MIT WORLD PEACE UNIVERSITY

Object Oriented Programming with Java and C++
Second Year B. Tech, Semester 1

IMPLEMENTATION OF BUCKET SORT

PRACTICAL REPORT

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A2, PA 20

October 9, 2022

Contents

1 Objectives

1. To understand use of an array of structures for maintaining records
2. To implement, analyze and compare linear and binary search.
3. To implement, analyze and compare selection, insertion sort and shell sort.

2 Problem Statements

Write a C program to create a student database using an array of structures. Apply searching (Linear and Binary Search) and sorting techniques(Insertion Sort, Selection Sort, Shell sort).

3 Theory

3.1 Bucket Sort

Bucket Sort is a sorting algorithm that divides the unsorted array elements into several groups called buckets. Each bucket is then sorted by using any of the suitable sorting algorithms or recursively applying the same bucket algorithm.

Finally, the sorted buckets are combined to form a final sorted array.

Advantages of Bucket sort

1. It is simple and performs faster computation for numbers or strings
2. Bucket sort can be used as an external sorting algorithm.
3. Once the list is grouped in buckets then it can be processed independently.

Disadvantages of Bucket Sort

1. You can't apply it to all data types because you need a good bucketing scheme.
2. Bucket sort's efficiency is sensitive to the distribution of the input values, so if you have tightly-clustered values, it's not worth it.
3. In many cases where you could use bucket sort, you could also use another specialized sorting algorithm like radix sort, counting sort, or burstsort instead and get better performance.
4. The performance of bucket sort depends on the number of buckets chosen, which might require some extra performance tuning compared to other algorithms.

3.2 Example

Here is an Example of how Bucket Sort Works.

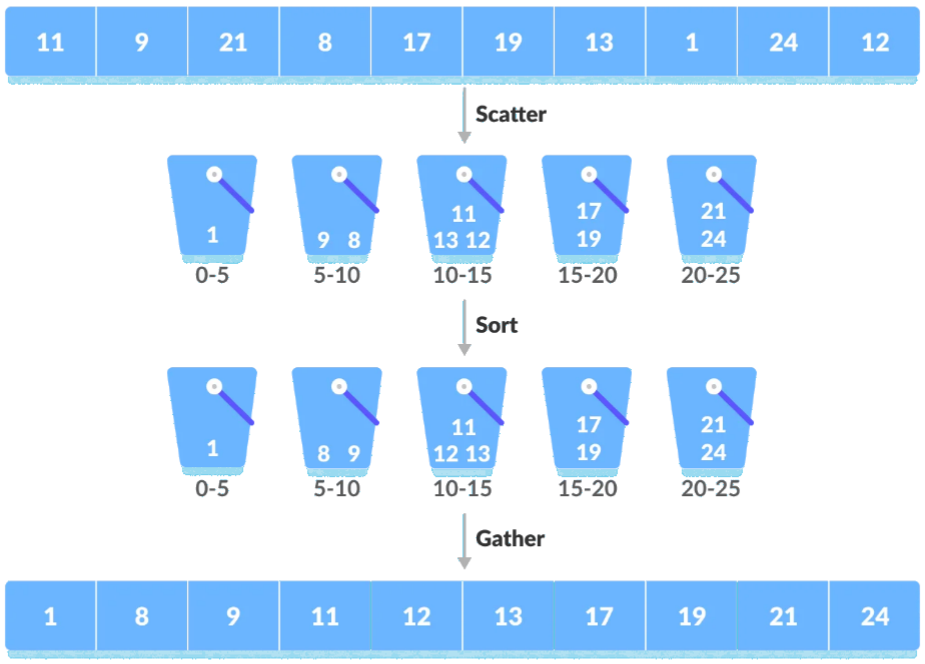

Figure 1: An Example of the Working of Bucket Sort

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers : gcc on linux for C

5 Input

- Array of Integer Elements
- Array of Floating Point Elements

6 Output

- Sorted Elements

7 Test Conditions

1. Input at least 10 records.
2. Test bucket sorting methods for at least 10 records.

8 Code

8.1 Pseudo Code

8.1.1 Pseudo Code for Implementation of Bucket Sort

```
void bucket_sort_integers(int arr[], int size)
{
  max = find_max(arr, size) + 1;
  int bucket[max];
  for (int i = 0; i < max; i++)
    bucket[i] = 0;
  for (int i = 0; i < size; i++)
    bucket[arr[i]]++
  for (int i = 0, j = 0; i < max; i++)
    for (; bucket[i] > 0; bucket[i]--)
    {
      arr[j] = i
      j++
    }
}
```

8.2 C Implementation of Problem Statement

```c
#include <stdio.h>

void accept(int arr[], int size)
{
    printf("\nAccepting the Array: \n");
    for (int i = 0; i < size; i++)
    {
        scanf("%d", &arr[i]);
    }
}

void display(int arr[], int size)
{
    printf("\nDisplaying the Array\n");
    for (int i = 0; i < size; i++)
    {
        printf("%d ", arr[i]);
    }
}

int find_max(int arr[], int size)
{
    int max = 0;
    for (int i = 0; i < size; i++)
    {
        if (arr[i] > max)
        {
            max = arr[i];
        }
    }
    return max;
}

```

```c
void bucket_sort_integers(int arr[], int size)
{
    int i, j;
    int max = find_max(arr, size) + 1;
    int bucket[max];
    // declaring bucket values to 0
    for (int i = 0; i < max; i++)
    {
        bucket[i] = 0;
    }
    for (int i = 0; i < size; i++)
    {
        bucket[arr[i]]++; // incrementing the values of bucket for each respective
     value of arr
    }
    for (int i = 0, j = 0; i < max; i++)
    {
        for (; bucket[i] > 0; bucket[i]--)
        {
            arr[j] = i;
            j++;
        }
    }
}

// Doesnt work
void radix_sort(int arr[], int size)
{
    int large, pass, div, bktno, count[10], bkt[20][20];
    large = arr[0];
    for (int i = 0; i < size; i++)
    {
        if (arr[i] > large)
        {
            large = arr[i];
        }
    }
    int passes = 0;
    while (large > 0)
    {
        passes++;
        large = large / 10;
        div = 1;
        for (int i = 0; i <= passes; i++)
        {
            for (int j = 0; j <= 9; j++)
            {
                count[i] = 0;
            }
            for (int j = 0; j < size; j++)
            {
                bktno = (arr[i] / div) % 10;
                bkt[bktno][count[bktno]] = arr[j];
                count[bktno]++;
            }
            int j = 0;
            for (bktno = 0; bktno <= 9; bktno++)
            {
                for (int k = 0; k < count[bktno]; k++)
                {
                    arr[j] = bkt[bktno][k];
                    j++;
                }
            }
            div = div * 10;
        }
    }
}

int main()
{
    int arr[10] = {9, 0, 8, 8, 1, 4, 3, 1, 4, 7};
    int arrf[10] = {9.23, 0.35, 8.95, 8.86, 1.50, 4.22, 4.96, 1.00, 4.55, 7.2};

    int size = 10;
    int sizef = 10;

    printf("Enter the size of the Array : \n");
    scanf("%d", &size);
    // accept(arr, size);
    display(arr, size);
    bucket_sort_integers(arr, size);
    // radix_sort(arrf, sizef);
    display(arr, size);
    return 0;
}
```

Listing 1: Main.Cpp

8.3 C Output

```
Enter the size of the Array :
10

Enter the Array:
9 0 8 8 1 4 3 1 4 7

Displaying the Array
9 0 8 8 1 4 3 1 4 7

Displaying the Array
0 1 1 3 4 4 7 8 8 9
```

Listing 2: Output

9 Time Complexity

Worst Case Complexity: O(n2)
When there are elements of close range in the array, they are likely to be placed in the same bucket. This may result in some buckets having more number of elements than others. It makes the complexity depend on the sorting algorithm used to sort the elements of the bucket. The complexity becomes even worse when the elements are in reverse order. If insertion sort is used to sort elements of the bucket, then the time complexity becomes O(n2).

Best Case Complexity: O(n+k)
It occurs when the elements are uniformly distributed in the buckets with a nearly equal number of

elements in each bucket. The complexity becomes even better if the elements inside the buckets are already sorted. If insertion sort is used to sort elements of a bucket then the overall complexity in the best case will be linear ie. O(n+k). O(n) is the complexity for making the buckets and O(k) is the complexity for sorting the elements of the bucket using algorithms having linear time complexity at the best case.

Average Case Complexity: O(n)
It occurs when the elements are distributed randomly in the array. Even if the elements are not distributed uniformly, bucket sort runs in linear time. It holds true until the sum of the squares of the bucket sizes is linear in the total number of elements.

10 Conclusion

Thus, implemented bucket sort and Radix Sort Algorithm

11 FAQs

1. What are the applications of bucket sort? Bucket sort is used when:
    - Input is uniformly distributed over a range.
    - There are floating point values
2. Compare bucket sorting with other sorting methods?
    (a) Bucket sort is different from other algorithms that we have learnt, in that it is not a comparison based algorithm like Bubble or selection sort.
    (b) The advantage of bucket sort is that once the elements are distributed into buckets, each bucket can be processed independently of the others. This means that you often need to sort much smaller arrays as a follow-up step than the original array.
    (c) Another advantage of bucket sort is that you can use it as an external sorting algorithm. If you need to sort a list that is so huge you can't fit it into memory, you can stream the list through RAM, distribute the items into buckets stored in external files, then sort each file in RAM independently.
    (d) You can't apply it to all data types because you need a good bucketing scheme.
    (e) Worst Case time Complexity is same as that of Bubble, Selection, Insertion and Quick sort.
    (f) Average Case Time Complexity is better than above mentioned algorithms.
    (g) Best Case time complexity is $\omega(n+k)$ which is worse than above mentioned algorithms, but better than selection sort.