Batch: B2 Roll No.: 16010124107

Experiment / assignment / tutorial No. 10

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
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan

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

Sorting simply means rearranging the elements of a data structure in a specific order. The most common orders are ascending or descending, although it is possible to have custom comparators that sort in any other orders as need be.

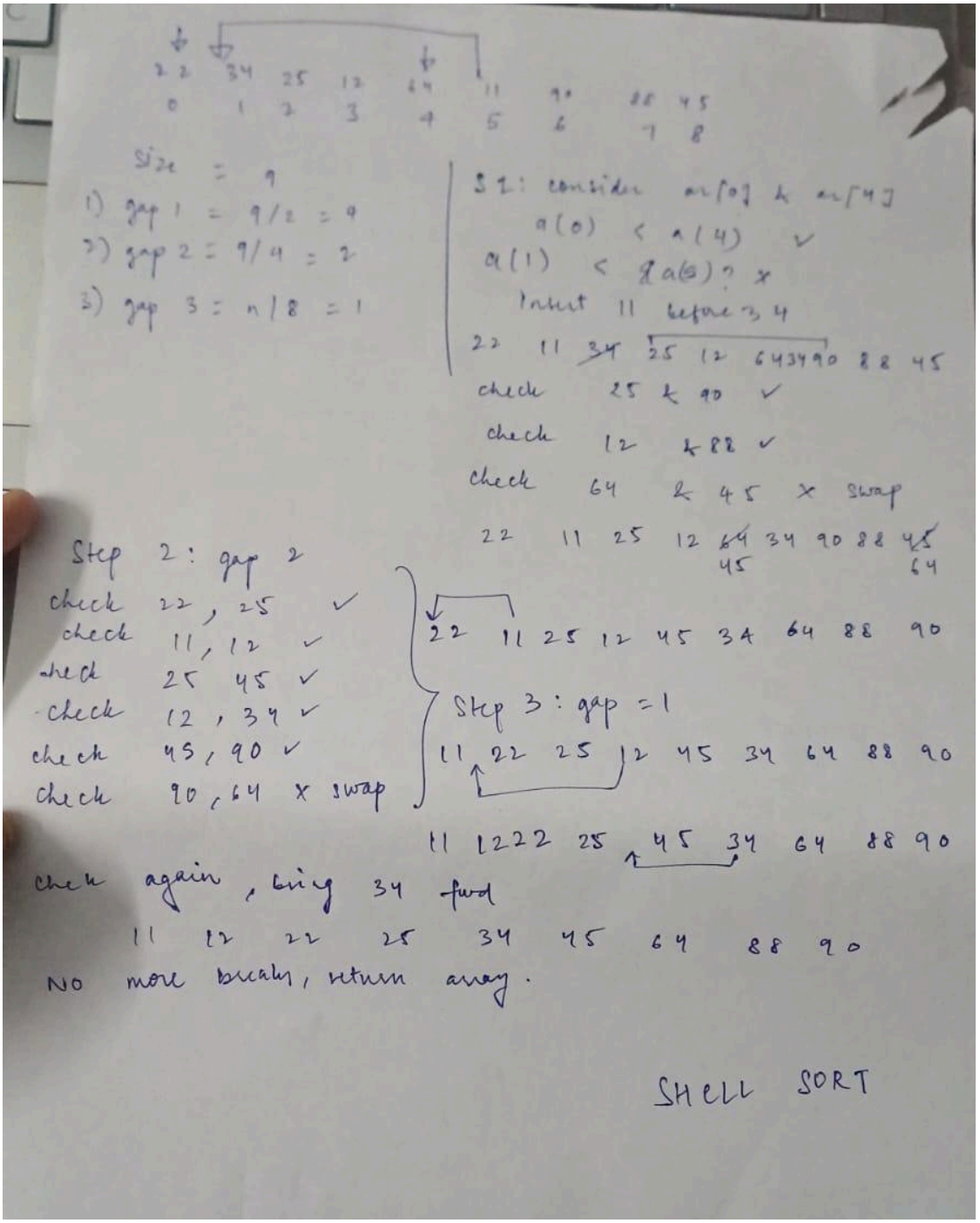
Applications of sorting:

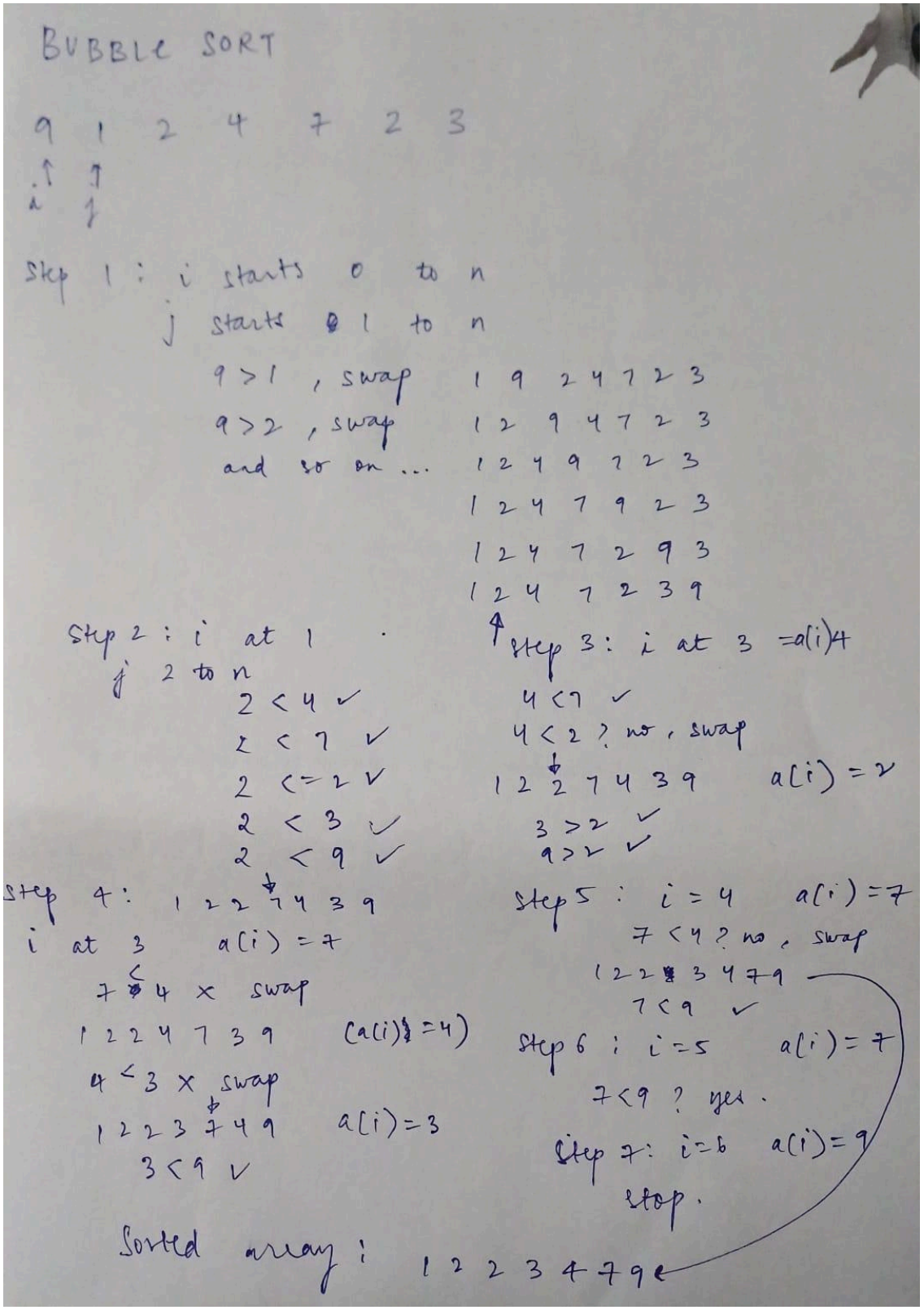
1. Searching for an element in a sorted array is faster
2. Sorted arrays give the ranks of students based on marks
3. Sorted strings are used in dictionaries to maintain alphabetical order
4. Sorting is useful in algorithms where binary search on answer is implemented.

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.


22 34 25 12 64 11 90 88 45
0 1 2 3 4 5 6 7 8
Size = 9
1) gap 1 = 9/2 = 4
2) gap 2 = 9/4 = 2
3) gap 3 = n/8 = 1
S1: consider a[0] & a[4]
a(0) < a(4) ✓
a(1) < a(5)? x
Insert 11 before 34
22 11 34 25 12 64 34 90 88 45
check 25 & 90 ✓
check 12 & 88 ✓
check 64 & 45 x swap
22 11 25 12 64 34 90 88 45
45 64
Step 2: gap 2
check 22, 25 ✓
check 11, 12 ✓
check 25 45 ✓
check 12, 34 ✓
check 45, 90 ✓
check 90, 64 x swap
22 11 25 12 45 34 64 88 90
Step 3: gap = 1
11 22 25 12 45 34 64 88 90
11 1222 25 45 34 64 88 90
check again, bring 34 fwd
11 12 22 25 34 45 64 88 90
No more breaks, return away.
SHELL SORT



Algorithm for Implementation:

Bubble sort

1. Start
2. Run a loop from i= 0 to n where n is the size of array
3. Run a loop inside this from j=i to n
4. Whenever a mismatched order is encountered, swap elements
5. Continue till the end, n^2 times.
6. End

Shell sort

1. Start
2. Choose a gap sequence
3. Sort elements at each gap
4. Reduce the gap and repeat until the gap becomes 1.
5. End

Program:

```
#include <iostream>

#include <vector>

using namespace std;


void bubble(vector<int>& arr) {

    int n = arr.size();


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

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

            if(arr[i]>arr[j]){

                swap(arr[i],arr[j]);

//can also use a third variable to sort

//int c=arr[i];

//arr[i]=arr[j];

//arr[j]=c;

            }

        }

    }

}


void print(const vector<int>& arr) {

    for (int num : arr)

        cout << num << " ";

    cout << endl;
```

```
}


int main() {
    cout << "Enter size of array, followed by its elements\n";
    int n;
    cin >> n;
    vector<int>arr(n);
    for(int &i:arr){
        cin >> i;
    }
    cout << "Before sorting: \n";
    print(arr);
    bubble(arr);
    cout << "After sorting: \n";
    print(arr);


    return 0;
}
```

Shell sort:

```
#include <iostream>
#include <vector>
using namespace std;

void shell(vector<int>& arr) {
```

```
    int n = arr.size();

    for (int gap = n / 2; gap > 0; gap /= 2) {

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

            int temp = arr[i];
            int j = i;

            while (j >= gap && arr[j - gap] > temp) {
                arr[j] = arr[j - gap];
                j -= gap;
            }

            arr[j] = temp;
        }
    }
}

void print(const vector<int>& arr) {
    for (int num : arr)
        cout << num << " ";
    cout << endl;
}

int main() {
    cout << "Enter size of array, followed by its elements\n";
    int n;
    cin >> n;
    vector<int>arr(n);
    for(int i:arr){
        cin >> i;
    }
    cout << "Before sorting: \n";
    print(arr);
    shell(arr);
    cout << "After sorting: \n";
    print(arr);
```

```
    return 0;
}
```

Output screenshots:

Bubble:

```
g++ sort.cpp -o sort } ; if ($?) { .\sort }
Enter size of array, followed by its elements
5
23 54 38 46 19
Before sorting:
23 54 38 46 19
After sorting:
19 23 38 46 54
PS C:\Users\syeda\OneDrive\Desktop\personal>
```

Shell:

```
g++ sort.cpp -o sort } ; if ($?) { .\sort }
Enter size of array, followed by its elements
4
23 213 12 31
Before sorting:
23 213 12 31
After sorting:
12 23 31 213
PS C:\Users\syeda\OneDrive\Desktop\personal>
```

Conclusion:-

Shell sort improves insertion sort by comparing distant elements. It reduces swaps and performs faster on medium-sized lists.

Bubble sort repeatedly swaps adjacent out-of-order elements. It's simple but slow for large datasets.

Post Lab Questions:

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

   Shell sort compares elements far apart, not just adjacent ones. It reduces large gaps first, then smaller ones for efficiency. Bubble sort only swaps neighboring elements each pass. Thus, Shell sort needs fewer passes and fewer swaps overall.

   The complexity of Bubble Sort is $O(n^2)$, while Shell Sort is $O(n \log n)$ on average.

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

   The gap controls how far apart elements are compared. Smaller gaps refine sorting after larger gaps reduce disorder. Efficient gap sequences greatly improve performance. Essentially with each pass, we are halving the gaps, hence reducing time by half with every iteration.

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

Shell sort is preferred for medium-sized, partially sorted datasets. It's faster than bubble sort when data is large or slightly unsorted.

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

Bubble sort: used in teaching sorting basics or tiny datasets.

Shell sort: used in embedded systems or moderate data sorting tasks.