1)postfix evaluate

```
#include<iostream>

#include<stack>

#include<cmath>

using namespace std;


int evaluatePostfix(const string& expr)

 {

   stack<int> stk;

   for (char c : expr)

       {

       if (isdigit(c))

           stk.push(c - '0');

       else

       {

           int b = stk.top(); stk.pop();

           int a = stk.top(); stk.pop();

           switch (c) {

               case '+': stk.push(a + b); break;

               case '-': stk.push(a - b); break;

               case '*': stk.push(a * b); break;

               case '/': stk.push(a / b); break;

               case '^': stk.push(pow(a, b)); break;

               default: cout << "Invalid operator\n"; return -1;

           }
```

```
        }
    }
    return stk.top();
}

int main()
{
    string expr;
    cout << "Enter postfix expression: ";
    getline(cin, expr);
    int result = evaluatePostfix(expr);
    if (result != -1) cout << "Result: " << result << endl;
    return 0;
}
```

2)Linear and binary recursive

*

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

int lsr(int arr[], int start, int end, int key) {
    if (start > end)
    {
        return -1;
    }
    if (arr[start] == key)
    {
        return start;
    }

    return lsr(arr, start + 1, end, key);
}

int main()
{
    int n;
    cout << "Enter the number of elements in the array: ";
    cin >> n;

    int arr[n];
    cout << "Enter " << n << " elements:\n ";
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }

    int key;
    cout << "Enter the key to search for: ";
    cin >> key;

    int result = lsr(arr, 0, n - 1, key);
    if (result != -1)
    {
        cout << "Element found at index " << result << endl;
    }
    else
    {
        cout << "Element not found in the array" << endl;
    }
    return 0;
}
```
****

2)BSR

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

int bsr(int arr[], int low, int high, int key)
{
if (low > high)
{
return -1;
}
int mid = low + (high - low) / 2;
if (arr[mid] == key)
 {
return mid;
}
else if (arr[mid] < key)
{

return bsr(arr, mid + 1, high, key);
}
else
{
return bsr(arr, low, mid - 1, key);
}
}

int main()
{
int n;
cout << "Enter the number of elements in the array: ";
cin >> n;

int arr[n];
cout << "Enter " << n << " sorted elements: " <<"\n";
for (int i = 0; i < n; i++)
{
cin >> arr[i];
}

int key;
```

```
cout << "Enter the key to search for: ";

cin >> key;

int result = bsr(arr, 0, n - 1, key);

if (result != -1)

{

cout << "Element found at index " << result << endl;

} else {

cout << "Element not found in the array" << endl;

}

return 0;

}
```

3) Linear and binary Non-recursive

```
#include <iostream>

using namespace std;

int linearSearch(int arr[], int n, int key)

{

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

{

if (arr[i] == key)

{

return i;

}

}

return -1;

}
```

```
int main()

{

int n;

cout << "Enter the number of elements in the array: ";

cin >> n;

int arr[n];

cout << "Enter " << n << " elements: "<<"\n";

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

{

cin >> arr[i];

}

int key;

cout << "Enter the key to search for: ";

cin >> key;

int result = linearSearch(arr, n, key);

if (result != -1)

{

cout << "Element found at index " << result << endl;

} else {

cout << "Element not found in the array" << endl;

}

return 0;

}
```

2)BSR Non

```
#include <iostream>

using namespace std;
```

```
int binarySearch(int arr[], int n, int key)
{
    int low = 0, high = n - 1;
    while (low <= high)
        {
        int mid = low + (high - low) / 2;

        if (arr[mid] == key)
        {
            return mid;
        }
        else if(arr[mid] < key)
        {
            low = mid + 1;
        }
        else
        {
            high = mid - 1;
        }
    }
    return -1;
}
int main()
{
    int n;
    cout << "Enter the number of elements in the array: ";
    cin >> n;
    int arr[n];
    cout << "Enter " << n << " sorted elements: "<<"\n";
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    int key;
    cout << "Enter the key to search for: ";
    cin >> key;
    int result = binarySearch(arr, n, key);
    if (result != -1)
    {
        cout << "Element found at index " << result << endl;
    }
else
    {
        cout << "Element not found in the array" << endl;
    }

    return 0;
}
```

4)Tower of Honai

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

```
void towerOfHanoi(int n, char source, char auxiliary, char destination) {
if (n == 1) {
cout << "Move disk 1 from rod " << source << " to rod " << destination << endl;
return;
}
towerOfHanoi(n - 1, source, destination, auxiliary);
cout << "Move disk " << n << " from rod " << source << " to rod " << destination << endl;
towerOfHanoi(n - 1, auxiliary, source, destination);
}
int main() {
int n;
cout << "Enter the number of disks: ";
cin >> n;
towerOfHanoi(n, 'A', 'B', 'C');
return 0;
}
```

5)BubbleSort

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

void bubbleSort(int arr[], int n)
{
for (int i=0; i<n-1; i++)
{
```

```
for (int j=0; j<n-i-1; j++)

{

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

{

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}


void printArray(int arr[], int size)

{

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

{

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

}

cout << endl;

}


int main()

{

int n;

cout << "Enter the number of elements: ";

cin >> n;
```

```
int arr[n];
cout << "Enter " << n << " elements:" << endl;
for (int i = 0; i < n; i++)
{
cin >> arr[i];
}

cout << "Original array: ";
printArray(arr, n);
bubbleSort(arr, n);
cout << "Sorted array: ";
printArray(arr, n);
return 0;
}
```

6)Selection Sort

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

void selectionSort(int arr[], int n)
{
for (int i = 0; i < n - 1; i++)
{
int minIndex = i;
for (int j = i + 1; j < n; j++)
{
if (arr[j] < arr[minIndex])
{
minIndex = j;
}
}

int temp = arr[i];
arr[i] = arr[minIndex];
arr[minIndex] = temp;
}
}

void printArray(int arr[], int size)
{
for (int i = 0; i < size; i++)
{
cout << arr[i] << " ";
}
cout << endl;
}

int main() {
int n;
cout << "Enter the number of elements: ";
cin >> n;
int arr[n];
cout << "Enter " << n << " elements:" << endl;
```

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

{

cin >> arr[i];

}


cout << "Original array: ";

printArray(arr, n);

selectionSort(arr, n);

cout << "Sorted array: ";

printArray(arr, n);

return 0;

}
```

7)Quick Sort

```
#include <iostream>

using namespace std;


int partition(int arr[], int low, int high)

{

int pivot = arr[high];

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;

}


void quickSort(int arr[], int low, int high)

{

if (low < high) {


int pi = partition(arr, low, high);


quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}


void printArray(int arr[], int size) {

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

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

}

cout << endl;

}

int main() {
```

```
int n;

cout << "Enter the number of elements: ";

cin >> n;

int arr[n];

cout << "Enter " << n << " elements:" << endl;

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

cin >> arr[i];

}

cout << "Original array: ";

printArray(arr, n);

quickSort(arr, 0, n - 1);


   cout << "Sorted array: ";

   printArray(arr, n);


   return 0;

}
```

8)Insertion Sort

```
#include <iostream>

using namespace std;

void insertionSort(int arr[], int n)

{

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

{

int key = arr[i];

int j = i - 1;
while (j >= 0 && arr[j] > key)

{

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

}


void printArray(int arr[], int size) {

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

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

}

cout << endl;

}

int main() {

int n;

cout << "Enter the number of elements: ";

cin >> n;

int arr[n];

cout << "Enter " << n << " elements:" << endl;

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

cin >> arr[i];

}

cout << "Original array: ";
```

```
printArray(arr, n);

insertionSort(arr, n);

cout << "Sorted array: ";

printArray(arr, n);

return 0;

}
```

9)Merge Sort

```
#include <iostream>

using namespace std;

void merge(int arr[], int l, int m, int r)

{

int n1 = m - l + 1;

int n2 = r - m;


int L[n1], R[n2];


for (int i = 0; i < n1; i++)

L[i] = arr[l + i];

for (int j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];


int i = 0;

int j = 0;

int k = l;

while (i < n1 && j < n2)
```

```
{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}


while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}


while (j < n2) {

arr[k] = R[j];

j++;

k++;

}
```

```
}

void mergeSort(int arr[], int l, int r) {
if (l < r) {

int m = l + (r - l) / 2;
/
mergeSort(arr, l, m);
mergeSort(arr, m + 1, r);

merge(arr, l, m, r);
}
}

void printArray(int arr[], int size)
{
for (int i = 0; i < size; i++) {
cout << arr[i] << " ";
}
cout << endl;
}
int main() {
int n;
cout << "Enter the number of elements: ";
cin >> n;
int arr[n];
```

```
cout << "Enter " << n << " elements:" << endl;

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

cin >> arr[i];

}

cout << "Original array: ";

printArray(arr, n);

mergeSort(arr, 0, n - 1);

cout << "Sorted array: ";

printArray(arr, n);

return 0;

}
```

10)Brute Force

```
#include <iostream>

#include <string>

using namespace std;


void bruteForceStringMatch(const string& text, const string& pattern)

{

int n = text.length();

int m = pattern.length();

int count = 0;

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

{

int j;

for (j = 0; j < m; ++j) {

if (text[i + j] != pattern[j])
break;
}
if (j == m)
{
count++;
cout << "Pattern found at index " << i << endl;
}
}
if (count == 0)
{
cout << "Pattern not found in the text." << endl;
} else {
cout << "Pattern found " << count << " time(s) in the text." << endl;
}
}
int main()
{
string text, pattern;
cout << "Enter the text: ";
getline(cin, text);
cout << "Enter the pattern to search for: ";
getline(cin, pattern);
bruteForceStringMatch(text, pattern);
return 0;
}
```