BANGALORE INSTITUTE OF TECNOLOGY

K.R.ROAD,V.V.PURAM,BANGALORE-560 004
(AFFILIATED TU VTU,BELGAVI)

Bangalore Institute of Technology

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

ANALYSIS &DESIGN OF ALGORITHM(BCS401)

Assignment On

Bingo sort & B-way sort

As per Choice Based Credit System(CBCS)
FOR IV SEMESTER AS PRESENTED BY VTU
ACADEMIC YEAR 2023-2024

Submitted by:-

BASAMMA HALLI

BHAGYASHRI

BHARAT GANGADHAR

BHARGAV B J

CHAITHRA P

Introduction to Bingo sort & B-way sort:

1. **Bingo sort, also known as random sort or stupid sort, is a highly inefficient sorting algorithm, generally used for educational purposes to demonstrate the importance of algorithm efficiency. It works by repeatedly shuffling the elements of the array until the array is sorted.**
2. **B-way sort, also known as multiway sort or polyphase sort, is an advanced sorting algorithm particularly useful for external sorting, where the dataset is too large to fit into main memory and must be managed on external storage like disks. It is designed to efficiently handle large volumes of data by dividing the dataset into smaller, manageable parts, sorting them individually, and then merging them in a systematic way.**

1.1 Explanation:

- **Function find_min:**

•Takes an array arr, a starting index start, and the size n of the array.
•Initializes min to the first element in the unsorted portion of the array.
•Iterates through the unsorted portion of the array to find the smallest element.
•Returns the smallest element found.

- **Function bingo_sort:**

•Takes an array arr and the size n of the array.
•Initializes sorted_count to 0, which keeps track of the number of sorted elements.
•While there are elements left to be sorted (sorted_count < n):
  •Finds the smallest element in the unsorted portion using find_min.
  •Iterates through the unsorted portion of the array and moves all occurrences of the smallest element to their correct positions.
  •Swaps each occurrence of the smallest element with the element at the sorted_count index.
  •Increments sorted_count for each occurrence moved.

- **Function print_array:**

•Takes an array arr and its size size.
•Iterates through the array and prints each element followed by a space.
•Prints a newline after printing all elements.

- **Main function:**

•Initializes an array arr and its size n.
•Prints the original array.
•Calls bingo_sort to sort the array.
•Prints the sorted array.

1.2 Algorithm for Bingo Sort:

//Input : enter the size of the array “n”, and enter the elements which has to be sorted.
//Output :an array of size “n” in a sorted order.
//Algorithm:

```
FUNCTION find_min(arr, start, n):
    min <- arr[start]
    FOR i <- start + 1 TO n - 1:
        IF arr[i] < min:
            min <- arr[i]
    RETURN min

FUNCTION bingo_sort(arr, n):
    sorted_count <- 0

    WHILE sorted_count < n:
        smallest <- find_min(arr, sorted_count, n)

        FOR i <- sorted_count TO n - 1:
            IF arr[i] == smallest:
```

```
 temp <- arr[i]
            arr[i] <- arr[sorted_count]
            arr[sorted_count] <- temp
            sorted_count <- sorted_count + 1

FUNCTION print_array(arr, size):
    FOR i <- 0 TO size - 1:
        PRINT arr[i], " "
    PRINT newline

MAIN:
    arr <- [4, 2, 5, 3, 2, 4, 1, 3]
    n <- length of arr

    PRINT "Original array: "
    CALL print_array(arr, n)

    CALL bingo_sort(arr, n)

    PRINT "Sorted array: "
    CALL print_array(arr, n)
```

1.3 Output:

main.c

Run

Output

Clear

```c
#include <stdio.h>

// Function to perform Bingo Sort
void bingoSort(int arr[], int n) {
    int max = arr[0];

    // Find the maximum value in the array
    for (int i = 1; i < n; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }

    int nextValue = max;
    int currentValue;

    do {
        currentValue = nextValue;
        nextValue = 0;

        for (int i = 0; i < n; i++) {
            if (arr[i] == currentValue) {
                arr[i] = nextValue;
                nextValue = currentValue;
            }
```

```
/tmp/dV3eeGUvp7.o
Original array:
29 10 14 37 13
Sorted array:
10 13 14 29 37


=== Code Execution Successful ===
```

2.1 Explanation:

B-way merge sort generalizes merge sort by dividing the array into B subarrays and recursively sorting each. The process involves splitting the array, sorting the subarrays, and then merging them.

1. Splitting: The `BWayMergeSort` function first checks if the array length is 1 or less, returning it if true. It then calculates the size of each subarray by dividing the array length by B. The array is split into B subarrays, each of which is recursively sorted using `BWayMergeSort`.

2. Merging: The `BWayMerge` function merges these B sorted subarrays. It initializes a min-heap to track the smallest elements of each subarray. The heap is populated with the first element of each subarray. The function then extracts the smallest element from the heap, appends it to the merged array, and inserts the next element from the same subarray into the heap. This continues until all elements are merged.

3. Min-Heap: The min-heap facilitates efficient merging by always providing the smallest element among the subarrays, ensuring the merged array is sorted.

This method ensures efficient sorting with a complexity of O(n log n) and is particularly useful for external sorting where B-way merging can reduce I/O operations.

2.2 Algorithm for B-way Sort:

//Input : Enter the size of an array and the elements of the array of size”n”.

//Output : The elements of array of size “n” in sorted order.

//Algorithm:

```
function BWayMergeSort(arr, B)
    if length(arr) <= 1
        return arr

    // Determine the size of each subarray
    subarray_size = ceil(length(arr) / B)

    // Divide the array into B subarrays
    subarrays = []
    for i = 0 to B-1
        start_index = i * subarray_size
        end_index = min(start_index + subarray_size, length(arr))
        if start_index < length(arr)
            subarray = slice(arr, start_index, end_index)
            subarrays.append(BWayMergeSort(subarray, B))

    // Merge B sorted subarrays
    return BWayMerge(subarrays)
```

```
function BWayMerge(subarrays)
    // Create a min-heap to store the smallest element from each subarray
    heap = MinHeap()
// Initialize the heap with the first element of each subarray
    for i = 0 to length(subarrays)-1
        if length(subarrays[i]) > 0
            heap.insert((subarrays[i][0], i, 0)) // (element, subarray_index, element_index)

    merged_array = []

    while heap is not empty
        // Extract the minimum element from the heap
        (min_element, subarray_index, element_index) = heap.extractMin()
        merged_array.append(min_element)

        // If there are more elements in the subarray, insert the next element into the heap
        if element_index + 1 < length(subarrays[subarray_index])
            next_element = subarrays[subarray_index][element_index + 1]
            heap.insert((next_element, subarray_index, element_index + 1))

    return merged_array

function MinHeap()
    // Implement a min-heap with standard operations: insert, extractMin, etc.
```

2.3 Output:

main.c

Run

Output

Clear

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void BWayMergeSort(int* arr, int n, int B);
void BWayMerge(int** subarrays, int* subarray_sizes, int B, int* merged_array,
    int total_size);
void MinHeapify(int* heap, int* indexes, int heap_size, int** subarrays, int*
    subarray_sizes);
void InsertMinHeap(int* heap, int* indexes, int* heap_size, int value, int
    subarray_index, int element_index);

int main() {
    int B = 3;  // Number of ways to split and merge
    int n;  // Size of the array

    // Input array size
    printf("Enter the size of the array: ");
    scanf("%d", &n);

    // Input array elements
    int* arr = (int*)malloc(n * sizeof(int));
    printf("Enter the elements of the array: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
```

```
/tmp/d8EGNbYePg.o
Enter the size of the array: 6
Enter the elements of the array: 42 34 98 56 21 87
Sorted array: 21 34 42 56 87 98
```

References

- **GeeksForGeeks**
- **Javatpoint**
- **Baeldung**