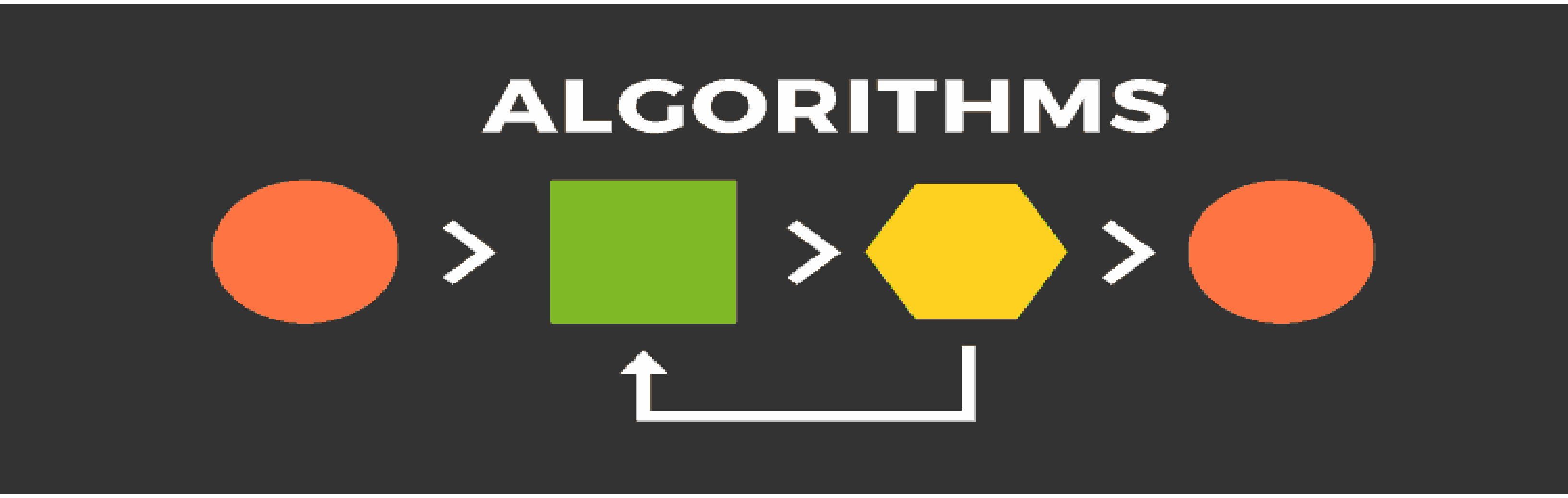
2022



Analysis and design of algorithms

[ASSIGNED]

D. KHALID AL-KAHSAH

الجمهورية اليمنية

وزارة التعليم العالي

جامعة إب

كلية العلوم

قسم تقنية المعلومات

تنفيذ خوارزميات مقرر تصميم وتحليل خوارزميات باستخدام لغة C++

Implementation of a course on analysis and design of algorithms using C++ language

إعداد الطلاب:

. عبد الله صادق محمد الهادي

. مرسل أحمد قاسم الجحافي

.أيمن عبده أحمد قايد الشراعي

.محمد أمين الصلاحي

تكليف الدكتور:

خالد الكحسه

1443هـ - 2022م

Computing the Greatest Common Divisor of Tow Integers

Euclid's Algorithm

```
// خوارزمية اقليدوس للقاسم المشترك
int Euclid(int num1, int num2) {
        if (num1 != 0 && num2 != 0) {
                int r;
                while (num2 != 0)
                {
                        r = num1 % num2;
                        num1 = num2;
                        num2 = r;

                }
                return num1;
        }
}
```

```
Microsoft Visual Studio Debug Console
Enter First Number 24
Enter Scound Number 60
greatest comman divider of by Algorithm Euclid 24 , 60  is : 12
```

Second Algorithm Gcd

```
// الخوارزمية الثانية لحساب القاسم المشترك الأكبر
int SoucondAlgorithm_Gcd (int num1, int num2)
{
        if (num1 != 0 && num2 != 0) {
                int r;
                r = Min(num1, num2);
                while (r!=0)
                {
                        if (num1 % r == 0)
                                if (num2 % r == 0)
                                {
                                        return r;
                                }
                        r--;
                }
                return r;
        }
}
```

```
Microsoft Visual Studio Debug Console
Enter First Number 24
Enter Scound Number 60
greatest comman divider of 24 , 60  is : 12
```

Middle-school procedure for computing the Greatest Common Divisor of Tow Integers

```
#include <iostream>
#include <math.h>
using namespace std;
const int Max = 100;//denamic size
int w = 0;//size the matrix new
//Function for find a max number
int Maxy(int num1, int num2)
{
        if (num1 < num2)
                return num2;
        return num1;
}
// A function to Find all prime factors of a
 //given number n
void primeFactors(int n,int a[])
{
        w = 0;
        // Print the number of 2s that divide n
        while (n % 2 == 0)
        {
                a[w] = 2;
                w++;
                n = n / 2;
        }
        for (int i = 3; i <= sqrt(n); i = i + 2)
        {
                while (n % i == 0)
                {
                        a[w] = i;
                        w++;
                        n = n / i;
                }
        }
        if (n > 2)
        {
                a[w] = n;
                w++;;
        }
}
//دالة تقوم بالفحص عن الرقم في المصفوفة المطلوبة
//وترجع عدد مرات وجودة
int find(int x[], int s, int k)
{
        int inc = 0;
        for (int i = 0; i < s; i++)
        {
                if (k == x[i])
                {
                        inc++;
                }
        }
        return inc;
}
// الخوارزمية المدرسية لحساب القاسم المشترك الأكبر
int ThirdAlgorithm_Gsd(int m, int n)
{       int r, s = 1;
        int c[Max] = { 0 }, b[Max]{0};
        primeFactors(m,c);
        int sizea = w;
        primeFactors(n,b);
        int sizeb = w;
                r = Maxy(sizea , sizeb);
                int k = 0;
        while(k < r)
        {
          if (find(c, sizea, c[k]) != 0 && find(b, sizeb, c[k])!=0)
            {
                if ((c[k] == c[k + 1]))
                 {
                    k++;
                 }
                else
                 {
                    if (find(c, sizea, c[k]) <= find(b, sizeb, c[k]))
                         {
                            for (int i = 0; i < find(c, sizea, c[k]); i++)
                                 s *= c[k];
                         }
                    else if (find(c, sizea, c[k]) > find(b, sizeb, c[k]))
                         {
                            for (int i = 0; i < find(b, sizeb, c[k]); i++)
                                 s *= c[k];
                         }
                     k++;
                   }
            }
            else
             {
                k++;
             }
         }
        for (int k = 0; k < r; k++)
        {
                cout << c[k] << "| " << b[k]<<"\n";
        }
        return s;
}
//Main program
int main() {
        int num1, num2;
        cout << "Enter First Number ";
        cin >> num1;
        cout << "Enter Scound Number ";
        cin >> num2;
        if (num1 < num2)
        {
                swap(num1, num2);
        }
        cout << "greatest comman divider of by Algorithm " <<
                num1 << " , " << num2 << "  is : " <<
                ThirdAlgorithm_Gsd(num1, num2) << endl;
        return 0;
}
```

Application Middle-school procedure for computing the Greatest Common Divisor of Tow Integers

```
Select Microsoft Visual Studio Debug Console
Enter First Number 24
Enter Scound Number 60
2| 2
2| 2
3| 2
5| 3
greatest comman divider of by Algorithm 60 , 24  is : 12
```

```
Microsoft Visual Studio Debug Console
Enter First Number 12
Enter Scound Number 9
2| 3
2| 3
3| 0
greatest comman divider of by Algorithm 12 , 9  is : 3
```

```
Microsoft Visual Studio Debug Console
Enter First Number 1000000
Enter Scound Number 200
2| 2
2| 2
2| 2
2| 5
2| 5
2| 0
5| 0
5| 0
5| 0
5| 0
5| 0
5| 0
greatest comman divider of by Algorithm 1000000 , 200  is : 200
```

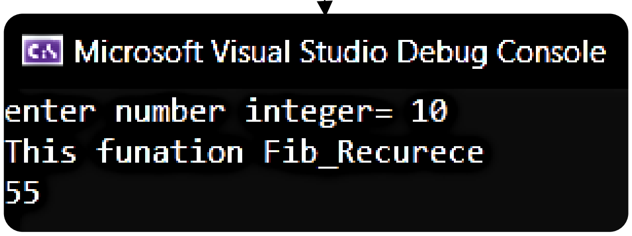
* نلاحظ: أن الخوارزمية المدرسية لإيجاد المضاعف المشترك الأكبر ذات كفاءة أقل مقارنة بالخوارزميات الباقية مثل خوارزمية اقليدس .

Algorithm Fibonacci

Fib_Recurece

```
// خوارزمية فيبوناتشي التراجعية
int Fib_Recurece(int n)
{
        if (n <= 1)
                return n;
        return (Fib_Recurece(n - 1) + Fib_Recurece(n - 2));
}
```

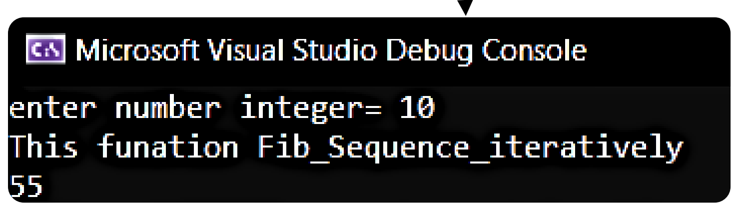
الكفاءة: $\theta(\emptyset^{n})$



Fib_Sequence_iteratively

```
// خوارزمية فيبوناتشي التكرارية
int Fib_Sequence_iteratively(int n)
{       //int *a = new int();
        int a[max];
        a[0] = {0}; a[1] = {1};
        for (int i = 2; i <= n; i++)
        {
                a[i] = a[i - 1] + a[i - 2];
        }
        return a[n];
}
```

الكفاءة: $\theta(n)$



* نلاحظ: أن الخوارزمية فيبوناتشي التراجعية ذات كفاءة أقل مقارنة بالخوارزمية التكرارية.

Sequential search

```
#include <iostream>
using namespace std;
// Sequential search
int Sequential(int a[], int n, int k)
{
	int i = 0;
	while (i < n && a[i] != k)
	{
		i++;
	}
	if (i < n)
		return i;
	else
		return -1;

}
int main() {
	int arr[] = { 7,5,99,30,1,2 };
	int n = sizeof(arr) / sizeof(arr[0]);
	int key = 2;
	int result = sequential(arr, n, key);
	if (result == -1)
		cout << key << "  Not Found" << endl;
	else
		cout << key << " Found At: " << result << endl;
	return 0;
}
```

الكفاءة تختلف بحسب حالات الكفاءة الأفضل والأسوأ والمتوسط

```
Microsoft Visual Studio Debug Console
Maxtrix ={ 7 5 99 30 1 2 }
2 Found At: 5
```

Maximum Element

```cpp
#include <iostream>
using namespace std;
//العنصر الأكبر في المصفوفة
int Max_Element(int a[], int n)
{
	int maxx = a[0];
	for (int i = 0; i < n; i++)
	{
		if (a[i] > maxx)
			maxx = a[i];
	}
	return maxx;
}
int main()
{
	int arr[] = { 7,5,99,30,1,2 };
	int m = sizeof(arr) / sizeof(arr[0]);
	cout << "Matrix ={ ";
	int n = sizeof(arr) / sizeof(arr[0]);
	for (int i = 0; i < n; i++)
	{
		cout << arr[i] << " ";
	}
	cout << "} " << endl;
	cout << "Maximum Element: " <<
	Max_Element(arr, m) << endl;
	return 0;
}
```

الكفاءة: (n)

```
Microsoft Visual Studio Debug Console
Maxtrix ={ 7 5 99 30 1 2 }
Maximum Element: 99
```

Minimum Element

```cpp
#include <iostream>
using namespace std;
//العنصر الأصغر في المصفوفة
int Min_Element(int a[], int n)
{
	int minx = a[0];
	for (int i = 0; i < n; i++)
	{
		if (a[i] < minx)
			minx = a[i];
	}
	return minx;
}
int main()
{
	int arr[] = { 7,5,99,30,1,2 };
	int m = sizeof(arr) / sizeof(arr[0]);
	cout << "Matrix ={ ";
	int n = sizeof(arr) / sizeof(arr[0]);
	for (int i = 0; i < n; i++)
	{
		cout << arr[i] << " ";
	}
	cout << "} " << endl;
	cout << "Minimum Element: " <<
Min_Element(arr, m) << endl;
	return 0;
}
```

الكفاءة: (n)

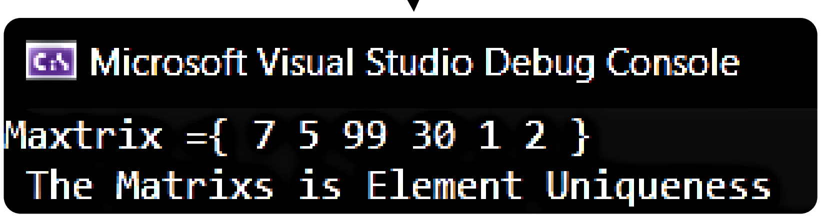
```
Microsoft Visual Studio Debug Console
Maxtrix ={ 7 5 99 30 1 2 }
Minimum Element: 1
```

Element Uniqueness

الكفاءة: $\theta(n^2)$

```cpp
#include <iostream>
using namespace std;
//دالة لفحص تفرد العناصر في مصفوفة أحادية
bool Element_Uniqueness(int arr[], int n)
{
	for (int i = 0; i < n - 2; i++)
		for (int j = i + 1; j < n; j++)
		{
			if (arr[j] == arr[i])
			{
				return false;
			}
		}
	return true;
}

int main()
{
	int arr[] = { 7,5,99,30,1,2 };
	int m = sizeof(arr) / sizeof(arr[0]);
	cout << "Matrix ={ ";
	int n = sizeof(arr) / sizeof(arr[0]);
	for (int i = 0; i < n; i++)
	{
		cout << arr[i] << " ";
	}
	cout << "} " << endl;
	if (Element_Uniqueness(arr,m)==false)
	{
		cout << " The Matrixs is not Element Uniqueness";
	}
	else
		cout << " The Matrixs is Element Uniqueness";
	return 0;
}
```



Matrix Multiplication

الكفاءة: $\theta(n^3)$

```
void multiply_Matrices(int fMat[][s], int sMat[][s], int mult[][s], int rowF, int columnF, int rowS, int columnS)
{
	int i, j, k;
	while (columnFirst != rowSecond)
	{
		cout << "Error! column of first matrix not equal to row of second !" << endl;
		return;
	}

	// Initializing elements of matrix mult to 0.
	for (i = 0; i < rowF; ++i)
	{
		for (j = 0; j < columnS; ++j)
		{
			mult[i][j] = 0;
		}
	}

	// Multiplying matrix firstMatrix and secondMatrix and storing in array mult.
	for (i = 0; i < rowF; ++i)
	{
		for (j = 0; j < columnS; ++j)
		{
			for (k = 0; k < columnF; ++k)
			{
				mult[i][j] += fMat[i][k] * sMat[k][j];
			}
		}
	}
}
```

Microsoft Visual Studio Debug Console

```
Enter rows for first matrix: 2
Enter column for first matrix: 2
Enter rows for second matrix: 2
Enter column for second matrix: 2

Enter elements of matrix 1:
Enter elements a[1][1]: 2
Enter elements a[1][2]: 4
Enter elements a[2][1]: 1
Enter elements a[2][2]: 4

Enter elements of matrix 2:
Enter elements b1][1]: 1
Enter elements b1][2]: 4
Enter elements b2][1]: 1
Enter elements b2][2]: 3
Output Matrix After Mult Tow Matrix:
6 20

5 16
```

*** شرط ضرب المصفوفتين:** أن يكون أعمدة المصفوفة الأولى يساوي صفوف المصفوفة الثانية.

C:\Users\USERWD\source\repos\Matrix Mult\Debug\Matrix Mult.exe

```
Enter rows for first matrix: 3
Enter column for first matrix: 4
Enter rows for second matrix: 5
Enter column for second matrix: 2
Error! column of first matrix not equal to row of second.
```

Counting Binary Digits

Recursive Function To Counting Binary Digits

الكفاءة: $\theta(\log n)$

```
// Recursive function to count set bits
int CountingBD_recursive(int n)
{
        if (n == 0)
                return 0;
        else
                return 1 + CountingBD_recursive((n /2));
}
```

```
Microsoft Visual Studio Debug Console
Enter a positive decimal integer: 20
By Recursive function:
Counting Binary Digits of (20)= 5
```

```
Microsoft Visual Studio Debug Console
Enter a positive decimal integer: 35
By Recursive function:
Counting Binary Digits of (35)= 6
```

```
Microsoft Visual Studio Debug Console
Enter a positive decimal integer: 0
By Recursive function:
Counting Binary Digits of (0)= 0
```

Iterative Function To Counting Binary Digits

الكفاءة: $\theta(\log n)$

```
// Iterative function to count set bits
int CountingBD_Iterative(int n)
{       int count = 0;
        while (n > 0)
        {
                count = count +1;
                n= (n/2);
        }
        return count;
}
```

```
Microsoft Visual Studio Debug Console
Enter a positive decimal integer: 20
By Iterative function:
Counting Binary Digits of (20)= 5
```

```
Microsoft Visual Studio Debug Console
Enter a positive decimal integer: 35
By Iterative function:
Counting Binary Digits of (35)= 6
```

```
Microsoft Visual Studio Debug Console
Enter a positive decimal integer: 0
By Iterative function:
Counting Binary Digits of (0)= 0
```

Find Factorial of a Number

Recursive Function To Find Factorial of a Number

الكفاءة: $\theta(n)$

```
unsigned long long int Find_Factorial_recursive(unsigned long long int n)
{
        if (n == 0)
                return 1;
        return n * Find_Factorial_recursive(n - 1);
}
```

```
C:\Users\USERWD\source\repos\Counting Binary
        By Recursive function:
Enter a positive decimal integer: 0
Factorial of (0)= 1
1
Factorial of (1)= 1
2
Factorial of (2)= 2
5
Factorial of (5)= 120
7
Factorial of (7)= 5040
9
Factorial of (9)= 362880
10
Factorial of (10)= 3628800
20
Factorial of (20)= 2432902008176640000
40
Factorial of (40)= 18376134811363311616
```

Iterative Function To Find Factorial of a Number

الكفاءة: $\theta(n)$

```
unsigned long long int Find_Factorial_Iterative(unsigned long long int  n)
{
        long long int res = 1, i;
        for (i = 2; i <= n; i++)
                res *= i;
        return res;
}
```

```
C:\Users\USERWD\source\repos\Counting Binary
        By Iterative function:
Enter a positive decimal integer: 1
Factorial of (1)= 1
2
Factorial of (2)= 2
3
Factorial of (3)= 6
4
Factorial of (4)= 24
5
Factorial of (5)= 120
6
Factorial of (6)= 720
7
Factorial of (7)= 5040
8
Factorial of (8)= 40320
9
Factorial of (9)= 362880
10
Factorial of (10)= 3628800
11
Factorial of (11)= 39916800
12
Factorial of (12)= 479001600
13
Factorial of (13)= 6227020800
```

The Tower of Hanwi Puzzel

الكفاءة: $\theta(2^n)$

```
// The Tower of Hanwi Puzzel
#include <iostream>
using namespace std;

void hanoi(int n, char a, char b, char c)
{
	if (n == 1)
	{
		cout << "\nMove disk " << n << " from tower " << a << " to tower " << b;
	}
	else
	{
		hanoi(n - 1, a, c, b);
		cout << "\nMove disk " << n << " from tower " << a << " to tower " << b;
		hanoi(n - 1, c, b, a);
	}
}
int main() {
	int n;
	cout << "How many disks : ";
	cin >> n;
	cout << n << endl;
	hanoi(n, 'A', 'C', 'B');
	return 0;
}
```

```
Microsoft Visual Studio Debug Console
How many disks : 3
3

Move disk 1 from tower A to tower C
Move disk 2 from tower A to tower B
Move disk 1 from tower C to tower B
Move disk 3 from tower A to tower C
Move disk 1 from tower B to tower A
Move disk 2 from tower B to tower C
Move disk 1 from tower A to tower C
```

The Sorting Algorithm

الكفاءة: $\theta(n^2)$

Selection Sort Algorithm

```
//Selection sort
void selectionsort(int sequenza[], int n)
{
        int min, temp;
        for (int i = 0; i <= n-2; i++)
        {
                min = i;
                for (int j = i + 1; j <= n - 1; j++)
                {
                        if (sequenza[j] < sequenza[min])
                                min = j;
                }
                temp = sequenza[i];
                sequenza[i] = sequenza[min];
                sequenza[min] = temp;

        }
}
```

```
Microsoft Visual Studio Debug Console
     *Using Selection sort Algorithm*
The matrix befor sorting: {89 45 68 90 29 34 17 }
The matrix befor sorting: {17 29 34 45 68 89 90 }
```

Bubble Sort Algorithm

```
//bubble Sort
void bubbleSort(int a[], int n)
{
        int i, j, t;
        for (i = 0; i < n - 1; i++)
        {
                for (j = 0; j < n - 1 - i; j++)
                {
                        if (a[j + 1] < a[j])
                        {
                                t = a[j + 1];
                                a[j + 1] = a[j];
                                a[j] = t;
                        }
                }
        }
}
```

```
Microsoft Visual Studio Debug Console
     *Using Bubble sort Algorithm*
The matrix befor sorting: {89 45 68 90 29 34 17 }
The matrix befor sorting: {17 29 34 45 68 89 90 }
```

* نلاحظ: أن الترتيب الفقاعي والترتيب بالاختيار لهما نفس الكفاءة لكن في المتوسط فإن أداء الترتيب الفقاعي أفضل بكثير.

Brute-Force String Matching

الكفاءة: $\theta(n)$

```
int BruteForceStringMatch(string T, string P, int n, int m)
{
        int i, j;
        int h = n - m;
        for (i = 0; i < h; i++)
        {
                j = 0
                        ;
                while (j < m && P[j] == T[i + j])
                {
                        j = j + 1
                                ;
                }
                if (j == m)
                        return i;
        }
        return -1;
```

```
Microsoft Visual Studio Debug Console
        *Search of The Pattern in Text*
The Text ={N O B O D Y _ N O T I C E D - H I }
The Pattern ={N O T}
Size Text= 34
Size Pattern= 5
The pattern found in Text in index = 14
```

```
Microsoft Visual Studio Debug Console
        *Search of The Pattern in Text*
The Text ={N O B O D Y _ N O T I C E D - H I }
The Pattern ={H O T}
Size Text= 34
Size Pattern= 5
The pattern not found in Text !~
```

Closest-Pair Algorithm

الكفاءة: $\theta(n^2)$

```cpp
// find the smallest distance using brute force.
#include <iostream>
using namespace std;
class Point
{
public:
        int x, y;
};
float Dist(Point p1, Point p2)
{
        return sqrt((p1.x - p2.x) * (p1.x - p2.x) +
                (p1.y - p2.y) * (p1.y - p2.y));
}
float BruteForce(Point P[], int n)
{
        float min = FLT_MAX;
        for (int i = 0; i < n; ++i)
                for (int j = i + 1; j < n; ++j)
                        if (Dist(P[i], P[j]) < min)
                                min = Dist(P[i], P[j]);
        return min;
}
int main()
{
        Point P[] = { {2, 3}, {12, 30},
                                {40, 50}, {5, 1},
                                {12, 10}, {3, 4} };
        int n = sizeof(P) / sizeof(P[0]);
        cout << "The smallest distance is "
                << BruteForce(P, n);
        return 0;
}
```

```
Microsoft Visual Studio Debug Console
The smallest distance is 1.41421
```

Traveling Salesman Problem(TSP)

الكفاءة: $\theta(v^2)$

```cpp
#include<iostream>
using namespace std;
#define MAX 9999
int n = 4; // Number of the places want to visit
 //Next distan array will give Minimum distance through all the position
int distan[10][10] = {
                                        {0, 2, 5, 7},
                                        {2, 0, 8, 3},
                                        5, 8, 0, 1},
                                        {7, 3, 8, 0}
                          };
int completed_visit = (1 << n) - 1;
int DP[16][4];
int min(int x, int y)
{
            if (x < y)
                        return x;
            return y;
}
int  TSP(int mark, int position)
{
            if (mark == completed_visit)
            {
                        return distan[position][0];
            }
            if (DP[mark][position] != -1)
            {
                        return DP[mark][position];
            }
            int answer = MAX;
            for (int city = 0; city < n; city++)
            {
                        if ((mark & (1 << city)) == 0)
                        {
                                    int newAnswer = distan[position][city] + TSP(mark | (1 << city), city);
                                    answer = min(answer, newAnswer);
                        }
            }
            return DP[mark][position] = answer;
}

int main()
{
            cout << "      Emplemantion for Traveling Salesman Problem(TSP)..." << endl;
            cout << "Matrix containing the value edges of vertices\n";
            cout << "\ta-> 0, 2, 5, 7      \n\tb-> 2, 0, 8, 3\n\tc-> 5, 8, 0, 1\n\tc-> 7, 3, 8, 0\n";
            for (int i = 0; i < (1 << n); i++)
            {
                        for (int j = 0; j < n; j++)
                        {
                                    DP[i][j] = -1;
                        }
            }
            cout << "Minimum Distance Travelled by you is " << TSP(1, 0);
            return 0;
}
```

```
Microsoft Visual Studio Debug Console
      Emplemantion for Traveling Salesman Problem(TSP)..
Matrix containing the value edges of vertices
        a-> 0, 2, 5, 7
        b-> 2, 0, 8, 3
        c-> 5, 8, 0, 1
        c-> 7, 3, 8, 0
Minimum Distance Travelled by you is 11
```

Knapsack Problem

الكفاءة: $\theta(n \log n)$

```
//program to solve fractional Knapsack Problem
#include <iostream>
using namespace std;
struct Item {
        int value, weight;
        // Constructor
        Item(int value, int weight)
        {
                this->value = value;
                this->weight = weight;
        }
};
double fractionalKnapsack(int W, struct Item arr[], int n);
// Comparison function to sort Item according to val/weight
// ratio
bool cmp(struct Item a, struct Item b)
{
        double r1 = (double)a.value / (double)a.weight;
        double r2 = (double)b.value / (double)b.weight;
        return r1 > r2;
}
int main()
{
        int W = 50; //    Weight of knapsack
        Item arr[] = { { 60, 10 }, { 100, 20 }, { 120, 30 } };
        cout << "     Value |  Weight |  ratio" << endl;
        int n = sizeof(arr) / sizeof(arr[0]);
        cout << "Maximum value we can obtain = "
                << fractionalKnapsack(W, arr, n);
        return 0;
}
```

```
double fractionalKnapsack(int W, struct Item arr[], int n)
{
        //sorting Item on basis of ratio
        //sort(arr, arr + n, cmp);
        for (int i = 0; i < n; i++)
        {
                cout << "\t" << arr[i].value << "\t" <<
arr[i].weight << "\t"
                        << ((double)arr[i].value /
arr[i].weight) << endl;
        }
        int curWeight = 0; // Current weight in knapsack
        double finalvalue = 0.0; // Result (value in
Knapsack)
        // Looping through all Items
        for (int i = 0; i < n; i++)
        {
                if (curWeight + arr[i].weight <= W)
                {
                        curWeight += arr[i].weight;
                        finalvalue += arr[i].value;
                }
        // If we can't add current Item, add fractional part
                // of it
                else {
                        int remain = W - curWeight;
                        finalvalue += arr[i].value
                                * ((double)remain
                                        /
(double)arr[i].weight);
                        break;
                }
        }
        // Returning final value
        return finalvalue;
}
```

```
Microsoft Visual Studio Debug Console
     Value |  Weight |  ratio
        60      10      6
        100     20      5
        120     30      4
Maximum value we can obtain = 240
```

Mergesort

الكفاءة: $\theta(n\log n)$

Mergesort Function

```
const int max = 100;
//Funcion merge elments tow array
void merge(int b[], int c[], int a[], int mid)
{
        int i = 0, j = 0, k = 0;
        while (i <= mid && j <= mid) {
                if (b[i] <= c[j]) {
                        a[k] = b[i];
                        i++;

                }
                else {
                        a[k] = c[j];

                        j++;
                }
                k++;
        }
        while (j <= mid) {
                a[k] = c[j];
                j++;
                k++;
        }
        while (i <= mid) {
                a[k] = b[i];
                i++;
                k++;
        }
}
```

Merge Function

```
//Funcion mergesort elments array
void mergesort(int a[], int n)
{
        int mid;
        int b[max];
        int c[max];
        if (n > 1) {
                mid = n / 2;
                copy(a + 0, a + mid, b);
                copy(a + mid, a + n, c);
                mergesort(b, mid);
                mergesort(c, mid);
                merge(b, c, a, mid - 1);
        }
}
```

Microsoft Visual Studio Debug Console

```
        Emplemantion Mergesort...
the matrix befor Mergesort= {8 3 2 9 7 1 5 4 }

the matrix After Mergesort= {1 2 3 4 5 7 8 9 }
```

Quicksort

$\theta(n^2)$:الكفاءة

Quicksort Function

```
//function to Sort matrix by quickSort
void quickSort(int arr[], int low, int high)
{
        if (low < high)
        {
                /* pi is partitioning index, arr[p] is now
                at right place */
                int pi = partition(arr, low, high);
                // Separately sort elements before
                // partition and after partition
                quickSort(arr, low, pi - 1);
                quickSort(arr, pi + 1, high);
        }
}
```

Partitioning Function

```
//Frist  function to partition matrix
int partition(int arr[], int start, int end)
{
        int pivot = arr[start];
        int count = 0;
        for (int i = start + 1; i <= end; i++) {
                if (arr[i] <= pivot)
                        count++;
        }
        // Giving pivot element its correct position
        int pivotIndex = start + count;
        swap(arr[pivotIndex], arr[start]);
        // Sorting left and right parts of the pivot element
        int i = start, j = end;
        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--]);
                }
        }
        return pivotIndex;
}
```

```
//Second  function to partition matrix
int partition2(int arr[], int low, int high)
{
        int pivot = arr[high]; // pivot
        int i = (low - 1);
        for (int j = low; j <= high - 1; j++)
        {
                if (arr[j] < pivot)
                {
                        i++;
                        swap(&arr[i], &arr[j]);
                }
        }
        swap(&arr[i + 1], &arr[high]);
        return (i + 1);
}
```

```
Microsoft Visual Studio Debug Console
     Emplemantion Quick Sort...
we using Frist function to partition matrix
the matrix befor Sort= {15 22 13 27 22 10 20 25 }
the matrix Sorted= {10 13 15 20 22 22 25 27 }
```

```
Microsoft Visual Studio Debug Console
     Emplemantion Quick Sort...
we using Second  function to partition matrix
the matrix befor Sort= {15 22 13 27 22 10 20 25 }
the matrix Sorted= {10 13 15 20 22 22 25 27 }
```

Binary Search

Binary Search Function Iterative

الكفاءة: $\theta(n)$

```
//تكرارية
int binarysearchIterative(int nums[], int n, int Key)
 {
		int low = 0, high = n - 1;
		while (low <= high)
		{
			int mid = (low + high) / 2;
			if (Key == nums[mid])
			{
				return mid;
			}
			else if (Key < nums[mid])
			{
				high = mid - 1;
			}
			else
			{
				low = mid + 1;
			}
		}
		// Key doesn't exist in the array
		return -1;
 }
```

```
Microsoft Visual Studio Debug Console
      Binary search
we used Iterative Implementation
This is Matrix = {2 3 5 7 8 10 12 15 18 20 }
The Element we want to search in arry = 10
Element is present at index 5
```

Binary Search Function Recursive

الكفاءة: $\theta(n)$

```
//تعاودية
int binarySearchRec (int arr[], int l, int r, int x)
{
	if (r >= l)
	{
		int mid = l + (r - l) / 2;
		if (arr[mid] == x)
			return mid;
		if (arr[mid] > x)
			return binarySearchRec (arr, l, mid - 1, x);
			return binarySearchRec (arr, mid + 1, r, x);
	}
		// We reach here when element is not
		// present in array
		return -1;
}
```

```
Microsoft Visual Studio Debug Console
      Binary search
we used Recursive Implementation
This is Matrix = {2 3 5 7 8 10 12 15 18 20 }
The Element we want to search in arry = 10
Element is present at index 5
```

* نلاحظ: أنه إذا كانت المصفوفة مرتبة فإن كفاءة البحث الثنائي تزيد لتصبح $\theta(\log n)$

Tree

Node for Tree

```
/* class definition for tree node */
class TNode
{
public:
        int data;
        TNode* left;
        TNode* right;

        TNode(int data) : data(data)
        {
                left = right = NULL;
        }
};
```

Definition for Tree

```
/* class definition for tree */
class Tree
{
public:
        TNode* root;
        Tree()
        {
                root = NULL;
        }
        // اضافة ابن ايسر لليسار
        void addLeftChild(TNode* p, TNode* c);
        //-------------------------------------------
        // اضافة ابن ايمن لليمين
        void addRightChild(TNode* p, TNode* c);
        //حساب ارتفاع الشجرة
        int getHeight(TNode* node);
        // دالة لاجتياز الشجرة بالترتيب السابق
        void Preorder(TNode* root);
        //دالة لاجتياز الشجرة بالترتيب الداخل
        void Inorder(TNode* root);
        //Function to visit nodes in Postorder
        // دالة لاجتياز الشجرة بالترتيب اللاحق
        void Postorder(TNode* root);
};
```

Add left Son

```
// اضافة ابن ايسر لليسار
        void Tree::addLeftChild(TNode* p, TNode* c)
        {
                p->left = c;
        }
```

Add Right Son

```
// اضافة ابن ايمن لليمين
        void Tree::addRightChild(TNode* p, TNode* c)
        {
                p->right = c;
        }
```

Tree height calculator

```
//حساب ارتفاع الشجرة
        int Tree::getHeight(TNode* node)
        {
                int lheight, rheight;
                if (node == NULL)
                        return 0;
                lheight = getHeight(node->left);
                rheight = getHeight(node->right);
                if (lheight > rheight)
                        return 1 + lheight;
                else
                        return 1 + rheight;
        }
```

Follow Tree

Tree Preorder traversal

```
// دالة لاجتياز الشجرة بالترتيب السابق
	void Tree::Preorder(TNode* root)
	{
		if (root == NULL) return;
		cout << root->data << " ";
		// Print data
		Preorder(root->left);
		// Visit left subtree
		Preorder(root->right);
		// Visit right subtree
	}
```

Tree Order traversal

```
//دالة لاجتياز الشجرة بالترتيب الداخل
	void Tree::Inorder(TNode* root)
	{
		if (root == NULL) return;
		Inorder(root->left);
		//Visit left subtree
		cout << root->data << " ";
		//Print data
		Inorder(root->right);
		// Visit right subtree
	}
```

Tree Postorder traversal

```
// دالة لاجتياز الشجرة بالترتيب اللاحق
	void Tree::Postorder(TNode* root) {
		if (root == NULL) return;

		Postorder(root->left);
		// Visit left subtree
		Postorder(root->right);
		// Visit right subtree
		cout << root->data << " ";
		// Print data
	}
```

Tree Postorder traversal

```
//دالة لعرض الشجرة
	void showTree(TNode* node, int space = 0, int t = 0)
	{
		int count = 3;
		if (node == NULL)
			return;

		space += count;
		showTree(node->right, space, 1);
		for (int i = count; i < space; ++i)
		{
			cout << "  ";
		}
		if (t == 1)// right child
		{
			cout << " |-- " << node->data << " "<<endl;
		}
		else if (t == 2) //left child
		{
			cout << " |-- " << node->data << " "<<endl;
		}
		else  // root node
		{
			cout << node->data << "->"<<endl;
		}
		showTree(node->left, space, 2);
	}
```

Follow Tree

Emplemantion For Tree

Microsoft Visual Studio Debug Console

```
                |-- 17
         |-- 13
                |-- 16
11->
                |-- 15
         |-- 12
                |-- 14

Height : 3
preorder:
11 12 14 15 13 16 17
inorder:
14 12 15 11 16 13 17
postorder:
14 15 12 16 17 13 11
```

Multiply Large Integers Divide and Conquer

الكفاءة: $\theta(n^2)$

```cpp
// C++ program to multiply two largr numbers
#include<iostream>
#include<string>
#include<vector>
using namespace std;
string multiply(string num1, string num2)
{
	int len1 = num1.size();
	int len2 = num2.size();
	if (len1 == 0 || len2 == 0)
		return "0";
	vector<int> result(len1 + len2, 0);
	int i_n1 = 0;
	int i_n2 = 0;
	for (int i = len1 - 1; i >= 0; i--)
	{
		int carry = 0;
		int n1 = num1[i] - '0';
		i_n2 = 0;
		for (int j = len2 - 1; j >= 0; j--)
		{
			int n2 = num2[j] - '0';
			int sum = n1 * n2 + result[i_n1 + i_n2] + carry;
			carry = sum / 10;
			result[i_n1 + i_n2] = sum % 10;
			i_n2++;
		}
		if (carry > 0)
			result[i_n1 + i_n2] += carry;
		i_n1++;
	}
	int i = result.size() - 1;
	while (i >= 0 && result[i] == 0)
		i--;
	if (i == -1)
		return "0";
	string s = "";
	while (i >= 0)
		s += std::to_string(result[i--]);
	return s;
}
```

```cpp
// Main code
int main()
{
	string str1;
	string str2;
	cout << "Multiply Tow Large Integers
|Implementation\n";
	cout << "Enter First Number: ";
	cin >> str1;
	cout << "Enter Scond Number: ";
	cin >> str2;
	cout << "The Result= " <<multiply(str1, str2);
	return 0;
}
```

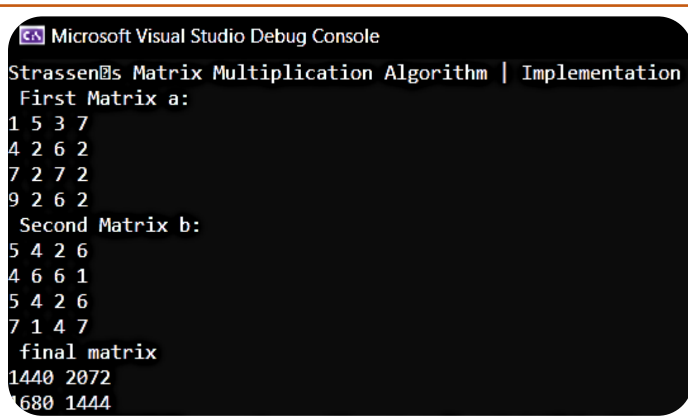
```
Microsoft Visual Studio Debug Console
Multiply Tow Large Integers |Implementation
Enter First Number: 18324697834297329343469
Enter Scond Number: 23423243465767867867859
The Result= 429223858809375519066545319846512457316662871
```

Strassen’s Matrix Multiplication

الكفاءة: $\theta(n^{2.807})$

```
#include<iostream>
using namespace std;
double a[4][4]={ 1, 5, 3, 7 ,4, 2, 6, 2,7, 2, 7, 2,9, 2,
6, 2 };
double b[4][4]= { 5, 4, 2, 6,4, 6, 6, 1,5, 4, 2, 6,7, 1,
4, 7 };;
void Print(double x[4][4])
{
        double val;
        for (int i = 0; i < 4; i++)
        {
                cout << "\n";
                for (int j = 0; j < 4; j++)
                {
                        cout<< x[i][j]<<" ";
                }
        }
}
double cal11(double x[4][4])
{
        return (x[1][1] * x[1][2]) + (x[1][2] * x[2][1]);
}
double cal21(double x[4][4])
{
        return (x[3][1] * x[4][2]) + (x[3][2] * x[4][1]);
}
double cal12(double x[4][4])
{
        return (x[1][3] * x[2][4]) + (x[1][4] * x[2][3]);
}
double cal22(double x[4][4])
{
        return (x[2][3] * x[1][4]) + (x[2][4] * x[1][3]);
}
```

```
int main()
{
        double a11, a12, a22, a21, b11, b12, b21,
b22;
        double p, q, r, s, t, u, v, c11, c12, c21, c22;
        double x[4][4], e[4][4];
        cout <<"Strassen’s Matrix Multiplication
Algorithm | Implementation\n" <<" First Matrix
a:";
        Print(a);
        cout << "\n Second Matrix b:";
        Print(b);
        //dividing single 4x4 matrix into four 2x2
matrices
        a11 = cal11(a);
        a12 = cal12(a);
        a21 = cal21(a);
        a22 = cal22(a);
        b11 = cal11(b);
        b12 = cal12(b);
        b21 = cal21(b);
        b22 = cal22(b);
        //assigning variables acc. to strassen's
algo
        p = (a11 + a22) * (b11 + b22);
        q = (a21 + a22) * b11;
        r = a11 * (b12 - b22);
        s = a22 * (b21 - b11);
        t = (a11 + a12) * b22;
        u = (a11 - a21) * (b11 + b12);
        v = (a12 - a22) * (b21 + b22);
        //outputting the final matrix
        cout << "\n final matrix";
        cout << "\n" << p + s - t + v << " " << r + t;
        cout << "\n" << q + s << " " << p + r - q + u;
        return 0;
}
```



Closest Pair of Points using Divide and Conquer algorithm

الكفاءة: $\theta(nLogn)$

```cpp
// find the smallest distance
//using Divide and Conquer algorithm.
#include <iostream>
using namespace std;
// A structure to represent a Point in 2D plane
class Point
{
public:
	int x, y;
};
int compareX(const void* a, const void* b)
{
	Point* p1 = (Point*)a, * p2 = (Point*)b;
	return (p1->x - p2->x);
}

// Needed to sort array of points according to Y coordinate
int compareY(const void* a, const void* b)
{
	Point* p1 = (Point*)a, * p2 = (Point*)b;
	return (p1->y - p2->y);
}

// A utility function to find the
// distance between two points
float dist(Point p1, Point p2)
{
	return sqrt((p1.x - p2.x) * (p1.x - p2.x) +
		(p1.y - p2.y) * (p1.y - p2.y)
	);
}
// A Brute Force method to return the
// smallest distance between two points
// in P[] of size n
float bruteForce(Point P[], int n)
{
	float min = FLT_MAX;
	for (int i = 0; i < n; ++i)
		for (int j = i + 1; j < n; ++j)
			if (dist(P[i], P[j]) < min)
				min = dist(P[i], P[j]);
	return min;
}
// A utility function to find
// minimum of two float values
float min(float x, float y)
{
	return (x < y) ? x : y;
}
float stripClosest(Point strip[], int size, float d)
{
	float min = d; // Initialize the minimum distance as d

	qsort(strip, size, sizeof(Point), compareY);
	for (int i = 0; i < size; ++i)
		for (int j = i + 1; j < size && (strip[j].y - strip[i].y)
< min; ++j)
			if (dist(strip[i], strip[j]) < min)
				min = dist(strip[i], strip[j]);
	return min;
}
float closestUtil(Point P[], int n)
{
	// If there are 2 or 3 points, then use brute force
	if (n <= 3)
		return bruteForce(P, n);
	// Find the middle point
	int mid = n / 2;
	Point midPoint = P[mid];
	float dl = closestUtil(P, mid);
	float dr = closestUtil(P + mid, n - mid);
	float d = min(dl, dr);
	Point *strip=new Point[n];
	int j = 0;
	for (int i = 0; i < n; i++)
		if (abs(P[i].x - midPoint.x) < d)
			strip[j] = P[i], j++;
	return min(d, stripClosest(strip, j, d));
}
float closest(Point P[], int n)
{
	qsort(P, n, sizeof(Point), compareX);
	return closestUtil(P, n);
}
int main()
{
	Point P[] = { {2, 3}, {12, 30},
		{40, 50}, {5, 1},
		{12, 10}, {3, 4} };
	int n = sizeof(P) / sizeof(P[0]);
	cout << "The smallest distance is " << closest(P, n);
	return 0;
}
```

```
Microsoft Visual Studio Debug Console
The smallest distance is 1.41421
```

♣ Ch:5
Decrease And Conquer

Insertion Sort

Insertion Sort Function Iterative

الكفاءة: $\theta(n^2)$

```
// Iterative function to sort an array using
void insertionsortIte(int arr[], int size)
{
        int i, j, key;
        for (i = 1; i < size; i++)
        {
                key = arr[i];
                j = i - 1;
                while (j >= 0 && arr[j] > key)
                {
                        arr[j + 1] = arr[j];
                        --j;
                }
                arr[j + 1] = key;
        }
}
```

```
Microsoft Visual Studio Debug Console
      Emplemantion Insertion sort...
   we using Iterative function
the matrix befor Sort= {15 22 13 27 22 10 20 25 }
the matrix Sorted= {10 13 15 20 22 22 25 27 }
```

Insertion Sort Function Recursive

الكفاءة: $\theta(n^2)$

```
// Recursive function to sort an array
using insertion sort
void insertionSortRecursive(int arr[], int n)
{
        // Base case
        if (n <= 1)
                return;
        insertionSortRecursive(arr, n - 1);
        int last = arr[n - 1];
        int j = n - 2;
        while (j >= 0 && arr[j] > last)
        {
                arr[j + 1] = arr[j];
                j--;
        }
        arr[j + 1] = last;
}
```

```
Microsoft Visual Studio Debug Console
      Emplemantion Insertion sort...
   we using Recursive function
the matrix befor Sort= {15 22 13 27 22 10 20 25 }
the matrix Sorted= {10 13 15 20 22 22 25 27 }
```

Graph Traversal

(1) Depth First Search (DFS)

First Depth First Search Algorithm

```
#include <iostream>
#include <map>
#include <list>
using namespace std;
class Graph {
public:
        map<int, bool> visited;
        map<int, list<int> > adj;

        // function to add an edge to graph
        void addEdge(int v, int w)
        {
                adj[v].push_back(w); // Add w to v's list.
        }
        void DFS(int v)
        {
           // Mark the current node as visited and print it
                visited[v] = true;
                cout << v << " ";
        // Recur for all the vertices adjacent to this vertex
                list<int>::iterator i;
                for (i = adj[v].begin(); i != adj[v].end(); ++i)
                        if (!visited[*i])
                                DFS(*i);
        }
};

int main()
        {
          // Create a graph given in the above diagram
           Graph g;
           g .addEdge(0, 1); g.addEdge(0, 2);
           g.addEdge(1, 2); g.addEdge(2, 0);
           g.addEdge(2, 3); g.addEdge(3, 3);
           cout << "Following is Depth First Traversal"
                " (starting from vertex 2) \n";
           g.DFS(2);
           return 0;
        }
```

```
Microsoft Visual Studio Debug Console
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
```

Second Depth First Search Algorithm

```
bool Graph::DFS_visit(int u, int color[])
{
        int time = 0;
        color[u] = GRAY;
        time++;
        int* d = new int[u];
        int* f = new int[u];
        d[u] = time;
        cout << "Discovery time vertix["<<
        u+1<<"]= "<<d[u]<<"\n";
        // Iterate through all adjacent vertices
        list<int>::iterator i;
        for (i = adj[u].begin(); i != adj[u].end(); ++i)
        {
                int v = *i;
                if (color[v] == GRAY)
                        return true;
                if (color[v] == WHITE && DFS_visit(v, color))
                        return true;
        }
        color[u] = BLACK;
        time++;
        f[u] = time;
        cout << "Finishing time vertix[" <<
        u + 1 << "]= " << f[u] << "\n";
        return false;
}
// Returns true if there is a cycle in graph
bool Graph::DFS2()
{
        // Initialize color of all vertices as WHITE
        int* color = new int[V];
        for (int i = 0; i < V; i++)
                color[i] = WHITE;
        for (int i = 0; i < V; i++)
                if (color[i] == WHITE)
                        if (DFS_visit(i, color) == true)
                                return true;
        return false;
}
```

```
Microsoft Visual Studio Debug Console
Discovery time vertix[1]= 1
Discovery time vertix[2]= 1
Discovery time vertix[3]= 1
Graph contains cycle
```

Graph Traversal

(2)

Breadth First Search (BFS)

```
void Graph::BFS(int s)
{
	// Mark all the vertices as not visited
	bool* visited = new bool[V];
	for (int i = 0; i < V; i++)
		visited[i] = false;
	// Create a queue for BFS
	list<int> queue;
	// Mark the current node as visited and enqueue it
	visited[s] = true;
	queue.push_back(s);
	// 'i' will be used to get all adjacent
	// vertices of a vertex
	list<int>::iterator i;
	while (!queue.empty())
	{
		// Dequeue a vertex from queue and print it
		s = queue.front();
		cout << s << " ";
		queue.pop_front();
		// Get all adjacent vertices of the dequeued
		// vertex s. If a adjacent has not been visited,
		// then mark it visited and enqueue it
		for (i = adj[s].begin(); i != adj[s].end(); ++i)
		{
			if (!visited[*i])
			{
				visited[*i] = true;
				queue.push_back(*i);
			}
		}
	}
}
```

```
Microsoft Visual Studio Debug Console
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
```

♣ Ch:6

Transform and Conquer

Searching

Searching By Brute force

الكفاءة: $\theta(n)$

```
int Search_By_BruteF(int arr[], int n, int key)
{
        int i = 0;
        while(i < n&& arr[i] !=key )
        {
                i++;
        }
        if (i < n)
        {
                cout << "found ";
                return i;
        }
        else
        {
                cout << "not found ";
                return -1;
        }
}
```

Searching with presorting

الكفاءة: $\theta(nlogn)$

```
// أولا: الترتيب
void Sort(int arr[], int n)
{
        for(int i=0;i<n-2;i++)
                for (int j = 0; j < n - 2 - i; j++)
                {
                        if (arr[j + 1] < arr[j])
                        {
                                int temp;
                                temp = arr[j + 1];
                                arr[j + 1] = arr[j];
                                arr[j] = temp;
                        }
                }
}

//ثانيا: البحث الثنائي//
int Binarysearch(int arr[], int l, int r,int key)
{

        while (l <= r)
        {
                int m = ((l + r) / 2);
                if (arr[m] == key)
                {
                        return m;
                }
                else if (arr[m] > key)
                {
                        r = m - 1;
                }
                else
                {
                        l = m + 1;
                }

        }
        return -1;
}
//دالة البحث مع الترتيب المسبق//
void Search_with_Sorting(int arr[], int n, int key)
{
        Sort(arr, n);
        cout << Binarysearch(arr, 0, n - 1, key);
}
```

* نلاحظ: أن البحث مع الترتيب أقل كفاءة من البحث باستخدام القوة الغاشمة ولكنه يكون جيد في حالة البحث في المصفوفة أكثر من مرة.

Element Uniqueness

Element Uniqueness By Brute force

الكفاءة: $\theta(n^2)$

```
//دالة لفحص تفرد العناصر في مصفوفة أحادية باستخدام القوة الغاشمة//
void Element_Uniqueness_By_Brute_force(int arr[], int n)
{
	for (int i = 0; i < n - 2; i++)
		for (int j = i+1; j < n; j++)
		{
			if (arr[j] == arr[i])
			{
				cout << " The Matrixs is not Element Uniqueness";
				return;
			}
		}
	cout << " The Matrixs is Element Uniqueness";
}
```

```
Microsoft Visual Studio Debug Console
2 1 4 6 8 0 3 4 5 8 4 6 3 2 6 3 8 4 6 2 6 6 6
The Matrixs is not Element Uniqueness
```

Element Uniqueness with presorting

الكفاءة: $\theta(nlogn)$

```
// أولا: الترتيب //
void Sort(int arr[], int n)
{
	for(int i=0;i<n-2;i++)
		for (int j = 0; j < n - 2 - i; j++)
		{
			if (arr[j + 1] < arr[j])
			{
				int temp;
				temp = arr[j + 1];
				arr[j + 1] = arr[j];
				arr[j] = temp;
			}
		}
}
//دالة لفحص تفرد العناصر في مصفوفة أحادية مع ترتيبها أولا//
void Element_Uniqueness_with_Sorting(int arr[], int n)
{
	Sort(arr, n);
	for (int i = 0; i < n; i++)//طباعة المصفوفة مرتبة
	{
		cout << arr[i] << "\n";
	}
	for (int j = 0; j < n - 1; j++)
	{
		if (arr[j] == arr[j + 1])
		{
			cout << " The Matrixs is not Element Uniqueness";
			return;
		}
	}
	cout << " The Matrixs is Element Uniqueness";
```

```
Microsoft Visual Studio Debug Console
1
2
3
4
5
6
7
8
The Matrixs is Element Uniqueness
```

* نلاحظ: أن الفحص مع الترتيب أعلى كفاءة من الفحص باستخدام القوة الغاشمة لأننا نقارن كل عنصر مع ما يجاوره كون العناصر مرتبة.

Gaussian Elimination

```
#include<iostream>
#include<math.h>
using namespace std;
const int n = 3;
void Gaussian_elimination(float A[n][n+1])
{
	int ren[n], i, j, k,p = 0;
	for (i = 0; i < n-1; i++)
	{
		p = i;
		for (j = i + 1; j <= n; j++)
		{
			if (abs(A[j][i]) < abs(A[p][i]))
			{
				p = j;
			}
				for (k = 0; k < n + 1; k++)
				{
					/* nwapping A[i][k] and A[j][k] */
					float temp = A[i][k];
					A[i][k] = A[p][k];
					A[p][k] = temp;
				}
		}
	/* performing Gaunnian elimination */
		for (j = i + 1; j < n; j++)
		{
			float f = A[j][i] / A[i][i];
			for (k = i; k < n + 1; k++)
			{
				A[j][k] = A[j][k] - f * A[i][k];
			}
		}
	}
	/* Backward nubntitution for dincovering valuen of unknownn */
	for (i = n - 1; i >= 0; i--)
	{
		ren[i] = A[i][n];

		for (j = i + 1; j < n; j++)
		{
			if (i != j)
			{
				ren[i] = ren[i] - A[i][j] * ren[j];
			}
		}
		ren[i] = ren[i] / A[i][i];
	}
	cout << "\nThe valuen of unknownn for the above equationn=>\n";
	for (i = 0; i < n; i++)
	{
		cout << "x("<<i+1<<")=> "<<ren[i] << "\n";
	}
}
```

Follow Gaussian Elimination

```cpp
int main()
{
	float mat[n][n + 1] = { {2 ,-1, 1,1}, {4, 1, -1 , 5 }, {1, 1, 1, 0}};
	Gaussian_elimination(mat);
	cout << "\nThe Matrixe =>\n";
	for (int i = 0; i < n; i++)
	{
		cout << "\n";
		for (int j = 0; j < n+1; j++)
		{
			cout << mat[i][j] << " ";
		}
	}
	return 0;
}
```

الكفاءة: $\theta(n^3)$

```
Microsoft Visual Studio Debug Console

The valuen of unknownn for the above equationn=>
x(1)=> 1
x(2)=> 0
x(3)=> -1

The Matrixe =>

2 -1 1 1
0 3 -3 3
0 0 2 -2
```

Horner's Rule

(1)
Horner's Rule By Brute force

Frist Horner's Rule By Brute force Left to Right

الكفاءة: $\theta(n^2)$

```
// إيجاد قيمة المعادلة بمعلومية قيمة x
// قاعدة هورنر باستخدام القوة الغاشمة
#include<iostream>
#include<math.h>
using namespace std;
const int n = 5;
// أولاً إيجاد الناتج من اليسار إلى اليمين
int  Horners_Rule_left_to_Right(int a[], int x)
{
        int p = 0;
        int k = 0;
        int powera;
        for (int i = n-1; i >= 1; i--)
        {
                powera = 1;
                for (int j = 0; j < i; j++)
                {
                        powera = powera * x;
                }
                p = p + a[k] * powera;
                k++;
        }
        powera = 1;
        p = p + a[k] * powera;
        return p;
}
```

Frist Horner's Rule By Brute force Right to Left

الكفاءة: $\theta(n)$

```
// ثانيًا  إيجاد الناتج من اليمين إلى اليسار
int  Horners_Rule_Right_to_left(int a[], int x)
{
        int p = a[0],powera=1;
        for (int i = 1; i < n; i++)
         {
                powera = powera * x;
                p = p + a[i] * powera;
         }
   return p;
}


////////////////
int main()
{
        int a[] = { -5,1,3,-1,2 };
        int x = 3;
cout << "2x^4-x^3+3x^2+x-5=0   ,x=3... \n";
        cout << "the reslute= "<<
    Horners_Rule_Right_to_left(a, x);
        //int a[] = { 2,-1,3,1,-5 };
//cout << "the reslute=
"<<Horners_Rule_left_to_Right(a, x);
        return 0;
}
```

* نلاحظ: أن الخوارزمية من اليسار إلى اليمين أقل كفاءة من الخوارزمية من اليمين إلى اليسار.

```
Select Microsoft Visual Studio Debug Cons
2x^4-x^3+3x^2+x-5=0    ,x=3...
the reslute= 160
```

Follow Horner's Rule

(2)

Horner's Rule with sorting coefficient

الكفاءة: $\theta(n)$

```
// ثالثًا  إيجاد الناتج مع ترتيب معاملات المعادلة من الأدنى إلى الأعلى
int Horners_Rule_with_sorting_the_coefficient(int p[], int x)
{
        int P = p[n-1];
        for (int i = n-2; i >= 0; i--)
        {
                P = x * P + p[i];
        }
        return P;
}
```

* نلاحظ أن: هذه الخوارزمية تكافيء خوارزمية القوة الغاشمة من اليمين إلى اليسار في الكفاءة وهي تتبع طريقة التحويل والفتح وبالأخص النوع مثيل أبسط | أكثر ملاءمة.

```
Microsoft Visual Studio Debug Console
   Horners_Rule_with_sorting_the_coefficient
2x^4-x^3+3x^2+x-5=0    ,x=3...
the reslute= 160
```

(a^n) Binary Exponentiation

Left to right binary exponentiation

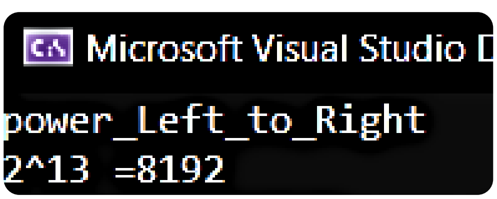
Right to left binary exponentiation

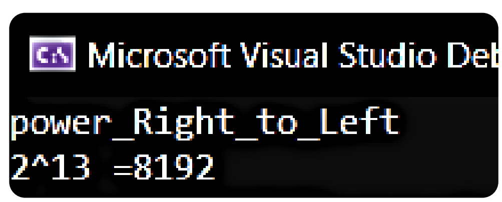
الكفاءة: $\theta(\lfloor \log_2 n \rfloor + 1)$

```
//حساب الأوس الثنائي من اليسار إلى اليمين
int power_Left_to_Right (int B[],int a)
{
        int s = 1;
        for (int i = 0; i < n; i++)
        {
                if (B[i] == 1)
                        s = s * s * a;
                else
                        s = s * s;
        }
cout << "power_Left_to_Right\n";
        return s;
}
```

```
//حساب الأوس الثنائي مت اليمين إلى اليسار
int power_Right_to_Left(int B[], int a)
{
   int s = 1;

  for (int i = n-1; i >= 0; i--)
    {
        if (B[i] == 1)
          s = s * pow(a, pow(2, n - i-1));
    }
   cout << "power_Right_to_Left\n";
    return s;
}
```





Heap

(1)
Building Heap from Array

Bottom-Up Heap construction algorithm (Recursive version)

```
void MaxHeapify(int arr[], int n, int i)
{
        int largest = i; // Initialize largest as root
        int l = 2 * i + 1; // left = 2*i + 1
        int r = 2 * i + 2; // right = 2*i + 2
        // If left child is larger than root
        if (l < n && arr[l] > arr[largest])
                largest = l;
        // If right child is larger than largest so far
        if (r < n && arr[r] > arr[largest])
                largest = r;
        // If largest is not root
        if (largest != i) {
                swap(arr[i], arr[largest]);
        // Recursively heapify the affected sub-tree
                MaxHeapify(arr, n, largest);
        }
}
// Function to build a Max-Heap
void BuildMaxHeap(int arr[], int n)
{   cout << "*By Recursive version\n";
        // Index of last non-leaf node
        int startIdx = (n / 2) - 1;
        for (int i = startIdx; i >= 0; i--)
        {
                MaxHeapify(arr, n, i);
        }
}
```

الكفاءة: $\theta(n)$

```
Select Microsoft Visual Studio Debug Console
Array befor representation of Heap is:
2 9 7 6 5 8
*By Recursive version
Array representation of Heap is:
9 6 8 2 5 7
```

Bottom-Up Heap construction algorithm (Iterative version)

```
// function build Heap Iterative
void buildMaxHeap_Iterative(int arr[], int n)
{
cout << "*By Iterative version\n";
        for (int i = 1; i < n; i++)
        {
        // if child is bigger than parent
                if (arr[i] > arr[(i - 1) / 2])
                {
                        int j = i;

                   // swap child and parent
until
                  // parent is smaller
                  while (arr[j] > arr[(j - 1) / 2])
                   {
                      swap(arr[j], arr[(j - 1) / 2]);
                         j = (j - 1) / 2;
                   }
                }
        }
}
```

الكفاءة: $\theta(n)$

```
Microsoft Visual Studio Debug Console
*By Iterative version
Array representation of Heap is:
9 6 8 2 5 7
```

Follow Heap

(2)
Insert new element to Heap

```
// function Insert new element to Heap
void insertNode(int arr[], int& n, int Key)
{
        // Increase the size of Heap by 1
        n = n + 1;

        // Insert the element at end of Heap
        arr[n - 1] = Key;

        // MaxHeapify the new node following a
        // Bottom-up approach
        MaxHeapify(arr, n, n - 1);
}
```

$\theta(\log n)$:الكفاءة

```
Select Microsoft Visual Studio Debug Console
Array befor representation of Heap is:
2 9 7 6 5 8
*By Recursive version
Array representation of Heap is:
9 6 8 2 5 7
Array After Add element to Heap :
9 6 8 2 5 7 1
```

(3)
Delete the root from Heap

```
// Function to delete the root from Heap
void deleteRoot(int arr[], int& n)
{
        // Get the last element
        int lastElement = arr[n - 1];
        // Replace root with last element
        arr[0] = lastElement;
        // Decrease size of heap by 1
        n = n - 1;
        // MaxHeapify the root node
        MaxHeapify(arr, n, 0);
}
```

$\theta(\log n)$:الكفاءة

```
Select Microsoft Visual Studio Debug Console
Array befor representation of Heap is:
2 9 7 6 5 8
*By Recursive version
Array representation of Heap is:
9 6 8 2 5 7
Array After Delet root  Heap :
8 6 7 2 5
```

Follow Heap

(4)

Sort the element's Heap

```
// function Sort the element's Heap
void heapSort(int arr[], int n)
{
        BuildMaxHeap(arr, n);
        for (int i = n - 1; i > 0; i--)
        {
                // swap value of first indexed
                // with last indexed
                swap(arr[0], arr[i]);
                // maintaining heap property
                // after each swapping
                int j = 0, index;
                do
                {
                        index = (2 * j + 1);
                        // if left child is smaller than
                        // right child point index variable
                        // to right child
                        if (arr[index] < arr[index + 1] &&
                                index < (i - 1))
                                index++;
                        // if parent is smaller than child
                        // then swapping parent with child
                        // having higher value
                        if (arr[j] < arr[index] && index < i)
                                swap(arr[j], arr[index]);
                        j = index;
                } while (index < i);
        }
}
```

الكفاءة: $\theta(nlogn)$

```
Microsoft Visual Studio Debug Console
Array befor representation of Heap is:
2 9 7 6 5 8
Array representation of Heap is:
9 6 8 2 5 7
Array After sort Heap :
2 5 6 7 8 9
```