8. Searching Algorithm

What is search algorithm?
A search algorithm is an algorithm for finding an item among a collection of items.

Sequential/Linear Search Algorithm:
It examines the first element in the list and then second element and so on until a much is found.

Pseudo code:
```
int sequentialSearch(a[],n,t) //It returns the location of the target t in the array a[] with n elements.
  for i = 0 to n-1
    if (a[i]=t) return i;
  next i
  return -1;
```

C++ Implementation:
```
#include<iostream.h>
#include<conio.h>

int sequentialSearch(int *a, int n, int t)
{
  int i;
  for (i = 0; i < n; i++)
    if (a[i]==t) return i;
  return (-1);
}

void main ()
{
  clrscr();
  int num[] = {4, 65, 2, -31, 0, 99, 2, 83, 782, 1};
  int n = 10;
  int t = 99;
  cout<<t<<" is found at "<<sequentialSearch(num, n, t)<<".";
}
```

Output: 99 is found at 5.

Binary Search Algorithm:
Here the elements should be in (ascending) order and the elements should be saved in a randomly accessible data structure like array.
The basic algorithm is to find the middle element of the list, compare it against the key/target, decide which half of the list must contain the key, and repeat with that half.

Pseudo code:
```
int binarySearch(a[],l,u,t) //It returns the location of t in the array a[] from the index l to u.
     p = ( l + u) / 2;
     while(a[p] ≠ t AND l<=u)
         if (a[p] > t)
             u = p - 1
         else
             l = p + 1
         p = (l + u) / 2
     end while
    if (l <= u)
        return p
     else
        return -1
```

C++ Implementation:

```
int binarySearch(int* a, int l, int u, int t)
{
   int p;
   p = ( l + u) / 2;
   while((a[p] != t) && (l<=u))
   {
      if (a[p] > t)
         u = p - 1;
      else
         l = p + 1;
      p = (l + u) / 2;
   }
   if (l <= u)
      return p;
   else
      return (-1);
}

void main ()
{
   clrscr();
   int num[] = {1, 2, 7, 9, 50, 99, 100, 150, 190, 200};
   int n = 10;
   int t = 99;
   cout<<t<<" is found at "<<binarySearch(num, 0, n-1, t)<<".";
}
```

Output: 99 is found at 5.

Recursive Pseudo Code:

```
int recBinarySearch(a[],l,u,t) //It returns the location of t in the array a[] from the index l to u.
   if l>u then
      return -1
   else
      mid=(l+u)/2
      if t=a[mid] then
         return mid
      else if t<a[mid] then
         return recBinarySearch(a[],l,mid-1,t)
      else
         return recBinarySearch(a[],mid+1,u,t)
      end if
   end if
```

Recursive C++ Code:

```
int recBinarySearch(int* a, int l, int u, int t)
{
   int mid;
   if (l>u)
      return (-1);
   else
   {
      mid=(l+u)/2;
      if (t==a[mid])
            return mid;
      else if (t<a[mid])
            return recBinarySearch(a,l,mid-1,t);
```

```
    else
        return recBinarySearch(a,mid+1,u,t);
  }
}

void main ()
{
  clrscr();
  int a[] = {1, 2, 7, 9, 50, 99, 100, 150, 190, 200};
  int n = 10;
  int t = 99;
  cout<<t<<" is found at "<<recBinarySearch(a, 0, n-1, t)<<".";
}
```

Output: 99 is found at 5.

Efficiency of the Search Algorithms (Best, Worst and Average Cases):

Searching Technique	***Best case***	***Average Case***	***Worst Case***
Sequential Search	O(1)	O(n)	O(n)
Binary Search	O(1)	O (log n)	O(log n)

The difference between O(log(N)) and O(N) is extremely significant when N is large.
For example, suppose your array contains 2 billion values, the sequential search would involve about a billion comparisons; binary search would require only 32 comparisons!