DSA Lab Assessment - 4

Name : Mahesh Jagtap

Reg.No.: 24MCS1017

Imagine you are a data analyst working for a logistics company that manages the delivery of packages across a complex network of routes. The company stores data about the delivery times (in minutes) for each route in a list. This data needs to be efficiently sorted and searched regularly to optimize the delivery process.

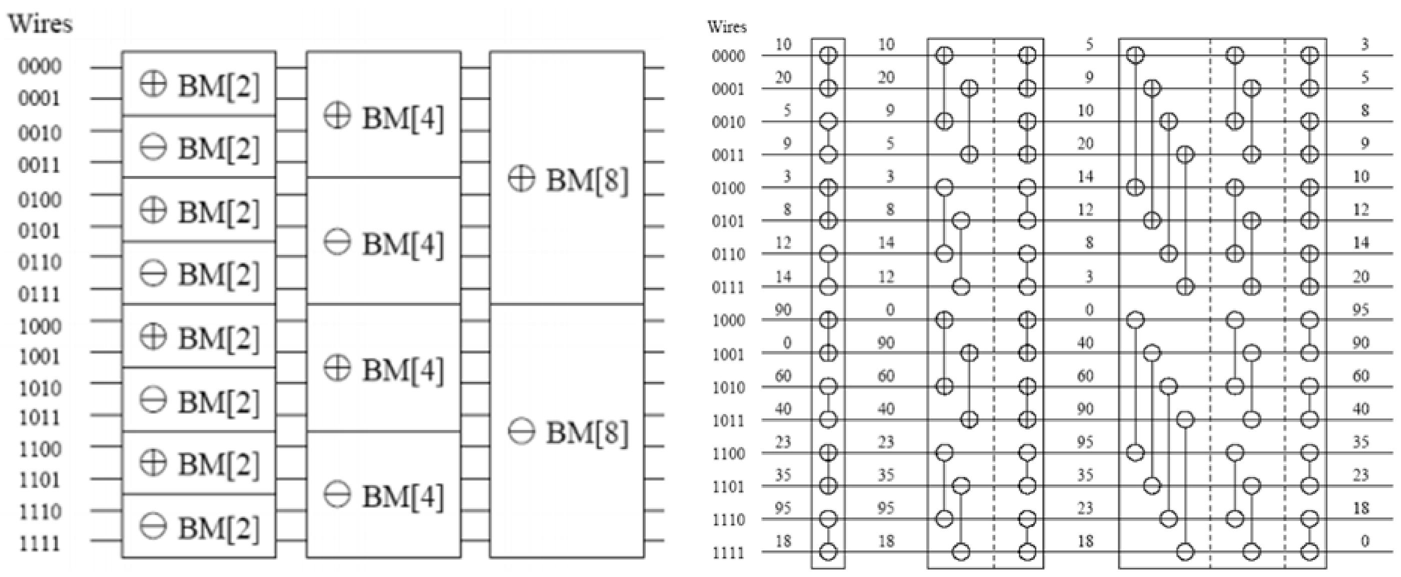
The current unsorted data of delivery times for 16 different routes is: [10, 20, 5, 9, 3, 8, 12, 14, 90, 0, 60, 40, 23, 35, 95, 18]. Your task is to determine whether a specific target delivery time X exists within this dataset and, if it does, identify its location.

Question:

You decide to first sort the delivery times using a bitonic sort network and then perform an interpolation search to locate the target delivery time X.

1. **Using the provided bitonic sort network diagram, implement how the delivery times are arranged into a bitonic sequence and sorted step by step.** ***Note: Bitonic sort can only be used when the number of elements to sort is .***

Bitonic Sort Network Diagram

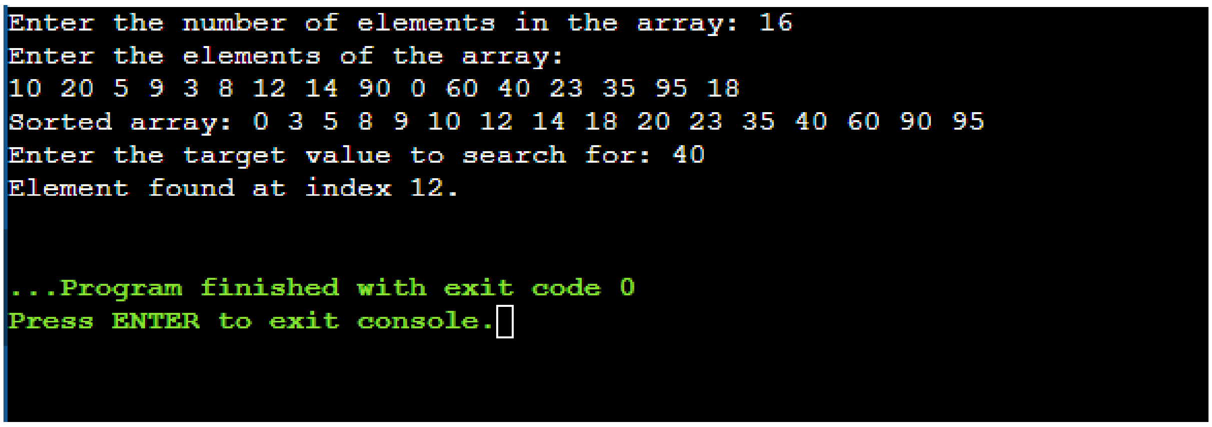


2. **Once the list is sorted, implement interpolation search to efficiently locate the target delivery time X. Assume X=40.**

3. **Consider a situation where the delivery times are not uniformly distributed, with some clusters of routes having very similar times. Analyze how the time complexity of bitonic sort and interpolation search performs in such a scenario.**

ANSWER:

```
#include <iostream>
#include <algorithm>

using namespace std;

// Function to perform a bitonic merge
void bitonicMerge(int arr[], int low, int cnt, bool up) {
    if (cnt > 1) {
        int k = cnt / 2;
        for (int i = low; i < low + k; i++) {
            if ((arr[i] > arr[i + k]) == up) {
                swap(arr[i], arr[i + k]);
            }
        }
        bitonicMerge(arr, low, k, up);
        bitonicMerge(arr, low + k, k, up);
    }
}

// Function to perform bitonic sort
void bitonicSort(int arr[], int low, int cnt, bool up) {
    if (cnt > 1) {
        int k = cnt / 2;
        bitonicSort(arr, low, k, true);
        bitonicSort(arr, low + k, k, false);
        bitonicMerge(arr, low, cnt, up);
    }
}

// Function to perform interpolation search
int interpolationSearch(const int arr[], int n, int target) {
    int low = 0;
    int high = n - 1;

    while (low <= high && target >= arr[low] && target <= arr[high]) {
        if (arr[high] == arr[low]) {
            break;
        }

        int pos = low + ((target - arr[low]) * (high - low) / (arr[high] - arr[low]));
        if (arr[pos] == target) {
            return pos;
        }
        if (arr[pos] < target) {
            low = pos + 1;
        }

        else {
            high = pos - 1;
        }
    }

    return -1;
}

int main() {
    int n;

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

    if (n & (n - 1)) {
        cout << "Number of elements must be a power of 2 for bitonic sort." << endl;
        return 1;
    }

    int* arr = new int[n];

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

    bitonicSort(arr, 0, n, true);

    cout << "Sorted array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

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

    int result = interpolationSearch(arr, n, target);
    if (result != -1) {
        cout << "Element found at index " << result << ".\n";
    } else {
        cout << "Element not found in the array.\n";
    }

    delete[] arr;

    return 0;
}
```



```
Enter the number of elements in the array: 16
Enter the elements of the array:
10 20 5 9 3 8 12 14 90 0 60 40 23 35 95 18
Sorted array: 0 3 5 8 9 10 12 14 18 20 23 35 40 60 90 95
Enter the target value to search for: 40
Element found at index 12.


...Program finished with exit code 0
Press ENTER to exit console.
```

Time complexity analysis:

1)Bitonic Sort

Bitonic sort sorts data by first arranging it into a bitonic sequence (which means it’s partly increasing and partly decreasing) and then merging these sequences into a completely sorted list.

It always takes O($\log^2$ n) time, no matter how the data is distributed. This is because the algorithm's steps are predetermined and don’t adapt to data patterns.

Whether the data is clustered or scattered, bitonic sort will perform the same number of operations. It doesn’t change based on how similar or different your data points are, so its performance remains steady but doesn't improve or worsen with the data distribution.

2)Interpolation Search

Interpolation search tries to estimate where the target value might be based on its value relative to the rest of the data. It works best when data is evenly distributed.On average, it’s very fast, with a time complexity of O(log(logn) but this is assuming the data is well spread out.

If the data is clustered (e.g., many similar values grouped together), interpolation search might struggle. The estimated position can be way off, leading to a lot more comparisons. In the worst-case scenario, it could end up performing as poorly as a linear search, which is O(n).