DSA Assessment - 5

Name : Jagtap Mahesh

Reg No. 24MCS1017

Objective:

To understand and implement both the Radix Sort algorithm for sorting, and the Exponential Search algorithm, which finds the range where the target element might be present and then applies binary search within that range.

Problem Statement 1:

You are given an unsorted array of n elements: [34, 12, 57, 65, 7, 74, 93, 88, 81, 100]. Your task is to:

Sort the Array: First, implement the Radix Sort algorithm to sort the given array.

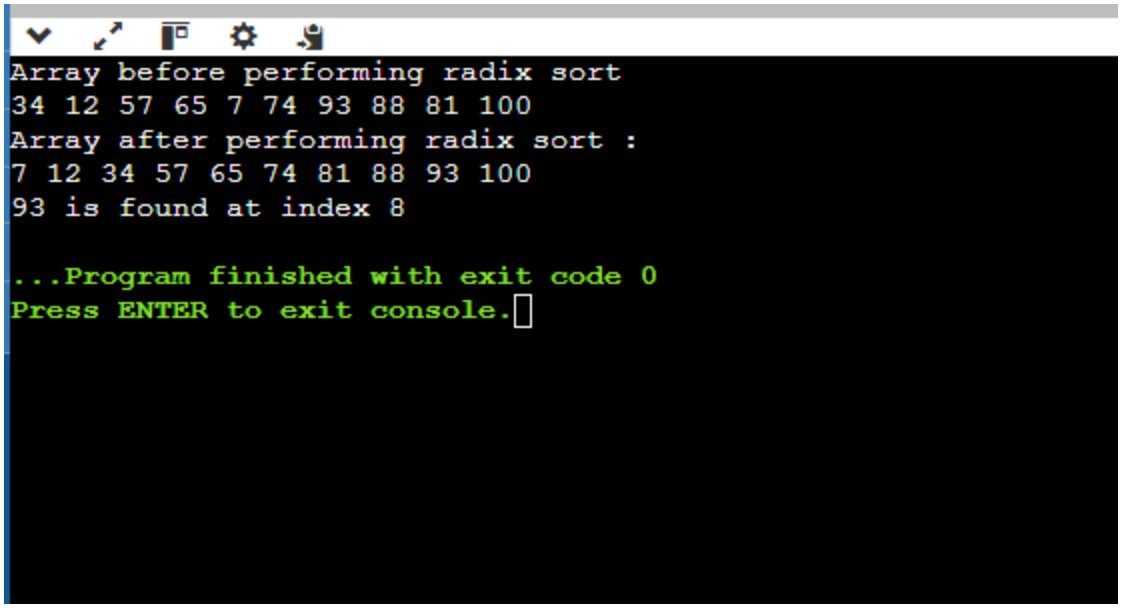
Search the Target Value: After sorting the array, apply the Exponential Search algorithm to find the position of the target value x = 93. If 93 is present, return its index in the sorted array; otherwise, return -1.

```
#include <bits/stdc++.h>
using namespace std;

void count_sort(int arr[], int n, int pos)
{
        int count[10] = { 0 };

        // count the frequency of each distinct digit at
        // given place for every element in the original array
        for (int i = 0; i < n; i++) {
                count[(arr[i] / pos) % 10]++;
        }

        //  perform prefix sum and update the count array
        for (int i = 1; i < 10; i++) {
                count[i] = count[i] + count[i - 1];
        }

        // store our answer in the ans array
        int ans[n];
        for (int i = n - 1; i >= 0; i--) {
```

```
		ans[--count[(arr[i] / pos) % 10]] = arr[i];
	}

	// copy the contents of ans array to our
	// original array
	for (int i = 0; i < n; i++) {
		arr[i] = ans[i];
	}
}

// function to implement radix sort
void radix_sort(int arr[], int n)
{
	int k = *max_element(arr, arr + n);

	for (int pos = 1; (k / pos) > 0; pos *= 10) {
		count_sort(arr, n, pos);
	}
}
int binarySearch(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 binarySearch(arr, l, mid-1, x);
		return binarySearch(arr, mid+1, r, x);
	}
	return -1;
}

int exponentialSearch(int arr[], int n, int x)
{
	if (arr[0] == x)
		return 0;

	int i = 1;
	while (i < n && arr[i] <= x)
		i = i*2;

	return binarySearch(arr, i/2, min(i, n), x);
}
```

```
int main()
{
	int arr[] = { 34, 12, 57, 65, 7, 74, 93, 88, 81, 100 };
	int n = sizeof(arr) / sizeof(arr[0]);
	cout << "Array before performing radix sort"<<endl;
	for (int i = 0; i < n; i++) {
		cout << arr[i] << " ";
	}
	cout<<endl;
	radix_sort(arr, n);

	cout << "Array after performing radix sort : " << endl;
	for (int i = 0; i < n; i++) {
		cout << arr[i] << " ";
	}
	cout<<endl;

	int x=93;
  int result = exponentialSearch(arr, n, x);
  if (result==-1){
    cout<<"element is not present in the array.";
  }
  else {
    cout<<x<<" is found at index "<<result;
  }
	return 0;
}
```

OUTPUT:



Problem Statement 2: You are given an unsorted array of n elements: [7, 5, 11, 1, 3, 9, 17, 13, 15, 19, 21]. Your task is to:

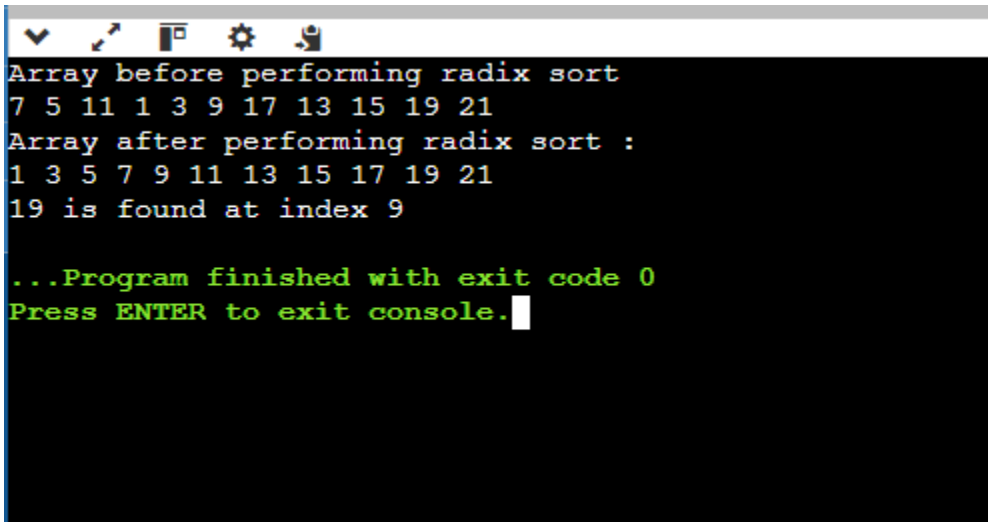
Sort the Array: First, implement the Radix Sort algorithm to sort the given array.

Search the Target Value: After sorting the array, apply the Exponential Search algorithm to find the position of the target value x = 19. If 19 is present, return its index in the sorted array; otherwise, return -1.

```
#include <bits/stdc++.h>
using namespace std;

void count_sort(int arr[], int n, int pos)
{
        int count[10] = { 0 };

        // count the frequency of each distinct digit at
        // given place for every element in the original array
        for (int i = 0; i < n; i++) {
                count[(arr[i] / pos) % 10]++;
        }
```

```
        //  perform prefix sum and update the count array
        for (int i = 1; i < 10; i++) {
                count[i] = count[i] + count[i - 1];
        }

        // store our answer in the ans array
        int ans[n];
        for (int i = n - 1; i >= 0; i--) {
                ans[--count[(arr[i] / pos) % 10]] = arr[i];
        }

        // copy the contents of ans array to our
        // original array
        for (int i = 0; i < n; i++) {
                arr[i] = ans[i];
        }
}

// function to implement radix sort
void radix_sort(int arr[], int n)
{
        int k = *max_element(arr, arr + n);

        for (int pos = 1; (k / pos) > 0; pos *= 10) {
                count_sort(arr, n, pos);
        }
}
int binarySearch(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 binarySearch(arr, l, mid-1, x);
                return binarySearch(arr, mid+1, r, x);
        }
        return -1;
}
```

```
int exponentialSearch(int arr[], int n, int x)
{
	if (arr[0] == x)
		return 0;

	int i = 1;
	while (i < n && arr[i] <= x)
		i = i*2;

	return binarySearch(arr, i/2, min(i, n), x);
}

int main()
{
	int arr[] = { 7, 5, 11, 1, 3, 9, 17, 13, 15, 19, 21 };
	int n = sizeof(arr) / sizeof(arr[0]);
	cout << "Array before performing radix sort"<<endl;
	for (int i = 0; i < n; i++) {
		cout << arr[i] << " ";
	}
	cout<<endl;
	radix_sort(arr, n);

	cout << "Array after performing radix sort : " << endl;
	for (int i = 0; i < n; i++) {
		cout << arr[i] << " ";
	}
	cout<<endl;

	int x=19;
  int result = exponentialSearch(arr, n, x);
  if (result==-1){
    cout<<"element is not present in the array.";
  }
  else {
    cout<<x<<" is found at index "<<result;
  }
	return 0;
}
```



Specify the worst-case time complexity for radix sort and exponential search algorithm

1. **Radix Sort:**
   Worst-case time complexity: **O(d * (n + k))**
   - Where n is the number of elements, d is the number of digits in the largest number, and k is the range of the digits (e.g., base 10 for decimal numbers).

2. **Exponential Search:**
   Worst-case time complexity: **O(log n)**
   - After finding the range where the element might be, a binary search is performed, which has a worst-case time complexity of O(logn).