for(i=0;i<m;i++){
            printf("\n");
            for(j=0;j<n;j++){
                printf("%d\t",a[i][j]);
        }
     }
     printf("\nThe Second matrix is\n");
```

The Exponential Function:

When we have an exponential function in a programming activity, the most common base for the function is b = 2. Usually, this is not the case for a typical math problem you would use as a base in everyday moments. The function f(n) = b ^ n is used to perform an operation and double the number of operations within each iteration when we start our loop. However, it is not practical function for everyday programmer to use. For example, we can use a tower algorithm to implement the function efficiently.



```cpp
#include <iostream>

using namespace std;


int moves(0);
void Hanoi(int m, char a, char b, char c);

void Hanoi(int m, char a, char b, char c){
  moves++;
  if(m == 1)
  {
    cout << "Move disc " << m << " from " << a << " to " << c << endl;
  }

  else
  {

    Hanoi(m-1, a,c,b);
    cout << "Move disc " << m << " from " << a << " to " << c << endl;
    Hanoi(m-1,b,a,c);
  }
}

int main()
{

  int discs;
  cout << "Enter the number of discs: " << endl;
  cin >> discs;
  Hanoi(discs, 'A', 'B', 'C');
  cout << "It took " << moves << " moves. " << endl;

  system("pause");

}
```



Part II: (75 Points)

Describe in your own words and in detail with example what are the following two:

Asymptotic Notation: asymptotic notation are languages that help us to analyze and understand a typical C++ algorithm's time by looking for its increasing input size. They identify the behavior of a algorithm by using typical function like logarithmic, liner, quadratic, polynomial, and exponential function. It is often used in O-notation for an upper bound examples like f(n) = O(g(n)) if any constant and n is greater than 0. However, it is used in worst case analysis. Here is an example of that scenario.

2N^2 = O(n^3) when c =1 and n = 2

Asymptotic Analysis: Asymptotic analysis is used as a method to describing the limiting behavior of the function. In mathematics, it is often used in analysis on a long run scenario of random variables in a problem. In computer science, it is however applied to data in which have a very big value to performance of an algorithm. For example, we used to analyze a running time of a input size of a program to estimate the time or space of the function given in a program. In this example, we use T(n) as time function as n gets large as possible.

$\boldsymbol{T(n)= c*n^2+k}$

T '(n) = c'*n*log2(n) + k'

Part III: (25 Points)

What would be the output of the following sequence of queue operations:

enqueue(5), enqueue(3),dequeue(),enqueue(2),enqueue(8),dequeue(),

dequeue(),enqueue(9),enqueue(1),dequeue(),enqueue(7),enqueue(6),

dequeue(),dequeue(),enqueue(4),dequeue(),dequeue(),

Output:

```
 Enqueue  (5), Enqueue  (3 ), Dequeue  ( ), Enqueue(2),   Enqueue(8  ), Dequeue  ( ), Dequeue  ( ),
   5                  5,3               3                  3,2                3,2,8        2,8                  8
Enqueue(9  ), Enqueue(  1), Dequeue  ( ), Enqueue  (7), Enqueue  (6), Dequeue  ( ), Dequeue  ( ),
    8,9             8,9,1                9,1               9,1,7          9,1,7,6          1,7,6                  7,6
Enqueue(4  ), Dequeue  ( ), Dequeue  ( ),
    7,6,4          6,4                    4
```