**Batch: C3 Roll No.: 16010123217**

**Experiment / assignment / tutorial No. 11**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:**  Implementation of sorting Algorithms. |

**Objective:** To Understand and Implement Bubble & Shell Sort

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| 4 | Demonstrate sorting and searching methods. |

**Books/ Journals/ Websites referred:**

1. *Fundamentals Of Data Structures In C –* Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications –* Jean Paul Tremblay,

Paul G. Sorenson

1. *Data Structures A Pseudo Approach with C –* Richard F. Gilberg & Behrouz A. Forouzan
2. *https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-sorting-algorithm/*

**Abstract**: (Define sorting process, state applications of sorting)

Ans. Sorting is a key operation in computer technology that involves arranging data in a specific order, either ascending or descending. This process enhances data management and retrieval efficiency, allowing for quicker searches and optimized information processing. There are various sorting algorithms, each designed to cater to different data structures and requirements.

Applications of Sorting Algorithms:

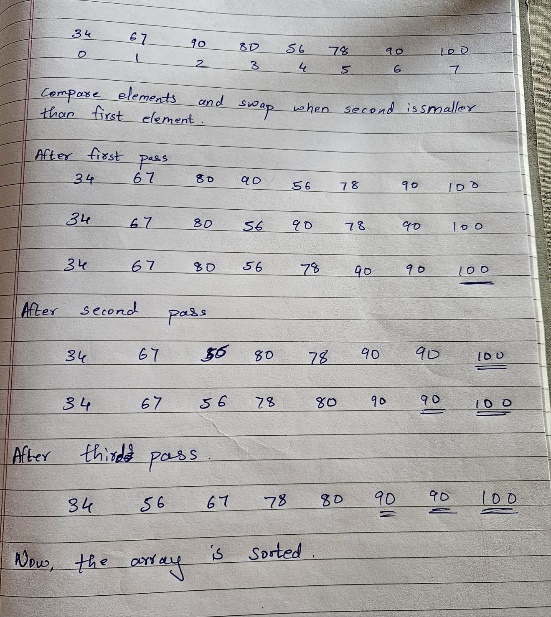
1. Data management: Sorting data makes it easier to search, retrieve, and analyze.
2. Database optimization: Sorting data in databases improves query performance. We typically keep the data sorted by primary index so that we can do quick queries.
3. Machine learning: Sorting is used to prepare data for training machine learning models.
4. Operating Systems: Sorting algorithms are used in operating systems for tasks like task scheduling, memory management, and file system organization.

**Example:**

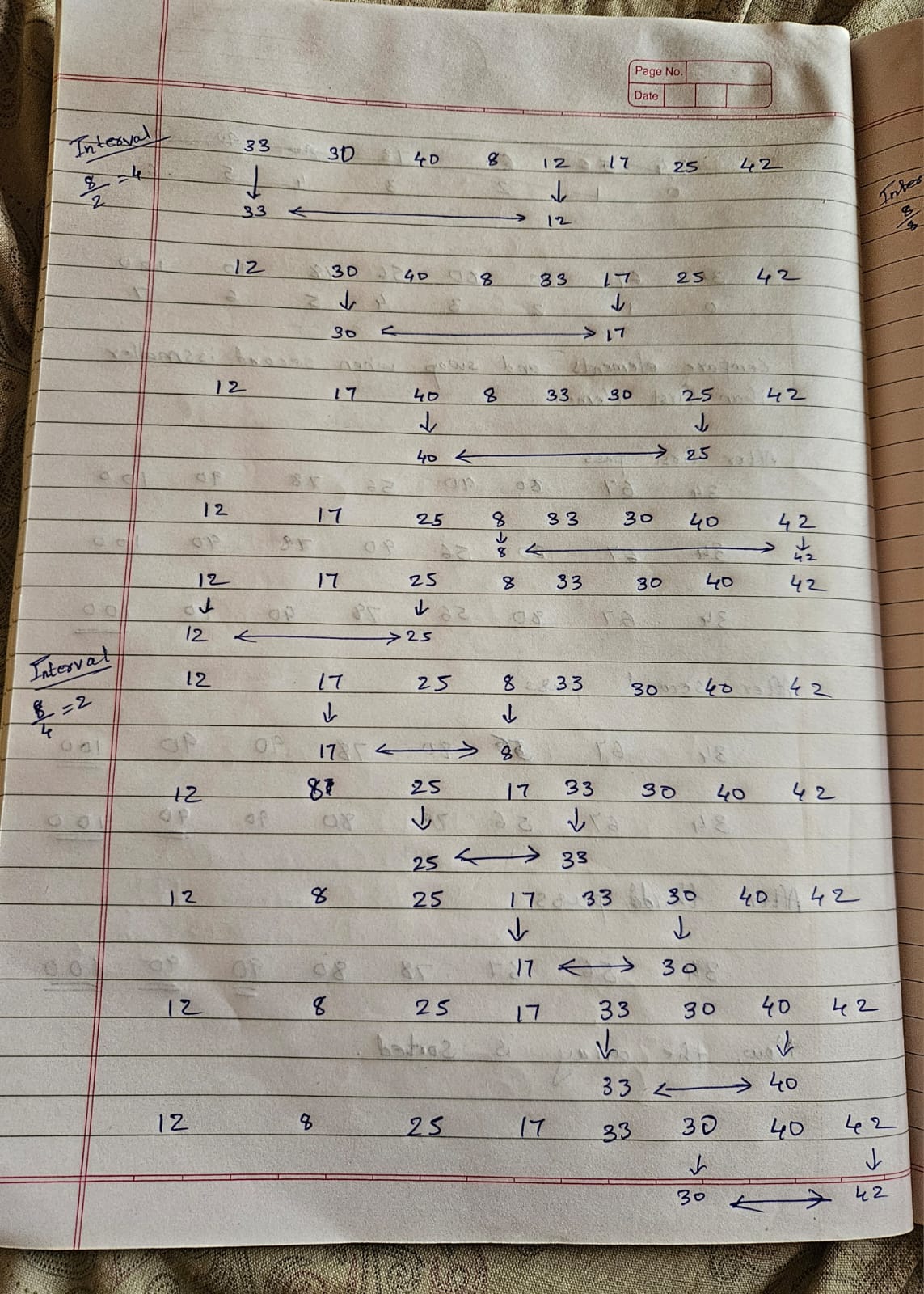
*Take any random unsorted sequence of numbers and solve by using the Bubble and Shell Sort. Clearly showcase the sorted array after every pass.*

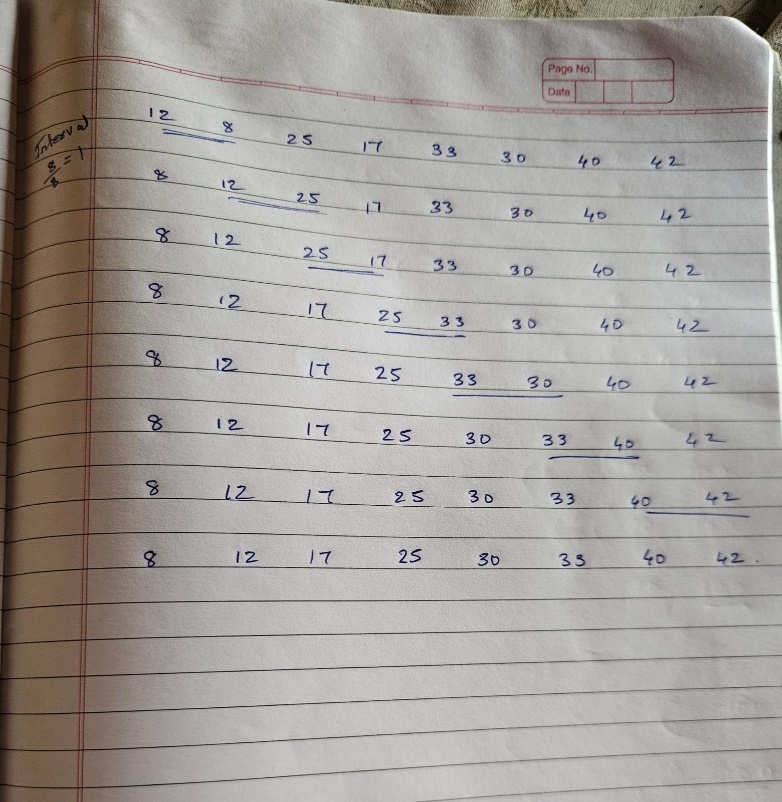
*The above is a pen-paper activity, take a picture of the solution and put it here.*

*Bubble Sort:*



Shell Sort:





**Algorithm for Implementation:**

Shell sort:  
ShellSort(a, n) // 'a' is the given array, 'n' is the size of array

for (gap = n/2; gap >= 1; gap /= 2)

for (j = gap; j < n; j += 1)

for (i = j - gap; i >= 0; i -= gap)

if (a[i + gap] > a[i])

break

else

swap(a[i + gap], a[i])

End ShellSort

Bubble Sort:  
BubbleSort(a, n) // 'a' is the given array, 'n' is the size of the array

for(int i = 0; i < n - 1; i++) {

for(int j = 0; j < n - 1 - i; j++) {

if(a[j] > a[j + 1]) {

swap(a[j], a[j + 1]);

}

}

}

End BubbleSort

**Program:**

**Bubble Sort**

#include <bits/stdc++.h>

using namespace std;

#define int long long

#define endl "\n"

const int MOD = 1e9 + 7;

const int INF = LLONG\_MAX >> 1;

signed main(){

    cout << "Enter the number of elements: ";

    int n;

    cin >> n;

    vector<int> a(n);

    cout << "Enter the elements:\n";

    for(int i = 0; i < n; i++) {

        cin >> a[i];

    }

    for(int i = 0; i < n - 1; i++) {

        for(int j = 0; j < n - 1 - i; j++) {

            if(a[j] > a[j + 1]) {

                swap(a[j], a[j + 1]);

            }

        }

    }

    cout << "Sorted elements:\n";

    for(auto i : a) {

        cout << i << " ";

    }

    cout << endl;

    return 0;

}

**Shell Sort**

#include <bits/stdc++.h>

using namespace std;

#define int long long

#define endl "\n"

const int MOD = 1e9 + 7;

const int INF = LLONG\_MAX >> 1;

signed main(){

    ios::sync\_with\_stdio(false); cin.tie(NULL);

    cout << "Enter the number of elements: ";

    int n;

    cin >> n;

    vector<int> arr(n);

    cout << "Enter the elements:\n";

    for(int i = 0; i < n; i++){

        cin >> arr[i];

    }

    for(int gap = n / 2; gap >= 1; gap /= 2) {

        for(int j = gap; j < n; j++) {

            for(int i = j - gap; i >= 0; i -= gap) {

                if(arr[i + gap] > arr[i]) {

                    break;

                } else {

                    swap(arr[i + gap], arr[i]);

                }

            }

        }

    }

    cout << "Sorted elements:\n";

    for(auto i : arr) {

        cout << i << ' ';

    }

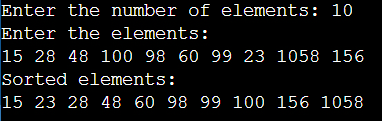
    cout << endl;

    return 0;

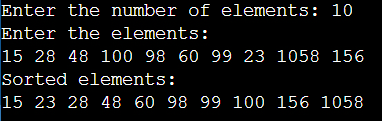
}

**Output screenshots:**

**1) Bubble Sort**

****

**2) Shell Sort**

****

**Conclusion:-**

From this, experiment we learned about different sorting techniques like shell sort and bubble sort and the differences between them and features of them.

**Post Lab Questions:**

1. Describe how shell sort improves upon bubble sort. What are the main differences in their approaches?

Ans.

* Efficiency: Shell sort is significantly more efficient than bubble sort, especially for larger datasets.
* Comparison Approach:
* Bubble Sort: Repeatedly compares adjacent elements, leading to O(n²) time complexity.
* Shell Sort: Compares elements at intervals (gaps), reducing the number of comparisons and swaps, leading to better performance (average O(n log n) or better with optimal gap sequences).
* Data Movement: Shell sort allows non-adjacent exchanges, which helps in moving elements to their correct position faster.

1. Explain the significance of the gap in shell sort. How does changing the gap sequence affect the performance of the algorithm?

Ans.

 **Role of Gap:** The gap determines which elements are compared and moved during sorting.

 **Impact on Performance:**

* Larger gaps allow elements to move closer to their final positions more quickly.
* Different gap sequences (e.g., Knuth, Hibbard) can lead to varying performance outcomes, with some yielding better sorting efficiency than others.

1. In what scenarios would you choose shell sort over bubble sort? Discuss the types of datasets where shell sort performs better.

Ans.

 **When to Choose Shell Sort:**

* For larger datasets where efficiency is critical.
* When partially sorted data is expected, as Shell sort performs better in such scenarios.

 **Better Performance Datasets:**

* Large datasets with random distributions.
* Data that requires frequent updates or insertions, as Shell sort handles such changes more efficiently than bubble sort.

1. Provide examples of real-world applications or scenarios where bubble sort or shell sort might be utilized, considering their characteristics.

Ans.

 **Bubble Sort Applications:**

* Educational purposes for teaching sorting concepts.
* Small datasets where simplicity is prioritized over efficiency.

 **Shell Sort Applications:**

* Sorting large datasets in database systems or data analytics tools.
* Real-time systems where moderate performance is necessary without complex implementations, such as embedded systems.